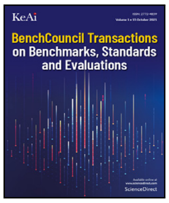
Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: www.keaipublishing.com/en/journals/benchcouncil-transactions-on-benchmarks-standards-and-evaluations/



Research Article

HPC AI500 V3.0: A scalable HPC AI benchmarking framework

Zihan Jiang [a,b,*], Chunjie Luo [a], Wanling Gao [a], Lei Wang [a], Jianfeng Zhan [a,b]

[a] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[b] University of Chinese Academy of Sciences, Beijing, China

ARTICLE INFO

Keywords:
Artificial intelligence
High performance computing
Benchmarking
Scalability

ABSTRACT

In recent years, the convergence of High Performance Computing (HPC) and artificial intelligence (AI) makes the community desperately need a benchmark to guide the design of next-generation scalable HPC AI systems. The success of the HPL benchmarks and the affiliated TOP500 ranking indicates that scalability is the fundamental requirement to evaluate HPC systems. However, being scalable in terms of these emerging AI workloads like deep learning (DL) raises nontrivial challenges. This paper formally and systematically analyzes the factor that limits scalability in DL workloads and presents HPC AI500 v3.0, a scalable HPC AI benchmarking framework. The HPC AI500 V3.0 methodology is inspired by bagging, which utilizes the collective wisdom of an ensemble of base models and enables the benchmarks to be adaptively scalable to different scales of HPC systems. We implement HPC AI500 V3.0 in a highly customizable manner, maintaining the space of various optimization from both system and algorithm levels. By reusing the representative workloads in HPC AI500 V2.0, we evaluate HPC AI500 V3.0 on typical HPC systems, and the results show it has near-linear scalability. Furthermore, based on the customizable design, we present a case study to perform a trade-off between AI model quality and its training speed. The source code of HPC AI500 V3.0 is publicly available from the HPC AI500 project homepage https://www.benchcouncil.org/aibench/hpcai500/.

1. Introduction

*D*eep *L*earning (DL) has been a dominating technology in *A*rtificial *I*ntelligence (AI) as its huge success in many challenging AI problems, such as image classification [1–3], object detection [4–6], and natural language processing [7–9]. DL allows building a computational model composed of multiple processing layers with trainable weights to learn the presentation of data [10]. To harness larger datasets and achieve higher model quality (e.g., Top1 accuracy), in recent years, tremendous DL models have been proposed endlessly, both for commercial applications [11–16] and scientific computing [17–20]. These giant models usually have deeper layers and billions of weights, which is extremely computation-intensive. Hence, academia and industry are greatly interested in designing and building next-generation HPC systems to run these emerging AI workloads for their computation requirement [21, 22]. Benchmark plays an important role in this process, as it provides the input and methodology for evaluation [23].

In the past three decades, the HPL benchmark [24] and the affiliated TOP500 ranking [25] witnessed the thriving of HPC systems. From CM-5 (1993) [26] to Fugaku (2020) [27], the FLOPS performance of the NO.1 supercomputer on the TOP500 list improves by more than $10^6\times$. HPL has become the measurement standard [28] in the HPC field for thirty years and will continue to be. The reason for its success is twofold. On the one hand, HPL solves a (random) dense linear system in double precision, which captures the general characteristic that many scientific applications share. We conclude this property as *relevancy*. On the other hand, HPL can adapt to scalable systems by adjusting the input matrix size. We summarize this property as *scalability*. The HPL lesson indicates that *relevancy* and *scalability* are two significant properties for an ideal benchmark. Most of the previous work [29–34] in AI benchmarking focus on relevancy and select represent workloads in real-world AI applications. However, they ignored the scalability issue.

Scalability is difficult to guarantee for AI workloads. According to the experiences in the previous researches [36,46], each AI workload has the best training batchsize, which is irrelevant to the system scale, to achieve state-of-the-art quality. This observation indicates that no matter how the scale of the system changes, the amount of parallel computation processed remains the same. Although many system optimizations [13,47–52] are proposed, all they can do is process this constant amount of computation as fast as possible by utilizing various parallel techniques (e.g., data parallelism [53]). Therefore, with the continuous growth of system scale, the speed of training existing AI workloads is rapidly accelerated. As shown in Fig. 1, from 2017 to 2021, with the development of HPC AI systems, the training time of

* Corresponding author at: Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.
E-mail addresses: jiangzihan@ict.ac.cn (Z. Jiang), luochunjie@ict.ac.cn (C. Luo), gaowanling@ict.ac.cn (W. Gao), wanglei_2011@ict.ac.cn (L. Wang), zhanjianfeng@ict.ac.cn (J. Zhan).

https://doi.org/10.1016/j.tbench.2022.100083
Received 17 November 2022; Received in revised form 23 December 2022; Accepted 23 December 2022
Available online 29 December 2022

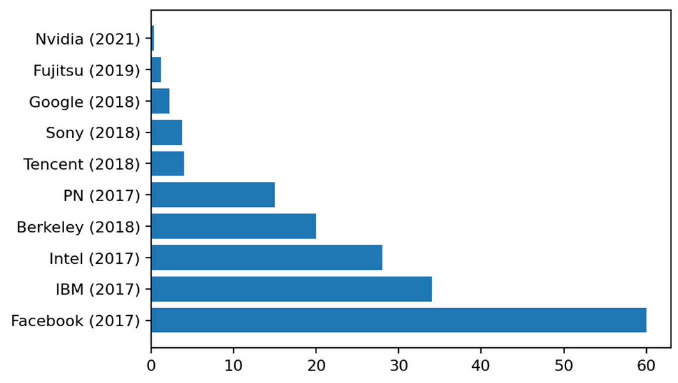


Fig. 1. ImageNet/ResNet-50 is a popular showcase for optimizing HPC AI systems from academia [35] and industry [36–44]. PN refers to Preferred Networks [45]. The x-axis refers to the training time measured in minutes.

ResNet-50 [2] has dropped exponentially, and the result of Nvidia [44] shows that it now can be done in under half a minute. From the benchmarking perspective, such a short running time does not allow for a thorough and endurable evaluation. Furthermore, the fixed amount of computation is distributed on the HPC system with a growing scale, which makes the resource utilization of each computing node extremely unsaturated.

Two prior works attempt to address the scalability problem in HPC AI benchmarking, namely AIPerf [54] and HPL-AI [55]. However, they both have their own flaws. AIPerf uses network architecture search (NAS) [56] as the primary workload. NAS automatically searches the network architecture with a predefined probability, introducing randomness to the benchmarking process. HPL-AI allows mixed-precision LU decomposition to solve a linear equation system and tends to be irrelevant to most AI workloads [57].

Bagging (Bootstrap Aggregation) [58] is designed to improve the stability and quality of the prediction by utilizing the collective wisdom of an ensemble of base models. As a meta-algorithm of ensemble learning [59], a critical feature of bagging is the independence between each base model. This independence makes bagging can be implemented as a highly parallel way to scale out with the number of nodes in an HPC system. Another merit of bagging is its flexibility and not being bound to any AI algorithm. In other words, we can easily and quickly achieve relevancy by integrating a state-of-the-art or state-of-the-practice algorithm into our bagging-based benchmarking framework. Considering the advantages above, this paper presents a bagging-based scalable AI benchmarking framework, which we call HPC AI500. HPC AI500 V3.0 extends our previous works: HPC AI500 V1.0 [50] and HPC AI500 V2.0 [57]. Table 1 summarizes the differences between HPC AI V3.0 from the other related works. HPC AI500 V3.0 not only leverages the advantages of bagging to achieve scalability and relevancy but also maintains user-customizable parallel optimization opportunities. HPC AI500 V3.0 implements two modules, bagging management (BM) and model parallelism management (MPM), to achieve this customizability. BM determines the algorithm adopted in data sampling and the number of base models. MPM determines the degree of parallelism inside each base model. Through these two modules, users can customize the number of base models and the degree of parallelism to make the trade-off between the model quality and training speed. Based on HPC AI500 [57], we evaluate HPC AI500 V3.0 on typical HPC systems to show its scalability and customizability.

Our main contributions are summarized as follows:

- According to the unique challenges of HPC AI Benchmarking, we reformulated the HPC AI scalability issue (Section 2).
- We propose the bagging approach in HPC AI benchmarking to achieve relevancy and scalability and implement HPC AI500 V3.0, a scalable and customizable framework for HPC AI benchmarking (Section 3).
- We evaluate HPC AI500 V3.0 by reusing HPC AI500 v2.0 workloads on typical HPC systems to show its scalability and customizability (Section 4).

2. Background and challenge

2.1. Deep learning preliminary

The whole training process of modern DL models is essentially a non-convex optimization. Mathematically, it can be represented as:

$$\min_{\forall x \in \mathbb{R}^n} f(x) := \frac{1}{N} \sum_{i=1}^{N} f_i(x), \tag{1}$$

where f_i is a loss function for data point $i \in \{1, 2, 3, \dots, N\}$, which measures the deviation of the model prediction from the data. x is the vector of weights being optimized. The process of optimizing the loss function is called training and is performed iteratively.

2.1.1. Mini-batch stochastic gradient descent

Stochastic Gradient Descent (SGD) is the dominant method for training DL models. Vanilla SGD updates weight x by adding the gradient computed on a single data point of the whole dataset. Since only one random data point is processed at one iteration, this approach has two disadvantages. First, such a noisy update makes the training process unstable [62]. Second, the computation is inefficient, especially when using computing devices such as GPUs. Mini-batch SGD is proposed to remedy these two deficiencies. It minimizes the loss function f iteratively in the following form:

$$x_{k+1} = x_k - \eta_k \left(\frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i(x_k) \right) \tag{2}$$

where $B_k \in \{1, 2, 3, \dots, N\}$ is the batch sampled from the whole dataset and η_k is the learning rate of iteration k. $|B_k|$ refers to the batchsize. The ratio of N and $|B_k|$ determines the number of iterations in a training epoch.

2.2. The scalability issue

With the convergence of AI and HPC, both academia and industry players [63–65] leverage the computing power of HPC systems to speed up the training process of DL models. However, SGD training has a significant drawback, limited by the batchsize.

2.2.1. The limitation of batchsize

Although there are millions of data in a DL dataset with the size N [66], the intrinsic sequential property of SGD only allows a batch with size B_k (e.g., $B_k = 256$) of data to be processed in parallel in an iteration. We call the computation cost required by a batch as the *Amount of Parallel Computation in an Iteration(in short, APC)*. Compared to Linpack, whose APC can be tuned by the size of the input matrix, the APC of DL workloads is usually a constant and can be represented as:

$$APC_{dl} = \sum_{j=1}^{|B_k|} Computation(f_j(x)) \tag{3}$$

where $j \in \{1, 2, 3, \dots, |B_k|\}$ is data that is randomly sampled from the DL dataset with the size N and included in batch B_k. And $Computation(f_j(x))$ is the computation cost required by the DL model to process a single data and can be measured by FLOPs.

Eq. (3) indicates that APC_{dl} is determined by the $|B_k|$. However, the value of B_k is usually a small number, where $|B_k| \ll N$. Specifically, $B_k \in \{16, 32, 64, 256...512\}$ in many DL applications such as image classification [2] and object detection [4,5]. In this context, it is hard to fully utilize the computing power of HPC systems, which are usually equipped with hundreds or even thousands of nodes. Taking

Table 1
Comparison of HPC AI500 V3.0 against HPC AI500 V1.0, V2.0, and other HPC AI benchmarks. The equivalence, affordability, representativeness, and repeatability issues are resolved in our previous work HPC AI500 V2.0 [57]. HPC AI500 V3.0 is an HPC AI benchmarking framework which inherits and extends HPC AI500 V2.0 with scalability. HPC AI500 V3.0 can naturally integrate other HPC AI benchmarks. "✗" and "✓" indicate whether they have the corresponding properties. "-" indicates not verified.

Related work	Equivalence	Representativeness	Affordability	Repeatability	Scalability
HPC AI500 V1.0 (2018) [50]	✗	✓	✓	✗	✗
HPL-AI (2019) [55]	✓	✗	✓	✓	✓
Deep500 (2019) [60]	✗	✗	✓	–	✗
HPC AI500 V2.0 (2020) [57]	✓	✓	✓	✓	✗
AIPerf (2020) [54]	✓	✓	✓	–	✓
MLPerf (HPC) (2021) [61]	✓	✓	✓	✓	✗
HPC AI500 V3.0	✓	✓	✓	✓	✓

the ImageNet/ResNet-50 training on Summit [50,57] as an example. $|B_k| = 512$, according to Eq. (3), the APC of ImageNet/ResNet-50 is 11 776 GFLOPs. Considering Summit has 4608 nodes (six Nvidia Tesla GPUs in each node), each node only can allocate the computation of $\frac{11776}{4608} = 2.55$ GFLOPs, which is far away from the peak performance of six V100 GPUs.[1]

Naively enlarging $|B_k|$ to improve APC_{dl} leads to a degradation in the model quality due to the sharp minima [36,46,67]. The tricks proposed in [36,46,67] indeed increase $|B_k|$ to a larger number, but it is still far from the peak performance of the HPC system, leading to poor resource utilization. Furthermore, the proposed tricks are empirical, lack generalization ability, and depend on a specific DL workload. So far, no research can systematically and theoretically quantify the relationship between B_k and model quality.

2.2.2. *The reformulation of HPC AI scalability*

Based on the aforementioned analysis, we reformulate the HPC AI scalability from the following two perspectives. In the previous work [57], we have discussed how to resolve equivalence, representativeness, affordability, and repeatability issues.

- The APC_{dl} should be large enough to accommodate the scale and computing capability of HPC systems. To be specific, it is necessary to maintain a high resource utilization and near-linear speed up.
- The model quality should be maintained or improved while increasing the APC_{dl} and $|B_k|$. Otherwise, the whole training process is meaningless.

Compared to the traditional HPC scalability, which focuses on scale efficiency and resource utilization [24], the reformulated HPC AI scalability emphasizes the restraint of model quality and batchsize $|B_k|$.

2.3. *Prior work*

In addition to the other AI benchmarks [29–34], MLPerf (HPC) [61], HPL-AI [55], AIPerf [54], and HPC AI500 [50,57] are representative HPC AI benchmarking works. Among them, the earliest work is the HPC AI500 V1.0 [50], dating back to 2018. HPC AI500 V1.0 [50] and V2.0 [57] and MLPerf(HPC) fail to tackle the scalability issue and focus on selecting typical HPC AI applications and parallel-based optimizations. HPL-AI and AIPerf manage to achieve scalability but bring other problems. HPL-AI evaluates HPC systems by performing mixed-precision LU decomposition at the kernel level. Same to HPL, it can increase the APC by adjusting the size of the input matrix. However, LU decomposition is irrelevant to most AI workloads [57]. The AIPerf methodology is inspired by AutoML, whose core process is performed by NAS. Although AutoML can scale automatically with the number of nodes, the high randomness of NAS (Fig. 2) calls into question whether AutoML is desirable as an HPC AI benchmark. Table 1 summarizes the related work chronologically and compares our work with other related work in five dimensions.

[1] The peak performance of six V100 GPUs in terms of FLOPS is: $6 \times 15.7 \times 10^3$ GFLOPS $= 94.2 \times 10^3$ GFLOPS.

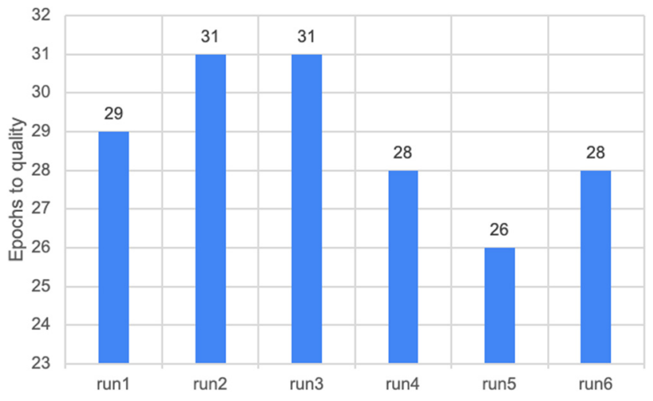


Fig. 2. The randomness of NAS. In different runs, the amount of computation required to train NAS to the target model quality varies, which leads to unfair and unrepeatable evaluation.

3. HPC AI500 V3.0

This section first presents the HPC AI500 v3.0 methodology. Then we detail the design, workflow, and customizable configuration. Finally, we introduce the measurement method and the proposed metrics.

3.1. *Methodology*

3.1.1. *Ensemble learning and bagging*

The ensemble learning idea is to solve a common problem by combining the predictions of a group of base models. Rather than making decisions depending on a single model, a group of models makes it possible for ensemble learning to reduce the variance of predictions [59], so-called the wisdom of crowds [68]. Bagging (Bootstrap AGGregatING) is a fundamental paradigm of Ensemble learning. As its name suggests, bagging consists of two parts: bootstrapping and aggregating. Bootstrapping is essentially a data sampling process with replacement from the original dataset. The data generated through this process is called the bootstrapped dataset. The training process of bagging is highly parallel as each base model in the ensemble is trained based on its corresponding bootstrapped dataset rather than the original dataset. After finishing the training, the final decision is aggregated by averaging all the predictions of the base models.

3.1.2. *Applying bagging in HPC AI benchmarking*

For HPC AI benchmarking, to tackle the scalability problem, the first thing is to enlarge the APC to keep up with the increasingly larger scale of HPC systems. Inspired by the Bagging, we introduce *the base model ensemble* on the basis of the training of a single model in the previous AI benchmark like HPC AI500 V2.0. We rewrite Eq. (3) in the following bagging form:

$$APC_{dl} = \sum_{m=1}^{M} \sum_{j=1}^{|B_k|} Computation(f_{m,j}(x)) \quad (4)$$

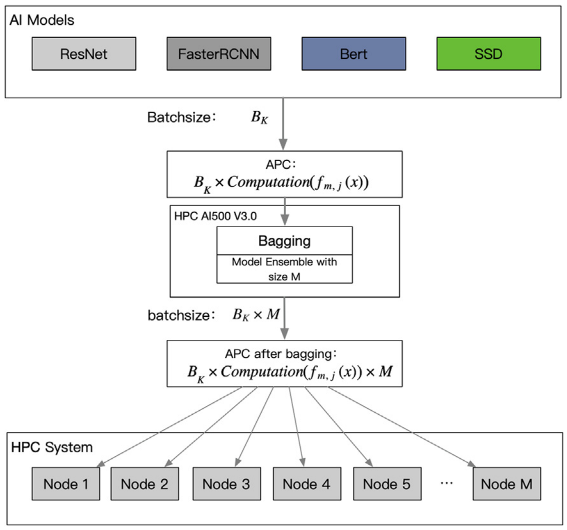

AI Models
ResNet
FasterRCNN
Bert
SSD
Batchsize: B_K
APC: $B_K \times Computation(f_{m,j}(x))$
HPC AI500 V3.0
Bagging
Model Ensemble with size M
batchsize: $B_K \times M$
APC after bagging: $B_K \times Computation(f_{m,j}(x)) \times M$
HPC System
Node 1
Node 2
Node 3
Node 4
Node 5
···
Node M

Fig. 3. The system overview of HPC AI500 V3.0. APC refers to the amount of parallel computing in an iteration.

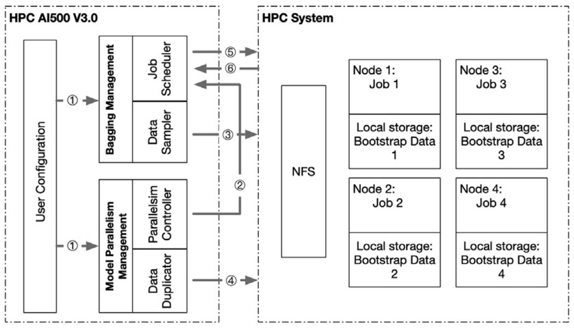


Fig. 4. System design and workflow of HPC AI500 V3.0. NFS refers to the Network File System of HPC systems that each node shares.

where M is the number of the base models in the ensemble, f_m is the m_{th} base model. Note that each base model is the instance of the original model, so the computation cost of each base model is equivalent to that in Eq. (3). Compared to AutoML, the re-sampled bootstrapped dataset makes every base model dissimilar, but the computational logic of each model is consistent, guaranteeing no randomness shown in Fig. 2. All the base model in the ensemble is trained independently, enlarging the $|B_k|$ by M times, and so does APC_{dl}. Considering each base model may train in a distributed manner across several nodes, the ensemble size M and the parallelism degree inside a base model P_degree should satisfy Eq. (5), where Sys_{scale} refers to how many nodes are contained in an HPC system.

$$Sys\_scale = M \times P\_degree \tag{5}$$

3.2. System overview

Based on the Bagging approach, we present HPC AI500 V3.0 and the system overview shown in Fig. 3. HPC AI500 V3.0 does not focus on workload selection and construction as previous AI benchmarks [29,31,34]. Instead, it is a framework that is compatible with these efforts. We briefly introduce the positioning and role of HPC AI500 V3.0 through Fig. 3. This figure shows that HPC AI500 V3.0 scales out the upper-layer AI workloads on lower-layer HPC systems by adaptively increasing APC_{AI}. Specifically, the batchsize of each AI workload is initially only a fixed B_k. After Bagging, a set of M base models are generated, which increases APC_{dl} by concurrently running M base models. This way, thereby, achieves higher resource utilization. In addition, the size of the base model set, M, can be adjusted according to the system size, corresponding to the same adjustable input matrix size in HPL, to adapt to the future growth of the HPC system scale.

3.3. System design and workflow

HPC AI500 V3.0 consists of three components, namely, User Configuration (UC), Bagging Management (BM), and Model Parallelism Management (MPM). BM focuses on managing Bagging, including Job Controller and Data Sampler. Job Controller schedules M jobs to the corresponding nodes, then launch training, and finally aggregates the predictions. Note that each job corresponds with a base model training. Data Sampler controls the data sampling algorithm. MPM is divided into Parallelism Controller and Data Duplicator. Parallelism Controller sets the parallel mode and P_degree. Data Duplicator is responsible for copying and migrating data according to parallelism-related configuration. As shown in Fig. 4, we summarize the workflow of HPC AI500 V3.0 as follows:

1. UC sends the configurations to BM and MPM. BM receives the configurations, including job number, equal to ensemble size M, and saves the DL model and original dataset that needs to be trained. MPM receives the configurations, such as parallelism mode, P_{degree}, and Sys_{scale}.
2. Parallelism Controller in MPM checks if M, P_{degree}, and Sys_{scale} satisfy Eq. (5) and generates the mapping of the jobs to the nodes according to the received messages (e.g., $Task1 \rightarrow Node1$), then sends this mapping to Job Scheduler in BM.
3. Data Sampler in BM determines the sampling algorithm and generates the bootstrap data for each task. All the generated data is sent to the NFS of the HPC system.
4. Data Duplicator in MPM duplicates the bootstrap data according to the mapping that Parallelism Controller generates. For example, $Job1->Node1$ means the bootstrap data in Job1 only need

Table 2
The Customizable Configuration of HPC AI500 V3.0. $Node\_acc$ refers to the number of accelerators equipped in a node of the HPC system.

Type	Default setting	Alternatives
Basic	$P_{degree} = Node\_acc$ $M = \frac{Sys_{scale}}{P_{degree}}$	Any M and P_{degree} that satisfy Eq. (5)
Learning Rate Scheduler	warm-up schema and linear scaling [69]	LARS [35], LAMB [70]
Optimizer	SGD with momentum	Adam [71], AdaGrad [72]
Data Precision for Training	FP16	mixed-precision, Int8
Data Precision for Communication	FP32	FP16, Int8
Parallel Mode	data parallelism	model parallelism, pipeline parallelism [73], mixed parallelism
Communication Mode	synchronous all-reduce	2D-Torus [41], Hierarchical all-reduce [40]
Framework	TensorFlow [74]	PyTorch [75], Mindspore [76]

to be duplicated once. All the duplicated data is sent to the local storage of the corresponding node.

5. Job Scheduler sends the job to the corresponding nodes and launches the training of the whole ensemble.
6. After the training is finished, Job Scheduler collects all the ensemble output and then makes the final prediction.

3.4. Customizable configuration

In order to maintain the optimization space, in addition to the basic configuration, such as M and $|P_{degree}|$, we summarize other customizable configurations in Table 2. We provide a default setting and some alternatives in each configuration type. Note that alternatives just list the favored option, and the user can customize the efficient implementation according to their situation.

3.5. Metrics

Same as HPL, we use *FLOPS* (Floating point operations per second) as our primary metric:

$$FLOPS = \frac{\sum_{i=1}^{N/|B_k|} APC_{dl}}{T_{epoch}} \tag{6}$$

where T_{epoch} refers to the training time of one epoch and $N/|B_k|$ refers to the number of iterations in one training epoch. In addition to FLOPS, we also adopt a metric that considers both system throughput and model quality, namely Valid FLOPS (VFLOPS) [57]. The definition of *VFLOPS* is shown as follows:

$$VFLOPS = FLOPS * penalty\_coefficient \tag{7}$$

$$penalty\_coefficient = (achieved\_quality/target\_quality)^n \tag{8}$$

where $penalty\_coefficient$ is used to penalize or award the FLOPS based on the achieved quality. $achieved\_quality$ refers to the actual model quality achieved in the evaluation. $target\_quality$ is predefined in the Table 4. The value of n defines the sensitivity to the model quality. According to the setting of HPC AI500 V2.0 [57], we set n as 10 for Extreme Weather Analytics and 5 for Image Classification.

Table 3
The FLOPs calculation rules for primary operators in a DL model. K refers to the kernel size, C_{in} and C_{out} refers to the input and output channel, H and W refers to the data size, $Group_{size}$ refers to the group size of the convolution, and FL refers to the flatten layer used in the Fully-connected.

Operators	FLOPs
Convolution	$2 \times K^2 \times C_{in} \times H \times W \times C_{out}$
Depth-wise Convolution	$2 \times K^2 \times C_{in} \times H \times W$
Group Convolution	$\frac{2\times K^2\times C_{in}\times H\times W\times C_{out}}{Group_{size}}$
Fully-connected	$FL_{in} \times FL_{out}$
Element-wise	$C_{out} \times H \times W$
Pooling	$C_{in} \times H \times W$
Normalization	$C_{in} \times H \times W$

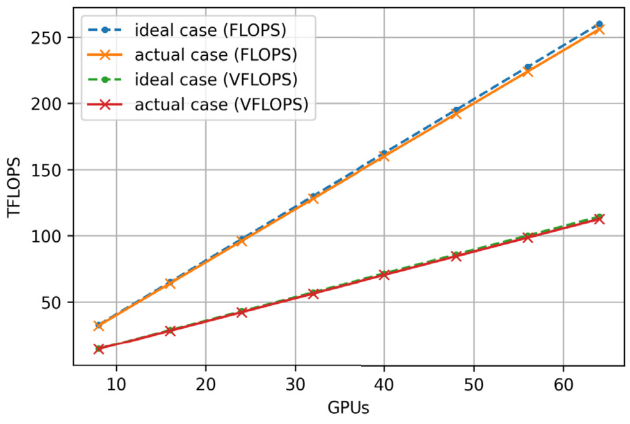
3.6. Measurement

According to Eq. (4) and Eq. (6), to determine the $FLOPS$, we need to first measure the $Computation(f(x))$. Although profiling tools such as Nsight [77] are able to count the FLOPs by kernel replay, it is dependent on the Nvidia hardware. In order to reduce the influence of the hardware and the hardware-specific optimizations performed by bundled low-level libraries (e.g., CuDnn for Nvidia GPUs), we present an analytical method to calculate the FLOPs that a DL model requires.

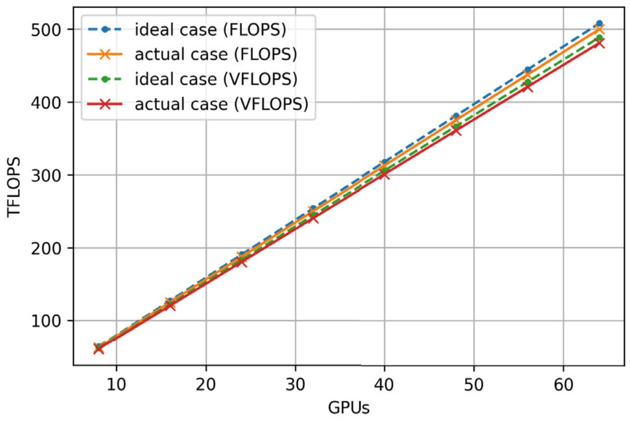
Modern AI frameworks, such as TensorFlow, describe the computation of a DL model using a directed acyclic graph (DAG) that consists of multiple nodes and edges. The Node in the DAG represents a kind of operator, and the edge represents the data flow. Each operator defines a computation logic and receives the data from the input edge, and then sends the intermediate result to the next operator after finishing its computation. Unlike HPL, which has only one kind of operator (LU decomposition), a DL model usually consists of multiple operators with different kinds. Hence, we summarize the most frequent operators in DL as shown in Table 3. In addition to these listed operators, we ignore other low-proportion operators contained in the DL model. Based on this table, we can calculate the $Computation(f(x))$ by traversing the DAG.

3.7. Implementation details

Job scheduler of the Bagging management module is based on SLURM (Simple Linux Utility for Resource Management) [78]. SLRUM is the most commonly used scheduling system in HPC AI systems, fault-tolerant and highly scalable, and suitable for Linux clusters of different sizes. We implement the submitted job script based on the $sbatch$ interface of SLRUM and use $sinfo$ and $smap$ to monitor the training progress of the base model in each job, and the basic unit of job scheduling is a container implemented by Docker [79]. According to the literature [58], the implemented random sampling algorithm guarantees that the i_{th} training sample is selected n ($n \in \{0, 1, 2...\}$) times. The probability of the times approximates the Poisson distribution of $\lambda = 1$, so the probability of at least one occurrence of the i_{th} sample is $1 - (\frac{1}{e}) = 0.632$. So for any Bagging base classifier, about 36.8% of the samples of the original dataset will not be used at the time of training. The default parallel implementation in the parallel management module uses data parallelism implemented by Horovod and OpenMPI, which is also the most common parallel method in HPC AI systems [17–19]. The measurement of bandwidth is divided into intra-node communication and inter-node communication, and we use Nvidia-smi (NVIDIA System Management Interface) tool [80] to monitor communication within nodes and use iftop tool [81] to monitor communication between nodes.



(a) Extreme Weather Analytics.



(b) Image Classification.

Fig. 5. The scalability experiments of HPC AI500 V3.0 in terms of FLOPS and VFLOPS. The $penalty\_coefficient$ is 0.44 for Extreme Weather Analytics and 0.96 for Image Classification.

4. Evaluation

4.1. Experimental setup

4.1.1. Hardware

Our experiments are conducted on a 64GPUs-cluster, consisting of eight nodes, each of which is equipped with one Intel(R) Xeon(R) Platinum 8268 CPU and eight NVIDIA Tesla V100 GPUs. Each GPU in the same node has 32 GB HBM memory, connected by NVIDIA NVLink—a high-speed GPU interconnection whose theoretical peak bi-directional bandwidth is 300 GB/s. The nodes are connected with Ethernet networking with a bandwidth of 10 Gb/s. Each node has 1.5 TB system memory and an 8 TB NVMe SSD disk.

4.1.2. Software

We use TensorFlow v1.14, compiled with CUDA v10.1 and cuDnn v7.6.2 backend. We use Horovod v0.16.4 for synchronous distributed training, compiled with OpenMPI v3.1.4 and NCCL v2.4.8. NCCL is short for the NVIDIA Collective Communications Library, which is a closed-source library of multi-GPU collective communication primitives that are topology-aware.

4.2. Workloads

HPC AI500 V3.0 is a benchmarking framework, which means any AI benchmark can be integrated into this framework in a bagging manner. Here, our default implementation is based on HPC AI500 V2.0 [57], a well-received HPC AI benchmark that mainly consists of two workloads, covering AI applications in business and scientific computing. As shown in Table 4, Image Classification uses ResNet-50 [2]and ImageNet [66] for training, which is a well-known showcase for optimizing HPC AI systems. Extreme Weather Analytics [82] is a representative scientific application, it uses Faster-RCNN for detecting the extreme weather in the climate image. Each climate image in Extreme Weather Dataset consists of 16 channels and contains four extreme weather patterns.

4.3. The scalability experiments

The scalability experiments are conducted with the default setting of HPC AI500 V3.0, as shown in Table 2. We set the $P_degree = 8$, which is equal to the number of GPUs in a node. In each node, a job is distributed to 8 GPUs by using data parallelism. We perform the experiment sequentially on different system scales, typically the $Sys_scale = 8, 16, 24, 32, 40, 48, 56, 64$ GPUs. According to Eq. (5), the corresponding job number is $M = 1, 2, 3, 4, 5, 6, 7, 8$. The results of scalability experiments are shown in Fig. 5. As we can see, HPC AI500 V3.0 shows near-linear scalability in both FLOPS and VFLOPS. Note that, in Fig. 5(a), the $penalty\_coefficient = 0.44$ leads to a gap between the VFLOPS line and FLOPS of Extreme Weather Analytics. Furthermore, we measure the GPU utilization by Nsight at the scale of 64 GPUs and the result is shown in Fig. 6. Both Extreme Weather Analytics and Image Classification achieve high GPU utilization.

4.4. The customizability experiments

4.4.1. Trade-off between the model quality and training speed

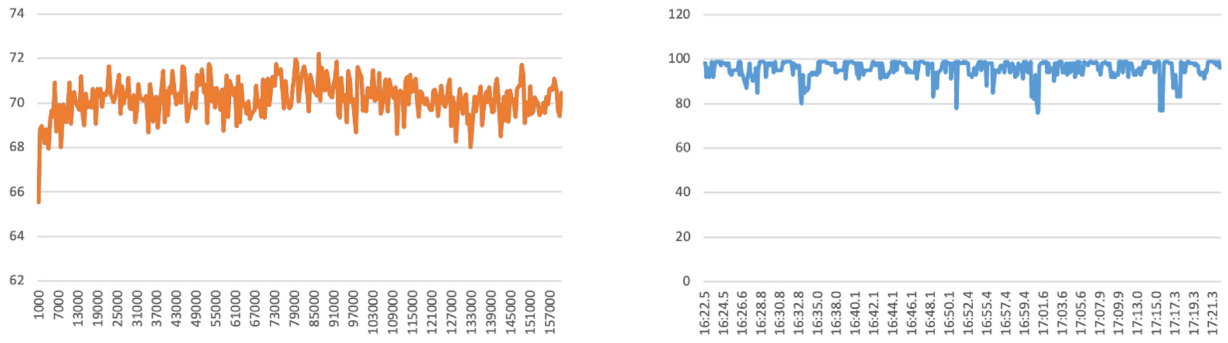
To exhibit this trade-off, we take Image Classification as the showcase. We set the $M = 8, 4, 1$ while the corresponding $P_{degree} = 8, 16, 64$. As shown in Fig. 7, the training speed increases along with a decrease in M. When $M = 1$, the process becomes training a single model through the whole cluster, achieving the highest training speed. However, since only one model makes decisions in the ensemble, the model quality suffers about a 3% drop compared to the case of $M = 8$. In practical scenarios, users can choose appropriate M and P_{degree} according to their training speed and model quality requirements.

4.4.2. Optimizations

To show the customizability of HPC AI500, we implement two frequently-used optimizations, mixed-precision training, and communication compression. The former utilizes Tensor Cores in Nvidia Volta architecture to accelerate the model's fully-connected and convolution layer, allowing a fused-multiply-add computation. When performing mixed precision training with a Tensor Core, we use FP16 for calculation and FP32 for accumulation. The latter is the communication compress-on that compresses the tensor precision for synchronizing from 32FP to 16FP to reduce communication overhead. We configure the optimization experiments in the same way as Section 4.3, and the results are shown in Fig. 8. We compared the optimized version to the original version to observe the corresponding effect. Since mixed-precision Extreme Weather Analysis leads to a significant loss of the model quality, here we only report the performance of the model compression. As we can see, mixed-precision training brings about 2x speed up for Image Classification. As for communication compression, it brings about 1.2x for Extreme Weather Analytics but barely has any speed up on Image Classification. The size of the communication tensor in Extreme Weather Analytics is 1.6x larger than that of Image Classification, allowing Extreme Weather Analytics to get a notable benefit.

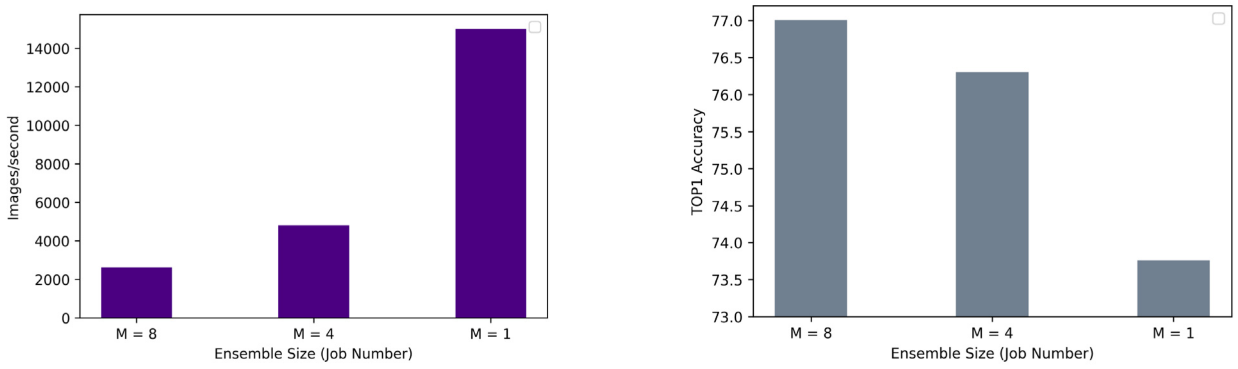
Table 4
The Specification of HPC AI500 V2.0 workloads [57]. HPC AI500 V3.0 can integrate any HPC AI benchmarks. In our evaluation, we reuse the HPC AI500 V2.0 workloads for testing.

Problem domains	Models	Datasets	Target quality
Image Classification	ResNet-50	ImageNet	TOP1 Accuracy = 0.763
Extreme Weather Analytics	Faster-RCNN	Extreme Weather Dataset	mAP@[IoU=0.5] = 0.35



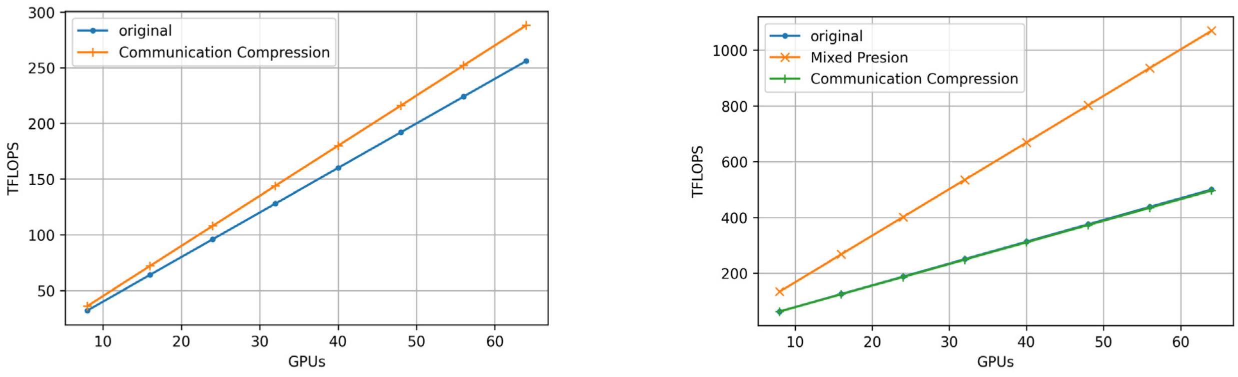
(a) Extreme Weather Analytics. **(b)** Image Classification.

Fig. 6. GPU utilization (%) of HPC AI500 V3.0. The X-axis represents different time steps.



(a) The training speed of different configurations. **(b)** The model quality of different configurations.

Fig. 7. The trade-off between the training speed and model quality. The workload is Image Classification. We use images per second to indicate how fast the training is.

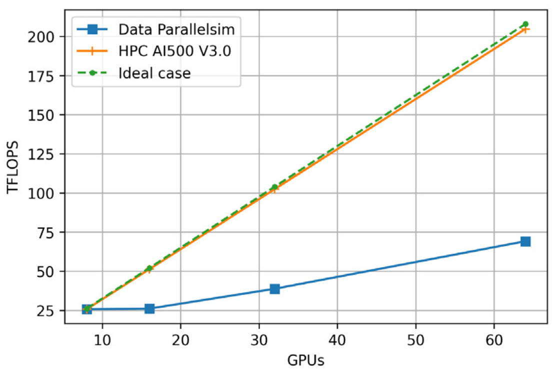


(a) Extreme Weather Analytics. **(b)** Image Classification.

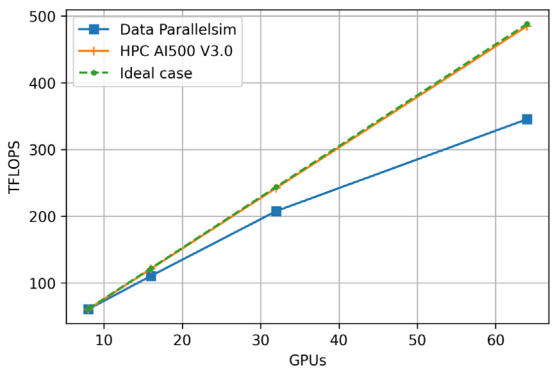
Fig. 8. The optimization experiments of HPC AI500 V3.0. In Fig. 8(b), the lines of the original and mixed precision overlap for their similar performance.

4.5. Comparison experiments

We compare our work with data parallelism (DP), which is a mainstream parallel method used in many previous work [17,18,47,61]. In this experiment, we focus on scale efficiency in terms of FLOPS. The system scales from 8 GPUs to 64 GPUs. As shown in Fig. 9, the scaling efficiency of DP is much lower than our approach in both Extreme Weather Analysis and Image Classification. The heavy communication



(a) Extreme Weather Analytics.



(b) Image Classification.

Fig. 9. The comparison experiments between HPC AI500 V3.0 against a setting using data parallelism.

overhead of DP is the main reason for this phenomenon because all the model copies of DP need to be synchronized globally at the end of each training step. The base model in the model ensemble of HPC AI500 V3.0 is trained highly independently without synchronization, so the communication overhead is avoided.

5. Conclusion

In this paper, we reformulate the HPC AI scalability issue and present HPC AI500 V3.0, a scalable and customizable framework for HPC AI benchmarking. The methodology of HPC AI500 V3.0 allows users to integrate existing AI benchmarks in a bagging manner, a meta-algorithm of ensemble learning with intrinsic high parallelism, leading to scalable benchmarking. The bagging management and model parallelism management of HPC AI500 V3.0 gives users the flexibility to control the size of model ensembles and the degree of model parallelism, enabling various optimizations from both system and algorithm levels. Based on HPC AI500 V2.0, which tackles the equivalence, representativeness, affordability, and repeatability issues, HPC AI500 V3.0 provide a complete HPC AI benchmarking framework. Reusing the workloads of HPC AI500 V2.0, we evaluate HPC AI500 V3.0 on a typical HPC system and the experimental results show the scalability and customizability of the proposed benchmarking framework.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. 25 (2012).
[2] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[3] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.
[4] S. Ren, K. He, R. Girshick, J. Sun, Faster R-CNN: Towards real-time object detection with region proposal networks, Adv. Neural Inf. Process. Syst. 28 (2015).
[5] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, A.C. Berg, SSD: Single shot multibox detector, in: European Conference on Computer Vision, Springer, 2016, pp. 21–37.
[6] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 779–788.
[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, Adv. Neural Inf. Process. Syst. 30 (2017).
[8] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, 2018, arXiv preprint arXiv:1810.04805.
[9] K. Han, A. Xiao, E. Wu, J. Guo, C. Xu, Y. Wang, Transformer in transformer, Adv. Neural Inf. Process. Syst. 34 (2021).
[10] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[11] OpenAI, OpenAI: AI and Compute, https://openai.com/blog/ai-and-compute/.
[12] A. Gholami, Medium: AI and Memory Wall, https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8/.
[13] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, B. Catanzaro, Megatron-LM: Training multi-billion parameter language models using model parallelism, 2019, arXiv preprint arXiv:1909.08053.
[14] T. Brown, B. Mann, N. Ryder, M. Subbiah, J.D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Adv. Neural Inf. Process. Syst. 33 (2020) 1877–1901.
[15] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, Z. Chen, Gshard: Scaling giant models with conditional computation and automatic sharding, 2020, arXiv preprint arXiv:2006.16668.
[16] W. Fedus, B. Zoph, N. Shazeer, Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021, arXiv preprint arXiv:2101.03961.
[17] A. Mathuriya, D. Bard, P. Mendygral, L. Meadows, J. Arnemann, L. Shao, S. He, T. Kärnä, D. Moise, S.J. Pennycook, et al., CosmoFlow: Using deep learning to learn the universe at scale, in: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2018, pp. 819–829.
[18] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, et al., Exascale deep learning for climate analytics, in: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2018, pp. 649–660.
[19] W. Jia, H. Wang, M. Chen, D. Lu, L. Lin, R. Car, E. Weinan, L. Zhang, Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning, in: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2020, pp. 1–14.
[20] Z. Guo, D. Lu, Y. Yan, S. Hu, R. Liu, G. Tan, N. Sun, W. Jiang, L. Liu, Y. Chen, et al., Extending the limit of molecular dynamics with ab initio accuracy to 10 billion atoms, 2022, arXiv preprint arXiv:2201.01446.
[21] Oak Ridge National Laboratory, Summit, https://www.olcf.ornl.gov/summit/.
[22] Fujitsu, Fugaku, https://www.fujitsu.com/global/about/innovation/fugaku/.
[23] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, Elsevier, 2011.
[24] J.J. Dongarra, P. Luszczek, A. Petitet, The LINPACK benchmark: Past, present and future, Concurr. Comput.: Pract. Exper. 15 (9) (2003) 803–820.
[25] J. Dongarra, Top500 Website, https://www.top500.org/.
[26] J. Dongarra, CM-5 in TOP500 List, https://www.top500.org/lists/top500/1993/06/.
[27] J. Dongarra, Fugaku in TOP500 List, https://www.top500.org/news/japan-captures-top500-crown-arm-powered-supercomputer/.
[28] J. Zhan, Call for establishing benchmark science and engineering, 2021, arXiv preprint arXiv:2112.09514.
[29] R. Adolf, S. Rama, B. Reagen, G.-Y. Wei, D. Brooks, Fathom: Reference workloads for modern deep learning methods, in: 2016 IEEE International Symposium on Workload Characterization, IISWC, IEEE, 2016, pp. 1–10.
[30] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, M. Zaharia, Dawnbench: An end-to-end deep learning benchmark and competition, Training 100 (101) (2017) 102.
[31] H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Phanishayee, B. Schroeder, G. Pekhimenko, TBD: Benchmarking and analyzing deep neural network training, 2018, arXiv preprint arXiv:1803.06905.

[32] W. Gao, F. Tang, L. Wang, J. Zhan, C. Lan, C. Luo, Y. Huang, C. Zheng, J. Dai, Z. Cao, et al., AIBench: An industry standard internet service AI benchmark suite, 2019, arXiv preprint arXiv:1908.08998.
[33] V.J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, et al., Mlperf inference benchmark, in: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture, ISCA, IEEE, 2020, pp. 446–459.
[34] P. Mattson, C. Cheng, G. Diamos, C. Coleman, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf, et al., Mlperf training benchmark, Proc. Mach. Learn. Syst. 2 (2020) 336–349.
[35] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, K. Keutzer, Imagenet training in minutes, in: Proceedings of the 47th International Conference on Parallel Processing, 2018, pp. 1–10.
[36] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, K. He, Accurate, large minibatch SGD: Training imagenet in 1 hour, 2017, arXiv preprint arXiv:1706.02677.
[37] T. Akiba, S. Suzuki, K. Fukuda, Extremely large minibatch SGD: Training resnet-50 on imagenet in 15 minutes, 2017, arXiv preprint arXiv:1711.04325.
[38] M. Cho, U. Finkler, S. Kumar, D. Kung, V. Saxena, D. Sreedhar, Powerai DDL, 2017, arXiv preprint arXiv:1708.02188.
[39] V. Codreanu, D. Podareanu, V. Saletore, Scale out for large minibatch SGD: Residual network training on ImageNet-1K with improved accuracy and reduced time to train, 2017, arXiv preprint arXiv:1711.04291.
[40] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, et al., Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes, 2018, arXiv preprint arXiv:1807.11205.
[41] H. Mikami, et al., Imagenet/resnet-50 training in 224 seconds, 2018, arXiv preprint arXiv:1811.05233.
[42] C. Ying, S. Kumar, D. Chen, T. Wang, Y. Cheng, Image classification at supercomputer scale, 2018, arXiv preprint arXiv:1811.06992.
[43] M. Yamazaki, A. Kasagi, A. Tabuchi, T. Honda, M. Miwa, N. Fukumoto, T. Tabaru, A. Ike, K. Nakashima, Yet another accelerated SGD: Resnet-50 training on imagenet in 74.7 seconds, 2019, arXiv preprint arXiv:1903.12650.
[44] MLCommons, MLPerf-Training-Result-V1.1, https://mlcommons.org/en/training-normal-11//.
[45] Preferred networks website, https://www.preferred.jp/en/.
[46] N.S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, P.T.P. Tang, On large-batch training for deep learning: Generalization gap and sharp minima, 2016, arXiv preprint arXiv:1609.04836.
[47] A. Sergeev, M. Del Balso, Horovod: Fast and easy distributed deep learning in TensorFlow, 2018, arXiv preprint arXiv:1802.05799.
[48] J. Rasley, S. Rajbhandari, O. Ruwase, Y. He, Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 3505–3506.
[49] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, et al., Mesh-tensorflow: Deep learning for supercomputers, Adv. Neural Inf. Process. Syst. 31 (2018).
[50] Z. Jiang, W. Gao, L. Wang, X. Xiong, Y. Zhang, X. Wen, C. Luo, H. Ye, X. Lu, Y. Zhang, et al., HPC AI500: A benchmark suite for HPC AI systems, in: International Symposium on Benchmarking, Measuring and Optimization, Springer, 2018, pp. 10–22.
[51] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N.R. Devanur, G.R. Ganger, P.B. Gibbons, M. Zaharia, PipeDream: Generalized pipeline parallelism for DNN training, in: Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019, pp. 1–15.
[52] Z. Jia, M. Zaharia, A. Aiken, Beyond data and model parallelism for deep neural networks, Proc. Mach. Learn. Syst. 1 (2019) 1–13.
[53] data-parallelim, https://en.wikipedia.org/wiki/Data_parallelism.
[54] Z. Ren, Y. Liu, T. Shi, L. Xie, Y. Zhou, J. Zhai, Y. Zhang, Y. Zhang, W. Chen, AIPerf: Automated machine learning as an AI-HPC benchmark, Big Data Min. Anal. 4 (3) (2021) 208–220.
[55] S. Kudo, K. Nitadori, T. Ina, T. Imamura, Prompt report on exa-scale HPL-AI benchmark, in: 2020 IEEE International Conference on Cluster Computing, CLUSTER, IEEE, 2020, pp. 418–419.
[56] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, 2016, arXiv preprint arXiv:1611.01578.
[57] Z. Jiang, W. Gao, F. Tang, L. Wang, X. Xiong, C. Luo, C. Lan, H. Li, J. Zhan, HPC AI500 v2. 0: The methodology, tools, and metrics for benchmarking HPC AI systems, in: 2021 IEEE International Conference on Cluster Computing, CLUSTER, IEEE, 2021, pp. 47–58.
[58] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.
[59] Z.-H. Zhou, Ensemble learning, in: Machine Learning, Springer, 2021, pp. 181–210.
[60] T. Ben-Nun, M. Besta, S. Huber, A.N. Ziogas, D. Peter, T. Hoefler, A modular benchmarking infrastructure for high-performance and reproducible deep learning, in: 2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS, IEEE, 2019, pp. 66–77.
[61] S. Farrell, M. Emani, J. Balma, L. Drescher, A. Drozd, A. Fink, G. Fox, D. Kanter, T. Kurth, P. Mattson, et al., MLPerf™ HPC: A holistic benchmark suite for scientific machine learning on HPC systems, in: 2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments, MLHPC, IEEE, 2021, pp. 33–45.
[62] S. Ruder, An overview of gradient descent optimization algorithms, 2016, arXiv preprint arXiv:1609.04747.
[63] R. Farber, AI-HPC is Happening Now, InsideHPC Special Report, InsideHPC, LLC, 2017.
[64] E.A. Huerta, A. Khan, E. Davis, C. Bushell, W.D. Gropp, D.S. Katz, V. Kindratenko, S. Koric, W.T. Kramer, B. McGinty, et al., Convergence of artificial intelligence and high performance computing on NSF-supported cyberinfrastructure, J. Big Data 7 (1) (2020) 1–12.
[65] H. Lee, A. Merzky, L. Tan, M. Titov, M. Turilli, D. Alfe, A. Bhati, A. Brace, A. Clyde, P. Coveney, et al., Scalable HPC & AI infrastructure for COVID-19 therapeutics, in: Proceedings of the Platform for Advanced Scientific Computing Conference, 2021, pp. 1–13.
[66] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, Ieee, 2009, pp. 248–255.
[67] I. Kandel, M. Castelli, The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset, ICT Express 6 (4) (2020) 312–315.
[68] J. Surowiecki, The Wisdom of Crowds, Anchor, 2005.
[69] A. Krizhevsky, One weird trick for parallelizing convolutional neural networks, 2014, arXiv preprint arXiv:1404.5997.
[70] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, C.-J. Hsieh, Large batch optimization for deep learning: Training bert in 76 minutes, 2019, arXiv preprint arXiv:1904.00962.
[71] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[72] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (7) (2011).
[73] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q.V. Le, Y. Wu, et al., Gpipe: Efficient training of giant neural networks using pipeline parallelism, Adv. Neural Inf. Process. Syst. 32 (2019).
[74] 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: 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 16, 2016, pp. 265–283.
[75] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, Adv. Neural Inf. Process. Syst. 32 (2019).
[76] Huawei, Mindspore, https://www.mindspore.cn/.
[77] Nvidia, Nsight system, https://developer.nvidia.com/nsight-systems.
[78] Lawrence Livermore National Laboratory, SLURM, https://slurm.schedmd.com/.
[79] T. Combe, A. Martin, R. Di Pietro, To docker or not to docker: A security perspective, IEEE Cloud Comput. 3 (5) (2016) 54–62.
[80] Nvidia, Nvidia-smi, https://developer.nvidia.com/nvidia-system-management-interface.
[81] iftop, https://en.wikipedia.org/wiki/Iftop.
[82] E. Racah, C. Beckham, T. Maharaj, S. Ebrahimi Kahou, M. Prabhat, C. Pal, Extremeweather: A large-scale climate dataset for semi-supervised detection, localization, and understanding of extreme weather events, Adv. Neural Inf. Process. Syst. 30 (2017).