Not All Explorations Are Equal: Harnessing Heterogeneous Profiling Cost for Efficient MLaaS Training

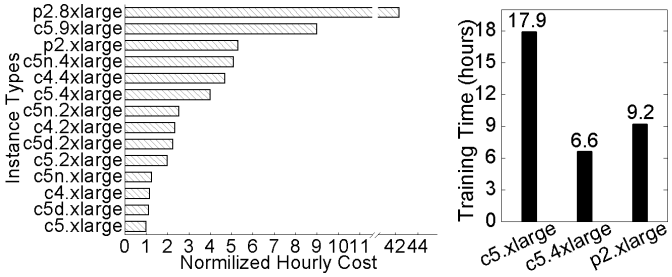
Jun Yi[†], Chengliang Zhang[‡], Wei Wang[‡], Cheng Li[*], Feng Yan[†]
[†]University of Nevada, Reno [‡]Hong Kong University of Science and Technology
[*]University of Science and Technology of China

***Abstract*—Machine-Learning-as-a-Service (MLaaS) enables practitioners and AI service providers to train and deploy ML models in the cloud using diverse and scalable compute resources. A common problem for MLaaS users is to choose from a variety of training deployment options, notably scale-up (using more capable instances) and scale-out (using more instances), subject to the budget limits and/or time constraints. State-of-the-art (SOTA) approaches employ analytical modeling for finding the optimal deployment strategy. However, they have limited applicability as they must be tailored to specific ML model architectures, training framework, and hardware. To quickly adapt to the fast evolving design of ML models and hardware infrastructure, we propose a new Bayesian Optimization (BO) based method HeterBO for exploring the optimal deployment of training jobs. Unlike the existing BO approaches for general applications, we consider the heterogeneous exploration cost and machine learning specific prior to significantly improve the search efficiency. This paper culminates in a fully automated MLaaS training Cloud Deployment system (MLCD) driven by the highly efficient HeterBO search method. We have extensively evaluated MLCD in AWS EC2, and the experimental results show that MLCD outperforms two SOTA baselines, conventional BO and CherryPick, by 3.1× and 2.34×, respectively.**

I. Introduction

Machine-Learning-as-a-Service (MLaaS) [1] is widely supported in the leading cloud platforms (e.g., AWS, Azure, and Google Cloud) to facilitate training and serving user-developed models in a scalable manner. In MLaaS training, users are offered two general deployment options for their training jobs, *scale-up* and *scale-out*. Specifically, MLaaS users need to choose from an array of cloud instances with different resource configurations and hardware architectures such as CPU, GPU, and TPU (scale-up). In the meantime the users also determine *how many* instances should be used for distributed training (scale-out). More often than not, the decision makings of scale-up and scale-out are subject to the *budget constraints* and *training time goals*. For example, an MLaaS user may have $1000 to spare and would like to use it to train a ResNet-50 model in AWS *as fast as possible*.

Training large SOTA models can be expensive, for instance, it is reported that training Google's BERT model costs about $6,912 while training the XLNet model can cost up to $245,000 [2]–[4], which makes efficient training deployment vitally important. However, existing MLaaS platforms provide no guideline for users to deploy their training jobs using the scale-up and scale-out options, which is highly non-trivial yet critically important as it determines the performance and cost of model training. To illustrate this point, we refer to

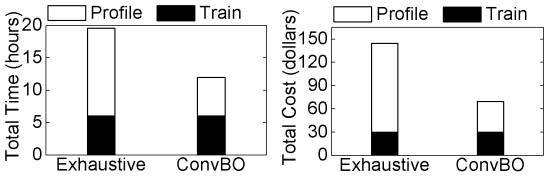


(a) (b)

Fig. 1: (a) Normalized hourly cost of different EC2 instances, where the cost of c5.xlarge is normalized to 1. (b) Training time of Char-RNN using 40 c5.xlarge, 10 c5.4xlarge, and 9 p2.xlarge instances, respectively. For a fair comparison, in all three settings, the per hour training cost remains the same.

Fig. 1(a) for a comparison of the hourly cost of popular CPU and GPU instances in AWS, where the cost of the cheapest CPU instance c5.xlarge is normalized to 1. We observe a significant cost variation across instance types, with the most costly GPU instance (p2.8xlarge) $42.5\times$ more expensive than CPU instance c5.xlarge. Despite the common belief that GPU instances offer more attractive performance-cost ratio than CPU instances for the training job [5], our experiments show that the actual performance and cost depend on many factors such as ML models, training datasets, learning platforms, and system configurations (e.g., parallelism, communication protocol). Fig. 1(b) compares the training time of a Char-RNN model [6] using 40 c5.xlarge, 10 c5.4xlarge, and 9 p2.xlarge instances, respectively. Note that for a fair comparison, the per-hour training cost in all three settings is equal. We see that the most performant and cost-effective training deployment (10 c5.4xlarge) is not given by a large number of cheap CPU instances (c5.xlarge), nor can it be achieved using a small number of costly, high-end GPU instances (p2.xlarge).

To find the optimal training deployment that balances between scale-up and scale-out, existing work employs two general approaches, *analytical modeling* and *profiling-based optimization*. The former establishes an analytical model and uses it to characterize the computation and communication time for a given ML model deployed in a cluster with known hardware architecture and resource configurations [7], [8]. However, this approach imposes heavy assumptions on specific ML model architectures and computing hardware, e.g., the



(a) Total Time. (b) Total Cost.

Fig. 2: Time and monetary cost of profiling and training using exhaustive search and conventional BO search for ResNet model using CIFAR-10 dataset. For exhaustive search, we only consider 180 deployment choices out of total 3,100 choices.

modeling and analysis for training CNNs on CPUs do not apply to training RNNs on GPUs. Analytical modeling therefore has limited applicability and is a poor fit for cloud users given the fast-evolving model architectures and the increasingly diversified instance hardware in MLaaS platforms.

Unlike analytical modeling, profiling-based optimization treats an ML model as a black box and searches for the optimal deployment by profiling the model training performance in a few iterations under various scale-up and scale-out configurations. Reinforcement-Learning (RL), Pareto-Optimization (PO), and Bayesian Optimization (BO) are widely used optimization methods. However, Reinforcement-Learning [9] usually requires extensive training samples and high computing resources while Pareto-Optimization [10] falls short in performance. In comparison, Bayesian Optimization [11] offers an appealing lightweight solution as it can judiciously determine the next search point (deployment configuration) based on the previous profiling results, striking a good balance between exploitation and exploration. Recent works have used BO to find the optimal configurations of cloud applications [12], [13].

However, directly applying BO to MLaaS training can be highly inefficient. First, conventional BO assumes that profiling each search point has a *uniform* cost. This is not the case in MLaaS training, e.g., profiling the training performance for a large number of high-end GPU instances can be orders of magnitude more expensive than that of a small number of low-cost CPU instances. As the conventional BO is *oblivious* to the *heterogeneous* profiling cost, it may "over explore" and hence cannot provide any guarantees for complying with the budget constraint, e.g., if the newly explored deployment scheme could not meet the training time requirements, the spending on this new exploration may result in insufficient fund to roll back to the previous best or even any explored deployments to finish training. Second, conventional BO assumes general applications, without incorporating ML-specific insights into search optimization. Failing to do so may result in excessive exploration cost, as exploring large scale-out deployment for training jobs can be extremely time-consuming and expensive. For example, Fig. 2 demonstrates the profiling and training time breakdown for both exhaustive search (profiling all points in the search space) and conventional BO. Though BO can find the best configuration as in exhaustive search in significantly shorter time, the profiling time and monetary cost are quite high—almost on par with training.

In this paper, we propose HeterBO, an intelligent profiling-based solution that efficiently searches the optimal deployment scheme for MLaaS training. Key to our design is a *cost-aware* search strategy that embeds the profiling cost of a deployment scheme into the BO's acquisition function. In this way, expensive profiling is naturally penalized in the search process and is performed cautiously. HeterBO supports various practical MLaaS training scenarios by providing guarantees for user-specified training time and monetary budget constraints. This is done by using a *protective mechanism* to prevent "over exploration." The protective mechanism reserves the training budget for the current best deployment scheme and only uses the remaining budget for exploration.

HeterBO also utilizes the MLaaS training specific prior to further improve the search efficiency. The prior is based on an important observation that the speedup brought by scaling out usually follows a concave-shape curve. This can be explained as when the distributed training scale (number of instances) is small, the computation is usually the bottleneck. Scaling-out thus helps mitigate the bottleneck and improve the training speed. On the other hand, as more instances are used for training, the communication overhead ramps up quickly, which eventually offsets the benefits of having more compute resources, thus resulting in decreased training speed. This prior is very prominent in reducing the exploration overhead as it helps limit the search in expensive range (e.g., large scale deployments) that usually dominates the search cost.

We built a fully-automated MLaaS training Cloud Deployment system (MLCD) on top of HeterBO. We prototyped MLCD on AWS and performed extensive experimental evaluations by training popular ML models such as Inception-V3 [14], AlexNet [15], ResNet [16], Char-RNN [6], and BERT [2] in the cloud. The experimental results show HeterBO reduces the total profiling and training cost by 3.1× and 2.34× respectively compared with conventional BO (ConvBO) and the state-of-the-art solution CherryPick [12].

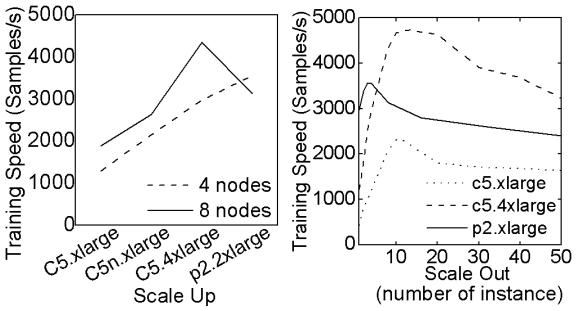
We summarize our main contributions as follows:

- We identify the opportunities and challenges of automating and optimizing MLaaS training deployment using BO.
- We provide a new BO-based methodology HeterBO for MLaaS training. HeterBO is significantly faster and more efficient than the existing approaches, and can provide guarantees for user-defined deployment requirements.
- We implement HeterBO in an automated MLaaS training deployment system (MLCD) atop AWS.
- We show through extensive evaluations that HeterBO outperforms ConvBO by 3.1× and CherryPick [12] by 2.34×.

II. Background and Motivation

A. ML Training Deployment in MLaaS

Sustained algorithmic advances in machine learning (ML) have enabled a growing number of new AI applications [17], [18]. These applications are backed by large ML models that contain up to billions of neural connections trained on a large amount of data. Training and serving ML models at



(a) Scale Up. (b) Scale Out.

Fig. 3: Training speed under different scale-up and scale-out deployment options for training Char-RNN model on AWS.

such a large scale require a significant amount of computing cycles and memory bandwidth [19], [20].Machine-Learning-as-a-Service (MLaaS) offers a scalable solution to facilitate training and serving ML models in the cloud, and is widely supported by leading providers such as AWS, Azure, and Google Cloud. In the *MLaaS training* stage, ML models are built from the training datasets using ML training platforms [21] in a cluster of machines with massive parallelism (model parallelism and data parallelism). In the *MLaaS serving* stage, the trained models are used to provide inference services such as classification and prediction [22], [23]. In this paper, we focus on the training stage in MLaaS.

ML model training is a costly and time-consuming process [8]. Optimally deploying an MLaaS training job with a good balance between scale-up and scale-out can dramatically accelerate the training process with significant cost savings. In our experiments on AWS, with the same monetary cost, training a Char-RNN model using the right deployment scheme can be $3\times$ faster than using a non-optimal scheme (see Fig. 1(b)). However, finding the optimal deployment scheme is a non-trivial task. For example, it is widely believed that GPU offers better performance-cost ratio than CPU for the training jobs. But we show in Fig. 1(b) that with the same monetary cost, using more CPU instances leads to faster training completion than using a few GPU instances. Our experiments further show that both scale-up and scale-out schemes have complex behaviors. Fig. 3 illustrates how the training speed of a Char-RNN model may vary using the two deployment schemes, where Fig. 3(a) shows the scale-up and Fig. 3(b) scale-out. We observe drastically different performance trends between the two schemes, both of which are non-linear. The things become even more complicated in practice, as users usually have specific training requirements, such as limited monetary budget or training deadlines.

B. Challenges of Modeling based Cloud Deployment

To find the best ML task deployment scheme that explores the trade-off between scale-up and scale-out, existing works employ analytical modeling to estimate the computation and communication time for a given ML model. For accurate modeling, those works usually need to know the exact hardware architecture, resource configurations, and the model architecture and hyperparameters [7], [8]. Despite the efforts to extend the analytical modeling to public cloud, the corresponding proposals can only support a limited number of instance types and ML models, due to the heavy assumptions that are necessary for modeling the ML model and underlying infrastructure. For example, Paleo [8] extended its analytical model for AWS, but only 3 models (Inception-V3, AlexNet V2, and NiN) are supported and a few instance types are applicable. We summarize the main reasons why the analytical modeling is not suitable for guiding cloud deployment as follows:

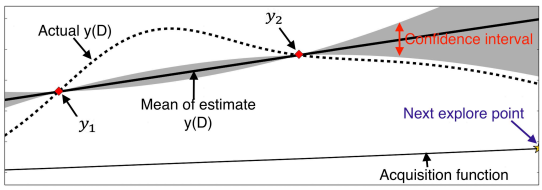
- It is difficult to capture the ever-changing and versatile yet nontransparent hardware and system architectures in the cloud;
- It is challenging to support new software features of machine learning platforms, such as the ring all-reduce communication protocol;
- It is hard to provide support for the fast evolving design of ML model architectures (e.g., the modeling and analysis for CNNs cannot be applied to RNNs directly) and large amount of hyperparameters.

C. Challenges of Exhaustive Profiling

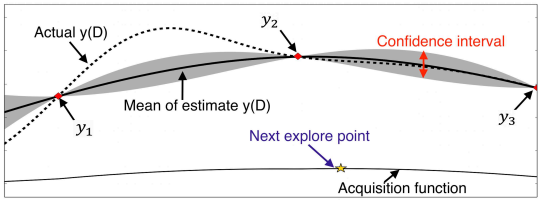
An alternative solution to analytical modeling is to search for the optimal deployment scheme by directly measuring the training throughput (i.e. profiling) in a specific configuration. However, as discussed in Sections II-A and II-B, there is a large number of different deployment choices (e.g., 3,100 possible configurations in total if we consider scale-out a training job over up to 50 nodes) and the training time and monetary cost depend on many factors such as model architectures, hyperparameters, ML system features, and the chosen cloud platform. Even if we only profile a very small set of deployment choices, e.g., 180 out of 3,100 deployment choices, the profiling cost can still be too expensive (see Fig. 2). To make matters worse, if there are any changes made in the training job (e.g., using a different batch size), the expensive search needs to re-performed again. We therefore rule out exhaustive profiling of a viable solution.

D. Opportunities and Challenges of Bayesian Optimization

Bayesian Optimization (BO) [24] is an observation-based method for solving problems with unknown objective functions, and can be potentially applied to solve the expensive search cost issue. Figure 4 illustrates the search process of conventional BO. Conventional BO starts by randomly selecting a few (e.g., two) deployment schemes for profiling as shown in Figure 4(a). Based on the profiling results $y1$ and $y2$, BO estimates the mean of target function $y(D)$ (the estimate is shown in bold solid line while the actual function is shown in dashed line) that represents here the training speed as a function of deployment scheme D with certain confidence interval. Closer to the profiled points has lower confidence interval, vice versa. BO judiciously selects the next points to explore based on the predefined *acquisition function*. There are three commonly used acquisition functions. EI (Expected Improvement) aims



(a) After Initial Profiling Step.



(b) After 2 Profiling Steps.

Fig. 4: An example of the work process for Conventional BO.

at maximizing the expected improvement from the new point over the current best. UCB (Upper Confidence Bound) chooses the next point whose confidence interval has the largest bound in the search space. POI (Probability of Improvement) [11], [25] seeks to maximize the possibility of improvement over the current best. After a profiling step, the estimated target function $y(D)$ as well as the confidence interval changes based on the new profiling results $y3$ (see Figure 4(b)). Meanwhile, the acquisition function is also updated for selecting the next profiling point. Such process repeats until the predefined stop condition meets, e.g., the expected improvement is smaller than a threshold when EI is used as an acquisition function.

Our experiments show that even though the conventional BO can significantly reduce the profiling cost compared to the exhaustive search, it still results in excessive profiling time and monetary cost, see Fig. 2. The reasons are multi-fold. First, conventional BO does not consider the heterogeneous profiling cost, e.g., it is more expensive to profile a high-end instance type (e.g., GPU) in a large cluster than cheap instance type in a small cluster. Such cost difference is particularly salient for MLaaS training as training a modern ML model is quite resource intensive [2]–[4] and today's ML training platforms can utilize powerful instances and large cluster to enable massive parallelism. Fig. 5 gives an example of using the conventional BO to search cloud deployments by demonstrating the cost saving and speedup change over profiling steps. The results show that most profiling steps do not bring benefits and can lead to lower performance. This suggests that conventional BO likely makes wrong judgements of the potential benefits versus the exploration cost.

Second, the general BO approaches do not consider ML specific prior. From Figure 3, we observe an important ML training prior that the speedup trend of scale-out follows a concave-shape curve. This is because the computation is usually the bottleneck when ML training is performed at a small scale, but when the scale expands, the communication overhead increases and eventually becomes the major bottleneck. We can utilize this prior to limit the search in the expensive scale-out region if we detect a declined training

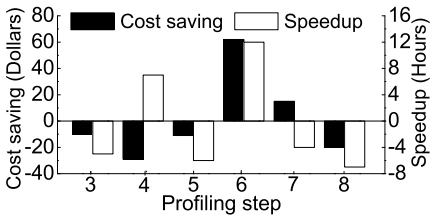


Fig. 5: Cost saving and speedup change over profiling steps using conventional BO to deploy AlexNet with CIFAR-10 dataset. The positive value means there is gain after the step while negative value means getting worse after the step.

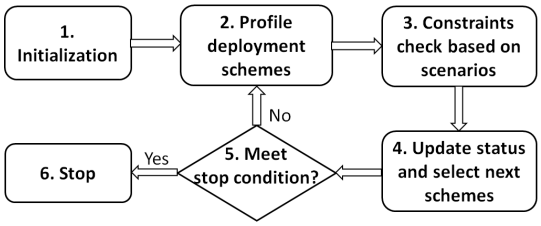


Fig. 6: Workflow of HeterBO.

speed between two neighboring deployments, i.e., down hill of the Concave-shape curve. However, scale-up may have a more complex behavior due to the complex memory hierarchy and hardware-specific optimization. Thus it is difficult to utilize this prior.

Finally, conventional BO methods do not incorporate the profiling cost into the search process, and thus they could not provide strict guarantees for user-defined training requirements such as a training deadline or monetary budget. Given the profiling can be costly due to the resource demanding nature of MLaaS training, ignoring the profiling cost may violate those user-defined constraints.

The above challenges make it difficult to apply conventional BO methods directly for finding the best deployment plans for MLaaS training. In this work, we will design a new BO based methods that can support fast and cost-effective MLaaS training deployment while at the same time providing guarantees for user-defined training requirements.

III. HeterBO Design

A. Overview

Our goal is to find the best deployment scheme in cloud for any given ML training job based on the user defined requirements. To this end, we propose a new BO-based method HeterBO that takes into consideration of heterogeneous exploration cost and MLaaS specific prior to enable quick and cost-effective search of the best deployment schemes according to the user defined requirements. In particular, we demonstrate how HeterBO supports the common practical scenarios outlined in Section II, and HeterBO can be extended to support other scenarios. It is worth noticing that we only change cloud deployment (i.e., scale-up and scale-out schemes), but we do not make changes to ML model or ML training platform parameters, such as model hyperparameter (e.g., batch size,

learning rate), synchronization schemes, communication protocols. This is because these configurations may potentially change the eventual accuracy of the trained model. Instead, we use these parameters as input when search for the best cloud deployment schemes.

In this paper, we consider three common scenarios for MLaaS users to deploying training jobs:

- *Scenario-1*: finish the training job *as fast as possible* with *unlimited* monetary budget.
- *Scenario-2*: finish the training job before a *deadline* at the *lowest cost*.
- *Scenario-3*: finish the training job *as fast as possible* within a budget.

B. Problem Formulation

For a machine learning training task, HeterBO tries to find the optimal MLaaS training deployment scheme based on the specific target and constraints. Formally, we denote m and n as the instance type (scale-up) and quantity of the selected instance type (scale-out), respectively. We define $D(m,n)$ as the deployment scheme. The deployment scheme search space depends on specific cloud provider. Using AWS as an example, there are 62 scale-up options and a rule of thumb for scale-out is 50, so there are in total 3,100 deployment schemes. We further define $P(m)$ as the instance price per unit time for the chosen instance. For a given deployment D, Let $T(D)$ be the total time, and $C(D)$ the total monetary cost to finish the given job. Let $D(m,n)$ be the deployment strategy which directly impacts $T(D)$ and $P(m)$. Let T_{max} be the deadline (if any) and C_{max} the maximum budget (if given). We define our optimization problem for the three scenarios as follows.

$$\text{Scenario-1:} \quad \underset{D(m,n)}{\text{minimize}}\ T(D) \quad \text{subject to } D \in D(m,n),\ m \in M,\ n \in [1,\infty]; \tag{1}$$

$$\text{Scenario-2:} \quad \underset{D(m,n)}{\text{minimize}}\ C(D) = T(D) \times P(m) \quad \text{subject to } T(D) \le T_{max}, D \in D(m,n), m \in M,\ n \in [1,\infty]; \tag{2}$$

$$\text{Scenario-3:} \quad \underset{D(m,n)}{\text{minimize}}\ T(D) \quad \text{subject to } C(D) = T(D) \times P(m) \le C_{max}, D \in D(m,n),\ m \in M,\ n \in [1,\infty]. \tag{3}$$

Here, M is the collection of instance types. Note that there are 3,100 deployment options in $D(m,n)$ and each takes a few minutes to measure, which is too expensive to profile all of them in practice. We propose a BO based method HeterBO to enable lightweight profiling that can support user specified requirements (if any).

C. HeterBO Search Method

We give an overview of HeterBO by showing its workflow in Fig. 6. HeterBO starts the search process by randomly selecting initial deployment schemes (e.g., two) and profiling their training speed. The profiling results are fed into the BO model and are used to update the function estimate and acquisition function by considering the user-specified constraints. After status updates, HeterBO selects the next deployment scheme to profile based on the acquisition function with heterogeneous profiling cost, expected improvement and its confidence, as well as user specified constraints. HeterBO evaluates whether profiling the selected next scheme meets the stop condition. If so, the search process stops, otherwise HeterBO performs the profiling and continue the iterative process until the stop condition is met.

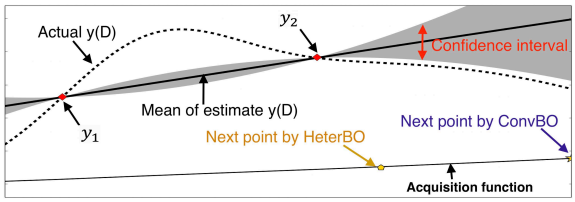


Fig. 7: Comparison of the next explore point selection between HeterBO and Conventional BO (ConvBO).

Prior function. We follow the convention of using Gaussian Process as the prior function [24], [26], i.e., the found function is a sample from Gaussian Process, because of its good flexibility and tractability.

Acquisition function with constraints. We employ the conventional Expected Improvement (EI) [24], [26] as the base acquisition function to derive the acquisition function with user-specified constraints. We choose EI as it does not require hyperparameter tuning and it is easier for setting the stop condition (introduced later) to guarantee the constraint compliance. The conventional EI is defined as:

$$EI(D) = (y_o - \mu(D))\Phi(\gamma(D)) + \sigma(D)\phi(\gamma(D)), \tag{4}$$

where $\mu(\cdot)$ and $\sigma(\cdot)$ are predictive mean function and predictive standard deviation function, respectively; y_o is the best current value at $argmin_{(D)}y(D)$; $\gamma(D) = \frac{y_o - \mu(D)}{D}$; $\Phi(\cdot)$ and $\phi(\cdot)$ are predictive cumulative distribution function of standard normal and probability density function of standard normal.

To support user specified constraints, we need to take profiling cost into consideration when compute the total cost. We define $T_{profile}$ as the profiling time (includes instance cluster setup and warm-up time) and $C_{profile}$ as the profiling monetary cost. We adjust the acquisition function to reflect the constraints as follows. For Scenario 2 with deadline T_{max}, the True Expected Improvement (TEI) is

$$TEI(D) = T_{max} - T_{profile} - \frac{S}{EI(D)}, \tag{5}$$

where S is total data samples for training. For Scenario 3 with budget C_{max}, the True Expected Improvement (TEI) is

$$TEI(D) = C_{max} - C_{profile} - \frac{S}{EI(D)} \times P(m). \tag{6}$$

Heterogeneous search cost. In conventional BO, it assumes every point in the search space takes the same amount of exploration cost. However, in MLaaS training in cloud, different deployment strategy $D(m,n)$ may cause significantly different profiling time and monetary cost. We embed the penalty function PL in the profiling time $T_{profile}$ and cost

$C_{profile}$ to penalize the exploring of expensive deployment options. Assuming the profiling time for deployment $D(m, n)$ is $t(m, n)$, we write the profiling time penalty as

$$PL_T = T_{profile} = t(m, n). \quad (7)$$

For the profiling monetary cost penalty, we have

$$PL_C = C_{profile} = P(m) \times n \times T_{profile}. \quad (8)$$

Note that unlike existing work [12], [27], this penalty function does not trim any search space based on experience, but rather guide the search process to avoid unnecessary expensive profiling cost due to randomly jumping into expensive profiling regions that do not meet requirements. We give an example to illustrate how the acquisition function with constraints and heterogeneous search cost work in Figure 7. Similar to Figure 4(a), $y1$ and $y2$ are profiled points. However, instead of selecting the maximum point in acquisition function as the next point to profile as in conventional BO, HeterBO also considers the user constraints and heterogeneous search cost and thus selects a different next point to profile—in this case, a point with much smaller scale and thus much less cost.

Initial points. Different from the existing works [11] that choose random points as initial points to profile, here we select a single node of each instance type as our initial explore points to avoid unnecessary large cost in consideration of the heterogeneous search space.

Stop condition. Existing works [11]–[13] use the expected improvement threshold and fixed number of points as stop condition. To guarantee that the constraints are not violated due to over exploration, we reserve the necessary training cost required to finish training from the optimal point found in the previous explorations. We also use the confidence interval (i.e., 95%) of the expected improvement to reduce the chance of unsuccessful exploration.

Optimization with prior. Motivated by the ML specific prior, i.e., concave-shape curve speedup pattern (Section II), we further optimize the search process by leveraging this prior. Once HeterBO detects two nearby deployments with declining training speed, i.e., predicting it is on the down slope of the Concave-shape curve, it prevents exploring further scale-out deployments to avoid unnecessary overheads. As large scale-out deployment schemes usually dominate the profiling time and monetary cost, this prior utilizes the ML training patterns to judiciously limit the search in a small range.

IV. MLCD Implementation

We provide the design overview of MLCD by demonstrating its system architecture in Fig. 8 and introducing its main components below. **Profiler.** The Profiler takes the deployment information from HeterBO Deployment Engine and executes the model training for certain iterations. It records the training time and monetary cost and feedback these measurements to the HeterBO Deployment Engine. To achieve statistic stability of profiling, Profiler monitors the training throughput across iterations and extends the profiling time when large discrepancy

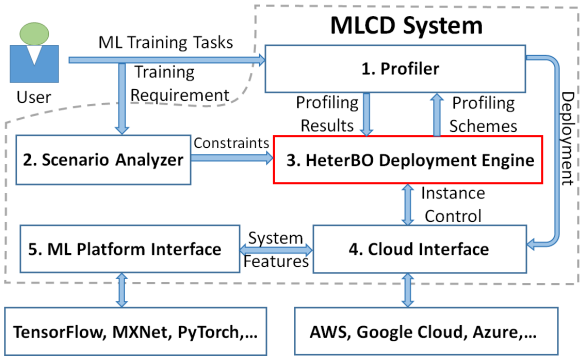


Fig. 8: Overview of MLCD system architecture.

is observed. **Scenario Analyzer.** The Scenario Analyzer takes the training requirements from user (e.g., training deadline, budget) and forms them into the search constraints and feeds them into the HeterBO Deployment Engine. **HeterBO Deployment Engine.** We use HeterBO search method to drive the deployment engine to search for the best deployment schemes based on the Profiler's feedback. **Cloud Interface.** MLCD supports different cloud services through Cloud Interface (e.g., AWS, Google Cloud, Azure). It provides the cloud control operations such as launch/suspend/manage instance, collect measurements through cloud tools (e.g., CloudWatch in AWS) and ML platform tools. **ML Platform Interface.** MLCD supports popular ML training platforms (such as TensorFlow [28], MXNet [29], PyTorch [30]) and connects them with the Cloud Interface to enable various ML platform features, such as PS and all-reduce communication protocols.

V. Evaluation

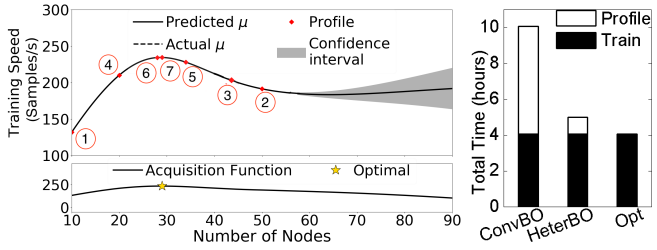
We prototype MLCD atop AWS and conduct extensive experimental evaluation to validate the effectiveness and robustness of HeterBO in practical use scenarios.

A. Experiment Setup

Testbed. We use AWS as the testbed and choose various CPU and GPU instance types as our scale-up options, including CPU instances like compute optimized `c5`, network enhanced `c5n`, last generation compute optimized `c4`, and GPU instances `p3` and `p2` featuring NVIDIA V100 and K80 accelerators respectively. In our experiments, up to 100 `c5`, `c5n`, `c4` instances and 50 `p2`, `p3` instances are used.

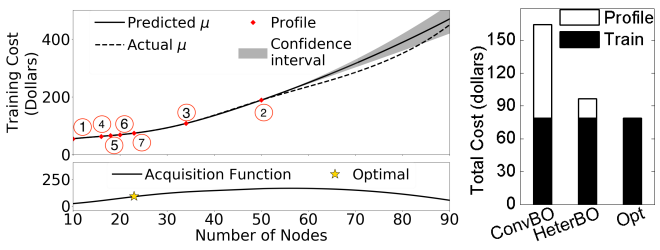
ML models. To verify that HeterBO is general and robust, we perform the evaluation use a diverse selection of popular ML models, such as AlexNet, ResNet, Inception-v3, Char-CNN, and BERT. The models vary from each other substantially in terms of the size, architecture, and task field.

ML platforms. To prove that HeterBO is platform-independent, we implement BERT on two popular ML platforms TensorFlow and MXNet. To demonstrate that HeterBO does not rely on any specific communication protocols or implementations, we examine HeterBO's performance under the two widely used ML distribution topologies, namely, parameter server (PS), and ring all-reduce. We use strong-scaling to avoid the scale-out level impacting accuracy.



(a) Search Process (b) Total Time

Fig. 9: Scenario1. (a) HeterBO search process. (b) Total time comparison with profiling and training time breakdown.



(a) Search Process (b) Total Cost

Fig. 10: Scenario2. (a) HeterBO search process. (b) Total cost comparison with profiling and training cost breakdown.

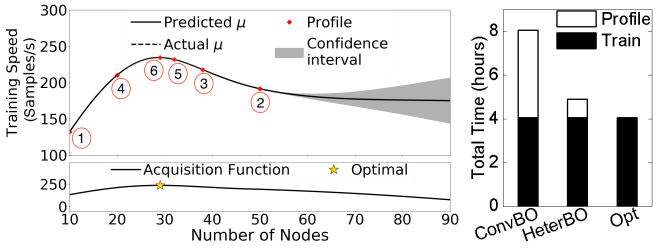
Baseline. We use exhaustive search to obtain the optimal deployment under each setting for reference. We perform a quick comparison with random profiling to show the statistical significance of HeterBO. We focus on making comparison with the analytical model based Paleo and BO based CherryPick as they are the state-of-the-art approaches in their category. Paleo builds individual analytical models for all supported models. The model estimates training time and cost based on model architecture, hardware configuration, hyperparameter, cluster size, and ML platform. CherryPick, on the other hand, employs ConvBO and extends it by utilizing prior knowledge. We also include ConvBO's results for reference.

Profiler. To ensure fairness, the profiling time is the same across different approaches. For single node, each profiling takes 10 minutes (including initial setup and warm-up), we add extra 1 minute to the profiling time for every increase of 3 extra nodes to offset the longer setup and warm-up time as well as the randomness in measurement.

B. Effectiveness of HeterBO under Practical Scenarios

We first verify the effectiveness of HeterBO's under the three common scenarios outlined in Section II. For clarity, we use scale-out (i.e., we already found the optimal scale-up is `c5.4xlarge`) as an example to illustrate how to deploy ResNet with CIFAR10 dataset using TensorFlow training platform in AWS. We report the experimental results in Fig. 9-11 for the three scenarios respectively.

Results of *Scenario1* (finding the fastest training deployment with unlimited budget) is shown in Fig. 9. Fig. 9(a) shows the snapshot of the final profiling step and also the search steps to demonstrate how HeterBO performed the search process. The plot shows after the 2 initial profiling points, HeterBO picks the third point in between to discover the curve shape. HeterBO is able to narrow down the search space to the interval between point 1 and 3 by fitting point 1, 2, 3 into a Concave-shape curve, which is the prior HeterBO employs. By doing so, HeterBO avoids the expensive regions on the right side, i.e., after point 2 and 3, no further left region is explored. HeterBO then further narrows down the search space iteratively until the best deployment scheme is found. Fig. 9(b) shows HeterBO is able to find the optimal deployment with only 16% profiling cost compared to conventional BO, thanks to the heterogeneous profiling cost aware and MLaaS training prior aware search strategies employed by HeterBO.



(a) Search Process (b) Total Time

Fig. 11: Scenario3. (a) HeterBO search process. (b) Total time comparison with profiling and training time breakdown.

Fig. 10 depicts *Scenario2* – how HeterBO finds the cheapest deployment when the total time (profiling plus training) needs to be under 6 hours. HeterBO keeps track of the time spent in profiling to make sure that the training can finish within the time limit. As shown in Fig. 10(b), HeterBO manages to find the cheapest deployment scheme with only 20% of ConvBO's profiling cost while complying with the user defined time limit, whereas ConvBO runs over time limit by 3.4 hours. Fig. 11 shows *Scenario3* – how HeterBO accomplishes to find the fastest configuration when the total monetary budget (profiling plus training) is capped by $100. HeterBO only uses 21% profiling time compared to ConvBO with $96, under the given monetary cost budget while ConvBO spends $225 in total, significantly violates the given budget. Also note that though *Scenario3* and *Scenario1* both targets at minimizing the total time, due to the different monetary budget, the search process and found deployment scheme are also different.

We also compare HeterBO with random search to show the statistical significance, see Figure 12. Random search exhibits significant variance when the number of profiling steps is small. When larger number of steps are used, the profiling cost increases, which results in larger total time. In addition, in practice, it is difficult to know how many steps strikes the best balance between variance and optimality. As expected, HeterBO consistently out performs random search in all cases by using intelligent search strategies.

C. HeterBO vs. state-of-the-art approaches

We first compare HeterBO with Paleo and use ConvBO as a reference in the scenario of limited budget of $80. Fig. 13 demonstrates the total cost and total time comparison. Since Paleo models distributed ML directly, there is no profiling

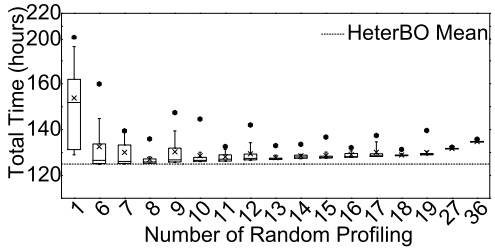
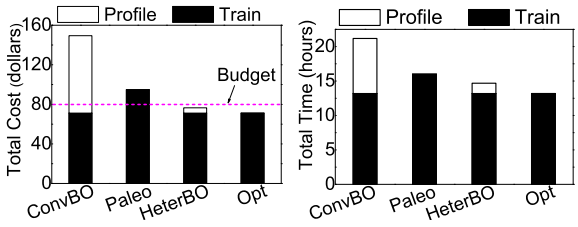


Fig. 12: Total time using random search with different number of profiling steps compared with HeterBO. We use whisker plot to demonstrate the distribution. I.e., box shows the first quartile to the third quartile and the horizontal line inside box shows the median, the circle dots and x dots are the maximum and medium value respectively.



(a) Total Cost (b) Total Time

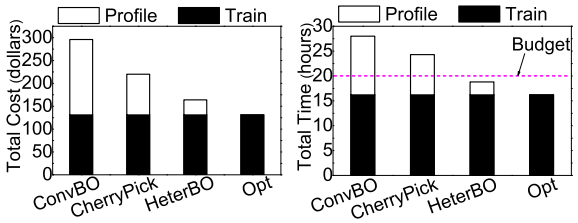
Fig. 13: Comparison of (a) total cost and (b) total time among ConvBO, Paleo, and HeterBO under the scenario of total budget capped by $80 using Inception-v3 with ImageNet dataset and TensorFlow platform. Opt is the optimal training cost/time for reference.

cost. However, as the cluster grows bigger, nuances like communication topology demonstrates bigger impacts on training. These nuances are particularly hard to capture by analytical modeling. Given Paleo does not consider these nuances, it fails to find the optimal configuration. HeterBO, on the other hand, finds an almost optimal configuration, while keeping the overall cost under budget.

We then compare HeterBO to the state-of-the-art BO based approach CherryPick under the scenario of limit time – 20 hours. CherryPick is also built atop of ConvBO with prior information, but instead of considering ML specific prior, it trims search space based on experience. Here we exclude the worse performing instance types in search to favor CherryPick as such prior is difficult to obtain in practice. The evaluation results are exhibited in Fig. 14. CherryPick runs over the budget even with narrowed search space using prior information as it does not consider the heterogeneous profiling cost nor aware of constraints when deciding the next profiling point. As expected, HeterBO achieves the training time limit with low profiling cost, thanks to its awareness of heterogeneous profiling cost and smart mechanism to protect "over profiling".

D. Robustness and Adaptivity of HeterBO

In this section, we verify the robustness of HeterBO by applying it to different ML models, training platforms, communication protocols, and different use scenarios. Fig. 15



(a) Total Cost (b) Total Time

Fig. 14: Comparison of (a) total cost and (b) total time among ConvBO, CherryPick, and HeterBO under the scenario of total time limit of 20 hours using Char-RNN model and TensorFlow platform. Opt is the optimal training cost/time for reference.

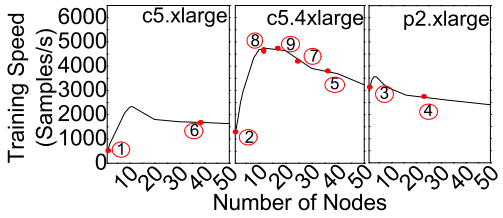


Fig. 15: How HeterBO searches for the optimal Char-RNN over TensorFlow configuration with both horizontal and vertical scaling options. HeterBO progressively narrows down the search space with a budget of $120 in mind.

illustrates how HeterBO finds the optimal configuration for training Char-CNN in 9 steps. HeterBO first profiles each instance type with only 1 instance to get a sense of their performance in the interest of profiling cost. HeterBO then starts exploring with step 4, 5, 6, trying to find an interval to exploit. Finally, HeterBO starts exploiting within the space between step 2 and 5 for the optimal configuration.

We then extend the evaluations to recently proposed BERT model, and implement it using both TensorFlow and MXNet. BERT model has more than 340 million trainable parameters, and it is trained with ring all-reduce communication topology instead of parameter server. The results are depicted in Fig. 16 and Fig.17, respectively. Similar exploring and exploiting procedures can be seen in both experiments, confirming that HeterBO is general and robust in terms of handling various ML models, platforms, and communication typologies.

Lastly, we evaluate the adaptivity of HeterBO by changing the budget requirements when training ResNet with CIFAR10 dataset in cloud. We compare HeterBO with the standard budget-oblivious ConvBO and CherryPick as well as their strengthened version by improving them to be budget-aware, named as BO_imprd and CP_imprd respectively. Again, we favor CherryPick by eliminating the sub-optional instance types and narrow down to only search within the optimal `c5n.4xlarge` instance type (i.e., no need to search scale-up dimension).

The results are shown in Fig. 18. Fig 18(a) shows the total cost comparison and Fig 18(b) shows the corresponding total time. As expected, for the total cost, strengthened ConvBO and CherryPick would stop the profiling process in time to

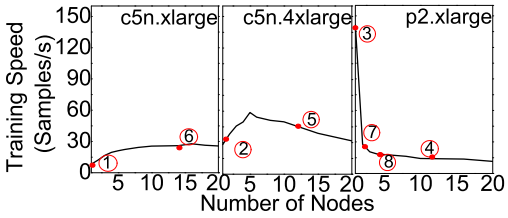


Fig. 16: How HeterBO searches for the optimal BERT over TensorFlow configuration with both horizontal and vertical scaling options. The budget is $100.

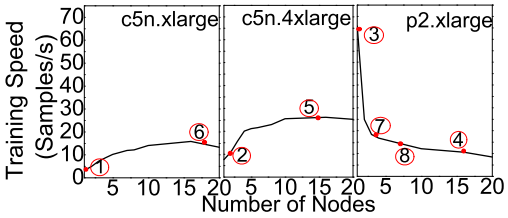


Fig. 17: How HeterBO searches for the optimal BERT over MXNet configuration with both horizontal and vertical scaling options. The budget is $120.

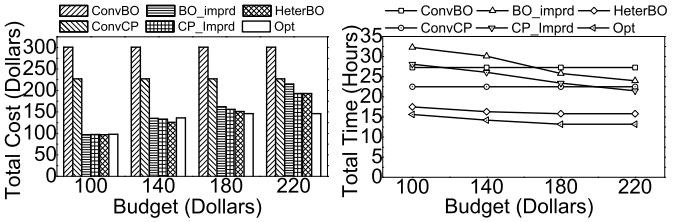
comply with the budget constraint. However, HeterBO still outperforms them in larger budget as HeterBO tries to narrow down the search space through the most cost-efficient way by taking into consideration of the heterogeneous profiling cost and ML specific prior. As we can see, with higher budget, HeterBO manages to find better configurations without significantly increasing cost as opposed to other methods. For total time, HeterBO outperforms all other approaches in all budget ranges, and up to 3.1× and 2.34× compared with ConvBO and CherryPick respectively to be specific.

E. Scalability of HeterBO

We evaluate the scalability of HeterBO by demonstrating the speedup and cost saving of HeterBO over ConvBO for models with different number of parameters: 6.4M(AlexNet), 60.3M(ResNet), 340M(Bert), 8B and 20B (ZeRO [31]). Due to the resource limitation, the results of model size 8B and 20B are simulated based on the training speed and system settings from ZeRO. The speedup improves form 1.3× to 6.5× and the cost saving increases from 69% to 92% as model size increases. Larger model size results in larger deployment search space, therefore, HeterBO achieves much better scalability compared to ConvBO, thanks to HeterBO's search cost awareness and ML prior employment.

VI. Related Work

Proteus [32] exploits transient revocable resources on public cloud to train ML models cheaper and faster. *Optimus* [33] dynamically schedules deep learning training tasks in a shared cluster to minimize job completion time. *MArk* [34] is a system supports cost-efficient ML inference. [7] employs performance modeling and scalability optimization to achieve efficient ML training. HeterBO significantly differs from the above work as it targets at quick and efficient ML training task deployment



(a) Total cost VS budget. (b) Total time VS budget.

Fig. 18: Sensitivity analysis of (a) total cost VS different budget constraints and (b) total time VS different budget constraints. We compare ConvBO, CherryPick, strengthened ConvBO (BO_imprd), strengthened CherryPick (CP_imprd), HeterBO against the optimal (Opt).

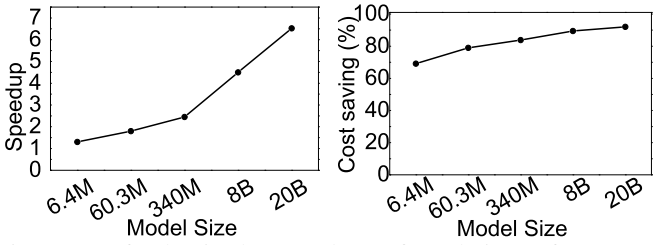


Fig. 19: Left plot is the speed-up of total time of HeterBO over ConvBO as model size increases. The right plot is the total cost saving by using HeterBO as model size increases.

on public cloud supporting user defined performance and cost objective and constraints.

Bayesian Optimization (BO) has been developed to find the optimal solution for black box functions [35], [36]. BO is employed in finding optimal Deep Neural Network structures for Deep Learning (DL) jobs [11], [37], [38], adaptive MCMC [39], information extraction [40] and hyper-parameter tuning [41]–[43]. However, our solution uses BO methods in a different way. Instead of changing the internals of ML structures or the training algorithms, which may lead the trained models to be less accurate or the training jobs not to converge, HeterBO aims to find an optimal deployment for speeding up the training process.

General approaches have been recently studied for optimizing the deployment of jobs in cloud. *PARIS* [44] provides performance prediction by combining a large amount of offline and online performance statics across different workloads and VM types. *Arrow* [13] proposes an augmented BO method by adding low level metrics into the performance model and using Extra-Trees algorithm as the surrogate model. *CherryPick* [12] is the most related work that builds BO based model to identify the most suitable configuration for targeting workloads. We want to emphasize that HeterBO employs a very different search strategy than CherryPick: CherryPick manually limits the search space based on experience while HeterBO does not trim any search space according to experience. Instead, HeterBO makes judicious decisions by considering a heterogeneous search space and uses a protective mechanism to reduce the risk of "randomly" exploring the more expensive search

region to avoid unnecessary high exploring cost.

VII. Conclusion

To support efficient MLaaS training deployment in public cloud with user defined time and cost requirements, we develop a fully automated MLaaS training Cloud Deployment system (MLCD). MLCD is driven by a highly efficient HeterBO search method that takes into account the heterogeneous exploration cost and machine learning specific prior, which are neglected by existing solutions. Experimental evaluation in AWS demonstrates the effectiveness and robustness of the HeterBO search method and MLCD system.

Acknowledgment

This work is supported in part by the following grants: National Science Foundation CCF-1756013, IIS-1838024 (using resources provided by Amazon Web Services as part of the NSF BIGDATA program), Hong Kong RGC ECS grant 26213818, and Amazon Web Services Cloud Credits for Research Award. Cheng Li is supported by the National Nature Science Foundation of China (Grant No. 61802358). We thank the anonymous reviewers for their insightful feedback.

References

[1] M. Ribeiro, K. Grolinger, and M. A. Capretz, "Mlaas: Machine learning as a service," in *ICMLA*. IEEE, 2015.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *arXiv*, 2018.

[3] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *NIPS*, 2019.

[4] Synced, "The staggering cost of training sota ai models," https://syncedreview.com/2019/06/27/the-staggering-cost-of-training-sota-ai-models/, 2019.

[5] W. Tan, L. Cao, and L. Fong, "Faster and cheaper: Parallelizing large-scale matrix factorization on gpus," in *HPDC*, 2016.

[6] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *ISCA*, 2010.

[7] F. Yan, O. Ruwase, Y. He, and T. Chilimbi, "Performance modeling and scalability optimization of distributed deep learning systems," in *ACM SIGKDD*, 2015.

[8] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks," in *ICLR*, 2017.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," in *Nature*, 2015.

[10] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann, "Predicting cloud performance for hpc applications: A user-oriented approach," in *IEEE/ACM CCGRID*, 2017.

[11] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *NIPS*, 2012.

[12] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics," in *NSDI*, 2017.

[13] C.-J. Hsu, V. Nair, V. W. Freeh, and T. Menzies, "Arrow: Low-level augmented bayesian optimization for finding the best cloud vm," in *ICDCS*, 2018.

[14] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016.

[15] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, and V. K. Asari, "The history began from alexnet: A comprehensive survey on deep learning approaches," in *arXiv*, 2018.

[16] http://bit.ly/2OQzWIG, accessed: 2020-02-18.

[17] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," in *NIPS*, 2012.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[19] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *arXiv*, 2017.

[20] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *OSDI*, 2014.

[21] Microsoft, "Microsoft Azure cloud computing platform & services," https://azure.microsoft.com/en-us/, 2019.

[22] D. C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *IJCAI*, 2011.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[24] M. A. Osborne, R. Garnett, and S. J. Roberts, "Gaussian processes for global optimization," in *LION3*, 2009.

[25] R. Baptista and M. Poloczek, "Bayesian optimization of combinatorial structures," in *arXiv*, 2018.

[26] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," in *Journal of Global Optimization*, 2001.

[27] C.-J. Hsu, V. Nair, T. Menzies, and V. W. Freeh, "Scout: An experienced guide to find the best cloud configuration," *arXiv*, 2018.

[28] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *OSDI*, 2016.

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

[30] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS*, 2017.

[31] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimization towards training a trillion parameter models," in *arXiv*, 2019.

[32] A. Harlap, A. Tumanov, A. Chung, G. R. Ganger, and P. B. Gibbons, "Proteus: agile ml elasticity through tiered reliability in dynamic resource markets," in *EuroSys*, 2017.

[33] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in *EuroSys*. ACM, 2018.

[34] C. Zhang, M. Yu, W. Wang, and F. Yan, "Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving," in *USENIX ATC*, 2019.

[35] J. Mockus, *Bayesian approach to global optimization: theory and applications*. Springer Science & Business Media, 2012.

[36] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," in *arXiv*, 2010.

[37] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *arXiv*, 2009.

[38] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams, "Scalable bayesian optimization using deep neural networks," in *ICML*, 2015.

[39] N. Mahendran, Z. Wang, F. Hamze, and N. De Freitas, "Adaptive mcmc with bayesian optimization," in *AIS*, 2012.

[40] Z. Wang, B. Shakibi, L. Jin, and N. de Freitas, "Bayesian multi-scale optimistic optimization," in *AIStat*, 2014.

[41] K. Swersky, J. Snoek, and R. P. Adams, "Multi-task bayesian optimization," in *NIPS*, 2013.

[42] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag, "Collaborative hyperparameter tuning," in *ICML*, 2013.

[43] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. De Freitas, "Bayesian optimization in high dimensions via random embeddings," in *IJCAI*, 2013.

[44] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz, "Selecting the best vm across multiple public clouds: a data-driven performance modeling approach," in *SoCC*, 2017.