BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 06/06 pp208–220
Volume 4, Number 3, September 2021
DOI: 10.26599/BDMA.2021.9020004

# AIPerf: Automated Machine Learning as an AI-HPC Benchmark

Zhixiang Ren*, Yongheng Liu, Tianhui Shi, Lei Xie, Yue Zhou, Jidong Zhai,
Youhui Zhang, Yunquan Zhang, and Wenguang Chen*

**Abstract:** The plethora of complex Artificial Intelligence (AI) algorithms and available High-Performance Computing (HPC) power stimulates the expeditious development of AI components with heterogeneous designs. Consequently, the need for cross-stack performance benchmarking of AI-HPC systems has rapidly emerged. In particular, the de facto HPC benchmark, LINPACK, cannot reflect the AI computing power and input/output performance without a representative workload. Current popular AI benchmarks, such as MLPerf, have a fixed problem size and therefore limited scalability. To address these issues, we propose an end-to-end benchmark suite utilizing automated machine learning, which not only represents real AI scenarios, but also is auto-adaptively scalable to various scales of machines. We implement the algorithms in a highly parallel and flexible way to ensure the efficiency and optimization potential on diverse systems with customizable configurations. We utilize Operations Per Second (OPS), which is measured in an analytical and systematic approach, as a major metric to quantify the AI performance. We perform evaluations on various systems to ensure the benchmark's stability and scalability, from 4 nodes with 32 NVIDIA Tesla T4 (56.1 Tera-OPS measured) up to 512 nodes with 4096 Huawei Ascend 910 (194.53 Peta-OPS measured), and the results show near-linear weak scalability. With a flexible workload and single metric, AIPerf can easily scale on and rank AI-HPC, providing a powerful benchmark suite for the coming supercomputing era.

**Key words:** High-Performance Computing (HPC); Artificial Intelligence (AI); automated machine learning

## 1 Introduction

Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) have drawn tremendous attention in recent years. DL requires a training process[1], which is essentially a multidimensional fitting, automatically adjusting the weights (parameters) of a neural network. As learnable data grow at an unprecedented rate, High-Performance Computing (HPC) machines are needed in large AI models to harness big data and extract complex abstractions[2]. Hybrid HPC models with AI surrogates reveal a collection of unique and novel opportunities for scientific breakthroughs and unforeseeable discoveries[3], business innovations, and other societal benefits. The increase in algorithmic advances of AI algorithms, available computing power, and data collections, as well as the demand for scalable and data-driven solutions, stimulate the convergence of AI and HPC machines[4]. However, the convergence[5] still faces multiple challenges, such as the effective and parallel implementation of algorithms on large-scale clusters, high bandwidth, and low latency communications between distributed workers, and high-speed interconnections to the Network File

• Zhixiang Ren, Yongheng Liu, and Yue Zhou are with Peng Cheng National Laboratory, Shenzhen 518000, China. E-mail: renzhx@pcl.ac.cn; yongheng.liu@pcl.ac.cn; zhouy@pcl.ac.cn.
• Tianhui Shi, Lei Xie, Jidong Zhai, Youhui Zhang, and Wenguang Chen are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: sth19@mails.tsinghua.edu.cn; xie-l18@mails.tsinghua.edu.cn; zhaijidong@tsinghua.edu.cn; zyh02@tsinghua.edu.cn; cwg@tsinghua.edu.cn.
• Yunquan Zhang is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100086, China. E-mail: zyq@ict.ac.cn.
* To whom correspondence should be addressed.
Manuscript received: 2021-03-01; accepted: 2021-03-12

System (NFS). HPC systems need to incorporate the support for various AI workloads on top of inconsistent accelerators and software frameworks for an AI-HPC adaption. Consequently, the need for an open and reliable benchmark suite to comprehensively evaluate the cross-stack performance of heterogeneous AI-HPC systems rapidly emerges, as shown in Fig. 1.

There are three major challenges in AI-HPC benchmarking. First, the benchmark workload needs to represent real problems running on AI-HPC, regarding hardware utilization, setup cost, and computing patterns. Second, the benchmark workload should preferably be auto-adaptive to various scales of machines without extra human effort. Third, a simple system metric on AI performance needs to be defined, and an approach to measure it needs to be designed. However, current HPC and AI benchmarks do not address these challenges. In particular, the de facto HPC benchmark, LINPACK, does not measure the cross-stack performance on AI without a representative workload. Furthermore, popular AI benchmarks, such as MLPerf[6], have representative AI workloads, but they have a fixed workload size and are often used to benchmark small systems. Moreover, they do not automatically scale with machines and require considerable human effort for tuning. This feature is fallacious because the increased computing power tends to be utilized to attack larger problems, instead of the same problem with less time. The fixed problems can not adapt to different scales of machines automatically, either.

Automated Machine Learning (AutoML) can search and optimize AI models more automatically and is receiving increasing attention in the AI community. As a representative AI application, AutoML basically contains all the critical components, regarding primary computing operations (e.g., sparse matrix multiplication), calculation precision (in FP-32 or lower),

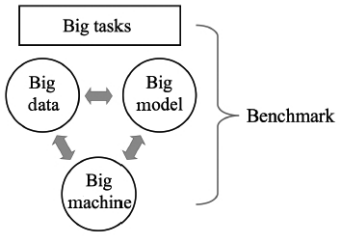



**Fig. 1 Convergence of AI and HPC with the growth of models, machines, data, and potential tasks. The benchmark should cover the system heterogeneity and reflect the cross-stack performance.**

and workflow in real AI scenarios. More importantly, AutoML can be implemented in a flexible style so that it automatically scales with the number of machines. Moreover, the pseudo-random generated architecture and extreme computational cost would address the evolving and diverse architectural designs in AI research and fully push the system limits in benchmarking. Considering all the advantages, AutoML is a desired workload, and we choose it to tackle the first two challenges. For the third challenge, we learn from the success of LINPACK and Top500, and utilize Operations Per Second (OPS) as our benchmark score to quantitatively measure the AI computing power. The OPS is measured in an analytical and systematic method to account for training and inference processes. With the auto-adaptive workload and single metric measurement, our benchmark can easily rank various sizes of machines from small clusters to large AI-HPC.

In summary, our main contributions are in the following:

- We propose AutoML as a representative and auto-adaptive workload to establish an end-to-end benchmark suite for AI-HPC.
- Our implementation is highly parallel and customizable to keep the optimization potential on diverse systems.
- We propose an analytical and speedy approach to calculate the operation rate of neural networks with different architectures.
- We evaluate our benchmark on various systems to ensure the benchmark's scalability and stability.

The rest of this paper is organized as follows. In Section 2, we review the existing HPC and AI benchmarks and point out their downsides for AI-HPC benchmarking. In Section 3, we briefly review AutoML and the popular frameworks for AI. In Section 4, we describe the details of our algorithms, implementations, and measurements. In Section 5, we evaluate our benchmark on different scales of machines. In Section 6, we summarize our work. The source code, specifications, and detailed procedures are publicly accessible on GitHub.

## 2 Related Work

### 2.1 HPC benchmarks

LINPACK is a popular HPC benchmark nowadays. It is essentially an algebra library that solves a dense system of linear equations, that is the heart of many

computational science problems. However, LINPACK is not suitable for benchmarking AI-HPC for three reasons: First, the problem size is usually manually decided and cannot be automatically scaled based on tested machines. Second, LINPACK provides little information about the setup cost and input/output (I/O) ability, which are critical data-intensive applications like AI. This concern is problematic because most algorithms perform more data motions than arithmetic[7]. Third, the calculation is performed in FP-64, whereas most AI applications typically only require FP-32 or even FP-16. HPL-AI mixed-precision benchmark[8] was developed based on LINPACK to highlight the third issue, but it still suffers from the other two issues. Other HPC benchmarks, including NASA Parallel Benchmarks, SLALOM[9], and HINT[7], do not utilize workloads that can represent real AI scenarios and therefore share the same problems as LINPACK. Although we cannot use the existing HPC benchmarks for AI-HPC, they still inspire us in the benchmark design. For example, the biggest challenge in benchmarking is to create a single workload that can capture all the features of real applications and be auto-adaptive without a fixed problem size. Moreover, further performance optimization with customizable configuration is encouraged, as long as the user does not specialize the program to input data. Lastly, a single number metric is preferred for easy comparison and ranking.

### 2.2 AI benchmarks

A fair and inclusive comparison of machine computing power on AI applications is not trivial. As the opposite of monoculture, the system's heterogeneity, and the variety of AI workloads, as well as the stochastic nature of approaches, make the benchmarking complicated. Previous AI benchmarks attempt to highlight the challenges by incorporating different hardware systems[10–14], software frameworks[15] or AI algorithms[16–19]. More recently, end-to-end benchmarks[6, 20–23] were developed to simultaneously evaluate hardware systems and AI algorithms. MLPerf[6], the arguably most accepted AI benchmark so far, uses time-to-accuracy to measure the co-performance of hardware and software. This metric is an indirect quantification of the computing ability compared to OPS, which is our metric. Because MLPerf is composed of multiple micro-tasks, each one would result in a different measurement. Although this approach makes the benchmark more accurate on various applications, it also makes the comparison and ranking more difficult. Moreover, the limited workloads in MLPerf have insufficient scalability with fixed problem size. Other AI benchmarks have similar drawbacks as MLPerf. Overall, the existing AI benchmarks are not suitable as AI-HPC benchmarks because of the following reasons:

- Existing AI benchmarks have a fixed problem size and therefore limited scalability.
- Existing AI benchmarks do not provide a single and direct measurement to quantify performance.

## 3 Background

### 3.1 AutoML

The development of AI solutions has mostly relied on a complex model design, which heavily involves human expertise and is extremely time-consuming. To explore the architecture space more efficiently and optimize the model automatically, AutoML[24] has emerged as the AI model complexity has exponentially increased in recent years. Surprising as it may seem, AutoML is already mature enough to rival human experts to make a real impact on AI research. Overall, AutoML is inherently computing intensive, highly scalable, and representative of AI-like workflows. Considering all the unique advantages, we chose AutoML as our benchmark workload. As shown in Fig. 2, AutoML contains various parts[25]: The first part is data preparation, which involves data collection and data cleaning. The second part is feature engineering, including feature selection, feature construction, and feature extraction. Although data and features lay the foundations of AI performance, they depend on the application scenarios and are irrelevant to the machine computing power and are therefore not considered in our benchmark. The third part is to generate the neural architecture and optimal configuration (referred to as hyperparameters), which can have a significant impact on the performance. The

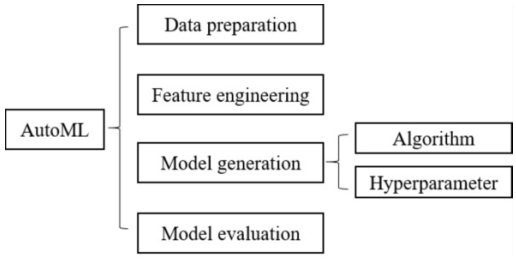



**Fig. 2 Overview of AutoML. We limit our attention to the model generation in this study.**

two main approaches for model generation are experts' manual design and the automated Neural Architecture Search (NAS)[26]. Without human intervention, NAS has the potential to generate novel architectures beyond imagination and can significantly boost performance. HyperParameter Optimization (HPO)[27] is essentially the optimization of the loss function over the complex configuration space. NAS and HPO can be implemented in a parallel manner to fully utilize the distributed resources. Finally, model evaluation measures the performance once the candidate model is generated. The simplest method is to conduct the inference on the test dataset for enough epochs. This is prohibitively expensive because there are numerous configurations for each neural architecture. In this study, we use warm-up and early stopping strategies[28] that stop the training once the validation loss flats. This method can allow quick measurements to a certain degree of accuracy.

### 3.2 Frameworks

**(1) DL frameworks:** DL frameworks provide a user-friendly application programming interface and transform programs in high-level languages into an internal representation of certain functionalities. The low-level efficient libraries, e.g., cuDNN, are invoked to execute primary operations, such as matrix multiplication. Multiple solutions with desired performance exist[29]; therefore, implementation and customized setups vary while maintaining similar results. The difference is critical as the training process is stochastic and approximate intrinsically. An open-source framework with enough community support would be a decent candidate for building the benchmark. According to GitHub, the most popular DL frameworks are TensorFlow[30], Keras[31], and PyTorch[32]. TensorFlow is an open-source library for low-level numerical calculation with static computational graphs, where operations are written as high-performance C++ binaries with high-level Python abstractions. Keras is a high-level library wrapper that is built on top of frameworks, such as TensorFlow, and provides off-the-shelf but often inflexible models. PyTorch utilizes dynamic computation graphs that are modifiable at runtime, but the "Pythonic" nature makes it less efficient for benchmarking purposes. After carefully comparing different frameworks[33,34], we choose TensorFlow in our benchmark evaluation for the following reasons:

- TensorFlow is so far the most popular open-source DL framework with a large and active community supported from Google.
- TensorFlow is efficient, user-friendly, and easy to debug (with TensorBoard) regarding the numerical computations for research and deployment.
- TensorFlow supports various systems with high performance and scalability.

**(2) AutoML frameworks:** Various works have been done to develop user-friendly AutoML frameworks[35,36], including Neural Network Intelligence (NNI), tree-based pipeline optimization tool, and auto-sklearn. NNI is a popular open-source toolkit that automates the DL model design process. One key feature is the rich collection of algorithms to generate neural architectures and optimizing hyperparameters, as well as a simple interface for more user-defined algorithms. Other frameworks focus on the AutoML pipeline optimization, especially data preprocessing and feature engineering, which is irrelevant for benchmarking the computing power. Therefore, we choose to build our own benchmark suite on top of NNI.

## 4 Methodology

### 4.1 NAS

Notable successes of neural architecture designs[37–42] in the past few years have drawn enormous attention in the AI research community. The manual design of neural architecture requires tremendous human effort, sometimes even domain knowledge in an ad-hoc fashion. By contrast, the architectures are automatically generated by selecting and combining primary operations (e.g., convolution) with NAS approaches that can be categorized into three abstraction levels[26]: search space, search strategy, and performance estimation strategy. The major search strategies (algorithms)[25] include random search[27], reinforcement learning[43], evolutionary[44], Bayesian optimization[45], and gradient-based method[46]. Research around NAS is typically exploring three dimensions of abstractions simultaneously using various algorithms to search for different combinations of building blocks. In the spirit of transfer learning and knowledge inheritance, Ref. [47] proposed network transformation that transforms a pre-trained parent network to a more complex child network while preserving the input and output consistency. The knowledge represented by the neural architecture is transformed from the parent network to the child network. Reference [48] first dubbed "network morphism" that can perform multiple transformation

operations including width, depth, kernel size, and skip operation. Reference [49] proposed an open-source framework (Auto-Keras), which is part of NNI, to perform network morphing guided by Bayesian optimization. Although every method has its own advantages, we choose the implementation in Ref. [48] as our baseline for developing the benchmark. We choose residual network (ResNet-50) as the initial model[37], because ResNet-50 is one of the de facto showcase models in the current DL community and contains basically all the AI-related computation primitives. We modify the morphism so that each transformation step adds a block (convolutional layer, batch normalization[50], and activation function all together) instead of just one layer. In addition, we adapt this implementation to suit benchmarking in a parallel and distributed way, which is explained later.

### 4.2 HPO

HPO problems can be viewed as the identification of optimal model configurations of all related hyperparameters. Similar to NAS, HPO has three abstractions[24]: search space, search approach, and evaluation method. Various search approaches can be applied to select the best hyperparameter combinations, including grid search[51], random search[27], Bayesian optimization[52], and heuristic search, such as evolutionary[44]. In our case, the search space is defined by the hyperparameters that are more directly related to the computational cost, including the batch size and kernel size, to reduce the randomness for benchmarking purposes. We use the stochastic gradient descent with momentum[53] as the optimizer because it requires less memory and is more efficient. We evaluated different optimization approaches and then compared the validation accuracy on the test dataset. The results of multiple experiments on CIFAR10 show that Bayesian optimization (in our case, tree-structured Parzen estimator) slightly outperforms other methods. Similar to NAS, we use this fixed algorithm to simultaneously optimize the batch size and kernel size. In our benchmark workflow, the HPO is separately performed after the NAS process on each worker.

### 4.3 Workflow

As mentioned, we choose NNI (V1.5) as a baseline to adapt to our benchmark suite. The original NNI framework is implemented with a "primary-replica" architecture and performs the NAS and HPO on the primary node, which is the bottleneck on large clusters. Moreover, not all operations in AutoML run on AI accelerators, such as model generation and data movement. Consequently, the AI accelerator idles because of the potential bottleneck on the CPU or disk I/O. In addition, the model generation is time-consuming and can be implemented with thread parallelism on CPUs. To address these problems and fully appreciate all computing resources in a balanced way, we need to effectively distribute the computations and use proper parallelism[54] on the CPU and AI accelerator. Therefore, we modify the NNI framework in various aspects, as shown in Fig. 3, including performing the model generation and training on replica nodes asynchronously, utilizing replica nodes' CPUs parallelly to generate new architectures, and performing training parallelly with all available AI accelerators on each replica node. We utilize data parallelism with a synchronous all-reduce strategy, so that all AI accelerators can train on different partitions of data and results in individual gradients, which are then aggregated all together at each step. We summarize our benchmark workflow as follows:

- User accesses the primary node through Secure SHell (SSH), collects information about replica nodes, and creates a Simple Linux Utility for Resource Management (SLURM) configuration script.

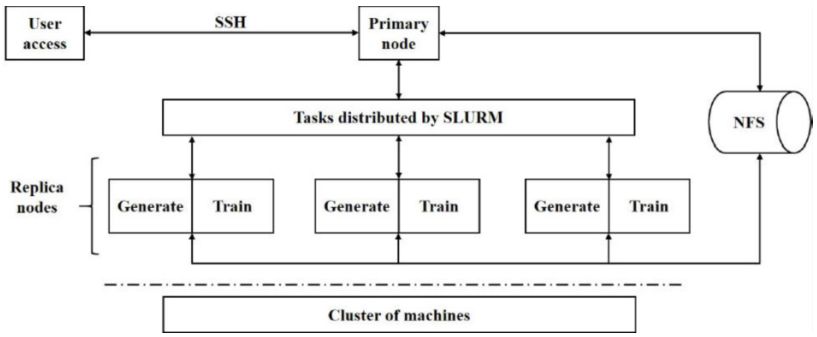



**Fig. 3 Schematic diagram of the benchmark workflow.**

• The primary node dispatches workloads with SLURM to replica nodes corresponding to the requested and available resources parallelly and asynchronously.

• The replica nodes receive the workloads and perform architecture searching and model training parallelly. The CPUs on replica nodes search for new architectures based on the current historical model list, which contains detailed model information and accuracy on the test dataset, and then store the candidate architecture in the NFS for later training.

• The AI accelerators on replica nodes load the candidate architecture and data, utilize data parallelism to train along with HPO, and then store the results in the historical model list.

• The running terminates once the condition is satisfied (e.g., reaching user-defined time). The final results are calculated based on the recorded metrics and then reported.

### 4.4 Measurement

At present, Floating-point Operations Per Second (FLOPS) or OPS is the most cited performance metric to reflect the overall computing ability of HPC. Our benchmark utilizes OPS as the major metric (score) to directly describe the computing power of AI accelerators. Because the processing time can be easily recorded, we only need to count the total operations, and all computations are required to be conducted with the floating points of at least FP-16 precision. Toolkits, such as NVIDIA profiling tools (nvprof), can record the executed operation count through a kernel replay, which is exceptionally slow. This method is also limited to NVIDIA hardware and is not suitable for various platforms. Inspired by LINPACK, we treat the operation counting as a mathematical problem, to calculate the operation count needed to finish the complex computation in the training and validation processes without any optimization. For a given dataset and model with specific hyperparameters, the theoretical operation needed to finish the training or validation is predetermined. If the hardware or software has any special optimization, then the operation count is reduced or the execution is faster, eventually resulting in a higher OPS.

To analytically calculate the operation count, we need to understand the training and validation processes. DL libraries, such as TensorFlow, use computational graphs to represent the computations and guide the workflow. A computational graph is a directed acyclic graph where nodes represent variables or operations and edges represent function arguments (data dependency). Each computation is essentially a node, so that variables feed values into operations and operations feed the outputs into other operations. Computational graphs can compose complex models with simple functions and enable automatic differentiation to train the neural networks. Backpropagation[55] is a reverse-mode automatic differentiation[56], which efficiently and recursively applies the chain rule to compute gradients of inputs and parameters and other intermediates along with computational graphs. As shown in Algorithm 1, backpropagation has two parts: Forward Pass (FP) that computes the results of operations and saves intermediate values needed for gradient computation in the memory, and Backward Pass (BP) that applies the chain rule to compute the gradients of the loss function with respect to the inputs (multiply Jacobian matrices by gradients).

The total operation count is the sum of that in FP and BP, which includes operations to calculate the gradients and the operations to update the parameters with gradient descent. Most computations in neural networks are matrix multiplication, whose ininer products $y = w[0] \times x[0] + w[1] \times x[1] + \cdots + w[n-1] \times x[n-1]$ have $n$ Multiply-ACcumulate (MAC) operations and correspond to roughly $2n$ operations. The gradient descent procedure can be described as repeat $\theta_i = \theta_i + \alpha \frac{\mathrm{d}L}{\mathrm{d}\theta_i}$, where $\theta_i$ and $\alpha$ are the weight and learning rate, respectively, until convergence, so the operation needed is equivalent to one MAC for one parameter

**Algorithm 1 Backpropagation[55]**

FP:

1. Define the computational graph where each node represents a variable (parameters and intermediates).
2. Visit each node in a topological order to compute the variables with corresponding operations and store the values at the nodes.

BP:

3. Initialize the loss gradients $\frac{\mathrm{d}L}{\mathrm{d}y}$ and all local partial derivatives $\frac{\mathrm{d}y}{\mathrm{d}x_i}$.
4. Visit each node in a reverse topological order to compute the loss gradients with respect to local variables with chain rule: $\frac{\mathrm{d}L}{\mathrm{d}x_i} = \frac{\mathrm{d}L}{\mathrm{d}y} \times \frac{\mathrm{d}y}{\mathrm{d}x_i}$.
5. Return $\frac{\mathrm{d}L}{\mathrm{d}x_i}$ for all variables.

in one BP. We break down the original and morphed models into several components (layers) and analytically compute the operation count needed of each layer in the FP, as listed in Table 1. For the convolutional layer, the input image dimension is $H_i \times W_i \times C_i$, the output dimension is $H_o \times W_o \times C_o$, and the kernel (filter) size is $K \times K$. For the dense layer, the input is $C_i$ and the output is $C_o$. Following the convention in Ref. [57], the operation weight of MAC is 2, the weight of add/subtract/multiply/comparison is 1, the weight of divide (div for short)/sqrt is 4, and the weight of the special operation, such as exponential (exp for short), is 8. The operation count is only an approximation. The detailed descriptions of each layer are presented in Refs. [37, 50].

The analytical method of the computing operation is more complicated in the BP process. The convolution in FP can be described as $O_{ij} = \sum_{m=1}^{k-1}\sum_{n=1}^{k-1} X(i-m, j-n)F(m,n)$, where $O_{ij}$ is the output, $X\ (i-m, j-n)$ is the input, and $F(m,n)$ is the filter (kernel). The partial derivatives of local parameters ($\frac{\partial O}{\partial F}$) and local intermediates ($\frac{\partial O}{\partial X}$) can be easily derived and are used in the gradient calculation. Applying the chain rule, we have the parameters' and intermediates' gradients by multiplying the loss gradients with the local gradients,

$$\begin{cases} \dfrac{\partial L}{\partial F_i} = \sum\limits_{k=1}^{m} \dfrac{\partial L}{\partial O_k} \times \dfrac{\partial O_k}{\partial F_i}, \\ \dfrac{\partial L}{\partial X_i} = \sum\limits_{k=1}^{m} \dfrac{\partial L}{\partial O_k} \times \dfrac{\partial O_k}{\partial X_i} \end{cases} \tag{1}$$

By substituting the derivatives ($\frac{\partial O}{\partial F}$ and $\frac{\partial O}{\partial X}$), we can express the backpropagation,

$$\begin{cases} \dfrac{\partial L}{\partial F} = \text{Convolution}\left(\text{Input } X, \text{Loss gradient } \dfrac{\partial L}{\partial O}\right), \\ \dfrac{\partial L}{\partial X} = \text{Full Convolution}\left(\text{Flipped } F, \text{Loss gradient } \dfrac{\partial L}{\partial O}\right) \end{cases} \tag{2}$$

Therefore, the total operation count needed to calculate all gradients is roughly twice as that in FP. The total parameter in the convolution layer (without bias) is $K \times K \times C_i \times C_o$, so the operation needed to update all parameters with the gradient descent method is $2 \times K \times K \times C_i \times C_o$. Considering all steps, we can have the total operation count in BP, as shown in Table 2. Because $K$, $C_i$, and $C_o$ are typically small values in the convolutional layers, the total operations in BP are roughly twice that of FP.

For the dense layer $Y = W^{\mathrm{T}}X + B$, where $W$ and $B$ are weight and bias, respectively, the intermediates gradients can be obtained by multiplying the loss gradients ($\frac{\partial L}{\partial Y}$) with the Jacobian matrices of intermediates ($\frac{\partial Y}{\partial X}$). Similarly, the weights' gradients are $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y} \times \frac{\partial Y}{\partial W}$. In both cases, the operation count needed is the same as that in FP. The bias gradient is $\frac{\partial L}{\partial B} = \frac{\partial L}{\partial Y} \times \frac{\partial Y}{\partial B} = \frac{\partial L}{\partial Y}$ since $\frac{\partial Y}{\partial B} = 1$, therefore resulting in no extra operation. The total parameter in a dense layer (with bias) is $(C_i + 1) \times C_O$, and the total count operation needed in the BP of the dense layer is shown in Table 2. Unlike the convolutional layer, the operation count of the dense layer in BP is more than tripled of that in FP (as shown in Table 3). The operation in the BP of the other layers, including batch normalization, activation function (ReLU), element-wise addition layer, max-pooling, global-pooling, and softmax layer, are all ignorable for practical purposes. We confirmed our analytical method by comparing the results of ResNet-50 on ImageNet[58]

**Table 1 Analytical operation count of each layer (per image) in the FP.**

| Layer | Operation count in the FP |
|---|---|
| Convolutional | MAC $= K \times K \times C_i \times H_o \times W_o \times C_o$ |
| Dense | MAC $= C_i \times C_o$ |
| Batch normalization | MAC = add = div $= H_i \times W_i \times C_i$ |
| ReLU | comparison $= H_o \times W_o \times C_o$ |
| Add | add $= H_o \times W_o \times C_o$ |
| Max-pooling | comparison $= K \times K \times H_o \times W_o \times C_o$ |
| Global-pooling | add $= H_i \times W_i \times C_i$; div $= C_i$ |
| Softmax | exp = add = div $= C_o$ |

**Table 2 Analytical operation count of each layer (per image) in the BP.**

| Layer | Operation count in the BP |
|---|---|
| Convolutional | MAC $= 2 \times (K \times K \times C_i \times H_o \times W_o \times C_o) + (K \times K \times C_i \times C_o)$ |
| Dense | MAC $= 2 \times C_i \times C_o + (C_i + 1) \times C_o$ |

**Table 3 Analytical operation count of each layer (per image) in the FP and BP together.**

| Layer | FP | BP | BP/FP | Total |
|---|---|---|---|---|
| Convolutional | $7.71\times10^9$ | $1.52\times10^{10}$ | 1.9755 | $2.29\times10^{10}$ |
| Dense | $4.10\times10^6$ | $1.23\times10^7$ | 3.0005 | $1.64\times10^7$ |
| Batch normalization | $7.41\times10^7$ | $1.91\times10^3$ | 0.000 03 | $7.41\times10^7$ |
| ReLU | $9.08\times10^6$ | 0 | 0 | $9.08\times10^6$ |
| Max-pooling | $1.81\times10^6$ | 0 | 0 | $1.81\times10^6$ |
| Average-pooling | $1.00\times10^5$ | 0 | 0 | $1.00\times10^5$ |
| Add | $5.52\times10^6$ | 0 | 0 | $5.52\times10^6$ |
| Softmax | $2.10\times10^4$ | 0 | 0 | $2.10\times10^4$ |
| Total | $7.81\times10^9$ | $1.52\times10^{10}$ | 1.9531 | $2.31\times10^{10}$ |

with a TensorFlow profiler (only computing operation in the FP) and NVIDIA profiling tools (computing operation in FP and BP). In our analytical method, we did not consider any hardware or software optimization that would result in such an effect. The operation count from this analytical approach is only related to the neural architecture, hyperparameter configuration, and data (e.g., image resolution). The optimizations that result in less operation count will speed up the training or validation processes and therefore higher final OPS. The details of other verifications of our OPS measure approach are elaborated in the Appendix.

The quantitative measurement of AI-HPC is not trivial, sometimes even conflicting, due to the diversity of workloads and metrics. One single metric, such as OPS, may not be sufficient to reflect the AI-HPC computation capabilities considering the hardware and software. For example, the data parallelism algorithm that is frequently applied in distributed ML will speed up the whole process at the cost of lower average AI accelerator utilization and OPS. Although one can separately present all relevant metrics, we intend to provide a metric to informatively characterize the system's overall performance. In general, an efficient AI-HPC would perform more computation and result in higher accuracy in less time. The empirical results[29] show that the accuracy on the validation dataset monotonically increases and then plateaus over time. In other words, the error (1-accuracy) decreases slowly over time. We would like to compensate for this effect with an increasing changing rate of the metric. Therefore, the absolute value of the partial derivative of the metric with respect to the error should increase with decreasing error. Furthermore, the partial derivative of the metric with respect to OPS should be independent of OPS to make the computation contribute to the metric uniformly. We use this metric as a regulated score in our benchmark to quantitatively measure the cross-stack performance of an AI-HPC, besides the OPS. According to the above conditions, we design our regulated score as

$$\text{regulated score} = -\ln(\text{error}) \times \text{OPS} \tag{3}$$

where error $\in (0, 1)$ and the negative sign keeps ln(error) positive. Consequently, the regulated score increases faster with lower error and linearly increases with OPS. For AI systems at the same machine scale but with different software optimizations, the regulated score can reflect the hardware and software co-performance. Therefore, we also provide it as a complementary result.

### 4.5 Fixed and customizable configuration

There are several rules in our benchmark for a fair comparison across various platforms. Using a "pencil-and-paper" manner[59], our benchmark also has customizable configurations that allow users to optimize the performance. First, the benchmark should run on a "primary-replica" architecture. The primary node is deployed on a strong server without any AI accelerator to dispatch tasks and collect all results from the replica nodes. The replica node is composed of one or multiple servers equipped with AI accelerator(s) and can be deployed with or without a container environment. The scale-up (multiple AI accelerators on each replica node) and scale-out (one AI accelerator on each replica node) configurations are supported. Second, the algorithms and search space used for AutoML are fixed, i.e., network morphism for NAS and Bayesian optimization for HPO, with the aforementioned operations and hyperparameters. The HPO only starts at the fourth round of training on each replica node, because the earlier rounds are trained insufficiently, which is also referred to as the warm-up process in this paper. A predicted accuracy, instead of the actual one, is used in the warm-up process. There is also a maximum limit on epoch and patience, which is the number of epochs to wait before an early stop if there is no progress on the validation dataset. Third, the dataset is fixed to be ImageNet, which has 1 281 167 and 50 000 $224 \times 224$ RGB images for training and validation, respectively. We kept the back-end DL framework and most hyperparameters open to further optimization. This method would partially relieve the performance dependency on manual designs and be more independent of the software part of the system. The data can be formatted in different ways corresponding to the DL framework. For example, the data loading with TFRecord is more efficient for TensorFlow. Fourth, our benchmark requires the minimum precision to be FP-16 and the maximum error to be 30%. A cumulative value of OPS was calculated at each timestamp (0.1-hour step), and the final value was considered as the score. The summarized configurations are shown in Table 4.

## 5 Evaluation

### 5.1 Setup

In our preliminary test, we verified our benchmark design (regarding algorithm and implementation) on our local machine with four NVIDIA 1080Ti GPUs based on the CIFAR10 dataset. The formal evaluation presented

**Table 4 Fixed and customizable configuration.**

| Configuration | Fixed and customizable setup |
|---|---|
| Server arrangement | Fixed: primary-replica |
| NAS method | Fixed: network morphism |
| HPO method | Fixed: Bayesian optimization |
| Dataset | Fixed: ImageNet |
| DL framework | Default: TensorFlow |
| Initial architecture | Fixed: ResNet-50 |
| Initial weight | Default: method in Ref. [60] |
| Batch size | Default: 448 |
| Optimizer | Default: gradient descent with momentum |
| Learning rate | Default: 0.1 with linear decay |
| Loss function | Default: categorical cross entropy |
| Maximum epoch | Default: 60 |
| Parallelism | Default: synchronous all-reduce |
| Parallel data transformation | Default: 48 |
| Minimum precision | Fixed: FP-16 |
| Maximum error | Fixed: 30% |

here was performed on two large clusters: GPU (NVIDIA V100) cluster and NPU (Huawei Ascend910) cluster. Both of them consist of multiple servers, each with two CPUs and eight AI accelerators (see Table 5 for hardware specifications). As a modern practice in AI research, we perform the evaluation in containers with the allocated resources and pre-assigned services for the consistency of the testing environment. We utilize Kubernetes to deploy the docker containers that wrap in all the dependencies, including the operating system, libraries, and workload codes, to provide the running environment. We use each physical server with the same hardware specifications as either a primary or a replica node for simplicity. The detailed information of the evaluation environment is shown in Table 6.

## 5.2 Performance

We run the benchmark on various scales of machines from 10 replica nodes with 80 GPUs up to 512 replica nodes with 4096 NPUs. All the intermediate results, including the generated architectures, hyperparameter configurations, accuracy at each epoch, and timestamps,

**Table 5 Hardware specifications of the two evaluated systems.**

| Component | GPU cluster | NPU cluster |
|---|---|---|
| Processor | Intel skylake 6151 | Huawei Kunpeng 920 |
| Memory | 2667 MHz DDR4 512 GB | 2933 MT/s DDR4 2048 GB |
| AI accelerator | NVIDIA V100 | Huawei Ascend910 |
| Storage | NVMe 5 TB | NVMe 5 TB |
| Ethernet network | InfiniBand 100 Gb/s | InfiniBand 100 Gb/s |

**Table 6 Evaluation environments of the two evaluated systems.**

| Component | CPU cluster | NPU cluster |
|---|---|---|
| Allocated resource | 30 CPU cores | 191 CPU cores |
| | 128 GB memory | 512 GB memory |
| | 8 NVIDIA V100 | 8 Huawei Ascend910 |
| Environment | Ubuntu 16.04 | Ubuntu 18.04 |
| | docker 18.09 | docker 19.03 |
| | SLURM 15.08 | SLURM 17.11 |
| | TensorFlow V2.2 | MindSpore V1.0 |
| | CUDA V10.1 | CANN V20.1 |
| | Python 3.5 | Python 3.7 |

are recorded in log files. Once the benchmarking process is finished, we run the data analysis toolkit to calculate the score along with other complementary results utilizing all the recorded information and then create a report.

In this study, we limit our evaluation to two major characteristics of the benchmark: stability and scalability. As for the stability characteristic, within the pre-assigned hours on various types and scales of AI accelerators, the cumulative OPS was calculated and is shown in Fig. 4 as the score. In both clusters, the cumulative OPS converges and increases steadily. The regulated score in Fig. 4 also converges because it is essentially

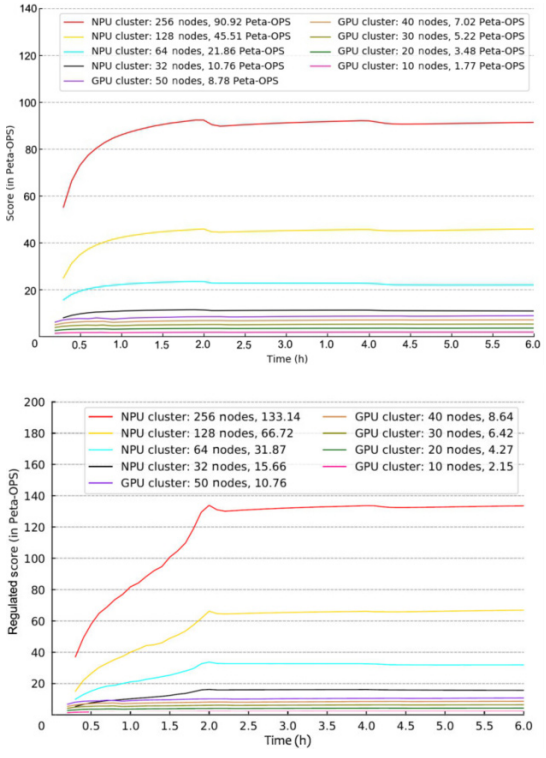



**Fig. 4 Benchmark scores and regulated scores (both in Peta-OPS) over time of evaluations with different scales of cluster nodes.**

just the OPS multiplied with the model performance as a coefficient. The regulated score has a similar behavior as the score. The results show the robustness and stability of our benchmark.

To ensure stability, we also monitored the GPU performance during the benchmarking process. We used the NVIDIA System Management Interface (nvidia-smi) to track the GPU utilization to show the percentage of time during which one or more kernels are occupied, along with the GPU memory utilization during the same time period. We developed a toolkit to extract real-time information with a 30 s sampling interval during the entire running time. As shown in Fig. 5, the GPU utilization and memory occupancy are both high during the training phase with the default benchmark configuration (for NVIDIA V100). As described in Section 4, the job size increases when the number of processing units increases. Hence, as for the scalability characteristic, the weak scaling test was performed, and the results are shown in Fig. 6. The benchmark shows a near-linear weak scalability on the two evaluated systems, which implies that our benchmark can evaluate even bigger systems, such as future exascale supercomputers. Due to the optimization of the benchmark configurations and system fluctuation, super-linear effect appears occasionally.

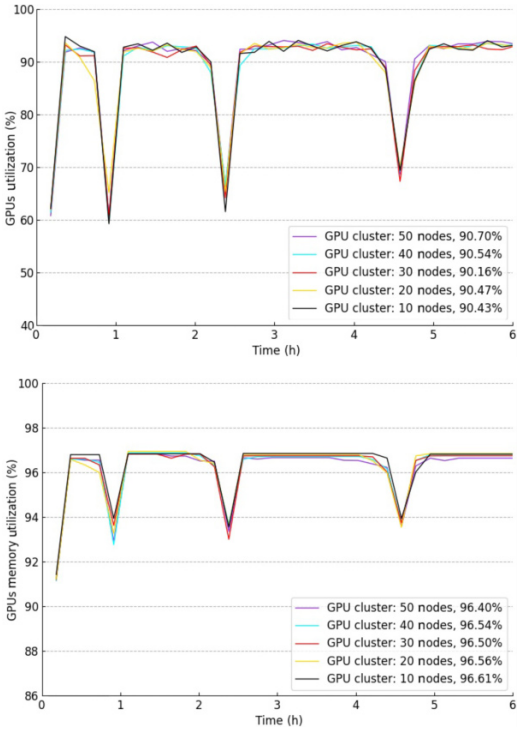



**Fig. 5 GPU utilization and their memory utilization of evaluations with different scales of machines measured by NVIDIA profiling tool. The average values are shown in the labels. The utilization drops during the inter-phase between the training stages come from the data loading and computational graph compilation.**

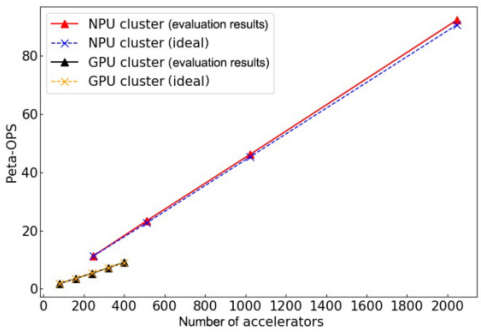



**Fig. 6 Sustained performance (in Peta-OPS) of AIPerf over different numbers of AI accelerators. The benchmark shows near-linear weak scalability on systems from dozens to thousands of GPUs or NPUs.**

## 6 Conclusion

The rise of the convergence of AI and HPC reveals new challenges state-of-the-art in benchmarking and future large-scale clusters for AI purposes. Here, we review the current HPC and AI benchmarks and explain why they do not address all the challenges. We choose AutoML, a highly scalable and representative AI application, as our benchmark workload and implemented the algorithms in a highly parallel manner. We also propose an analytical approach that is independent of DL frameworks and other software implementations to estimate the computation operation rate during training and validation processes. We utilize this rate as the benchmark score to construct the benchmark score to quantitatively measure the machine computing power on AI applications. We evaluate the benchmark on different types and scales of systems with a large dataset and verify the benchmark's stability and scalability. Moreover, the simple metric design allows us to easily compare and rank machines from small clusters to large AI-HPC.

### Appendix: OPS calculation

We compare our analytical approach of computing operations with the TensorFlow profiler (tf.profiler) and NVIDIA profiling tool (nvprof). The tf.profiler can only count operations in the FP. The nvprof can trace GPU activities and use the kernel replay to ensure all requested profile data, including the counts of addition, multiplication, MAC, and special operation. The

profiling process with nvprof is prohibitively expensive, so we need an approach to speed up the process.

Fortunately, we can utilize the iterative nature of DL computation and sample the profiling process based on a small partition of data. This method only gives an approximation, because the operations vary with the hyperparameter configurations. Table A1 shows the operations of ResNet-50 layers on ImageNet with the three approaches. The operations in BP are consistent between our analytical approach and nvprof, and the operations in FP are consistent among all the three approaches.

## References


[1] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] G. M. Yi and V. Loia, High-performance computing systems and applications for AI, *J. Supercomput.*, vol. 75, no. 8, pp. 4248–4251, 2019.

[3] E. A. Huerta, A. Khan, E. Davis, C. Bushell, W. D. Gropp, D. S. Katz, V. Kindratenko, S. Koric, W. T. C. Kramer, B. McGinty, et al., Convergence of artificial intelligence and high performance computing on NSF-supported cyberinfrastructure, *J. Big Data*, vol. 7, no. 1, p. 88, 2020.

[4] G. C. Fox, Perspectives on high-performance computing in a big data world, in *Proc. 28$^{th}$ Int. Symp. High-Performance Parallel and Distributed Computing*, Phoenix, AZ, USA, 2019, pp. 145–145.

[5] D. E. Womble, M. Shankar, W. Joubert, J. T. Johnston, J. C. Wells, and J. A. Nichols, Early experiences on summit: Data analytics and AI applications, *J. Reprod. Dev.*, vol. 63, no. 6, pp. 2:1–2:9, 2019.

[6] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. Patterson, H. L. Tang, G. Y. Wei, P. Bailis, V. Bittorf, et al., MLPerf training benchmark, arXiv preprint arXiv: 1910.01500, 2019.

[7] J. L. Gustafson and Q. O. Snell, HINT: A new way to measure computer performance, in *Proc. 28$^{th}$ Annu. Hawaii Int. Conf. System Sciences*, Wailea, HI, USA, 1995, pp. 392–401.


**Table A1 Comparison of the operation count of each layer of ResNet-50 on ImageNet (per epoch, batch size = 1, input shape = 224 × 224 × 3) with different approaches. The difference of FPs in the training and validation stage is due to the data size.**

| Procedure | Operation count from different approaches | | |
|---|---|---|---|
| | tf.profiler | nvprof | Analytical |
| FP (training) | $9.97\times10^{15}$ | $1.02\times10^{16}$ | $1.00\times10^{16}$ |
| BP (training) | – | $2.10\times10^{16}$ | $1.95\times10^{16}$ |
| BP / FP(training) | – | 2.0603 | 1.9533 |
| Total (training) | – | $3.12\times10^{16}$ | $2.95\times10^{16}$ |
| FP(validation) | $3.89\times10^{14}$ | $3.98\times10^{14}$ | $3.90\times10^{14}$ |
| Total (training+ validation) | – | $3.16\times10^{16}$ | $2.99\times10^{16}$ |


[8] E. Carson and N. J. Higham, Accelerating the solution of linear systems by iterative refinement in three precisions, *SIAM J. Sci. Comput.*, vol. 40, no. 2, pp. A817–A847, 2018.

[9] J. Gustafson, D. Rover, S. Elbert, and M. Carter, The first scalable supercomputer benchmark, *Supercomputing Review*, pp. 56–61, 1990.

[10] Baidu, Deepbench, https://github.com/baidu-research/DeepBench, 2020.

[11] S. Dong and D. Kaeli, DNNMark: A deep neural network benchmark suite for GPUs, in *Proc. 2017 General Purpose GPUs*, Austin, TX, USA, 2017, pp. 63–72.

[12] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, and L. Van Gool, AI benchmark: Running deep neural networks on android smartphones, in *Proc. Computer Vision – ECCV 2018 Workshops*, Munich, Germany, 2018, pp. 288–314.

[13] J. H. Tao, Z. D. Du, Q. Guo, H. Y. Lan, L. Zhang, S. Y. Zhou, L. J. Xu, C. Liu, H. F. Liu, S. Tang, et al., BENCHIP: Benchmarking intelligence processors, *J. Comput. Sci. Technol.*, vol. 33, no. 1, pp. 1–23, 2018.

[14] Y. X. Wang, Q. Wang, S. H. Shi, X. He, Z. H. Tang, K. Y. Zhao, and X. W. Chu, Benchmarking the performance and energy efficiency of AI accelerators for AI training, arXiv preprint arXiv: 1909.06842, 2019.

[15] HPE, Deep learning benchmarking suite, https://github.com/HewlettPackard/dlcookbook-dlbs, 2020.

[16] R. Adolf, S. Rama, B. Reagen, G. Y. Wei, and D. Brooks, Fathom: Reference workloads for modern deep learning methods, in *Proc. 2016 IEEE Int. Symp. Workload Characterization* (*IISWC*), Providence, RI, USA, 2016, pp. 1–10.

[17] AIIA-DNN-benchmark, https://github.com/AIIABenchmark/AIIA-DNN-benchmark, 2021.

[18] W. L. Gao, F. Tang, L. Wang, J. F. Zhan, C. X. Lan, C. J. Luo, Y. Y. Huang, C. Zheng, J. H. Dai, Z. Cao, et al., AIBench: An industry standard internet service AI benchmark suite, arXiv preprint arXiv: 1908.08998, 2019.

[19] W. Zhang, W. Wei, L. J. Xu, L. L. Jin, and C. Li, AI matrix: A deep learning benchmark for Alibaba data centers, arXiv preprint arXiv: 1909.10562, 2019.

[20] T. Ben-Nun, M. Besta, S. Huber, A. N. Ziogas, D. Peter, and T. Hoefler, A modular benchmarking infrastructure for high-performance and reproducible deep learning, in *Proc. 2019 IEEE Int. Parallel and Distributed Processing Symp.* (*IPDPS*), Rio de Janeiro, Brazil, 2019, pp. 66–77.

[21] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, Dawnbench: An end-to-end deep learning benchmark and competition, *Training*, vol. 100, no. 101, p. 102, 2017.

[22] Z. H. Jiang, L. Wang, X. W. Xiong, W. L. Gao, C. J. Luo, F. Tang, C. X. Lan, H. X. Li, and J. F. Zhan, HPC AI500: The methodology, tools, roofline performance models, and metrics for benchmarking HPC AI systems, arXiv preprint arXiv: 2007.00279, 2020.

[23] H. Y. Zhu, M. Akrout, B. J. Zheng, A. Pelegris, A. Phanishayee, B. Schroeder, and G. Pekhimenko, TBD: Benchmarking and analyzing deep neural network training, arXiv preprint arXiv: 1803.06905, 2018.

[24] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated*

*Machine Learning*: *Methods*, *Systems*, *Challenges*. Springer, 2019.

[25] X. He, K. Zhao, and X. Chu, Automl: A survey of the state-of-the-art, *Knowl.-Based Syst.*, vol. 212, p. 106622, 2021.

[26] T. Elsken, J. H. Metzen, and F. Hutter, Neural architecture search: A survey, arXiv preprint arXiv: 1808.05377, 2018.

[27] J. Bergstra and Y. Bengio, Random search for hyper-parameter optimization, *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 281–305, 2012.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[29] Y. I. Bengio, J. Goodfellow, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. F. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems, arXiv preprint arXiv: 1603.04467, 2016.

[31] F. Chollet, Keras: The python deep learning library, *Astrophysics Source Code Library*, doi: 2018ascl.soft06022C.

[32] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. M. Lin, A. Desmaison, L. Antiga, and A. Lerer, Automatic differentiation in pytorch, in *Proc. 31$^{st}$ Conf. Neural Information Processing Systems*, Long Beach, CA, USA, 2017, pp. 1–4.

[33] G. Nguyen, S. Dlugolinsky, M. Bobák, V. Tran, Á. L. García, I. Heredia, P. Malík, and L. Hluchý, Machine learning and deep learning frameworks and libraries for large-scale data mining: A survey, *Artif. Intell. Rev.*, vol. 52, no. 1, pp. 77–124, 2019.

[34] Z. B. Wang, K. Liu, J. Li, Y. Zhu, and Y. N. Zhang, Various frameworks and libraries of machine learning and deep learning: A survey, *Arch. Comput. Methods Eng.*, doi: 10.1007/s11831-018-09312-w.

[35] A. Truong, A. Walters, J. Goodsitt, K. Hines, C. B. Bruss, and R. Farivar, Towards automated machine learning: Evaluation and comparison of AutoML approaches and tools, arXiv preprint arXiv: 1908.05557, 2019.

[36] M. A. Zöller and M. F. Huber, Survey on automated machine learning, arXiv preprint arXiv: 1904.12054, 2019.

[37] K. M. He, X. Y. Zhang, S. Q. Ren, and J. Sun, Deep residual learning for image recognition, in *Proc. 2016 IEEE Conf. Computer Vision and Pattern Recognition*, Las Vegas, NV, USA, 2016, pp. 770–778.

[38] A. G. Howard, M. L. Zhu, B. Chen, D. Kalenichenko, W. J. Wang, T. Weyand, M. Andreetto, and H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv: 1704.04861, 2017.

[39] J. Hu, L. Shen, and G. Sun, Squeeze-and-excitation networks, in *Proc. 2018 IEEE/CVF Conf. Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, 2018, pp. 7132–7141, 2018.

[40] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, Densely connected convolutional networks, in *Proc. 2017 IEEE Conf. Computer Vision and Pattern Recognition*, Honolulu, HI, USA, 2017, pp. 4700–4708.

[41] K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv: 1409.1556, 2014.

[42] C. Szegedy, W. Liu, Y. Q. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, Going deeper with convolutions, in *Proc. 2015 IEEE Conf. Computer Vision and Pattern Recognition*, Boston, MA, USA, 2015, pp. 1–9.

[43] B. Zoph and Q. V. Le, Neural architecture search with reinforcement learning, arXiv preprint arXiv: 1611.01578, 2016.

[44] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, Large-scale evolution of image classifiers, in *Proc.34$^{th}$ Int. Conf. Machine Learning*, Sydney, Australia, 2017, pp. 2902–2911.

[45] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, Towards automatically-tuned neural networks, in *Proc. 2016 Workshop on Automatic Machine Learning*, New York City, NY, USA, 2016, pp. 58–65.

[46] H. X. Liu, K. Simonyan, and Y. M. Yang, DARTS: Differentiable architecture search, arXiv preprint arXiv: 1806.09055, 2018.

[47] T. Q. Chen, I. Goodfellow, and J. Shlens, Net2Net: Accelerating learning via knowledge transfer, arXiv preprint arXiv: 1511.05641, 2015.

[48] T. Wei, C. H. Wang, Y. Rui, and C. W. Chen, Network morphism, in *Proc. 33$^{rd}$ Int. Conf. Machine Learning*, New York City, NY, USA, 2016, pp. 564–572.

[49] H. F. Jin, Q. Q. Song, and X. Hu, Auto-Keras: An efficient neural architecture search system, in *Proc. 25$^{th}$ ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, Anchorage, AK, USA, 2019, pp. 1946–1956.

[50] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in *Proc. 32$^{nd}$ Int. Conf. Machine Learning*, Lille, France, 2015, pp. 448–456.

[51] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, An empirical evaluation of deep architectures on problems with many factors of variation, in *Proc. 24$^{th}$ Int. Conf. Machine Learning*, Corvalis, OR, USA, 2007, pp. 473–480.

[52] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, Algorithms for hyper-parameter optimization, in *Proc. 25$^{th}$ Annu. Conf. Neural Information Processing Systems*, Granada, Spain, 2011, pp. 2546–2554.

[53] N. Qian, On the momentum term in gradient descent learning algorithms, *Neural Netw.*, vol. 12, no. 1, pp. 145–151, 1999.

[54] D. Peteiro-Barral and B. Guijarro-Berdiñas, A survey of methods for distributed machine learning, *Prog. Artif. Intell.*, vol. 2, no. 1, pp. 1–11, 2013.

[55] R. Hecht-Nielsen, Theory of the backpropagation neural network, in *Neural Networks for Perception*: *Computation*, *Learning*, *and Architectures*, H. Wechsler, ed. Amsterdam, the Netherland: Elsevier, 1992, pp. 65–93.

[56] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, Automatic differentiation in machine learning: A

survey, *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 5595–5637, 2017.

[57] J. E. Huss and J. A. Pennline, *A Comparison of Five Benchmarks*. Cleveland, OH, USA: National Aeronautics and Space Administration, 1987.

[58] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and F. F. Li, ImageNet: A large-scale hierarchical image database, in *Proc. 2009 IEEE Conf. Computer Vision and Pattern Recognition*, Miami, FL, USA, 2009, pp. 248–255.

[59] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, et al., The NAS parallel benchmarks, *Int. J. High Perform. Comput. Appl.*, vol. 5, no. 3, pp. 63–73, 1991.

[60] K. M. He, X. Y. Zhang, S. Q. Ren, and J. Sun, Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification, in *Proc. 2015 IEEE Int. Conf. Computer Vision*, Santiago, Chile, 2015, pp. 1026–1034.

**Zhixiang Ren** received the PhD in high-energy astrophysics from the University of New Mexico in 2018, and then continued his research as a post-doctoral fellow in 2019. He is currently a research fellow at Peng Cheng National Laboratory. His main research interests include high-performance artificial intelligence and scientific computing. He has published more than 30 papers with extensive experience in the applications of high-performance scientific computing, machine learning, and deep learning in science.

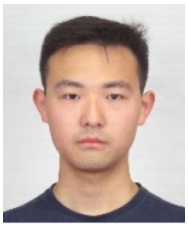
**Jidong Zhai** received the BEng degree in computer science from University of Electronic Science and Technology of China in 2003, and the PhD degree in computer science from Tsinghua University in 2010. He is currently an associate professor at the Department of Computer Science and Technology, Tsinghua University. His research interests include performance evaluation for high performance computers, performance analysis, and modeling of parallel applications.

**Youhui Zhang** received the BEng and PhD degrees in computer science from Tsinghua University, Beijing, China, in 1998 and 2002, respectively. He is currently a professor at the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include computer architecture and neuromorphic computing. He is a member of CCF, ACM, and IEEE.

**Yunquan Zhang** received the PhD degree in computer science from Chinese Academy of Sciences, China in 2000. He is currently a full professor of computer science at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include high-performance computing, with particular emphasis on large scale parallel computation and programming models, and high-performance parallel numerical algorithms.

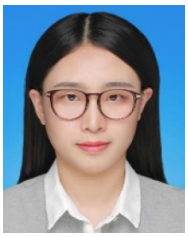
**Yongheng Liu** received the MEng degree in communication engineering from Xidian University, China in 2008. He is currently a senior engineer at Peng Cheng National Laboratory. His main research interests include cloud OS, high-performance computing, and performance evaluation.

**Tianhui Shi** received the BEng degree in computer science and technology from Tsinghua University, Beijing, China in 2018. He is currently pursuing the PhD degree in computer science and technology at Tsinghua University. His current research interests include graph mining systems and high-performance computing.



**Lei Xie** received the BEng degree in computer science and technology from Tsinghua University, Beijing, China in 2018. He is currently pursuing the PhD degree in computer science and technology at Tsinghua University. His current research interests include optimization for machine learning using high-performance computing techniques and mitigation of errors in quantum computers.

**Wenguang Chen** received the BEng and PhD degrees in computer science from Tsinghua University, in 1995 and 2000, respectively. He was the CTO at Opportunity International Inc. from 2000 to 2002. In January 2003, he joined Tsinghua University. He is currently a professor and an associate head at the Department of Computer Science and Technology, Tsinghua University. His research interests include parallel and distributed computing and programming model.



**Yue Zhou** received the BEng and MEng degrees in electronics engineering from Beijing Institute of Technology, China in 2016 and 2019, respectively. She is currently an engineer at Peng Cheng National Laboratory, China. Her research interests are image processing, machine learning, and high-performance computers.