Predicting CNN Training Latency on Cloud GPUs without Disclosing Model Architecture

Yoonseo Hur[1], Sungjae Lee[2], Seungwoo Kum[3], Kyungyong Lee[4*]

[1]Department of Computer Science, Kookmin University, Seoul, South Korea.
[2]Department of Computer Science and Engineering, The Ohio State University, Columbus, Ohio, USA.
[3]Media Research Center, Korea Electronic Technology Institute (KETI), Seongnam-si, Gyeonggi-do, Republic of Korea.
[4]Department of Data Science, Hanyang University, Seoul, South Korea.

*Corresponding author(s). E-mail(s): kyungyong@hanyang.ac.kr;
Contributing authors: yoonseo@kookmin.ac.kr; lee.10952@osu.edu; swkum@keti.re.kr;

Abstract

Training distributed deep neural networks (DNNs) demands substantial computational resources, leading to the increasing adoption of public cloud platforms as primary training environments. However, the rapid evolution and diversity of cloud-based hardware offerings pose significant challenges for algorithm developers in maintaining up-to-date and efficient training systems. As a result, there is a growing need for cloud service providers to offer optimized training environments that reduce the operational overhead for end users. To address this need, we present PROFET, a system that predicts the training latency of arbitrary Convolutional Neural Network (CNN) implementations across a wide range of GPU types and mini-batch sizes. In contrast to prior approaches, PROFET does not require access to CNN architecture details or source code, making it particularly well-suited for integration by public cloud providers. To improve prediction accuracy, we introduce a novel operation-name clustering heuristic that effectively resolves naming inconsistencies in computational graphs where semantically similar operations are labeled differently. Extensive experimental evaluations demonstrate that PROFET achieves superior prediction accuracy compared to existing state-of-the-art methods, with improvements ranging from 17% to 62%. Furthermore, we assess the practical utility of PROFET by comparing its recommended GPU selection against an *Oracle* strategy that assumes perfect knowledge of actual training latencies. PROFET attains comparable performance, with only marginal differences of 1.2% in average latency and 0.4% in training cost, highlighting its potential for real-world deployment in cloud-based DNN training systems.

Keywords: CNN training, performance model, GPU, cloud computing

1 Introduction

The success of deep learning across a range of domains, such as image recognition, natural language processing, and autonomous systems, can be attributed to advancements in DNN algorithms [1], the availability of powerful programming interfaces and development frameworks [2–4], and the proliferation of specialized hardware,

particularly GPUs. Among various DNN architectures, CNNs [5–7] are widely adopted in applications including object detection for autonomous vehicles and large-scale image classification. CNN training, in particular, demands substantial computational power due to the complexity and depth of modern network architectures, often requiring significant parallelism to process large volumes of data efficiently [8, 9]. Although newer architectures like Transformers [10] have gained popularity, CNNs remain fundamental and widely utilized in production-scale visual recognition tasks. Therefore, building a cost-effective training environment specifically optimized for the unique operator patterns of CNNs is crucial for improving developer productivity.

Although GPUs offer massive parallel processing capabilities, deploying and maintaining a dedicated GPU cluster can be prohibitively expensive. As a result, public cloud platforms have emerged as a viable alternative, enabling developers to leverage GPU instances on demand. This pay-as-you-go model allows for flexible scaling and reduces the upfront infrastructure investment, making it a compelling choice for CNN training workloads. The rate of innovation in cloud computing is accelerating, with public cloud vendors continuously introducing new types of services and instance families. For developers of deep learning algorithms, keeping pace with these frequent updates and understanding their implications for training performance and system integration pose a significant challenge. As a result, it is advantageous for cloud service providers to offer natively optimized environments for deep learning workloads. Such environments can abstract away low-level system details, enabling algorithm developers to concentrate on core model design and application logic rather than infrastructure management.

The platform-as-a-service (PaaS) model in cloud computing has been promoted as a way to reduce the operational overhead of managing development environments, thereby allowing developers to focus on core software tasks. In the context of deep learning, public cloud vendors offer pipeline platforms such as AWS SageMaker and Google Cloud Datalab to streamline training and deployment. However, these services remain suboptimal, as users are still required to manually select the types and numbers of cloud instances for training tasks. Moreover, the optimal configuration for DNN training varies significantly depending on model architecture, dataset characteristics, and hyperparameter settings, making the management of deep learning platforms both complex and error-prone.

To better understand the performance characteristics of CNN training on public cloud GPU instances, we conducted a series of measurements using a variety of GPU-equipped cloud instances. As shown in Figure 2, there exist substantial performance variations across different CNN architectures and GPU types, with up to a five-fold difference in training time between the best- and worst-performing instance types. Furthermore, training latencies do not scale linearly with changes in mini-batch sizes, further complicating performance prediction.

Despite these considerable differences, developers of deep learning algorithms often lack the necessary insights into system-level performance, leading to missed opportunities for training optimization. This issue is exacerbated by the fact that most DNN developers are not system experts and may incur significant overhead when required to manage low-level infrastructure configurations. As such, it becomes imperative for cloud service providers to offer optimized training environments that account for the specific implementation characteristics of the training workloads and alleviate the burden on algorithm developers.

Habitat [11], Paleo [12], NeuralPower [13], and MLPredict [14] propose algorithms that predict the training time of CNN models on GPUs. These systems utilize both the internal architecture of the CNN implementation and the hardware characteristics of the target GPU as input features, thereby adopting a *white-box* modeling approach. While these methods demonstrate the feasibility of accurately predicting CNN training latency across various GPU types, they are not well-suited to deployment scenarios in which deep learning algorithm developers and platform providers, such as public cloud vendors, are separate entities.

In public cloud settings, CNN architectures are often considered proprietary intellectual property and are unlikely to be disclosed to service providers. Existing performance prediction methods require full access to internal model details—including layer configurations, filter sizes, connectivity patterns, and weight

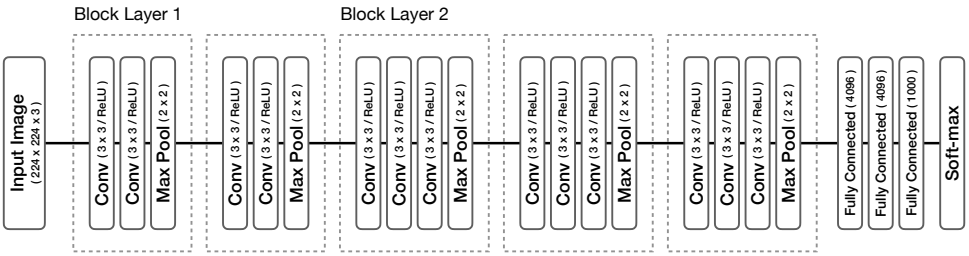


Fig. 1: A general architecture of a CNN model with an example of VGG16

structures—effectively demanding that algorithm developers surrender their entire model design to the cloud provider. This creates a fundamental barrier to adoption, motivating the need for a prediction method that can operate without any exposure of architectural internals.

To address the limitations of prior work, we propose **PROFET**, a system that predicts the training latency of arbitrary CNN implementations without requiring access to their internal architectures. This represents a fundamental shift from traditional *white-box* approaches and enables performance prediction in scenarios where model details are considered confidential as is often the case in public cloud environments. By decoupling performance modeling from architectural exposure, PROFET is well-suited for deployment by cloud service providers who lack visibility into users' proprietary models.

PROFET achieves this by leveraging abstracted profiling data collected during CNN training as input to a machine learning-based prediction model. To further improve prediction accuracy, the system introduces novel heuristics for modeling and clustering semantically equivalent operations that may appear under different names across implementations. Additionally, PROFET supports predictions across a wide range of GPU types and mini-batch sizes, making it applicable to diverse training scenarios in cloud environments.

Comprehensive experimental evaluations demonstrate that PROFET outperforms recent prior works, including Paleo [12], Habitat [11], and MLPredict [14], with improvements in prediction accuracy of 17%, 38%, and 62%, respectively. Based on these predictions, PROFET can recommend the most suitable GPU instance for training a given CNN model. Compared to the *Oracle* strategy, which assumes perfect knowledge of actual training latencies, PROFET achieves nearly equivalent results, with only 1.2% higher average training latency and 0.4% higher cost, demonstrating its practical utility in real-world cloud deployment settings.

In summary, the major contributions of this paper are as follows:

- We identify the need for *black-box* performance estimation of CNN training latency in public cloud environments, where access to internal model architectures is restricted.
- We design and implement a training latency prediction system capable of accurately modeling performance across diverse GPU types and mini-batch sizes.
- We propose a novel operation-name clustering heuristic that enhances prediction accuracy by resolving semantic inconsistencies in operator naming.
- We demonstrate the practical applicability of the proposed system by recommending cost-effective and low-latency GPU instances, achieving near-optimal performance compared to an *Oracle* approach.

2 Model Training on Cloud

CNN model training can take from several days to months, depending on architecture complexity and configuration [9, 15, 16]. Factors such as model size, dataset, and hardware influence training time, highlighting the need to understand performance variability for efficient environment setup.

2.1 Overview of CNN

CNNs are widely used for processing visual data such as images and videos. A typical CNN includes an input layer, an output layer, and multiple

hidden layers composed of convolution, activation, and pooling operations. These layers extract and abstract features hierarchically. Early models like LeNet-5 [5] had about 60,000 parameters, while succeeding architectures such as AlexNet [7], VGG [6], ResNet [17], and Inception [18] are much deeper and more complex, often exceeding hundreds of millions of parameters.

Figure 1 shows the architecture of VGG16 [6], which classifies 224 × 224 RGB images. The model includes convolution, ReLU activation [19], and max-pooling layers. During training, multiple images are processed in mini-batches, and weights are updated via backpropagation. The mini-batch size impacts both accuracy and training latency.

2.2 GPU Instances on Cloud

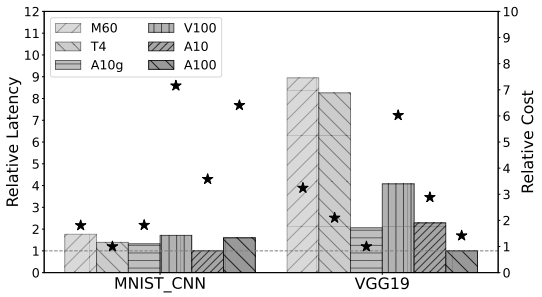
Due to the high computational demands of CNN training, accelerators such as GPUs, FPGAs, and ASICs are used to speed up training. GPUs are the most widely used, offering strong performance and mature software support. To avoid the high cost of dedicated hardware, many developers use GPUs via public cloud services.

Major cloud providers (e.g., AWS, Google Cloud, Azure) offer a wide range of GPU instances, from older models like K80 and P4 to recent ones like A100, L4, A10G, L40s, and H100. These GPUs differ in architecture, memory, bandwidth, and supported features, leading to significant differences in training performance and cost. Additionally, pricing varies widely, making it difficult for developers to choose the optimal GPU instance type.

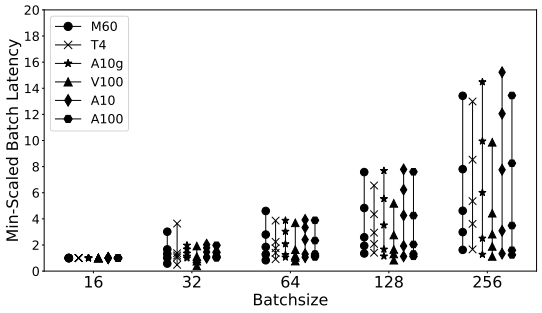
2.3 Understanding CNN Training Performance

The diversity of CNN architectures, configurations, and GPU cloud instances results in large variations in training time. We conducted experiments to analyze differences in latency and cost. Key findings are shown in Figure 2.

Figure 2a illustrates the training latency differences across cloud GPU instances for two CNN models: `MNIST_CNN` and VGG19 [6]. The input sizes were 32×32 for `MNIST_CNN` and 128×128 for VGG19, both using a mini-batch size of 16. Training latencies are shown as bars on the primary vertical axis, while the relative cost to complete



(a) Different model



(b) Different mini-batch size

Fig. 2: Performance variation of CNN model training using GPU cloud instances

training is shown as stars on the secondary vertical axis. Latencies for each instance type are normalized to the minimum value, with a horizontal line at 1.0 indicating the best-performing case. The GPUs are ordered as *M60*, *T4*, *A10g*, *V100*, *A10*, and *A100*.

When training the `MNIST_CNN` model, the A10 GPU achieves the lowest latency, while the T4 offers the best cost efficiency. For the larger VGG19 model, the A100 provides the fastest training, whereas the A10g is the most cost-effective, despite requiring nearly twice the training time of the A100. These results highlight that latency and cost efficiency vary significantly depending on the model characteristics.

In CNN training, the mini-batch size is a key hyperparameter that impacts training latency. It represents the number of samples processed simultaneously, and latency generally increases with batch size. A linear scaling is often expected. For instance, doubling the batch size from 32 to 64 should roughly double the latency. To examine this effect, we measured training latency for

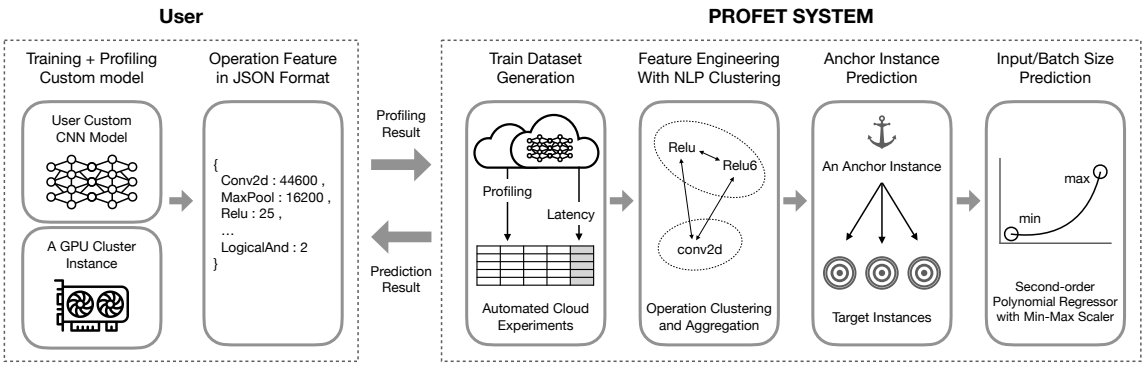


Fig. 3: High level description of the proposed system and working scenario

batch sizes of 16, 32, 64, 128, and 256 using the same model and instance. Figure 2b shows the relative latency ratios across different batch sizes. To compute the latency ratios, we used the training latency with a batch size of 16 as the baseline (1.0). Latencies for larger batch sizes were expressed relative to this baseline. For each GPU instance type, we report the minimum, 25th percentile, median, 75th percentile, and maximum of the latency ratios.

The results in Figure 2b show that the relationship between batch size and training latency is not linear and varies across models and instance types. For example, when training MobileNetV2 on a *V100* instance, increasing the batch size from 16 to 256 (16×) results in only a 1.45× increase in latency. In contrast, for VGG13 on a *T4* instance, the same batch size increase leads to a 13.5× latency increase. Notably, the *V100* instance consistently exhibits less latency growth compared to other GPUs. These observations suggest that latency does not scale linearly with batch size and that simple regression models may be insufficient for accurate performance prediction, motivating the need for more advanced approaches.

In summary, Figure 2 demonstrates substantial variability in CNN training latency among GPU-powered cloud instances, attributable to diverse model architectures and batch sizes. Given the rapid introduction of new cloud services, resources, and pricing mechanisms [20–23], CNN algorithm developers face considerable difficulty in promptly integrating these advancements into their training environments. Moreover, the assessment of CNN model performance becomes arduous when considering every instance type that necessitates a custom device library. Consequently, providing estimated performance guidance for custom CNN algorithm implementations on various cloud instance types is crucial for establishing optimal training environments.

3 Modeling CNN Training Time

The objective of PROFET is to accurately predict the training time for arbitrary CNN implementations on GPUs offered by public cloud service providers, while simultaneously minimizing the exposure of implementation details. This minimization is particularly critical in a public cloud environment where establishing a CNN development environment with GPUs demands significant system operation effort, which can be challenging for algorithm developers. Consequently, cloud service providers should assume responsibility for maintaining and operating an optimal development environment, aligning with the principles of cloud computing [24] as exemplified by serverless computing [25–27]. PROFET achieves this objective by leveraging abstracted operational information.

A set of characteristic vectors, denoted by $\boldsymbol{X}$, is generated from offline experiments to construct a prediction model. Here, $\boldsymbol{X}$ comprises N workloads, each represented by a feature vector of dimension D. Consequently, the dimension of $\boldsymbol{X}$ is $N \times D$. Each workload scenario is defined as $\boldsymbol{x}_i$, where $i \in \{1, \ldots, N\}$. To represent the j-th feature of workload i, where $j \in \{1, \ldots, D\}$, we use x_{ij}.

To generate diverse CNN workloads, we varied CNN training scenarios based on GPU instance types (G), model architectures (M), and batch sizes (B). The Cartesian product of G, M, and B yielded all possible workload scenarios, denoted as $G \times M \times B = \{(g, m, b) : g \in G, m \in M, b \in B\}$. However, not all combinations within $G \times M \times B$ were executable due to hardware or model constraints. After filtering, the final dataset was constructed with a cardinality of N. Each workload $\boldsymbol{x}_i$, a vector of length D, was derived from the underlying profiler [28] by extracting 65 aggregated high-level operations across these N executable workloads.

Figure 3 illustrates the end-to-end operational scenario of PROFET. A client initiates the process by executing their custom CNN implementation on a chosen anchor instance g_a with profiling enabled (Section 3.1). The resulting profiled output, which intentionally excludes internal model architecture details, provides high-level categorical latency information for various operations. This profiled result is then submitted to the PROFET prediction model.To evaluate PROFET, we constructed a comprehensive dataset with $N = 3,060$ data points, encompassing all possible combinations of 17 CNN architectures, 6 input resolutions, 5 batch sizes, and 6 GPU instances.

Utilizing these profile results, PROFET predicts training time across various scenarios. This involves: a) profiling CNN characteristics to generate a training dataset (Section 3.1); b) an operation name clustering module (Section 3.2) designed to group similar operations and thus enhance prediction accuracy; and c) latency prediction modules that model training latency across different instance types (Section 3.3.1) and batch sizes (Section 3.3.2). PROFET introduces novel prediction models, including DNN and scale-based second-order polynomial models, which are tailored to enhance prediction accuracy based on specific prediction scenarios.

3.1 Extracting CNN Characteristics on Cloud

In a PaaS cloud model, CNN algorithm developers are relieved of the burden of operating cloud infrastructures and DNN development platforms, as these responsibilities are borne by the cloud service provider. To furnish efficient deep learning platforms, cloud vendors must possess a comprehensive understanding of the distinct characteristics of the workloads submitted by their clients. However, within the PaaS paradigm, most CNN algorithm developers are disinclined to share the source code or internal architecture of their models.

PROFET uses only (Operation, Aggregated Time)

Operation	Operation Details				Aggregated Time (ms)
	Layer Name	Output Tensor	MEM Request	Time (ms)	
Conv2D	conv2d_0	[32, 96, 224, 224]	616562 KB	50	286
	conv2d_1	[32, 256, 112, 112]	411041 KB	45	
	conv2d_2	[32, 512, 58, 58]	220463 KB	39	
	...	...	...	...	
ReLU	activation_0	-	-	11	26
	activation_1	-	-	7	
	activation_2	-	-	3	
	...	...	...	...	
MaxPool	max_pooling2d_0	[32, 96, 112, 112]	154140 KB	6	14
	max_pooling2d_1	[32, 256, 56, 56]	102760 KB	4	
	max_pooling2d_2	[32, 512, 29, 29]	100663 KB	2	
	...	...	...	...	
and more operations (Conv2DBackpropFilter / MatMul / Softmax ...)					

Fig. 4: An example of profiling data generated from AlextNet model training using TensorFlow profiler

Consequently, previous work that relies on internal model architectures for predicting DNN training latency [12–14] cannot be directly applied in a PaaS environment. Therefore, PROFET is designed to characterize workloads while minimizing the disclosure of internal model architecture, thereby enabling the prediction of training latency for arbitrary CNN implementations within a public PaaS model.

Rather than referencing the internal model architecture to characterize CNN workloads, PROFET proposes leveraging performance metrics generated during the training phase. These metrics are furnished by readily available DNN programming platform profilers, such as TensorFlow [3], MXNet [29], Torch [2], and Theano [4]. Although the specific information provided by these metrics may exhibit slight variations across platforms, they consistently yield response times for core DNN-specific operations.

To better understand the performance metrics provided by a DNN development platform, Figure 4 presents an example of a CNN model profiling result generated by the TensorFlow profiler [28]. The profiling output typically comprises fields such as operation, operation detail, and latency. The `operation` field indicates the method utilized in the source code as specified by the development platform. The `operation detail` field contains extensive information about the

CNN model, including the method, layer location, input and output tensor sizes, memory usage, and other specifics. This detailed information could potentially reveal the internal architecture of a custom implementation, thereby allowing a model to be reconstructed.

PROFET proposes utilizing only the `operation` field and its corresponding aggregated time field as features to represent CNN workloads. Previous work by Hafeez et al. has already demonstrated that profiled results can effectively capture the characteristics of arbitrary CNN models [30]. By intentionally obscuring operation details, PROFET aims to alleviate the burden on CNN algorithm developers when sharing workload characteristics with public cloud PaaS vendors, ultimately enabling the provision of an optimal development environment for any CNN workload type. The privacy guarantee offered by PROFET is scoped to the minimization of architectural exposure, rather than absolute confidentiality. PROFET shares only operation names and their aggregated execution times, while withholding layer-level details such as filter sizes, connectivity patterns, and weight structures. Since the feature vector represents the sum of execution times per operation type, sequential and structural information — such as layer ordering, depth, and connectivity — is inherently removed during aggregation. Consequently, while coarse-level information about which operation types are present may be inferred, the precise architectural composition and model family cannot be reliably reconstructed from the aggregated feature vector alone. A formal quantification of information leakage, including whether an adversary could classify model family or estimate parameter scale from PROFET's feature vectors, remains a direction for future work.

Utilizing operation names and aggregated latencies as features, we construct input vectors $\boldsymbol{X}$, with the corresponding batch latencies serving as output vectors $\boldsymbol{Y}$. For a given CNN workload i, the profiling features are represented as $\boldsymbol{x}_i$, and y_i denotes the measured batch latency, where $i \in \{1, \ldots, N\}$. The term x_{ij} refers to a specific feature value j of workload i, where $i \in \{1, \ldots, N\}$, $j \in \{1, \ldots, D\}$, and D represents the dimension of the scalar feature space.

Generating feature vectors $\boldsymbol{X}$ from a deep learning platform inherently incurs additional overhead, which can influence the measured batch latency $\boldsymbol{Y}$. To mitigate this influence, we conducted two distinct sets of experiments for each workload i: one with profiling enabled and one with profiling disabled. In profiling experiments, we collected the characteristic vectors $\boldsymbol{X}$. Conversely, to obtain accurate batch latency values for $\boldsymbol{Y}$, we measured the actual training latency with profiling disabled, thereby avoiding any overhead introduced by the profiling process itself. Feature vectors $\boldsymbol{X}$, however, were collected with profiling enabled, as profiling is requisite for extracting operation-level statistics from the model. This clear separation between the generation of $\boldsymbol{X}$ and $\boldsymbol{Y}$ directly reflects the intended usage of PROFET: profiling is temporarily enabled solely to obtain features, while the model predicts training latency under the assumption that profiling is not active during actual execution.

To ensure the stability of the feature vectors $\boldsymbol{X}$ while minimizing profiling overhead, we established a measurement protocol within a 2-epoch execution for each workload. All measurements were conducted on dedicated cloud instances to eliminate external performance interference. We implemented a 2-epoch warm-up period to account for transient overheads, such as cuDNN auto-tuning and input pipeline stabilization. Following this stabilization, at the 1.5-epoch mark, profiling was conducted for a single batch to capture representative operation-level statistics. As summarized in Table 1, profiling increases the average end-to-end training latency across the 17 evaluated CNN models from 49.23 seconds to 50.04 seconds. This sub-second increase indicates that the overhead introduced by profiling is negligible, both in absolute terms and relative to the total training duration.

3.2 Feature Engineering by Name Clustering

During the profiling step, an offline execution of a model i generates its characteristic vector $\boldsymbol{x}_i$. Due to the distinct model architectures of various implementations, the feature set for each model can differ, even when some features exhibit high similarity. For instance, the activation functions *ReLU* [19] and *ReLU6* are analogous, differing only by the maximum value imposed by *ReLU6*.

Table 1: End-to-End Profiling Overhead

Model	E2E Latency (Profiling On)	E2E Latency (Profiling Off)
CIFAR10_CNN	26.34	25.65
EfficientNetB0	65.25	64.17
FLOWER_CNN	14.78	14.08
InceptionV3	61.33	60.08
LeNet5	13.44	12.78
MNIST_CNN	16.85	16.14
MobileNetV2	38.52	37.68
ResNet18	34.28	33.49
ResNet34	49.80	49.00
ResNet50	67.21	66.14
ResNetSmall	16.47	15.70
VGG11	52.40	51.67
VGG13	68.22	67.52
VGG16	79.81	79.37
VGG19	86.59	85.95
VGGSmall	30.46	29.60
Xception	83.37	82.53
Avg.	**50.04**	**49.23**

While *VGG16* utilizes a *ReLU* activation function, *MobileNetV2* [31] employs *ReLU6*. In the profiling stage, *ReLU* and *ReLU6* are treated as distinct features, precluding the capture of their inherent similarity. This scenario can degrade the prediction accuracy of PROFET, particularly when a model generates a feature vector containing unique features not observed during the training phase.

To address this challenge and incorporate operational similarity into PROFET's modeling, we propose measuring the quantitative similarity of operation names. This similarity value is then utilized as a metric to cluster analogous features. The clustered features are subsequently aggregated into a single feature by summing their profiled values.

3.2.1 Measurement of Operation Name Similarity

The operation names within profiling results represent distinct computational procedures. Therefore, we propose quantifying the similarity of operation name strings to ascertain the suitability of arbitrary operation combinations for clustering, utilizing the Levenshtein algorithm [32]. This algorithm quantifies the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one word into the other. A lower Levenshtein distance inherently implies a higher degree of similarity. For instance, to transform the operation name *ReLU* into *ReLU6*, only the character '6' must be inserted into *ReLU*, yielding a Levenshtein distance of one. Conversely, the distance between *ReLU* and *Conv2D* is six, as four characters from *ReLU* require replacement, and two additional characters ('2D') must be inserted to achieve identity. Employing the Levenshtein algorithm enables a quantitative comparison of operation-name similarity, thereby enhancing the clustering and merging of analogous operations.

Subsequent to its calculation, the Levenshtein distance is normalized by dividing it by the maximum string length between the two compared strings. This normalization, based on string length, prevents shorter strings from inherently exhibiting smaller distances, as such strings naturally require fewer edits for identity. By computing the normalized Levenshtein distance for all pairs of D features, we obtain a symmetric $D \times D$ distance matrix.

3.2.2 Feature Name Clustering by Distance

PROFET performs clustering by grouping similar operations for aggregation, leveraging the previously computed Levenshtein distance matrix. Hierarchical clustering [33] is applied to generate a tree structure based on inter-node distances. In PROFET's implementation, each operation name serves as a leaf node, with operations exhibiting similar names positioned closer to each other within the tree. A dendrogram [34] is subsequently generated, illustrating the hierarchical relationships between nodes. Within a dendrogram, the height at which two or more objects are joined signifies their similarity; a lower height indicates a higher degree of similarity.

When calculating the distance between two individual nodes, the process is straightforward, directly referencing the $D \times D$ distance matrix.

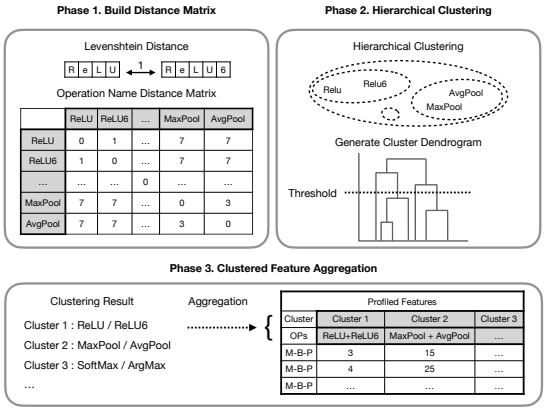


Fig. 5: Steps of clustering features based on the operation name similarity

However, this calculation becomes more intricate when multiple nodes are aggregated under a common parent node. In such instances, various linkage heuristics are employed, including the average, single, complete, and Ward methods [35].

Among these methods, PROFET utilizes the average linkage method, selected based on empirical analysis. The average method computes the pairwise distances between a new candidate node and all existing nodes within a cluster. These distances are then averaged to derive a height metric for dendrogram construction. For illustration, consider three operation names: *MaxPoolGrad*, *AvgPoolGrad*, and *ArgMax*. The Levenshtein distance between *MaxPoolGrad* and *AvgPoolGrad* is three, representing the smallest distance among all combinations. If these two operations are clustered, the distances between the new candidate operation *ArgMax* and the individual clustered operations are ten (for *MaxPoolGrad*) and eight (for *AvgPoolGrad*), respectively. Adopting the average linkage heuristic, the distance between the new candidate operation and the newly formed cluster is nine, derived from the average of the calculated pairwise distances.

3.2.3 Clustered Feature Aggregation

After constructing a hierarchical dendrogram tree, the generation of clusters based on text similarity becomes crucial. To define the cluster boundaries, we establish a maximum height, which directly influences both the number of resultant clusters and the composition of operations within each cluster. Empirical analysis revealed that a maximum height setting of six yielded optimal prediction accuracy. At this maximum height, all operations with a dendrogram height lower than six are aggregated into clusters.

For each established group of operations, a single feature is generated by summing the individual profiled operation values. Representative aggregated operations include [*FusedBatchNormV3*, *FusedBatchNormGradV3*], [*AssignSubVariableOp*, *AssignAddVariableOp*], [*Softmax*, *ArgMax*], [*MaxPoolGrad*, *AvgPoolGrad*], [*DepthwiseConv2dNativeBackpropInput*, *DepthwiseConv2dNativeBackpropFilter*], [*BiasAddGrad*, *BiasAdd*], [*MatMul*, *MaxPool*, *AvgPool*], and [*Relu6Grad*, *RsqrtGrad*, *ReluGrad*]. While the clustering results are generally reasonable, effectively grouping similar operations with minor naming variations, to further validate the semantic coherence of the clustering results, we categorized all 41 clusters produced at threshold 6 according to the functional meaning of each operation. Only 4.9% of clusters contained operations from semantically distinct groups, and even these cases involved forward/backward pass pairs sharing computational context. This confirms that Levenshtein-based clustering captures semantic proximity in the vast majority of cases (95.1%). A systematic comparison between semantically defined clusters and Levenshtein-based clusters as predictor inputs remains a direction for future work.

Figure 5 shows the feature clustering procedure. In *Phase 1*, a Levenshtein distance matrix is computed based on operation names. In *Phase 2*, similar names are grouped using a dendrogram. In *Phase 3*, operations in the same cluster are aggregated by summing their feature values. This clustering improves prediction accuracy, particularly when models use uncommon operations. For instance, aggregating *ReLU* and *ReLU6* helps generalize for models using *ReLU6*, which may not appear frequently in training data.

3.2.4 Infeasible Operation Name Change by Users

The effectiveness of clustering operation names relies on the assumption that operation names reflect their actual computational behavior. If

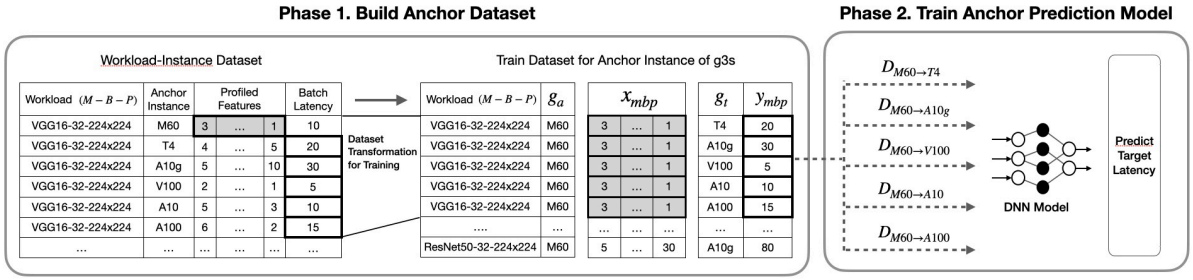


Fig. 6: Predicting the training latency on a target instance type based on profiled feature from an anchor instance

users could arbitrarily rename operations, clustering based on name similarity could degrade prediction accuracy. However, we confirmed that users cannot modify the names of predefined operations within the framework. Although it is technically possible to define new operations with custom names, such cases are uncommon in practice. Most users prefer using standard operations provided by the framework, which are typically more optimized and better supported. Therefore, our method does not consider scenarios involving custom-named operations.

3.3 Modeling CNN Performance on Cloud

PROFET aims to predict the training latency of arbitrary CNN implementations (M) across different GPU instance types (G) and batch sizes (B). To represent the input dataset (x_i) more precisely, we use subscripts to indicate each dimension: x_{mgb}, where $m \in M$, $g \in G$, and $b \in B$. For example, x_{mb} refers to the input data for model m with batch size b across all GPU types in G, omitting g in the subscript. The size of x_{mb} is $|G|$, representing the number of GPU instance types. The PROFET model consists of two main phases: (1) cross-instance performance prediction and (2) batch-size-specific latency prediction.

3.3.1 Cross-Instance Performance Modeling

The cross-instance performance modeling phase aims to predict the training latency of a workload x_{mb} across different GPU instance types. This phase requires a set of profiled features obtained by executing the target workload on a selected anchor instance g_a. Using the profiled results from g_a, PROFET predicts the training latency on a target instance g_t, where $g_a \neq g_t$ and $g_a, g_t \in G$, for the given workload x_{mb}.

We denote the prediction model that estimates the latency from anchor instance g_a to target instance g_t as $f_{g_a \to g_t}$. The training dataset used to construct this model is represented by $\mathcal{D}_{g_a \to g_t}$. To build this prediction model, we define the training dataset as follows.

$$\mathcal{D}_{g_a \to g_t} \triangleq \{(x_{mb|G=g_a}, y_{mb|G=g_t}) | mb \in (M \cup B)\}$$

For each workload defined by a combination of model architecture and batch size (mb), the input feature vector x is obtained by profiling the training execution on an anchor instance (g_a). The corresponding output y is a scalar value representing the observed batch training latency when the same workload is executed on a target instance (g_t).

Using the dataset $\mathcal{D}_{g_a \to g_t}$, collected from anchor-to-target instance experiments, PROFET trains a prediction model based on a DNN. Through empirical evaluation, we found that the best-performing model consists of three fully connected layers with 64, 32, and 16 hidden nodes followed by an output layer predicting training latency. The model uses the ReLU activation function [19] and is optimized using the Adam optimizer [36].

Figure 6 illustrates the overall procedure for predicting batch latency across GPU instances. The upper part of the figure shows the training data generation phase. In this phase, all workloads

are executed on each GPU instance in G, and their profiled features and corresponding batch latencies are collected. To construct input-output pairs for model training, we match the profiled features x_{mb} obtained from an anchor instance g with the batch latencies y_{mb} observed on various target instances.

In the example shown in the figure, the anchor GPU is *M60*, and the target instances include *T4*, *A10g*, *P3*, *V100*, *A10*, and *A100*. The same input features profiled from *M60* are used to predict the training latencies on these target instances, creating five distinct training cases. In the model construction phase, a separate DNN model is trained for each anchor-target pair using the corresponding dataset $\mathcal{D}_{g_a \to g_t}$. For example, models $f_{\text{M60}\to\text{T4}}$, $f_{\text{M60}\to\text{A10g}}$, $f_{\text{M60}\to\text{P3}}$, $f_{\text{M60}\to\text{V100}}$, and $f_{\text{M60}\to\text{A100}}$ are trained independently.

3.3.2 Modeling Latency of Batch Size Change

In the first phase of latency prediction, PROFET estimates the expected training latency of a given workload x_{mb} on any target instance, based on profiling results obtained from an anchor instance. In the second phase, using the predicted latency on a specific instance type, PROFET further predicts the training latency for various batch sizes on that instance.

Formally, let y_{mbg} denote the predicted training latency of workload mb on instance g. PROFET then estimates the latency $y_{mb'g}$ for other batch sizes $b' \in B$ on the same instance. This step requires capturing the non-linear performance scaling behavior across batch sizes and GPU types. In particular, the impact of batch size changes is modeled by analyzing the ratio of latency differences with respect to batch size values, allowing PROFET to generalize across different configurations effectively.

To build the training latency prediction model for various batch sizes, PROFET first identifies the minimum and maximum batch size values within the configuration range. Using these bounds, it applies min-max normalization to training latencies and fits a second-order polynomial regression model [37] to capture the relationship between normalized latency and batch size.

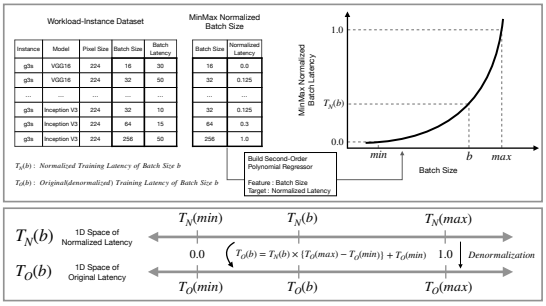


Fig. 7: Prediction of training latency with respect to the diverse batch sizes

The regression model is defined as $T_N(b) = \alpha_2 b^2 + \alpha_1 b + \alpha_0$, where $T_N(b)$ denotes the normalized training latency for batch size b, and α_2, α_1, and α_0 are the coefficients obtained by minimizing prediction error. PROFET trains a separate regressor for each combination of instance type and workload.

Using this model, PROFET predicts the normalized latency value $T_N(b)$ in the range $[0.0, 1.0]$. To convert the prediction back to the original scale, two reference latencies are required: the latency of the minimum batch size configuration, $T_O(\text{min})$, and the maximum, $T_O(\text{max})$. The denormalized latency $T_O(b)$ is then computed as follows.

$$T_O(b) = T_N(b) \times (T_O(max) - T_O(min)) + T_O(min)$$

Figure 7 provides an overview of the procedure for predicting training latency across varying batch sizes. In this example, the minimum and maximum batch sizes are 16 and 256, respectively. The upper part of the figure illustrates the first step, where the normalized latency $T_N(b)$ is predicted for an arbitrary batch size. To train the regression model, the normalized latencies of all workloads are used as input features, and the training data is grouped by instance type, resulting in one regressor per instance.

Once the normalized latency for the target batch size is predicted, the lower part of the figure depicts the denormalization step. The normalized value $T_N(b)$ is converted to the original latency $T_O(b)$ using the corresponding minimum and maximum latency values, $T_O(\text{min})$ and $T_O(\text{max})$, for the target configuration.

3.3.3 Predicting Latency of New GPU Model

New GPU types are frequently introduced and may not be available during the initial modeling phase. As a result, unlike prior work [11, 12, 14], PROFET does not rely on hardware specifications as input features and cannot directly predict training latency for GPU instances that were not part of the training dataset.

However, this limitation is less critical in a cloud-based deployment. In cloud environments, service providers control the release schedule of new instance types and have sufficient time to collect profiling data and train new latency prediction models. As cloud platforms continue to dominate deep learning infrastructure, it becomes the provider's responsibility to manage the complexity of system-level optimization.

While PROFET cannot perform zero-shot prediction for unseen GPU types, the cost of extending support to new instances is remarkably low. As demonstrated in Figure 11b, the PROFET prediction model requires only 2.46 seconds to train and ˜60ms for inference, meaning that onboarding a new GPU type requires minimal retraining overhead. Furthermore, the profiling step itself introduces negligible latency — an average overhead of less than 0.81 seconds across all evaluated models (Table 1). Since cloud providers already manage instance lifecycle and driver/CUDA updates as part of their operational responsibilities, triggering a lightweight retraining pipeline upon new instance deployment is consistent with standard cloud PaaS management practices. This is fundamentally different from white-box approaches, which would require re-engineering the entire prediction model whenever framework internals change.

4 Implementation : PROFET Service

All PROFET materials, including the source code for extracting profiled features from the TensorFlow profiler on AWS EC2 and the DNN model trainer, have been made publicly accessible.[1] We also implemented a publicly available end-to-end demonstration system leveraging a serverless architecture [25].

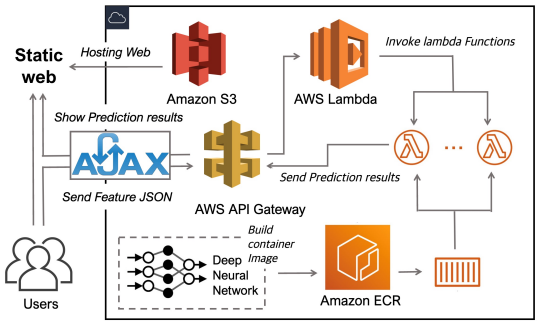


Fig. 8: The implementation of PROFET using serverless computing services

Figure 8 illustrates this system architecture. A static HTML file is served via Amazon S3, an object storage service offering a publicly accessible HTTP endpoint URL for stored objects. User prediction requests are handled by AWS Lambda, which provides a fully managed source code execution environment. To address AWS Lambda's limitations (e.g., restricted local storage), we adopted a custom container image for the function runtime, encompassing all necessary packages [38]. This custom container image is stored in the Amazon Elastic Container Registry (ECR) and can be accessed by AWS Lambda. We utilized Amazon API Gateway, which provides an HTTP endpoint with a RESTful API, to trigger PROFET's prediction model served within AWS Lambda.

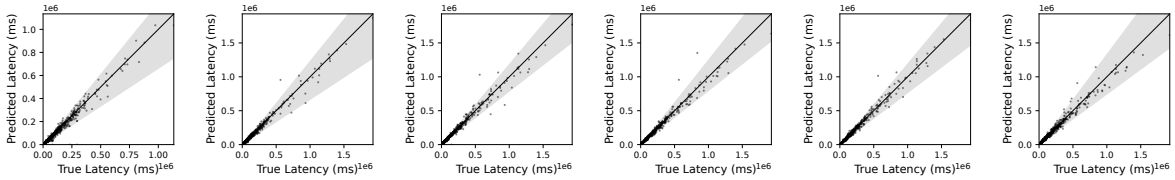
5 Evaluation

We conducted a thorough evaluation of PROFET to demonstrate its practicality and the effectiveness of its prediction performance. The evaluation focuses on Mean Absolute Percentage Error (MAPE) and Root Mean Squared Error (RMSE) metrics to assess the prediction model's accuracy.

5.1 Setup

Experiments were conducted on AWS GPU instances (M60, T4, A10g, V100), GCP (A100), and Alibaba Cloud (A10) that compose G. For the model architectures (M), we used well-known CNN models from the literature,

[1] http://profet.ddps.cloud



(a) M60 (b) T4 (c) A10g (d) V100 (e) A10 (f) A100

Fig. 9: True (x-axis) and predicted (y-axis) latencies for different anchor instances

including *Xception*, *EfficientNetB0*, *LeNet5*, *InceptionV3*, *FLOWER_CNN*, *MobileNetV2*, *MNIST_CNN*, *CIFAR10_CNN*, *ResNetSmall*, *ResNet18*, *ResNet34*, *ResNet50*, *VGG11*, *VGG13*, *VGG16*, *VGG19*, and *VGGSmall*. For batch size B, we used [16, 32, 64, 128, 256].

On AWS, we utilized the Deep Learning AMI, which includes the Nvidia GPU driver library (v. 470.57.02) and CUDA (v. 11.4). TensorFlow (2.5.0) served as the CNN development platform. For GCP, we employed an 'a2-highgpu-1g' instance featuring one Nvidia A100 GPU, 85 GB of RAM, and 12 virtual CPUs. In the Alibaba Cloud environment, we used a 'gn7i-c8g1.2xlarge' instance equipped with a single Nvidia A10 GPU, eight vCPUs, and 30 GB of memory. To ensure consistency, the software and hardware configurations in both Alibaba Cloud and GCP environments were set to match the AWS Deep Learning AMI specifications.

The GPU instances selected for evaluation represent the most widely deployed GPU lineup in major public cloud platforms at the time of this study, covering a broad spectrum from legacy to state-of-the-art hardware. The batch sizes evaluated encompass the range most commonly adopted in practical CNN training scenarios, including both memory-constrained and large-batch configurations. While TensorFlow was adopted as the primary development platform, PROFET's core mechanism relies on kernel execution logs produced by the profiler, rather than any TensorFlow-specific internals. Since comparable profiling interfaces are available in other major frameworks — such as PyTorch's profiler and MXNet's profiling API — PROFET's approach is structurally extensible to these frameworks with minimal adaptation.

5.2 Cross-Instance Latency Predictor

Figure 9 presents the prediction accuracy of cross-instance performance modeling. Each subfigure corresponds to a distinct anchor instance: Figures 9a, 9b, 9c, 9d, 9e, and 9f represent anchor instances M60, T4, A10g, V100, A10, and A100, respectively. The horizontal axis displays the true training latency, while the vertical axis shows the latency predicted by PROFET. The scatter plot indicates the relationship between real and predicted values. Values proximate to the line $y = x$ imply accurate predictions. As depicted, most predicted values are observed to be close to the actual CNN training time, indicating the effectiveness of PROFET's prediction accuracy.

Figure 10 compares the prediction accuracy of PROFET with multiple other prediction models. In this model accuracy measurement, a leave-one-model-out cross-validation approach was employed to evaluate PROFET's generalization capability on unseen architectures. Specifically, all data samples belonging to a single CNN model (e.g., VGG16) were entirely excluded from the training set and reserved solely for testing in each validation fold. This rigorous setup ensures that the evaluation reflects PROFET's performance on models it has never encountered during training. PROFET, which utilizes a DNN model, demonstrates superior accuracy compared to the other models. Specifically, the PROFET DNN model outperforms decision tree-based models such as Random Forest [39] by 45.4% and Extreme Gradient Boosting (XGBoost) [40] by 52.1%, and linear regression by 87.7%.

To evaluate the overhead of training and inference in PROFET, experiments were conducted to analyze prediction accuracy as the number of parameters and training epochs varied

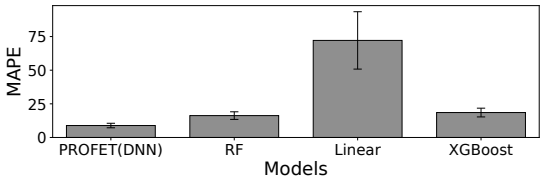
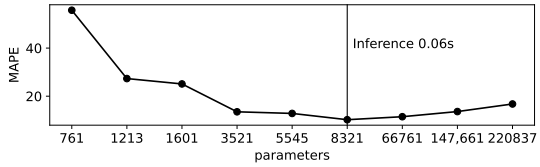
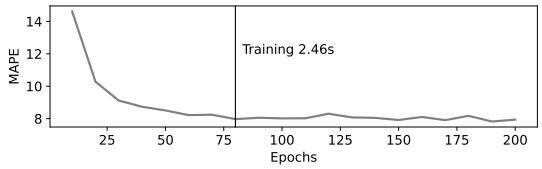


Fig. 10: Prediction accuracy comparison with different models



(a) MAPE change as the number of parameters of DNN model change



(b) The accuracy change as the train continues with more epochs

Fig. 11: Overhead of training a PROFET prediction model

(Figure 11). In Figure 11a, prediction precision is presented against the number of parameters in the DNN models. Prediction accuracy increases with the number of parameters, reaching a peak around 8,000. However, as the model size continues to increase, the prediction error rate rises, likely due to overfitting. Based on these experiments, the PROFET prediction model was finalized with 8,321 parameters. With this model, the average inference time is approximately 60 ms. Figure 11b presents the effect of the number of training epochs on training precision, indicating how MAPE improves as training progresses. The graph suggests that the MAPE value does not improve significantly beyond approximately 200 epochs. Training to this point took approximately 2.46 seconds. Both training and inference times were measured using a T4 GPU instance.

To validate the efficacy of operation clustering, we performed a sensitivity analysis by varying the

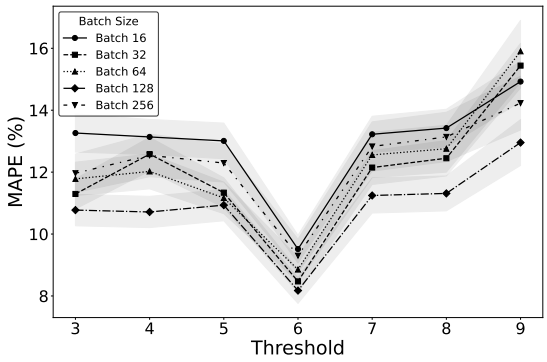
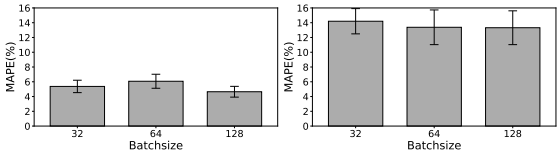


Fig. 12: Prediction MAPE by Threshold (95% CI)

clustering threshold from 3 to 9. While the threshold was initially fixed at 6 based on the trial and error discussed in Section 3.2.3, Figure 12 confirms a distinct V-shaped trend across all batch sizes, with MAPE being minimized at 6. This optimal point prevents both the feature fragmentation caused by lower thresholds and the over-aggregation of distinct operations at higher thresholds, empirically justifying 6 as the most effective value for maintaining predictive precision.

5.3 Batch Size Predictor Accuracy

We also conducted experiments to evaluate the accuracy of training time prediction across varying batch sizes, ranging from a minimum of 16 to a maximum of 256. We acknowledge that using the minimum and maximum batch sizes as normalization anchors reduces the number of intermediate evaluation points to three. However, the second-order polynomial regressor operates strictly as an interpolator within the observed range, which inherently constrains prediction error. The empirical results show a MAPE of approximately 5% when using true latency values, as shown in Figure 13a, suggesting that the model captures the general scaling trend within the evaluated range. Furthermore, a second-order polynomial is a deliberate design choice — a more complex model would risk overfitting given the limited number of batch size configurations available in practice. We note, however, that this normalization-anchor design constrains the present evaluation to three intermediate batch-size points, and a broader validation across a denser range of batch



(a) Using true values (b) Predicted values

Fig. 13: Average prediction accuracy of as the batch size varies. The prediction is made from a true and predicted train time of minimum and maximum batch size.

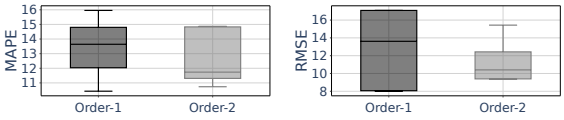
sizes remains a limitation to be addressed in future work. Figure 13 illustrates the prediction accuracy for intermediate batch sizes, excluding the minimum and maximum values used during training. The graph presents two scenarios: one utilizing measured latency values (*true*) and another employing PROFET's predicted values (*Predict*). Specifically, Figure 13a depicts the precision of a predictor based on measured latencies (*true*), while Figure 13b shows the accuracy of a predictor that uses latencies estimated by PROFET's cross-instance performance model (*predicted*). The MAPE value is presented on the vertical axis, with batch sizes on the horizontal axis.

As depicted, the MAPE value is approximately 5% when utilizing *true* latencies. When employing *predicted* latencies, the average MAPE value is about 13.6%. Achieving higher prediction accuracy (approximately 5% MAPE with *true* latencies) typically requires executing two training tasks on a target GPU instance: one at the minimum and one at the maximum batch size. However, PROFET's cross-instance latency prediction model can be leveraged to obtain these minimum and maximum batch-size latencies, thereby circumventing additional training task executions on the target instance. Although this additional prediction step introduces a slight reduction in overall precision, it still yields a 13.6% MAPE, which remains competitive compared to other state-of-the-art approaches.

Moreover, PROFET employs a second-order polynomial regressor coupled with a min-max scaler for predicting training latencies by batch size. We conducted a comparative analysis of various scaling heuristics, and the min-max scaler demonstrated superior performance. Table 2

	MAPE	RMSE
Standard Scaler [41]	87.59	143.95
MaxAbs Scaler [42]	24.80	23.90
Normalize [43]	19.77	17.34
MinMax(PROFET)	13.00	11.96

Table 2: Accuracy for different scaling methods



(a) MAPE (b) RMSE

Fig. 14: Prediction accuracy to different polynomial equation orders

presents this performance comparison across different scaling methods.

Figure 14 compares the prediction accuracy of first- and second-order polynomial regressors. The horizontal axis represents the order of the polynomial regressors, while the vertical axis of Figure 14a indicates the MAPE value and that of Figure 14b presents the RMSE value. The comparison indicates that the second-order polynomial regressor used in PROFET demonstrates a higher prediction accuracy. The median and maximum MAPE values of the second-order regressor are 14% and 6.77% better than those of the first-order regressor, respectively. Similar patterns are observed for the RMSE value.

5.4 Efficiency of Profiled Feature Clustering

We conducted experiments to evaluate the efficiency of clustering operation names in PROFET; Figure 15 presents the corresponding MAPE values. The solid gray bars represent MAPE values when feature clustering is disabled, whereas the upper right diagonal bars indicate prediction accuracy when feature clustering is enabled. The horizontal axis displays the names of the models utilized in the experiments.

Overall, most models exhibited improved prediction performance through feature clustering, with *ResNet18* showing the most significant gain

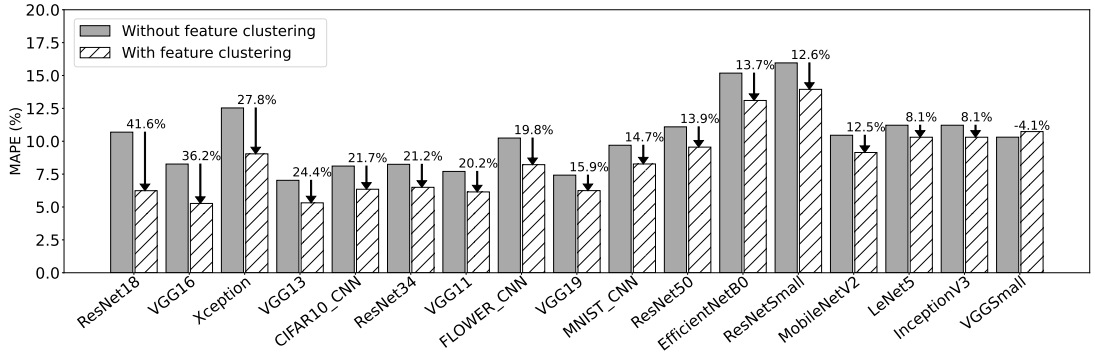


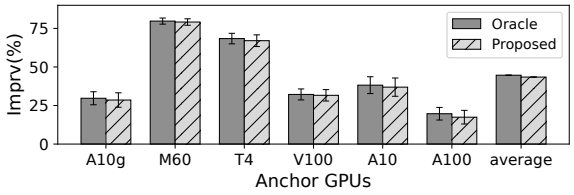
Fig. 15: Train latency prediction accuracy improvement as we apply feature name clustering. Most models show performance gain by aggregating similar operation names

at 41.6%. However, the impact of clustering varies depending on the specific model architecture. For instance, the *VGGSmall* model demonstrated a slight decrease in accuracy of 4.1% after applying clustering. Further investigation suggests that because *VGGSmall* possesses fewer unique operations and a simpler structure, the potential loss of distinct performance details might have marginally outweighed the benefits of feature generalization. These results indicate that while clustering generally enhances efficiency by reconciling naming inconsistencies across different implementations, its effectiveness can be influenced by the structural complexity of the target model.

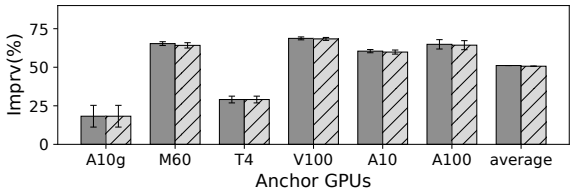
5.5 Optimal GPU Instance Recommendation

The predicted training latency can facilitate recommendations for an optimal target GPU instance, considering either latency or cost. This section explores a comparison between an *Oracle* approach and our proposed method for recommending the ideal target GPU instance. For the *Oracle* approach, we assumed prior knowledge of a model's training latency across all GPUs and utilized this information for recommendations, representing the theoretical best-performing method.

Figure 16 displays the performance improvement on the vertical axis, with anchor instances on the horizontal axis. The percentage improvement on the vertical axis is calculated using Equation 1. In this equation, TM represents the target metric for evaluation (latency or cost). The subscript *anchor* denotes the metric value TM of the anchor



(a) Latency improvement from GPU recommendation



(b) Cost improvement from GPU recommendation

Fig. 16: Comparing performance improvement when recommending optimal GPU instances for *oracle* and the proposed latency prediction heuristic

instance, while the subscript *best* indicates the metric value of the best-performing GPU. For the *Oracle* approach, the *best* GPU is determined using actual training time measurements; for the *proposed* approach, this determination is based on the latency predicted by PROFET. When calculating performance improvement, cases where the anchor GPU was already the optimal choice were excluded. We assumed that a user initially utilizes an anchor instance (as indicated on the horizontal axis) and subsequently receives a recommendation

for the best-performing GPU from either a latency or cost perspective.

$$Imprv(\%) = \frac{TM_{anchor} - TM_{best}}{TM_{anchor}} \times 100 \quad (1)$$

Figure 16a compares the latency improvement. The solid gray bar depicts the performance improvement of the *Oracle* method, while the bar with the upper right diagonal line indicates the improvement achieved using the predicted latency of the proposed method. Figure 16b similarly compares improvements based on cost. Across both latency and cost metrics, the proposed method demonstrates a performance improvement nearly identical to that of the *Oracle* method. Specifically, for latency, PROFET exhibits an average performance gain that is 1.2% lower. For cost, PROFET achieves an improvement that is 0.4% lower. This result substantiates the efficiency of the proposed training latency prediction system for instance recommendation. Furthermore, it is important to consider both the MAPE and practical performance gain values from the recommendation, as prediction errors may not directly correlate with recommendation quality.

5.6 Comparison to Other Work

To evaluate PROFET's prediction accuracy against recent related work, we conducted comparisons with Paleo [12], MLPredict [14], and Habitat [11].

While we attempted to re-implement Paleo, the significant architectural shifts in deep learning frameworks rendered a direct reproduction within contemporary environments infeasible. Thus, all Paleo accuracy figures reported in this section are taken directly from the original publication [12] rather than reproduced in our environment. As Paleo [12] adopts a white-box approach, layer-wise validation results are provided only for models whose architectures were explicitly characterized in its original work, namely AlexNet [7] and VGG-16 [6]. Since AlexNet is not included in PROFET's evaluation set, VGG-16 represents the sole model enabling a direct, apple-to-apple comparison between the two systems.

Table 3 compares the prediction accuracy of PROFET and Paleo for standard CNN models, VGG-16 [6]. For PROFET, we utilized both actual and predicted values of these standard models generated by the cross-instance prediction modeling algorithm. As shown in the table, PROFET outperforms Paleo across all three metrics: MAPE is 17% lower, and RMSE is 85% lower. An analysis of Paleo's experimental results suggests that its theoretical modeling may not accurately represent the operational characteristics of a CNN development platform like TensorFlow. In contrast, PROFET leverages profiled outcomes as features, which inherently represent practical execution characteristics.

	PALEO	PROFET
MAPE	10.1072	8.39951
RMSE	32.3637	4.92250

Table 3: The prediction accuracy of PALEO and PROFET

	MAPE (%)		RMSE	
Batch	MLPrdt	Ours	MLPrdt	Ours
16	15.68	10.08	90.82	3.9
32	17.89	8.4	135.63	4.92
64	24.27	7.28	408.81	8.96
128	115.39	11.47	2254.41	103.22

Table 4: The prediction accuracy of VGG16 model for diverse GPU instances of MLPredict and PROFET

Following Paleo [12], the MLPredict [14] algorithm predicts the training time of DNN models based on GPUs and batch sizes, reporting a notable improvement in precision. To compare PROFET's prediction accuracy with MLPredict, we implemented the MLPredict algorithm following the details in its paper [14] and engaged in discussions with its authors. Table 4 presents the MAPE and RMSE values for MLPredict (MLPrdt) and PROFET (Ours) across various batch sizes. Among the standard models, we specifically compared the prediction accuracy for VGG16.

PROFET exhibits higher prediction accuracy than MLPredict for both MAPE and RMSE values. The accuracy observed for MLPredict in our experiments is lower than that reported in its original article [14]. Following discussions with the

	Habitat	PROFET
T4 → V100	12.16	7.01
V100 → T4	7.99	5.25

Table 5: The prediction accuracy (MAPE) of Habitat and PROFET with different combination of anchor and target GPUs

authors, we concluded that this outcome is reasonable: MLPredict was originally optimized for small batch sizes, specifically ranging from 1 to 16, and the original authors confirmed that the observed error increase at larger batch sizes is expected and reasonable given its design scope. In real-world CNN training, however, small batch sizes are rarely adopted due to their tendency to increase overall training time, making accurate prediction at larger batch scales increasingly critical. The 62% improvement in MAPE against MLPredict should therefore be understood in this practical context.

With the increasing device memory of modern GPUs, larger batch sizes are commonly employed for many well-known DNN implementations, making accurate prediction for larger batches increasingly important. In summary, PROFET demonstrates an improvement in MAPE value of 62.20% compared to MLPredict.

Finally, we compared PROFET with Habitat [11], which models operation-wise latency prediction for training latency across various GPUs. Habitat does not support batch size prediction; therefore, we measured prediction accuracy using fixed batch sizes of 16, 32, and 64 for both PROFET and Habitat. We reproduced Habitat's experimental results, as its system implementation is open source. The results were also confirmed through discussions with its authors. Table 5 presents the prediction accuracy values for PROFET and Habitat.

For this comparison, we selected Nvidia T4 and V100 GPUs. A row denoted T4 → V100 indicates that T4 served as the anchor instance and V100 as the target instance for training latency prediction. For various anchor and target GPU pairs, we predicted the training latency of the *ResNet50*, *InceptionV3*, and *VGG16* models and presented the average MAPE value. The selected GPUs and CNN models were common to both PROFET and Habitat experiments. Both PROFET and Habitat exhibited decent accuracy. Overall, the average MAPE of PROFET is 38% lower than that of Habitat.

In summary, PROFET achieves high prediction accuracy among contemporary related work, demonstrating a reduction in MAPE value by 17% (compared to Paleo), 38% (compared to Habitat), and 62% (compared to MLPredict). This enhanced prediction accuracy in PROFET, even with less detailed information compared to white-box approaches, can be attributed to the novel definition of its input features and the targeted latency prediction for specific instance types. Furthermore, the feature clustering technique enhances the generality of the prediction model. We note that the comparison results in Tables 3, 4, and 5 are reported as point estimates. The Paleo results are taken directly from its original publication, while the MLPredict and Habitat results are obtained by reproducing the respective systems on a fixed set of models and GPU pairs, so that variance reporting across both systems is not feasible. Accordingly, the dispersion of PROFET's own predictions is reported separately, where it can be measured consistently, through the per-model confidence intervals in Table 6 and the threshold-sensitivity intervals in Figure 12.

5.7 Extensibility to a New GPU Generation

To empirically validate the claim in Section 3.3.3 that PROFET can be extended to previously unseen GPU types with minimal retraining, we evaluated the latest-generation *g6.xlarge* instance (NVIDIA L4, Ada Lovelace architecture). The overall average MAPE across the 17 evaluated architectures is 16.7%, which remains comparable to the accuracy observed on the original GPU lineup. As summarized in Table 6, the best-performing models, including the VGG family along with *CIFAR10_CNN*, *ResNet18*, and *Xception*, achieve an average MAPE of 14% or lower. Larger errors on ultra-lightweight networks such as *ResNetSmall* and *EfficientNetB0* arise from the short absolute iteration time on the L4 (under 100 ms), where 1 to 2 ms of system-level jitter inflates the percentage error.

Model	MAPE (%) (± 95% CI)
VGG11	9.40 ± 1.29
VGG13	10.98 ± 1.57
VGG16	12.93 ± 3.84
VGGSmall	9.99 ± 2.76
ResNet18	13.10 ± 3.27
CIFAR10_CNN	11.76 ± 2.03
Xception	14.01 ± 5.76

Table 6: The prediction accuracy of PROFET on the L4 GPU instance (*g6.xlarge*) for the best-performing CNN architectures. Values are reported as MAPE ± 95% confidence interval across the five mini-batch configurations (four for *Xception*).

6 Related Work

DNN Training Performance Prediction

: Predicting DNN training time is crucial for efficient resource management. Early works, often employing **white-box approaches**, rely on detailed internal model architectures and hardware specifications. Paleo [12] considers computational and communication latency alongside model and hardware characteristics. NeuralPower [13] uses a polynomial regressor with internal architecture to predict training time and power usage for resource-constrained environments. MLPredict [14] also predicts DNN training time based on internal model architectures and GPU features. However, these methods, including Habitat [11], often assume access to full model details, which is impractical in public cloud settings where proprietary information is protected. Such approaches may also impose significant operational overhead on developers, deviating from cloud computing's service-oriented model [11]. In contrast, PROFET adopts a partitioned black-box approach that predicts performance without requiring sensitive architectural blueprints. While traditional white-box methods demand full visibility into the model's internal graph, PROFET operates on abstracted execution traces—specifically, operation names and their corresponding latencies. By focusing on these high-level performance signatures rather than proprietary configurations (e.g., filter sizes, connectivity, or weights), PROFET achieves a balance between prediction accuracy and the preservation of model privacy, making it more suitable for service-oriented cloud environments.

Other studies in this area include Zancato et al. [44], who model convergence steps for fine-tuning pre-trained models, though without considering varied models or GPU types. AIGauge [45] estimates CNN training cost and time using decision trees, but its applicability is limited by reliance on vendor-specific data. DTT/B [46] estimates training time from DNN features via a genetic algorithm, supporting only a batch size of 32. Pei et al. [47] developed a model for multi-GPU CNN data-parallel training, which still requires detailed model architecture. Centimani [48] offers an AI accelerator selection framework via low-level kernel profiling, representing another white-box method requiring tight integration.

PowerTrain [49] builds generalizable time and power prediction models from profiling data to optimize DNN training on accelerated edge devices, demonstrating the practical value of profiling-driven prediction, though its focus remains on single-device edge accelerators rather than cross-instance prediction in public clouds. Jahani et al. [50] predict the training performance of deep learning applications on GPU-based cloud services, underscoring the growing importance of accurate latency estimation for the GPU-as-a-Service model. However, like other architecture-aware methods, their approach relies on network and hardware features rather than the architecture-agnostic, profiling-only formulation that PROFET adopts.

Profiler-based DNN Performance Analysis

: Profiling metrics from hardware (e.g., Nvidia Nsight [51]) or DNN platforms (e.g., the TensorFlow profiler [28]) provide valuable insights into workload characteristics and execution time. The DNN training benchmark (TBD) [52] analyzes profiled results across various DNN models and hardware, reporting throughput, utilization, and memory. Srivastava et al. [53] examined hardware acceleration's impact on training time. Yeung et al. [54] used GPU profiling for memory usage and DNN structure analysis, enabling GPU memory utilization prediction. Ruohan Wu [55] designed a performance prediction model for DNN workloads using profiling data of common operators.

Ceer [30] extracts CNN characteristics via DNN profilers to predict training time, similar to PROFET's use of profiled data. However, Ceer cannot predict cross-GPU latency, a critical capability in cloud environments. Liu et al. [56] analyzed resource utilization from profilers. PerfNetV2 [57] employs a DNN profiler for detailed operational data to build an execution time model, but it is limited to predefined operations. PROFET differentiates itself by supporting arbitrary new DNN models without imposing such operational restrictions. Li et al. [58] proposed an operation-wise inference latency predictor, but its white-box nature restricts public cloud deployment.

At a lower level, the execution characteristics that profilers expose are ultimately determined by how individual operators are realized on the hardware. Lopes [59] presents an open, efficient CUDA implementation of convolution operations for popular networks such as ResNet, VGG, and GoogLeNet, tuning kernel parameters against memory-bandwidth and register constraints. Such operator-level implementation studies complement PROFET's perspective: while they optimize the realization of individual operations, PROFET treats the aggregated, operation-level execution times exposed by the profiler as architecture-agnostic features for cross-instance latency prediction.

DNN Performance Estimation on Cloud

: Accurate DNN training time estimation significantly benefits cluster utilization and efficiency. Optimus [60] and Cynthia [61] maximize DNN training group utilization by recommending optimal placement in distributed clusters. Horus [62] proposes a DNN job-scheduling strategy minimizing interference on single GPUs. RubberBand [63] predicts job completion time and provisioning costs for DNNs, enabling elastic hyperparameter tuning. Chaudhary et al. [64] developed a GPU scheduling algorithm enhancing fairness and utilization, an area where PROFET could improve efficiency.

For distributed cloud systems, Srifty [65] models distributed training latency, suggesting independent computation and communication modeling which PROFET could adapt for parallel CNN processing. Hydra [66] presented a scheduling algorithm for heterogeneous GPUs, where PROFET's latency estimation could enhance efficiency. In the broader cloud context, Ernest [67] models big data ML job training time and suggests offline experimental design. CherryPick [68] offers an optimal environment algorithm using Bayesian optimization. MPEC [69] and DoS [70] focus on recommending optimal cloud environments for matrix multiplication tasks.

The challenge of selecting cost-effective cloud instances for AI workloads remains an active research direction. Maas and Lorenzon [71] conduct an extensive evaluation of performance-cost trade-offs across a broad set of CPU- and GPU-based instances from multiple providers, reporting that the cheapest instances rarely deliver the best efficiency and that workload-specific selection yields substantial gains. From the provider's perspective, Wang et al. [72] forecast GPU type allocation in heterogeneous clusters through demand feature extraction and a spatio-temporal LSTM, addressing resource provisioning rather than per-workload latency, complementing PROFET's user-facing prediction. Beyond point estimates, uncertainty-aware cloud workload forecasting via Bayesian deep learning [73] jointly predicts multiple resources together with their predictive uncertainty. This perspective aligns with the dispersion analysis we report for PROFET's predictions and points to a promising avenue for attaching confidence estimates to training-latency predictions.

While these cloud-centric studies share PROFET's goal of optimal cloud environments, DNN training favors a scale-up approach with specialized hardware like GPUs, necessitating a distinct performance estimation model like PROFET due to fundamental workload differences. Although PROFET currently targets single-GPU cloud instances, extending it to heterogeneous HPC settings remains future work. The quantitative study of Han et al. [74] could guide such an extension.

7 Discussions

PROFET excels at predicting the training time of CNN implementations across a wide range of GPUs and mini-batch sizes. Despite its extensive

prediction capabilities, some topics warrant further consideration to facilitate an optimal CNN model training environment in a public cloud.

Training Latency Prediction Using Multiple GPUs: A significant portion of deep neural network (DNN) training tasks are performed on a single GPU. According to the Microsoft Philly trace [16], approximately 86% of DNN tasks execute on a single GPU. Similarly, Hu et al. [75] analyzed SenseTime DNN training logs and found that around 70% of tasks utilize a single GPU. While single-GPU training is prevalent, leveraging multiple GPUs in parallel remains an option to accelerate training tasks. In parallel CNN model training, a substantial amount of time is dedicated to calculating weight parameters on each GPU and communicating these parameters among devices. To extend PROFET for parallel model training, we can incorporate a communication overhead estimation model proposed in the literature. Hafeez et al. [30] extensively investigated the training latency of arbitrary CNN algorithms while varying the number of GPUs during training. Their findings suggest that as more GPUs are added for CNN training, the performance gain ratio tends to become more static, irrespective of the GPU instance type. We anticipate that applying the static multiplier suggested by Hafeez et al. [30] to PROFET would enable accurate estimation of training latency when utilizing multiple GPUs.

Training Latency Prediction for Non-CNN Models: The current version of PROFET focuses exclusively on training latency prediction for CNN models. However, different types of DNN models, such as natural language processing (NLP) models (e.g., Transformer [10] and Bidirectional Encoder Representations from Transformers (BERT) [76]), involve distinct operational characteristics. Consequently, latency prediction models built solely using CNN profile results may not perform as effectively for these other model types. Although PROFET's proposed feature clustering algorithm can improve prediction accuracy to some extent, achieving precise performance across all model types requires equitable measurement. The CNN models used in our evaluation are intentionally selected from well-known standard architectures to enable reproducibility and fair comparison with prior work. We are currently working on extending PROFET to support general DNN models beyond CNNs, including more complex and custom architectures, which would further validate the generalizability of the proposed approach.

Framework Generalizability: Beyond model-level generalization, the current implementation of PROFET has been validated using TensorFlow's profiling interface, and all clustering results are derived from TensorFlow operation naming conventions. While the underlying mechanism — relying on operation names and aggregated execution times — is conceptually applicable to other frameworks such as PyTorch, MXNet, and Theano, the Levenshtein-based clustering heuristic may require re-calibration to account for framework-specific naming differences. Extending and validating PROFET across multiple deep learning frameworks remains an important direction for future work.

8 Conclusions

Training CNN models with GPUs has become standard due to their substantial computational demands, and public cloud providers offer diverse, flexible GPU options. However, the performance of arbitrary CNN implementations can vary dynamically across different GPUs, posing a challenge for algorithm developers seeking to create optimal training environments. This paper introduces PROFET, a system designed to predict the training latency of arbitrary CNN implementations across various GPU configurations. This capability assists in developing efficient CNN training environments. Furthermore, PROFET can predict training latency, even when algorithm developers adjust mini-batch sizes, without requiring disclosure of CNN model implementation details. Experimental results demonstrate that PROFET outperforms contemporary related models. We also quantitatively show the algorithm's efficiency when applied to GPU recommendations for training tasks. Beyond its quantitative advantages, PROFET's ability to predict training latency without revealing implementation specifics makes it particularly well-suited for cloud environments, where algorithm developers and resource providers often operate independently.

Acknowledgments

This work is supported by the National Research Foundation of Korea (NRF) or Institute of Information & communications Technology Planning & Evaluation (IITP) Grant funded by the Korean Government (MSIT) : NRF-2022R1A5A7000765, 2021-0-01578 (Smart Farm Platform for high-quality and traceable yield. A multi-purpose DDS based on proximal and remote sensing), RS-2022-00144309, RS-2025-25441560, and RS-2026-25492200.

References

[1] Schmidhuber, J.: Deep learning in neural networks: An overview. Neural Networks **61**, 85–117 (2015) https://doi.org/10.1016/j.neunet.2014.09.003

[2] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019)

[3] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., *et al.*: Tensorflow: a system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283 (2016)

[4] Theano Development Team: Theano: A Python framework for fast computation of mathematical expressions. arXiv e-prints **abs/1605.02688** (2016) [cs.SC]

[5] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9 (2015). https://doi.org/10.1109/CVPR.2015.7298594

[6] Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition (2015)

[7] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, vol. 25, pp. 1097–1105. Curran Associates, Inc., USA (2012). https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[8] Chatfield, K., Simonyan, K., Vedaldi, A., Zisserman, A.: Return of the devil in the details: Delving deep into convolutional nets. In: British Machine Vision Conference 2014 (2014)

[9] Wang, M., Meng, C., Long, G., Wu, C., Yang, J., Lin, W., Jia, Y.: Characterizing deep learning training workloads on alibaba-pai. In: 2019 IEEE International Symposium on Workload Characterization (IISWC), pp. 189–202 (2019). https://doi.org/10.1109/IISWC47752.2019.9042047

[10] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 30. Curran Associates, Inc., USA (2017). https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[11] Yu, G.X., Gao, Y., Golikov, P., Pekhimenko, G.: Habitat: A Runtime-Based computational performance predictor for deep neural network training. In: 2021 USENIX Annual Technical Conference (USENIX ATC 21), pp. 503–521. USENIX Association, USA (2021). https://www.usenix.org/conference/atc21/presentation/yu

[12] Qi, H., Sparks, E.R., Talwalkar, A.: Paleo: A performance model for deep neural networks. In: Proceedings of the International Conference on Learning Representations (2017)

[13] Cai, E., Juan, D.-C., Stamoulis, D., Marculescu, D.: Neuralpower: Predict and deploy energy-efficient convolutional neural networks. Asian Conference on Machine Learning (2017)

[14] Justus, D., Brennan, J., Bonner, S., McGough, A.S.: Predicting the Computational Cost of Deep Learning Models (2018)

[15] Hu, Q., Sun, P., Yan, S., Wen, Y., Zhang, T.: Characterization and prediction of deep learning workloads in large-scale gpu datacenters. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press, USA (2021)

[16] Jeon, M., Venkataraman, S., Phanishayee, A., Qian, J., Xiao, W., Yang, F.: Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In: 2019 USENIX Annual Technical Conference (USENIX ATC 19), pp. 947–960. USENIX Association, Renton, WA (2019). https://www.usenix.org/conference/atc19/presentation/jeon

[17] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90

[18] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. CoRR **abs/1512.00567** (2015) arXiv:1512.00567

[19] Xu, B., Wang, N., Chen, T., Li, M.: Empirical evaluation of rectified activations in convolutional network. CoRR **abs/1505.00853** (2015) arXiv:1505.00853

[20] Lee, K., Son, M.: Deepspotcloud: Leveraging cross-region gpu spot instances for deep learning. In: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), pp. 98–105 (2017). https://doi.org/10.1109/CLOUD.2017.21

[21] Alkharif, S., Lee, K., Kim, H.: Time-series analysis for price prediction of opportunistic cloud computing resources. In: Lee, W., Choi, W., Jung, S., Song, M. (eds.) Proceedings of the 7th International Conference on Emerging Databases, pp. 221–229. Springer, Singapore (2018)

[22] Ambati, P., Irwin, D., Shenoy, P.: No reservations: A first look at amazon's reserved instance marketplace. In: 12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20). USENIX Association, USA (2020). https://www.usenix.org/conference/hotcloud20/presentation/ambati

[23] Lee, S., Hwang, J., Lee, K.: Spotlake: Diverse spot instance dataset archive service. In: 2022 IEEE International Symposium on Workload Characterization (IISWC), pp. 242–255. IEEE Computer Society, Los Alamitos, CA, USA (2022). https://doi.org/10.1109/IISWC55918.2022.00029 . https://doi.ieeecomputersociety.org/10.1109/IISWC55918.2022.00029

[24] Schleier-Smith, J., Sreekanti, V., Khandelwal, A., Carreira, J., Yadwadkar, N.J., Popa, R.A., Gonzalez, J.E., Stoica, I., Patterson, D.A.: What serverless computing is and should become: The next phase of cloud computing. Communications of the ACM **64**(5), 76–84 (2021)

[25] Hellerstein, J.M., Faleiro, J.M., Gonzalez, J., Schleier-Smith, J., Sreekanti, V., Tumanov, A., Wu, C.: 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, USA (2019). http://cidrdb.org/cidr2019/papers/p119-hellerstein-cidr19.pdf

[26] Kim, J., Lee, K.: Functionbench: A suite of workloads for serverless cloud function service. In: 2019 IEEE 12th International Conference on Cloud Computing (CLOUD) (2019). https://doi.org/10.1109/CLOUD.2019.00091

[27] Park, S., Choi, J., Lee, K.: All-you-can-inference: Serverless dnn model inference suite. In: Proceedings of the Eighth International Workshop on Serverless Computing. WoSC '22, pp. 1–6. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3565382.3565878 . https://doi.org/10.1145/3565382.3565878

[28] Documents, T.: TensorFlow Profiler. https://www.tensorflow.org/guide/profiler

[29] Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., Zhang, Z.: Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. CoRR **abs/1512.01274** (2015) arXiv:1512.01274

[30] Hafeez, U., Gandhi, A.: Empirical analysis and modeling of compute times of cnn operations on aws cloud. In: 2020 IEEE International Symposium on Workload Characterization (IISWC), pp. 181–192. IEEE Computer Society, Los Alamitos, CA, USA (2020). https://doi.org/10.1109/IISWC50251.2020.00026 . https://doi.ieeecomputersociety.org/10.1109/IISWC50251.2020.00026

[31] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4510–4520 (2018). https://doi.org/10.1109/CVPR.2018.00474

[32] Levenshtein, V.I., *et al.*: Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet Physics Doklady, vol. 10, pp. 707–710 (1966). Soviet Union

[33] Johnson, S.C.: Hierarchical clustering schemes. Psychometrika **32**(3), 241–254 (1967)

[34] Phipps, J.: Dendrogram topology. Systematic zoology **20**(3), 306–308 (1971)

[35] Ackerman, M., Ben-David, S.: A characterization of linkage-based hierarchical clustering. The Journal of Machine Learning Research **17**(1), 8182–8198 (2016)

[36] Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization (2017)

[37] Ostertagová, E.: Modelling using polynomial regression. Procedia Engineering **48**, 500–506 (2012) https://doi.org/10.1016/j.proeng.2012.09.545 . Modelling of Mechanical and Mechatronics Systems

[38] Blog., A.: AWS Lambda – Container Image Support. https://aws.amazon.com/blogs/aws/new-for-aws-lambda-container-image-support/

[39] Breiman, L.: Random forests. Machine Learning **45**(1), 5–32 (2001) https://doi.org/10.1023/A:1010933404324

[40] Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '16, pp. 785–794. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939785 . https://doi.org/10.1145/2939672.2939785

[41] Raschka, S.: About feature scaling and normalization and the effect of standardization for machine learning algorithms (2014) https://doi.org/10.13140/2.1.4245.1849

[42] Ahsan, M.M., Mahmud, M., Saha, P., Gupta, K.D., Siddique, Z.: Effect of data scaling methods on machine learning algorithms and model performance. Technologies **9**, 52 (2021) https://doi.org/10.3390/technologies9030052

[43] Patro, S.G.K., Sahu, K.K.: Normalization: A preprocessing stage. CoRR **abs/1503.06462** (2015) 1503.06462

[44] Zancato, L., Achille, A., Ravichandran, A., Bhotika, R., Soatto, S.: Predicting training time without training. In: Proceedings of

the 34th International Conference on Neural Information Processing Systems. NIPS'20, vol. 33, pp. 6136–6146. Curran Associates Inc., Red Hook, NY, USA (2020)

[45] Dube, P., Suk, T., Wang, C.: Ai gauge: Runtime estimation for deep learning in the cloud. In: 2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), pp. 160–167 (2019). https://doi.org/10.1109/SBAC-PAD.2019.00035

[46] Pinel, F., Yin, J.-x., Hundt, C., Kieffer, E., Varrette, S., Bouvry, P., See, S.: Evolving a deep neural network training time estimator. In: International Conference on Optimization and Learning, pp. 13–24 (2020). Springer

[47] Pei, Z., Li, C., Qin, X., Chen, X., Wei, G.: Iteration time prediction for cnn in multi-gpu platform: Modeling and analysis. IEEE Access **7**, 64788–64797 (2019) https://doi.org/10.1109/ACCESS.2019.2916550

[48] Xie, Z., Emani, M., Yu, X., Tao, D., He, X., Su, P., Zhou, K., Vishwanath, V.: Centimani: Enabling fast AI accelerator selection for DNN training with a novel performance predictor. In: 2024 USENIX Annual Technical Conference (USENIX ATC 24), pp. 1203–1221. USENIX Association, Santa Clara, CA (2024). https://www.usenix.org/conference/atc24/presentation/xie

[49] S.K., P., Taluri, S., S, B., Karwa, L., Simmhan, Y.: Powertrain: Fast, generalizable time and power prediction models to optimize dnn training on accelerated edges. Future Generation Computer Systems **161**, 329–344 (2024) https://doi.org/10.1016/j.future.2024.07.001

[50] Jahani, A., Hasani, S.: Performance prediction in training of deep learning applications on gpu-based cloud services. Computing **108**(3) (2026) https://doi.org/10.1007/s00607-025-01613-w

[51] Developers, N.: Nsight Systems User Guide. https://developer.nvidia.com/nsight-systems

[52] Zhu, H., Akrout, M., Zheng, B., Pelegris, A., Jayarajan, A., Phanishayee, A., Schroeder, B., Pekhimenko, G.: Benchmarking and analyzing deep neural network training. In: 2018 IEEE International Symposium on Workload Characterization (IISWC), pp. 88–100 (2018). IEEE

[53] Srivastava, S., Divekar, A.V., Anilkumar, C., Naik, I., Kulkarni, V., Pattabiraman, V.: Comparative analysis of deep learning image detection algorithms. Journal of Big Data **8**(1), 66 (2021) https://doi.org/10.1186/s40537-021-00434-w

[54] Yeung, G., Borowiec, D., Friday, A., Harper, R., Garraghan, P.: Towards gpu utilization prediction for cloud deep learning. HotCloud'20. USENIX Association, USA (2020)

[55] Wu, R., Li, M., Li, H., Chen, T., Tian, X., Xu, X., Zhou, B., Chen, J., An, H.: 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), 25–34 (2022)

[56] Liu, J., Liu, J., Du, W., Li, D.: Performance analysis and characterization of training deep learning models on mobile device. In: 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS), pp. 506–515. IEEE Computer Society, Los Alamitos, CA, USA (2019). https://doi.org/10.1109/ICPADS47876.2019.00077 . https://doi.ieeecomputersociety.org/10.1109/ICPADS47876.2019.00077

[57] Wang, C.-C., Liao, Y.-C., Kao, M.-C., Liang, W.-Y., Hung, S.-H.: Toward accurate platform-aware performance modeling for deep neural networks. SIGAPP Appl. Comput. Rev. **21**(1), 50–61 (2021) https://doi.org/10.1145/3477133.3477137

[58] Li, Z., Paolieri, M., Golubchik, L.: Predicting inference latency of neural architectures

on mobile devices. In: Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering. ICPE '23, pp. 99–112. Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3578244.3583735 . https://doi.org/10.1145/3578244.3583735

[59] Lopes, P.: Open cuda convolution neural network inference implementation. Cluster Computing **29**(2), 105 (2026) https://doi.org/10.1007/s10586-025-05895-9

[60] Peng, Y., Bao, Y., Chen, Y., Wu, C., Guo, C.: Optimus: An efficient dynamic resource scheduler for deep learning clusters. In: Proceedings of the Thirteenth EuroSys Conference. EuroSys '18. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3190508.3190517 . https://doi.org/10.1145/3190508.3190517

[61] Zheng, H., Xu, F., Chen, L., Zhou, Z., Liu, F.: Cynthia: Cost-efficient cloud resource provisioning for predictable distributed deep neural network training. In: Proceedings of the 48th International Conference on Parallel Processing. ICPP '19. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3337821.3337873 . https://doi.org/10.1145/3337821.3337873

[62] Yeung, G., Borowiec, D., Yang, R., Friday, A., Harper, R., Garraghan, P.: Horus: An interference-aware resource manager for deep learning systems. In: International Conference on Algorithms and Architectures for Parallel Processing, pp. 492–508 (2020). Springer

[63] Misra, U., Liaw, R., Dunlap, L., Bhardwaj, R., Kandasamy, K., Gonzalez, J.E., Stoica, I., Tumanov, A.: Rubberband: Cloud-based hyperparameter tuning. In: Proceedings of the Sixteenth European Conference on Computer Systems. EuroSys '21, pp. 327–342. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3447786.3456245 . https://doi.org/10.1145/3447786.3456245

[64] Chaudhary, S., Ramjee, R., Sivathanu, M., Kwatra, N., Viswanatha, S.: Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning. In: Bilas, A., Magoutis, K., Markatos, E.P., Kostic, D., Seltzer, M.I. (eds.) EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020, pp. 1–16. ACM, USA (2020). https://doi.org/10.1145/3342195.3387555 . https://doi.org/10.1145/3342195.3387555

[65] Luo, L., West, P., Krishnamurthy, A., Ceze, L.: Srifty: Swift and Thrifty Distributed Training on the Cloud (2022)

[66] Yang, Z., Wu, H., Xu, Y., Wu, Y., Zhong, H., Zhang, W.: Hydra: Deadline-aware and efficiency-oriented scheduling for deep learning jobs on heterogeneous gpus. IEEE Transactions on Computers (01), 1–13 (5555) https://doi.org/10.1109/TC.2023.3242200

[67] Venkataraman, S., Yang, Z., Franklin, M., Recht, B., Stoica, I.: Ernest: Efficient performance prediction for large-scale advanced analytics. In: Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation. NSDI'16, pp. 363–378. USENIX Association, USA (2016)

[68] Alipourfard, O., Liu, H.H., Chen, J., Venkataraman, S., Yu, M., Zhang, M.: Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), pp. 469–482. USENIX Association, Boston, MA (2017). https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/alipourfard

[69] Kim, J., Son, M., Lee, K.: Mpec: Distributed matrix multiplication performance modeling on a scale-out cloud environment for data mining jobs. IEEE Transactions on Cloud Computing, 1–1 (2019) https://doi.org/10.1109/TCC.2019.2950400

[70] Choi, U., Lee, K.: Dense or sparse : Elastic

spmm implementation for optimal big-data processing. IEEE Transactions on Big Data (01), 1–17 (2022) https://doi.org/10.1109/TBDATA.2022.3199197

[71] Maas, W., Lorenzon, A.F.: Exploring cloud instance options for optimal performance-cost efficiency. Cluster Computing **28**(15), 984 (2025) https://doi.org/10.1007/s10586-025-05702-5

[72] Wang, S., Shi, Y., Chen, S., Liu, M.: Forecasting gpu type allocation via demand feature extraction in heterogeneous clusters. Cluster Computing **28**(7), 418 (2025) https://doi.org/10.1007/s10586-024-05068-0

[73] Rossi, A., Visentin, A., Carraro, D., Prestwich, S., Brown, K.N.: Forecasting workload in cloud computing: towards uncertainty-aware predictions and transfer learning. Cluster Computing **28**(4), 258 (2025) https://doi.org/10.1007/s10586-024-04933-2

[74] Han, J., Xu, L., Rafique, M.M., Butt, A.R., Lim, S.-H.: A quantitative study of deep learning training on heterogeneous supercomputers. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER), pp. 1–12 (2019). https://doi.org/10.1109/CLUSTER.2019.8890993

[75] Hu, Q., Sun, P., Yan, S., Wen, Y., Zhang, T.: Characterization and prediction of deep learning workloads in large-scale gpu datacenters. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '21. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3458817.3476223 . https://doi.org/10.1145/3458817.3476223

[76] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: NAACL 2019 (2019)