SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

September 2024, Vol. 67, Iss. 9, 192102:1–192102:12
https://doi.org/10.1007/s11432-023-3989-3

# OpBench: an operator-level GPU benchmark for deep learning

Qingwen GU[1], Bo FAN[2*], Zhengning LIU[3], Kaicheng CAO[2], Songhai ZHANG[1] & Shimin HU[1]

[1]*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084*, China;*
[2]*National Innovation Institute of Defense Technology, Academy of Military Science, Beijing* 100071*, China;*
[3]*Beijing Fitten Technology Co., Ltd., Beijing* 100080*, China*

Received 31 July 2023/Revised 30 December 2023/Accepted 22 March 2024/Published online 20 August 2024

**Abstract** Operators (such as Conv and ReLU) play an important role in deep neural networks. Every neural network is composed of a series of differentiable operators. However, existing AI benchmarks mainly focus on accessing model training and inference performance of deep learning systems on specific models. To help GPU hardware find computing bottlenecks and intuitively evaluate GPU performance on specific deep learning tasks, this paper focuses on evaluating GPU performance at the operator level. We statistically analyze the information of operators on 12 representative deep learning models from six prominent AI tasks and provide an operator dataset to show the different importance of various types of operators in different networks. An operator-level benchmark, OpBench, is proposed on the basis of this dataset, allowing users to choose from a given range of models and set the input sizes according to their demands. This benchmark offers a detailed operator-level performance report for AI and hardware developers. We also evaluate four GPU models on OpBench and find that their performances differ on various types of operators and are not fully consistent with the performance metric FLOPS (floating point operations per second).

**Keywords** deep learning, operator benchmark, model benchmark, GPU performance analysis, deep neural networks

## 1 Introduction

The past ten years have witnessed a resurgence of machine learning, particularly deep learning. Recently, deep neural networks, such as convolution neural networks (CNNs) [1–3] and transformers [4–6], have emerged as the leading technology, driving many new applications that have gained widespread popularity in the computing age. Deep learning has powered image generation [7] and human-machine conversation [6], helped develop autonomous vehicles and robots, and shown promise in healthcare applications, including disease diagnosis and medical image analysis.

Behind the success of deep learning are the increasingly large model sizes and complex model structures requiring tremendous computation and memory resources. With the burgeoning of AI applications, model training and inference have become a development bottleneck and a critical workload. To perform well, a deep neural network will often have millions of tunable parameters (ResNet50 [3] has up to 25 million parameters); there are even more parameters in the recently highly publicized large models, e.g., 175 billion parameters in GPT-3 [6]. A larger number of parameters means a greater computational load. To address these growing computational demands, hardware developers have been focusing on improving model training and inference performance by designing optimized GPU hardware for deep learning tasks, leading to a demand for GPU performance assessment metrics across various AI use cases.

Operators (such as convolutions and activation functions) have pivotal roles in deep neural networks. Every neural network is composed of a series of differentiable operators. ResNet50 is connected by 50 convolutional layers, and a typical convolutional layer is composed of operators that include convolution, batch normalization, and an activation function (like ReLU). In various tasks, different types of operators

* Corresponding author (email: oceanpearl@126.com)

 

differ in importance, which means certain operators in specific scenarios should be paid more attention to when performing optimizations. For example, convolution is commonly used in computer vision but rarely used in natural language processing. Thus, without a clear view of which operators are more likely to be used or more time-consuming in a given scenario, developers optimize GPU hardware or evaluate hardware performance on specific deep learning tasks with difficulty. Moreover, each GPU model may have its own way of optimizing a certain operator. Bianco et al. [8] tested the performance of some representative neural networks on two different GPUs (Titan Xp and Jetson TX1), and the results show that the ranking of model inference time is inconsistent on the two GPUs. For example, AlexNet [2] runs faster than SqueezeNet [9] on Titan Xp but vice versa on Jetson TX1. The discrepancy is observed mainly because these networks are composed of various types of operators that are optimized diversely on different GPUs. This diversity could confuse developers when they suffer a sudden drop in performance when running the network on a new GPU, and analyzing the performance bottleneck could be troublesome without the help of operator-level benchmarks. Thus, it is important to test GPU performance at the operator level and carefully design the metrics for operator benchmarks.

However, existing AI benchmarks such as MLPerf [10,11] mainly focus on accessing model training and the inference performance of machine learning systems on a single specific model. For example, MLPerf inference includes five common models, along with the corresponding datasets for developers to test their ML systems with. Developers who need to assess the performance of GPU hardware on designated operators will face a challenge. Some benchmarks include operator testing (e.g., SuperBench [12]), but they simply offer a few frequently used operators for users to test with. Because the importance of various operators differs under designated application scenarios, as mentioned before, it would confuse users about the benchmark results without telling them which operators are of more importance in their tasks.

Therefore, we conclude that GPU performance must be tested at the operator level for two reasons. First, the computing bottlenecks of GPU hardware are determined by operators, which are the units constituting the neural networks. If the contribution of each type of operator to different networks is analyzed, we can design a metric for operator testing that considers the different importance of operators. The test results can guide developers on which operators are more important and time-consuming and, thus, should be optimized with higher priorities. Second, if users want to select from different GPUs for their AI tasks but the specific network structure has not been determined yet, it would not be very suitable to do model benchmarking. However, the operators that should be paid more attention to can be determined from the specific tasks. For example, if it is a text-generation task that has a high probability of involving transformers, SoftMax and matrix multiplication would have a greater impact on efficiency among all the operators. So, with an operator benchmark to provide a clear view of the GPU performance at the operator level within the selected tasks, developers can find computing bottlenecks and evaluate GPU performance on specific deep learning tasks intuitively.

Deep learning operator benchmarking faces three major challenges. First, the number of operators in a neural network is often large, each with different types or with different input parameters. A single network such as SSD300 [13], which is a representative model for object detection, contains up to one thousand operators covering approximately 20 operator types, not to say that multiple networks need to be included in the benchmark. Second, the input shapes or parameters of operators would change with the input data size of the network, so how to analyze the operators is complicated depending on the input size required by users. Third, to achieve higher efficiency of model inference, operator fusion (or kernel/layer fusion) is the key optimization in many state-of-the-art execution frameworks for deep learning, which allows the combination of multiple operators into a single operator. This optimization makes operator benchmarking rather sophisticated because the fusion needs to be considered instead of seeing each operator separately. Proposing the first operator benchmark for deep learning, we managed to address the first two challenges and leave the issue of operator fusion to later work.

We highlight our contributions as follows:

• **An operator and model dataset for deep learning.** We provide an operator dataset that covers up to 200 types of operators commonly seen in deep learning tasks and 13 representative neural networks. Users can select from the given models and determine the number of calls of each operator.

• **An operator benchmark.** We present OpBench, an operator-level benchmark that allows users to choose from a given range of models and set input sizes according to their demands. The benchmark offers a detailed operator performance report to guide AI and GPU developers. Table 1 [10, 12, 14, 15] shows the difference between OpBench and some existing AI benchmarks.

**Table 1** Comparison among existing AI benchmarks[a)]

| | Scope | Model test | Op test | Custom input size | Op importance |
|---|---|---|---|---|---|
| MLPerf [10] | ML systems | ✓ | × | × | × |
| SuperBench [12] | ML systems | ✓ | ✓ | × | × |
| AIPerf [14] | HPC | ✓ | ✓ | ✓ | × |
| Benanza [15] | GPU | ✓ | ✓ | ✓ | × |
| OpBench (ours) | GPU | ✓ | ✓ | ✓ | ✓ |

a) The "Custom input size" column refers to whether the input of the neural networks or operators under test could be set by users. "Op importance" refers to whether the importance of different types of operators in various deep learning tasks and networks is considered in the benchmark. HPC: high performance computing.

The remainder of this paper is organized as follows. In Section 2, we review the existing AI benchmarks and microbenchmarks, noting their downsides in operator-level benchmarking. In Section 3, the prerequisite knowledge of operator testing is presented. In Section 4, we describe the details of our benchmark design, including the architecture, interactions, and measurements. In Section 5, we evaluate our benchmark on the GPUs of multiple architectures from different organizations. We summarize our work in Section 6. The proposed operator dataset and the OpBench source code are publicly accessible on GitHub[1)].

## 2 Related work

### 2.1 Deep learning benchmarks

To evaluate deep learning systems, which include hardware and software, end-to-end benchmarks are developed. As the most recognized machine learning benchmark to date, MLPerf [10] crafted a collection of time-to-accuracy metrics to measure the overall performance of AI systems. However, the workloads are still limited and cannot satisfy the various needs of deep learning applications. Moreover, this design brings trouble in evaluation and comparison. To provide a single and direct measurement that is auto-adaptively scalable to various scales of machines, AIPerf [14] uses operations per second (OPS) as the major metric to quantify high-performance computing (HPC) system performance. However, because memory operations and possible optimizations are neglected, the computing operation count cannot fully represent system consumption. Moreover, it ignores the difference in importance among various operators when measuring performance. SuperBench [12] provides not only a model benchmark but also a microbenchmark to measure primitive computation and communication, but the microbenchmark mainly focuses on primary functions at the bottom, such as GEMM and device-to-host memory copy, instead of the commonly used operators in AI tasks. In addition, the importance of different operators is neglected in SuperBench.

### 2.2 Microbenchmarks

To demystify GPU micro-architecture and access GPU performance precisely, researchers propose microbenchmarks that test primary functions on GPUs. Zhang et al. [16] developed a microbenchmark-based performance model for NVIDIA GeForce GPUs that identifies GPU program bottlenecks and quantitatively analyzes performance and thus allows programmers and architects to predict the benefits of potential program optimizations and architectural improvements. Gomez-Luna et al. [17] presented a detailed microbenchmark-based analysis of atomic additions in shared memory to quantify the impact of access conflicts on latency and throughput, which can be an example for testing other atomic operations on GPUs. These microbenchmarks mainly focus on primary functions and the micro-architecture of GPUs, which can elucidate how to build a microbenchmark for deep learning operators. Benanza [15], which also profiles model execution on GPU at the operator level, is the closest to our work. However, instead of analyzing operators, Benanza mainly focuses on layer-level potential optimizations and sources of inefficiency, such as cuDNN convolution algorithm selection and Tensor Core usage. It does not provide an overall view of the performance or distinctions of each type of operator. Moreover, Benanza analyzes one model at a time, whereas our work can test several models together.

1) https://github.com/SabretoothX/OpBench/.

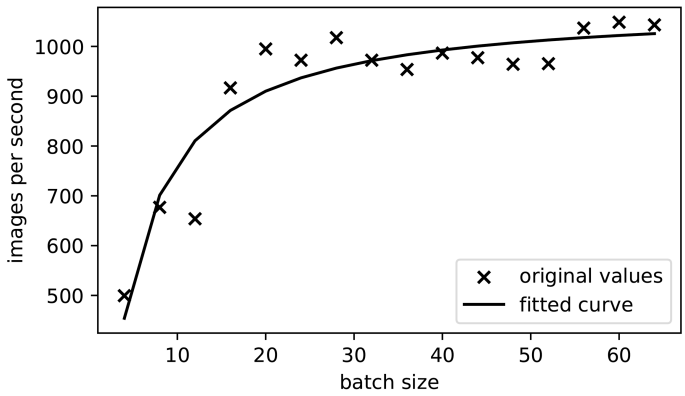


**Figure 1** With an increase in batch size, the change in images processed per second by ResNet50 [3] on NVIDIA GeForce RTX 3090, tested with Pytorch [19]. The batch sizes are set to multiples of 4.

# 3 Background

## 3.1 Operators in deep neural networks

Each deep learning model is a combination of differentiable operators. For example, a standard CNN comprises an input layer, hidden layers, and an output layer. The hidden layers include layers that perform convolutions, followed by other layers such as pooling, fully connected, and normalization layers. Each layer contains an operator which acquires input from the previous layer and calculates the product for subsequent layers. Thus, without considering optimizations, the execution time of a neural network should be equal to the sum of the running time of all operators in the network unless part of the network is structured in parallel, which is rarely seen.

However, the equation often does not hold because of operator fusion. To achieve high accuracy, DNN (deep neural network) models have become increasingly deep, with hundreds or even thousands of operator layers, leading to high memory and computational requirements for inference. Operator fusion (or kernel/layer fusion) is the key optimization in many state-of-the-art DNN execution frameworks, such as TensorFlow [18], Pytorch-Mobile [19], Jittor [20], and TVM [21], that aims to improve DNN inference efficiency. The basic idea of such fusion is to combine multiple operators into a single operator, with the advantage that intermediate results do not need to be stored in memory.

## 3.2 Kernel launch latency and overhead

When we run a model using a deep learning framework with the GPU enabled, the CPU runs the framework and function scheduling, while the GPU is responsible for most of the computational process of operators, which is called kernel execution. Normally, the GPU time greatly exceeds the CPU time when running a neural network. When the parallelism of the whole system is relatively good, the CPU time is negligible, so the sum of the kernel execution time is nearly identical to the total execution time of the network. In contrast, if the computational loads are rather light, the CPU time would not be considerably affected, but the GPU time would be substantially reduced, in which case the CPU time cannot be ignored compared to the GPU time. As Figure 1 shows, when the input batch size increases, there is an upward trend in the number of images processed by ResNet50 per second. Particularly when the batch size is small, the parallelism of the whole system is bad because of the relatively high overhead compared with the short CUDA kernel time. Ideally, we would like to reduce the bias caused by CPU time and GPU overhead to make the benchmark results more accurate, so the computational loads need to be as large as possible.

Overhead is defined as the time it takes to perform an operation that should ideally take zero time or the time spent (latency) without useful kernel work. When running an operator with GPUs enabled, three types of overhead are mainly used: CPU wrapper, memory, and GPU launch overhead [22].

• **CPU wrapper overhead.** This overhead is for the wrappers around a kernel on the host CPU side. Most profiling tools that are used to measure the kernel execution time inevitably add slight overhead to capture trace data, meaning that events might have a longer duration on the timeline than their actual execution time when running without profiling tools.

• **Memory overhead.** This overhead is for transferring data between the CPU and GPU, which includes the time required to read input tensors from the CPU and write output to DRAM. If the memory copy performance is unimportant, input tensors need to be moved to the GPU before testing.

• **GPU launch overhead.** This overhead is the time taken for the GPU to retrieve and start executing the command, excluding the time of actual kernel execution.

The overhead of launching kernels mainly impacts the situation of small kernels, while the GPU execution time mainly impacts the situation of large kernels [23]. To reduce the impact of unwanted overhead, we recommend a large input size when analyzing the kernel execution time of operators.

### 3.3 Deep learning frameworks

Because deep learning algorithms (particularly backward propagation) are often complicated to implement, various frameworks have been developed to provide researchers and developers with convenient ways of rapidly developing deep learning systems. To provide user-friendly APIs, deep learning frameworks transform programs in high-level languages into an internal representation of certain functionalities. The frameworks invoke the low-level efficient libraries, such as cuDNN, to execute primary operations such as matrix multiplication. An open-source framework with sufficient community support would be a decent candidate for building the benchmark. However, common deep learning frameworks such as TensorFlow [18] and PyTorch [19] only officially support Nvidia and AMD GPUs, while the adaptation on other GPUs is rather difficult. Pytorch has a profiling tool that can help analyze the number of calls of each operator and record information such as input shapes, CUDA time, and memory consumption. Because most of the common neural networks have an official Pytorch implementation, we choose Pytorch and Pytorch profiler as the analysis tools. Jittor is an open-source deep learning framework with high-performance operator compilers and tuners. We choose Jittor [20] to build our benchmark system for the following reasons:

• **Easy installation.** Unlike Pytorch and other frameworks, where users must manually select a version corresponding to the GPU driver version, Jittor can automatically detect and install the correct version, which is convenient when multiple types of GPUs need to be evaluated.

• **Official adaptations for various GPUs.** Jittor officially supports GPUs other than Nvidia and AMD, including Hygon DCU and Iluvatar Tiangai.

• **Kernel profiler.** Jittor provides a profile tool that can analyze the kernel execution time of operators on the GPU.

• **Built-in operators and custom operators.** In addition to more than 200 built-in operators commonly used in machine learning, Jittor supports easy compilation, execution, and profiling for custom operators. This capability is important because some neural networks may have custom operators that are not implemented in the frameworks.

## 4 Benchmark design

### 4.1 Benchmark architecture

To provide a clear view of deep learning applications, we design the benchmark architecture with three levels: deep learning areas, tasks, and neural networks. To meet industry needs, we choose the most common deep learning tasks and representative networks for the benchmark and allow users to select among the given range according to their demands. Once the networks have been chosen, the benchmark will only run within defined limits. Below is a brief introduction of three popular areas of deep learning applications covered by the benchmark, and the specific networks that we provide are found in Table 2 [3, 5, 6, 13, 24–31].

**Computer vision (2D).** Computer vision (CV) is a field that involves enabling computers to "comprehend" visual data such as images and videos. Typical CV tasks include image classification, object detection, and semantic segmentation, and the fundamental problem in CV is image feature extraction. Image classification and object detection tasks involve identifying the most likely class or detecting the type and location of objects in the image. The semantic segmentation task involves dividing an image into nonoverlapping regions or segments, each of which represents a distinct object or class. These techniques are widely used in production and everyday endeavors, such as autonomous monitoring, remote sensing,

**Table 2** Deep learning areas, tasks, and networks included in OpBench, along with the basic operator information of each network

| Area | Task | Network | No. of op types | No. of ops |
|---|---|---|---|---|
| Computer vision (2D) | Feature extraction (image classification)[a)] | VGG16 [24] | 5 | 21 |
| | | ResNet50 [3] | 4 | 99 |
| | Object detection | SSD300 [13] | 15 | 1169 |
| | | YOLOv5 [25] | 6 | 71 |
| | | Faster RCNN [26] | 22 | 546 |
| | Semantic segmentation | FCN [27] | 3 | 53 |
| Computer vision (3D) | Object classification | PointNet [28] | 5 | 47 |
| | | DGCNN [29] | 17 | 69 |
| | Scene segmentation | PointNet++ [30] | 14 | 15626 |
| | | DeepLab V3 [31] | 3 | 52 |
| Natural language processing | Machine translation and text generation | BERT [5] | 16 | 336 |
| | | GPT [6] | 14 | 355 |

a) Although VGG and ResNet are first designed to solve image classification tasks, they are much more widely used as image feature extraction modules in applications today.

medical image analysis, and image/video editing. In the past decades, deep-learning-based approaches, particularly CNNs, have developed rapidly and considerably improved the performance of CV tasks.

**Computer vision (3D).** As computer vision becomes increasingly popular in widely varying areas, researchers have recognized that depth information can provide a complete 3D understanding for autonomous robots and considerably enhance the accuracy of object recognition, detection, and modeling. This attribute is particularly crucial for applications that require the ability to accurately estimate distance or 3D object shapes, such as self-driving cars. To achieve a full 3D understanding, recent studies have highlighted the development of new CNN architectures and deep learning techniques [32]. The input for 3D vision networks typically comprises a point cloud, which is composed of points in a three-dimensional space coordinate system.

**Natural language processing (NLP).** NLP is a tract of artificial intelligence and linguistics devoted to making computers understand the statements or words written in human languages [33]. The importance of NLP stems from the vast amount of data available online, with at least 20 billion pages that can be used as a valuable resource, providing that meaningful information can be extracted from them through NLP techniques [34]. Common tasks of NLP include machine translation and text generation. OpenAI ChatGPT [6] is a successful application of NLP that can perform conversations with people rather intelligently.

### 4.2 Custom input shapes

Obviously, the shape of input tensors has a considerable impact on the kernel launch time of operators, and the input of operators is determined by the input of the network they belong to, so we allow users to set the input shapes of networks under test according to their needs. Because the input type differs among tasks, the size of the input image, point cloud, and sentence should be set separately. In CV (2D), the data size is determined by the image height and width ($H \times W$); in CV (3D), the data size is determined by the number of points in the point cloud ($N$); and in NLP, the data size is determined by the length of input sentence ($L$). The user also specifies the inference batch size $B$. We also provide a set of standard input shapes for ranking and reference.

Notably, when the batch size and input shapes are relatively small, the computational loads on the GPU will be light, and the CPU time and other possible overhead will be comparatively non-negligible. In this case, the benchmark result cannot reveal the actual time of running the models (which includes CPU time) because only the kernel launch time on GPU is recorded. Moreover, the profiling tool also brings overhead and will be recorded along with the kernel launch time. Consequently, we recommend a large batch or large input shapes to exploit the GPUs' performance.

### 4.3 Operator dataset

Because of the difference in complexity and amount, various types of operators usually have different weights in neural networks, and the same type of operators can differ in importance in diverse tasks. For example, there are 24 convolution operators in YOLOv5 [25] but only eight in SSD300 [13]. Moreover,

**Table 3** Statistical information of some operators in SSD300 [13], given by the operator dataset[a)]

| Operator type | Calls | Input shapes |
|---|---|---|
| unbind | 2 | [$B$, 3, 256, 256] |
| unbind | 1 | [$B$, 3, $H$, $W$] |
| select | 2 | [$B$, 3, 256, 256] |
| select | 2 | [$B$, 3, $H$, $W$] |
| unsqueeze | 2 | [3] |
| upsample | 1 | [$B$, 3, 256, 256] |
| conv2d | 1 | [$B$, 3, $H$, $W$], [64, 3, 3, 3] |
| conv2d | 1 | [$B$, 64, $H$, $W$], [64, 64, 3, 3] |
| conv2d | 1 | [$B$, 64, $H/2$, $W/2$], [128, 64, 3, 3] |
| conv2d | 1 | [$B$, 128, $H/2$, $W/2$], [128, 128, 3, 3] |
| conv2d | 1 | [$B$, 128, $H/4$, $W/4$, $H/4$, $W/4$], [256, 128, 3, 3] |
| conv2d | 2 | [$B$, 256, $H/4$, $W/4$, $H/4$, $W/4$], [256, 256, 3, 3] |
| relu | 2 | [$B$, 64, $H$, $W$] |
| relu | 2 | [$B$, 128, $H/2$, $W/2$] |
| relu | 3 | [$B$, 256, $H/4$, $W/4$, $H/4$, $W/4$] |
| maxpool2d | 1 | [$B$, 64, $H$, $W$] |
| maxpool2d | 1 | [$B$, 128, $H/2$, $W/2$] |
| maxpool2d | 1 | [$B$, 256, $H/4$, $W/4$, $H/4$, $W/4$] |
| ... | ... | ... |

a) The dimension of the input image tensor of the network is $[B, 3, H, W]$. The convolution operator has two values in the "Input shapes" column, the former and latter values being the shapes of the input tensor and the convolution kernel, respectively.

operator parameters and input shapes considerably impact the execution cost, so operators of the same type with different input shapes should be considered separately.

To accurately reveal the importance of different operators in various tasks, we construct an operator dataset by analyzing some frequently used networks in the industry, as shown in Table 2. The number of calls and the input shape of each operator are recorded separately for each network. Table 3 shows the information of some main operators in SSD300 provided by the dataset. According to the records, one easily finds the most important operators in specific networks. The benchmark is also constructed based on the dataset. Likewise, others can implement their operator benchmark system using our operator dataset.

### 4.4 Measurement

Once the models under test are specified, we can acquire the importance of each operator according to its average number of calls in the models selected. Commonly, the parameters and input shapes have an important impact on the execution cost of operators, so operators of the same type but with different parameters and input shapes should be considered separately. For example, the convolution with a kernel size of $3 \times 3$ and the convolution with a kernel size of $7 \times 7$ are considered different operators at this stage. As in Table 3, each line is a different operation with a specific type and specific input shapes. To represent the different importance of various operators, we define the weight of operator $i$ in a certain test as

$$W_i = \frac{N_i}{N_{\text{models}}}, \tag{1}$$

where $N_i$ indicates the total number of calls of operator $i$ among the selected models, and $N_{\text{models}}$ represents the number of models under test. The statistics are from the operator dataset introduced in Subsection 4.3.

After obtaining the weights of operators using (1), the next key step of benchmarking is to acquire the executing consumption of the operators under test, which is accomplished by running each operator with the corresponding parameters and input shapes using a timing wrapper. Each type of operator is executed several times to obtain an average performance.

Then, the score of operator type $n$ can be calculated as

$$S_n = \sum_i W_i \times C_i, \tag{2}$$

**Table 4** Specifications of the GPU models under test

| GPU model | TFLOP/s (fp32) | Memory (GB) | Memory bit width |
|---|---|---|---|
| NVIDIA GeForce RTX 3090 | 35.58 | 24 | 384 |
| NVIDIA Quadro RTX 6000 | 13.69 | 24 | 384 |
| HYGON DCU Z100 | 21.6 | 32 | – |
| Iluvatar Tiangai 100 | 37 | 32 | – |

where each operator $i$ is of the same operator type $n$ but with different parameters and input shapes, $C_i$ is the execution cost of operator $i$, and $W_i$ is the weight of operator $i$ obtained using (1). Likewise, the overall score is the sum of operator scores:

$$S = \sum_n S_n = \sum_n \sum_i W_{n,i} \times C_{n,i}. \tag{3}$$

When the selected models and input shapes are set identically, developers can compare different GPUs in terms of their performance on specific deep learning tasks. Because the score is proportional to time consumption, a lower score means better performance. Eq. (3) can also be written as

$$S = \frac{\sum_n \sum_i N_{n,i} \times C_{n,i}}{N_{\text{models}}}. \tag{4}$$

Thus, when only one network is under test, the overall score summarizes the GPU computing time of all the operators in the network, meaning that when the CPU time is negligible and independent operators are not executed in parallel, the score is equal to the network inference time. When multiple networks are involved, the score can reveal the average running time. Consequently, the benchmark can also be used for model-level tests.

## 5 Benchmark assessment

### 5.1 Setup

Benchmarking GPUs at the operator level can be challenging because different inference engines and kernel profiling tools can lead to different results. To address this issue and ensure reproducible results that are free from common errors, we develop an operator benchmark system using the Jittor [20] framework. Submitters only need to determine the deep learning areas or specific models under test along with the sizes of input; then the system would ensure that the scores are reliable to be used for peer review.

In order to demonstrate the necessity of testing operators alone instead of the common model benchmarking, We assessed our benchmark system on 4 different models of GPUs, which include NVIDIA GeForce RTX 3090, NVIDIA Quadro RTX 6000, Haiguang DCU Z100, and Iluvatar Tiangai 100. The detailed information of the GPUs under test is shown in Table 4. Among the GPU specifications, FLOPS (floating point operations per second) shows the theoretical computing power and processing speed which is an important reference for the operator test results.

### 5.2 Results

The test results of different GPUs are found in Table 5, which is not fully consistent with the FLOPS of each GPU. Because the temperature and workload of GPUs and CPUs may affect the performance, we test each GPU five times and take the minimum value as the result. It is interesting to discover that NVIDIA GeForce RTX 3090 outperforms Iluvatar Tiangai 100 by twofold in CV (2D), while Tiangai 100 performs considerably better in NLP, although the FLOPS of the two GPU models are close. Model benchmarks might also reveal this type of result, but only OpBench can show the reasons and problems behind it, guiding developers to optimize the bottleneck operators or choose appropriate GPUs for their tasks. According to the detailed score of each operator, the large margin in the CV (2D) and NLP results is mainly due to the discrepancy of two operations, convolution, and batch normalization. As shown in Figure 2, convolution and batch normalization are two main operations in CV (2D) but are rarely seen in NLP. The discrepancy in performance has two possible causes. First, the optimization techniques of the GPU hardware and software differ, which is the most likely cause. NVIDIA GeForce RTX 3090 might

**Table 5** OpBench scores of 4 GPUs on the neural networks from 3 deep learning research areas respectively. Detailed information of the networks and areas can be found in Table 2

| | CV (2D) ↓ | CV (3D) ↓ | NLP ↓ |
|---|---|---|---|
| NVIDIA GeForce RTX 3090 | 37.84 | 8.32 | 65.84 |
| NVIDIA Quadro RTX 6000 | 87.18 | 38.43 | 88.02 |
| HYGON DCU Z100 | 96.18 | 23.52 | 42.06 |
| Iluvatar Tiangai 100 | 80.06 | 24.83 | 24.82 |

a) ↓ indicates that the lower score shows better performance. CV (2D) input size: $[B, C, H, W]$, $B = 16$, $H = 512$, $W = 512$; CV (3D) input size: $[B, C, N]$, $B = 32$, $N = 1024$; NLP input size: $[B, C, L]$, $B = 1$, $L = 1024$.

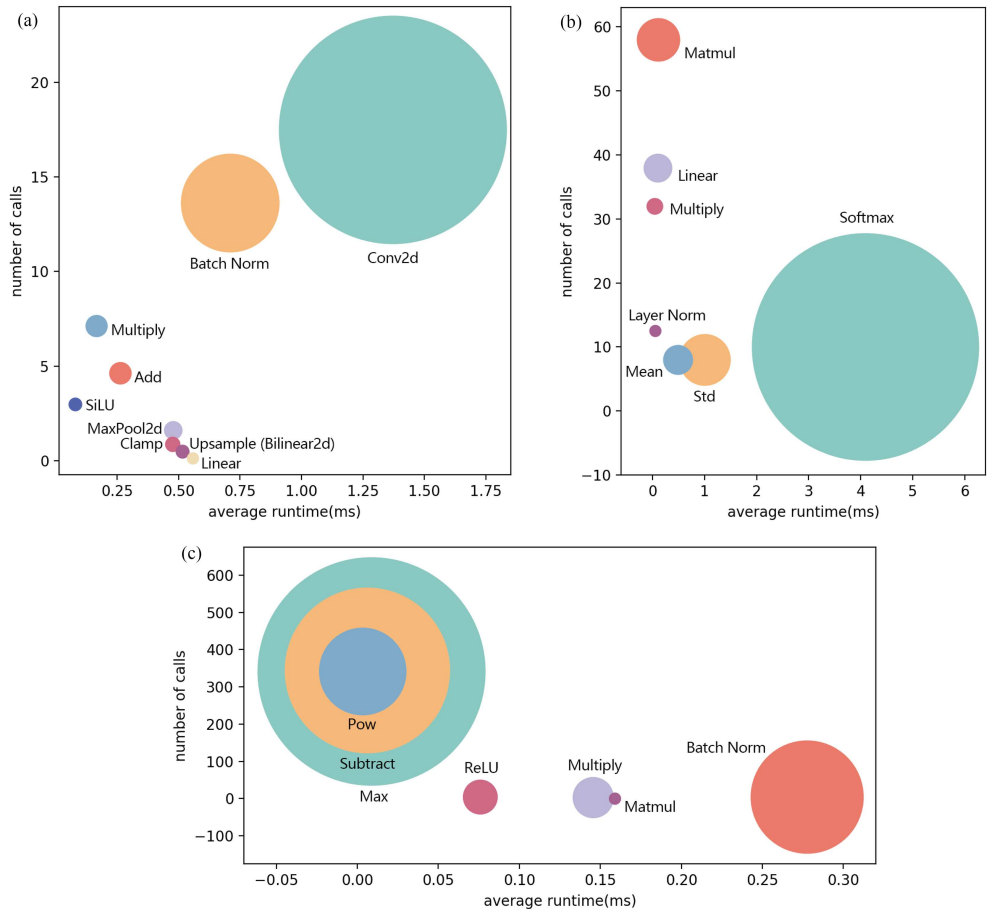


**Figure 2** (Color online) Visualization of the operator-level results of NVIDIA GeForce RTX 3090 on 3 deep learning areas separately given by OpBench. The batch size and input data size are the same as the test in Table 5. The $x$-axis represents the average time of executing the operator once, and the $y$-axis represents the average number of calls of the operator in a single neural network. The area of a point stands for the weighted runtime of the operator, which indicates its importance and overall time consumption. (a) CV (2D); (b) NLP; (c) CV (3D).

have better support for GEMM (general matrix multiplication, the fundamental mathematical component of convolution) than the other GPUs under test. Second, because GPUs provide various interfaces for programmers to invoke, Jittor might have different implementations of the operations for different GPUs, leading to the distinction in performance.

To reveal the importance and actual consumption of the deep learning operators in different areas, we visualize the results of running OpBench on NVIDIA GeForce RTX 3090 in Figure 2. A larger point represents a higher score, which means more importance and a greater share of the overall consumption, and the score (or weight) of each operator is calculated using (2). As seen from the visualization, although convolution plays a key role in CV (2D), the situation differs in CV (3D) and NLP. This type of difference may help developers choose suitable GPUs for their AI tasks, which perform best on the most weighted operators or guide developers on which operators to optimize first.

**Table 6** Comparison between testing operators by weighted runtime (ours) and operation count (AIPerf [14])

| Operator | Weighted runtime (ours) (ms) | | Operation count | |
|---|---|---|---|---|
| | Value | Ratio to minimum | Value | Ratio to minimum |
| Convolution | 16.27 | 60.26 | $4.93 \times 10^{11}$ | 4250.0 |
| Batch normalization | 13.04 | 48.30 | $4.74 \times 10^{9}$ | 40.86 |
| ReLU | 1.87 | 6.93 | $5.81 \times 10^{8}$ | 5.01 |
| Max-pooling | 0.27 | 1.0 | $1.16 \times 10^{8}$ | 1.0 |

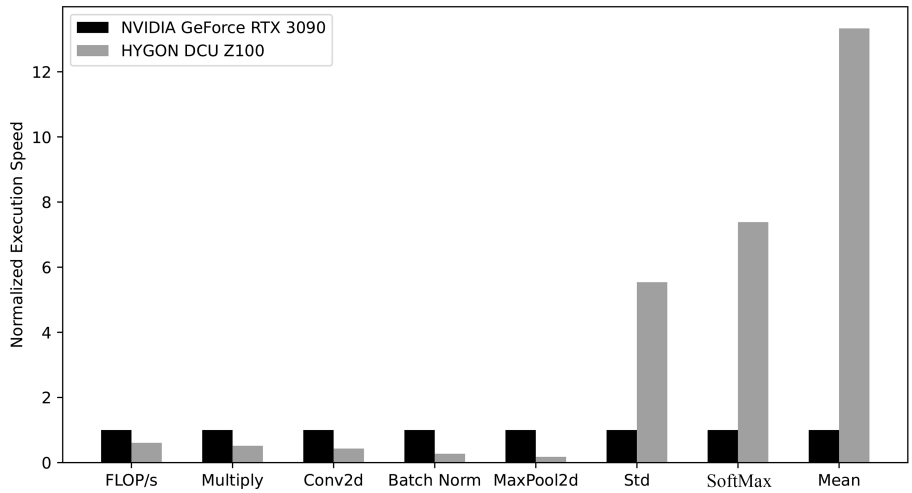


**Figure 3** Comparison between the operation running speeds of NVIDIA GeForce RTX 3090 and HYGON DCU Z100, including the theoretical speed (FLOPS) and the actual performance of different types of operations. The speed of the operations is the inverse of the running time tested by OpBench. Benchmark settings are identical to the test in Table 5. For comparison, the data are normalized with the speeds of RTX 3090 all set to 1.0.

### 5.3 Comparison

AIPerf [14] calculates the operation counts in ResNet50 and measures performance by OPS, which is convenient for assessing HPC systems. However, this type of evaluation metric might not be suitable for single GPUs. To confirm this supposition, we test and calculate the weighted runtime using our method and compare it with the operation count calculated according to the results in AIPerf. As shown in Table 6, a gap exists between the theoretical operation counts and the actual execution time, which might be caused by memory consumption and possible optimizations by GPUs and deep learning frameworks, which can be very complex and diverse. Thus, we choose to test the actual runtime of each operator instead of calculating operation counts.

To further reveal the relationship between FLOPS, the theoretical computing speed, and the actual operator running performance of GPUs, we choose NVIDIA GeForce RTX 3090 and HYGON DCU Z100 for comparison. The FLOPS of RTX 3090 is more than twice that of DCU Z100. We select operators whose performance is not very consistent with the theoretical speed, as shown in Figure 3. The performance varies among operators drastically, and DCU Z100 even considerably outperforms RTX 3090 in some cases. Thus, the actual running time of operators is not completely related to FLOPS, and solid tests are necessary. Together with the results in Figure 2, we find that Conv2d and Batch Norm are among the most time-consuming operators in CV (2D) and CV (3D), which RTX 3090 performs better on, so the scores of RTX 3090 in the two areas are higher than those of DCU Z100 (Table 5). In contrast, DCU Z100 is considerably faster on Std (standard deviation), Mean, and SoftMax, and they are important operators in NLP, leading to higher scores despite lower FLOPS. These results and discoveries cannot be easily reached by other benchmarks such as Benanza [15] because Benanza analyzes GPU performance at the layer level instead of the operator level and cannot test more than one model at a time.

## 6 Conclusion

The rise of deep learning in various areas reveals new challenges in benchmarking GPUs and finding bottlenecks for specific AI purposes. We review the current deep learning benchmarks and microbenchmarks and then explain why operator-level benchmarking is necessary. We provide a relatively complete operator dataset for deep learning, covering up to 200 types of operators commonly seen in neural networks, and conduct a statistical analysis of the operator information of 13 representative networks. We also design an operator-level benchmark, OpBench, allowing users to choose from a given range of models and set the sizes of input according to their demands. The benchmark on four GPU models and three popular deep learning areas is evaluated to show how the operator-level results can help AI and GPU developers access and optimize their systems for specific AI tasks, which cannot be achieved by previous model-based benchmarks. However, we do not address several challenges in OpBench that we would like to leave for future work, including how to test operators when considering operator fusion techniques and whether network structures can be optimized on specific GPUs with the help of operator benchmarks.

**References**


1 Lecun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition. Proc IEEE, 1998, 86: 2278–2324
2 Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks. Commun ACM, 2017, 60: 84–90
3 He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016. 770–778
4 Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. In: Proceedings of Advances in Neural Information Processing Systems, 2017. 30
5 Devlin J, Chang M W, Lee K, et al. BERT: pre-training of deep bidirectional transformers for language understanding. 2018. ArXiv:1810.04805
6 Schulman J, Zoph B, Kim C, et al. ChatGPT: optimizing language models for dialogue. OpenAI, 2022. https://openai.com/blog/chatgpt
7 Rombach R, Blattmann A, Lorenz D, et al. High-resolution image synthesis with latent diffusion models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022. 10684–10695
8 Bianco S, Cadene R, Celona L, et al. Benchmark analysis of representative deep neural network architectures. IEEE Access, 2018, 6: 64270–64277
9 Iandola F N, Han S, Moskewicz M W, et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size. 2016. ArXiv:1602.07360
10 Reddi V J, Cheng C, Kanter D, et al. MLPerf inference benchmark. In: Proceedings of ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), 2020. 446–459
11 Mattson P, Cheng C, Diamos G, et al. MLPerf training benchmark. In: Proceedings of Machine Learning and Systems, 2020. 2: 336–349
12 Cheng P, Xiong Y, Zhao G, et al. SuperBench: a validation and profiling tool for AI infrastructure. Microsoft, 2021. https://microsoft.github.io/superbenchmark/docs
13 Liu W, Anguelov D, Erhan D, et al. SSD: single shot multibox detector. In: Proceedings of the 14th European Conference on Computer Vision, Amsterdam, 2016. 21–37
14 Ren Z, Liu Y, Shi T, et al. AIPerf: automated machine learning as an AI-HPC benchmark. Big Data Min Anal, 2021, 4: 208–220
15 Li C, Dakkak A, Xiong J, et al. Benanza: automatic μbenchmark generation to compute "lower-bound" latency and inform optimizations of deep learning models on GPUs. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2020. 440–450
16 Zhang Y, Owens J D. A quantitative performance analysis model for GPU architectures. In: Proceedings of IEEE 17th International Symposium on High Performance Computer Architecture, 2011. 382–393
17 Gomez-Luna J, Gonzalez-Linares J M, Benitez J I B, et al. Performance modeling of atomic additions on GPU scratchpad memory. IEEE Trans Parallel Distrib Syst, 2012, 24: 2273–2282
18 Abadi M, Agarwal A, Barham P, et al. TensorFlow: large-scale machine learning on heterogeneous distributed systems. 2016. ArXiv:1603.04467
19 Paszke A, Gross S, Massa F, et al. PyTorch: an imperative style, high-performance deep learning library. In: Proceedings of Advances in Neural Information Processing Systems, 2019. 32
20 Hu S-M, Liang D, Yang G-Y, et al. Jittor: a novel deep learning framework with meta-operators and unified graph execution. Sci China Inf Sci, 2020, 63: 222103
21 Chen T, Moreau T, Jiang Z, et al. TVM: an automated end-to-end optimizing compiler for deep learning. 2018. ArXiv:1802.04799
22 Wilper H, Knight R, Cohen J. Understanding the visualization of overhead and latency in NVIDIA nsight systems. NVIDIA, 2020. https://developer.nvidia.com/blog/understanding-the-visualization-of-overhead-and-latency-in-nsight-systems
23 Zhang L, Wahib M, Matsuoka S. Understanding the overheads of launching CUDA kernels. ICPP19, 2019. https://www.hpcs.cs.tsukuba.ac.jp/icpp2019/data/posters/Poster17-abst.pdf
24 Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. 2014. ArXiv:1409.1556
25 Jocher G, Chaurasia A, Stoken A, et al. Ultralytics/yolov5: v7. 0-YOLOv5 SotA realtime instance segmentation. Zenodo, 2022. https://zenodo.org/records/7347926
26 Ren S, He K, Girshick R, et al. Faster R-CNN: towards real-time object detection with region proposal networks. In: Proceedings of Advances in Neural Information Processing Systems, 2015. 28
27 Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015. 3431–3440

28 Qi C R, Su H, Mo K, et al. PointNet: deep learning on point sets for 3D classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017. 652–660
29 Wang Y, Sun Y, Liu Z, et al. Dynamic graph CNN for learning on point clouds. ACM Trans Graph, 2019, 38: 1–12
30 Qi C R, Yi L, Su H, et al. PointNet++: deep hierarchical feature learning on point sets in a metric space. In: Proceedings of Advances in Neural Information Processing Systems, 2017. 30
31 Chen L C, Papandreou G, Schroff F, et al. Rethinking atrous convolution for semantic image segmentation. 2017. ArXiv:1706.05587
32 Vodrahalli K, Bhowmik A K. 3D computer vision based on machine learning with deep neural networks: a review. J Soc Info Display, 2017, 25: 676–694
33 Khurana D, Koli A, Khatter K, et al. Natural language processing: state of the art, current trends and challenges. Multimed Tools Appl, 2023, 82: 3713–3744
34 Chowdhary K R, Chowdhury G G. Natural language processing. In: Proceedings of Fundamentals of Artificial Intelligence, 2020. 603–649