BenchCouncil Transactions on Benchmarks, Standards and Evaluations 2 (2022) 100091

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

IoTBench: A data centrical and configurable IoT benchmark suite☆

Simin Chen [a,b], Chunjie Luo [a,*], Wanling Gao [a], Lei Wang [a]

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

ARTICLE INFO

Keywords:
IoT
Benchmark
Processor
Performance evaluation
Gem5

ABSTRACT

As the Internet of Things (IoT) industry expands, the demand for microprocessors and microcontrollers used in IoT systems has increased steadily. Benchmarks provide a valuable reference for processor evaluation. Different IoT application scenarios face different data scales, dimensions, and types. However, the current popular benchmarks only evaluate the processor's performance under fixed data formats. These benchmarks cannot adapt to the fragmented scenarios faced by processors. This paper proposes a new benchmark, namely IoTBench. The IoTBench workloads cover three types of algorithms commonly used in IoT applications: matrix processing, list operation, and convolution. Moreover, IoTBench divides the data space into different evaluation subspaces according to the data scales, data types, and data dimensions. We analyze the impact of different data types, data dimensions, and data scales on processor performance and compare ARM with RISC-V and MinorCPU with O3CPU using IoTBench. We also explored the performance of processors with different architecture configurations in different evaluation subspaces and found the optimal architecture of different evaluation subspaces. The specifications, source code, and results are publicly available from https://www.benchcouncil.org/iotbench/.

1. Introduction

Internet of Things (IoT) applications are becoming more and more common, such as smart wearable devices, smart cities, smart medical care, and smart homes. With the expansion of the IoT industry, the demand for microcontrollers and microprocessors has increased steadily. Unlike general-purpose processors, which are designed for a wide range of applications, microcontrollers and microprocessors used in IoT systems are application-specific. These processors need to process different data when facing different application scenarios. For example, applications for text sequence analysis mainly deal with one-dimensional data, applications for image processing deal with two-dimensional data, and applications for video processing deal with three-dimensional data.

To realize the function and purpose of the application, it is important to choose the proper microcontroller or microprocessor. Benchmarks are useful for evaluating processors' performance. However, the current benchmarks do not pay much attention to the impact of data scale, data dimension, and data type on processors' performance, so they cannot evaluate the processor's different performances when processing different kinds of data. For example, according to [1], Dhrystone [2] consists of integer-only code, which makes it useful for micro-controllers but far from real-world applications. On the other hand, although CoreMark [3]'s data scale can be adjusted, for standard runs, the data scale (TOTAL_DATA_SIZE) must be set to 2000 bytes.

This paper proposes a new benchmark, namely IoTBench. The IoTBench workloads cover three types of algorithms commonly used in IoT applications: matrix processing, list operation, and convolution. The concept of evaluation subspace is proposed. Considering the different characteristics of the data used in different scenarios, the data space is divided into multiple evaluation subspaces according to data type, data dimension, and data scale. A set of data scales, dimensions, and types defines an evaluation subspace, and the entire data space can be divided into countless evaluation subspaces. In practice, users only need to obtain certain evaluation subspace to run the bench according to the actual scenario requirements. The three parameters of the evaluation subspace can be modified in the definition. Meanwhile, different evaluation indicators are selected to evaluate processors' performance, such as the ratio of iterations to running time (Iterations/Sec), Cycle Per Instruction (CPI), and Cache Miss Rate. We regard IoTBench as a data-centrical configurable benchmark because the main characteristic of IoTBench is that it is built to face real IoT scenarios, and the data scale, data type, and data dimension can be modified.

In the experiments, we first analyze the impact of different data types, data dimensions, and data scales on processor performance. The results show that data type, data dimension, and data scale affect the

☆ This work is supported by the Strategic Priority Research Program of the Chinese Academy of Sciences, Grant No. XDA0320000 and XDA0320300.
* Corresponding author.
E-mail addresses: chensimin22z@ict.ac.cn (S. Chen), luochunjie@ict.ac.cn (C. Luo), gaowanling@ict.ac.cn (W. Gao), wanglei_2011@ict.ac.cn (L. Wang).

https://doi.org/10.1016/j.tbench.2023.100091
Received 11 January 2023; Received in revised form 5 March 2023; Accepted 6 March 2023
Available online 8 March 2023

performance distinctly. That is to say; the data features are important factors for IoT benchmarking. We then compare ARM with RISC-V and MinorCPU with O3CPU using IoTBench. We find that the ARM ISA is more efficient than RISC-V with the same micro-architecture configuration. We explored the performance of processors with different architecture configurations in different evaluation subspaces and found the optimal architecture of different evaluation subspaces.

The contributions of this paper are as follows:

- We design and implement IoTBench, which covers three types of algorithms commonly used in IoT applications: matrix processing, list operation, and convolution. We propose the concept of evaluation subspace, which is defined by a set of data scales, dimensions, and types.
- We analyze the impact of different data types, data dimensions, and data scales on processor performance. The results show that data type, data dimension, and data scale affect the performance distinctly. We also compare ARM with RISC-V and MinorCPU with O3CPU using IoTBench. We find that the ARM ISA is more efficient than RISC-V with the same micro-architecture configuration.
- We explored the performance of processors with different architecture configurations in different evaluation subspaces and found the optimal architecture of different evaluation subspaces.

The rest of this paper is organized as follows. Section 2 presents the related works. Section 3 describes the IoTBench design and implementation. Section 4 shows the experiment settings. Section 5 provides the analysis of the experimental results. Section 6 draws the conclusions and introduces future works.

2. Related work

Benchmark is an evaluation method that has been used in the computer field for a long time. Some of the widely used benchmarks are SPEC CPU2017[4], BigDataBench [5], an industry-standardized compute-intensive benchmark, and TPC-C [6,7], a test benchmark for comparing database platforms running medium-complexity online transaction processing (OLTP) workloads. OLxPBench [8] is a composite HTAP benchmark suite. Supermarq [9] is a scalable quantum benchmark suite. TSB-UAD [10] introduces an end-to-end benchmark suite for univariate time-series anomaly detection. Galli et al. [11] provides a benchmark framework in order to analyze and discuss the most widely used and promising machine/deep learning techniques for fake news detection AIoTBench [12,13] focuses on evaluating the AI inference ability of mobile and embedded devices. MLPerf inference [14] proposes a set of rules and practices to ensure comparability across systems with wildly differing architectures. MLPerf mobile [15] is derivative of MLPerf inference which aims to Benchmark for On-Device AI. ETH Zurich AI Benchmark [16,17] aims to evaluate the AI ability of Android smartphones. These benchmarks are application-level and focus on artificial intelligence. GeekBench [18], and Moby [19] focus on benchmarking mobile phones.

Zhan [20] summarized five types of benchmark tests: measurement standard, standardized data set with defined properties, representative workload, representative data sets, and best practices, which are widely available in multiple disciplines. The benchmark proposed in this thesis is related to representative workloads and measurement standards.

There are three main benchmarks for evaluating the performance of microcontrollers and microprocessors used in IoT systems: MIPS, Dhrystone [2], and CoreMark [3]. MIPS, which is the execution of millions of instructions per second, is the most direct indicator of the processor's computational speed. However, the number and instruction types differ for different instruction set architectures. The time consumed by the execution of different instructions is also different. The value is not representative and comparable even for processors of similar architectures. Because if the instruction sequences are artificially selected, for example, selecting instruction sequences with fewer branches, the measurement results obtained will be different from the actual working and cannot accurately reflect the processor performance. Consequently, this indicator has been gradually replaced by comprehensive benchmarks such as Dhrystone.

Dhrystone is a general-purpose performance benchmark originally developed by Reinhold Weicker in 1984 with the aim of creating a short benchmarking program to measure the performance of computer system programs. Its code is composed mainly of integer operations, string operations, logical decisions, and memory accesses. Dhrystone measures processor performance by testing how many times the processor runs the Dhrystone program per second, using the VAX 11/780 as the reference machine, and reporting the results as a ratio of the number of runs on the machine to be tested to the reference machine in "DMIPS/MHz". Although Dhrystone is more meaningful than MIPS in reflecting processor performance, it is still controversial. In fact, Dhrystone's results are not only influenced by the processor's performance but also affected by factors such as the efficiency of the compiler. This characteristic allows processor manufacturers to obtain a better score by using methods such as optimizing compilers. However, this does not mean that the results of Dhrystone are meaningless. York [21] points out that when the results of Dhrystone are used for comparison, it is necessary to clearly indicate the conditions under which the benchmark is run, such as the version of Dhrystone used, the C libraries used, and so on.

CoreMark was developed by Shay Gal-On of EEMBC in 2009 to replace Dhrystone as the industry standard. CoreMark has become popular, and its features provide a strong competitive advantage. First, its code is small, easy to understand, and has good portability to ensure it runs on all platforms. Second, CoreMark introduces data that cannot be pre-computed at compile time to avoid code elimination due to compilation optimization, making all computations driven by values provided at runtime. Third, CoreMark provides rules on how to run the code and a uniform reporting format to facilitate inter-processor performance comparisons.

At present, the evaluation of IoT processors' performance is generally based on two scores, Dhrystone and CoreMark. However, the above benchmarks' standard scores only reflect the computing speed of the processor under fixed data format and have not considered processors' characteristics of data processed in different IoT applications, so they cannot meet the needs of diverse IoT scenarios. Our IoTBench considers the different data characteristics used in different scenarios; the data space is divided into multiple evaluation subspaces according to data type, dimension, and scale. Table 1 shows the comparison of IoTBench, CoreMark, and Dhrystone.

3. IoTBench

3.1. Workloads and evaluation subspace

IoTbench is comprised of list processing, matrix processing, and convolution. List processing is a kind of basic operator which is widely used in IoT scenarios. When the sensor receives the data, data cleaning and preprocessing are often performed first, and then some simple statistical analysis is carried out. In this process, search and sorting based on lists are widely used. Typical IoT scenarios, e.g., smart cities, smart homes, smartphones, and smart medical care, involve tasks such as voice control, image processing, text processing, and face recognition. Those tasks heavily depend on machine learning and deep learning. As a result, we selected the most basic operators of machine learning and deep learning, namely convolution and matrix processing.

Besides the workload itself, we argue that the data should be considered in IoT benchmarking. Different IoT scenarios face different data dimensions. For example, in scenarios that require text processing, such as natural language processing, the word vector is one-dimensional

Table 1
Comparison of IoTBench, CoreMark, and Dhrystone.

Characteristic	CoreMark	Dhrystone	IoTBench
Written in C language, portable	✓	✓	✓
Provide a single easily reportable score, concise and intuitive	✓	✓	✓
Results are independent from libraries and compilers	✓	✗	✓
Cover convolution algorithm	✗	✗	✓
Various data types can be evaluated	✗	✗	✓
Various data dimensions can be evaluated	✗	✗	✓

"✓" represents that the benchmark has this characteristic, and "✗" represents that the benchmark does not have.

Table 2
Workloads and data space.

Category	Workload	Data type	Data scale	Data dimension
List processing	List search	INT/FLOAT	Any	1/2/3
List processing	List sort	INT/FLOAT	Any	1/2/3
Matrix processing	Matrix add constant	INT/FLOAT	Any	1/2/3
Matrix processing	Matrix multiply constant	INT/FLOAT	Any	1/2/3
Matrix processing	Matrix multiply matrix	INT/FLOAT	Any	1/2/3
Convolution	Convolution	INT/FLOAT	Any	1/2/3

data. The processed image is two-dimensional data in computer vision and image processing scenarios. The processed data is three-dimensional in medical imaging, video processing, and other scenarios. Different IoT scenarios also deal with different data types. For example, in order to save computing and storage resources, AI inference on end devices often compromises between machine precision and prediction accuracy; that is, low precision, such as INT, could be used instead of high precision, such as FLOAT, for calculation. Similarly, the scale of data generated in different scenarios is different. For example, wearable devices need to monitor human body data in real time, which will generate large-scale data.

Based on the above reasons, the entire data space is divided into different evaluation subspaces according to the data scale, data dimension, and data type. A set of data scales, dimensions, and types defines an evaluation subspace, and the entire data space can be divided into countless evaluation subspaces. In practice, users only need to obtain certain evaluation subspace to run the bench according to the actual scenario requirements. The three parameters of the evaluation subspace can be modified in the macro definition. Table 2 shows the workloads and data space details.

3.2. Implementation

The data space for list processing is divided into 2 parts, list items and data items are separately stored in the 2 parts. Data structures used in list processing are shown in Fig. 1(a) and are also similar to CoreMark's. The data to be calculated together with the index is stored in structure list_data. And the structure list_data is indexed by the structure list_node, which makes up the list.

List processing consists of searching and sorting.

- List searching contains two algorithms; one is searching based on value, and the other is based on an index. IoTBench traverses the list and returns all eligible items.
- List sort is realized by merge sort and can sort the list based on value or index. Merge sort is implemented in a non-recursive way. First, every two elements in the list are divided into a group for sorting. After the group is in order, every four elements in the list are divided into a group for sorting. Expand the range of sorting to twice the present size after sorting each time until it reaches the size of the whole list.

The data structure used in convolution is shown in Fig. 1(b). Pointer 'in' points to input data, pointer 'out' points to output data, 'inWidth' refers to the width of input data, 'filter_size' refers to the kernel dimension. If the data is two-dimensional or three-dimensional, 'inHeight' is used to indicate the height of input data. If the data is three-dimensional, 'inDepth' is used to indicate the depth of input data.

One-dimensional convolution, two-dimensional convolution, and three-dimensional convolution are completed in the convolution algorithm.

- One-dimensional convolution means that the kernel slides on the vector according to the stride, and the output value is the sum of the products of the corresponding elements plus the bias.
- Two-dimensional convolution means that the kernel slides in the two-dimensional input space according to stride, and the output value is the sum of the products of the corresponding elements in the window plus the bias.
- Three-dimensional convolution means that the kernel slides in the three-dimensional input space according to stride. 3D matrix multiplication is performed in each window, and the output value is the result obtained above, plus the bias.

The data structure used in matrix processing is shown in Fig. 1(c). 'N' refers to the dimension of the matrix A/B/C. Input data is stored in matrices A and B. Output data is stored in matrix C. Matrix adds constant, matrix multiplies constant, and matrix multiplication is completed in matrix processing.

- The "matrix adds constant" function adds a constant to matrix A, and the result is stored in matrix A.
- The "matrix multiplies constant" function multiplies each item of matrix A by a constant, and the result is restored in matrix C.
- The "matrix multiplication" function multiplies matrix A and matrix B and stores the result in matrix C.

The algorithm's time complexity is shown in Table 3. $OutWidth = (InWidth - filter\_size)/stride + 1$. $OutHeight = (InHeight - filter\_size)/stride + 1$.

4. Experiment

4.1. Gem5 simulator

Gem5 simulator [22] is a modular simulation platform for computer system architecture research, including system-level architecture and processor micro-architecture, which has been widely used in academia, industry, and teaching. Gem5 was originally formed by the merger of M5[23] and GEM [24], where M5 mainly studies CPU simulation, while Gem mainly studies memory systems. Gem5 aims to create a community tool focused on architecture modeling, with flexible modeling and wide availability.

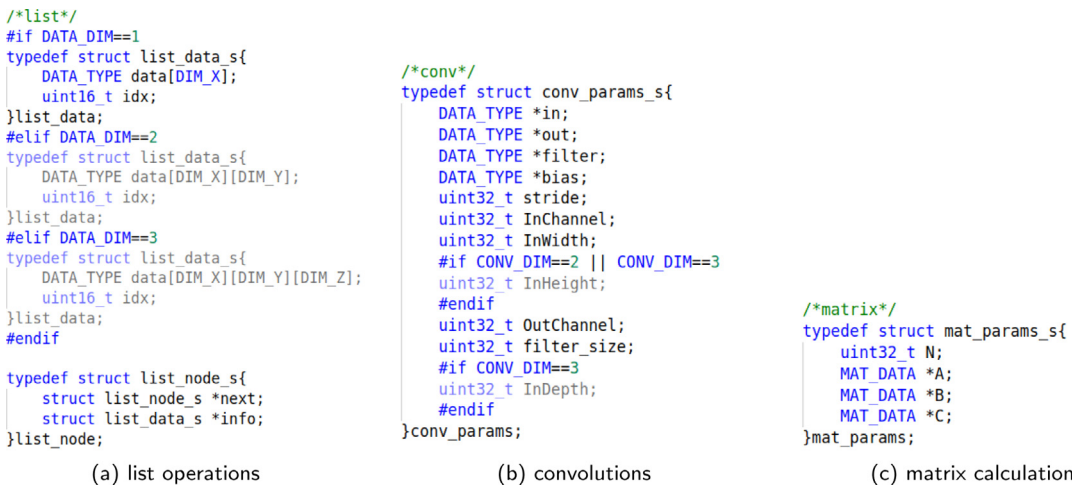

Fig. 1. Data Structure.

Table 3
Time complexity.

Algorithm	Time complexity
List search	$O(n)$
List sort	$O(n \log_2 n)$
One-dimensional convolution	$O(InChannel \cdot OutChannel \cdot filter\_size \cdot OutWidth)$
Two-dimensional convolution	$O(InChannel \cdot OutChannel \cdot filter\_size^2 \cdot OutWidth \cdot OutHeight)$
Three-dimensional convolution	$O(InChannel \cdot OutChannel \cdot filter\_size^2 \cdot OutWidth \cdot OutHeight)$
One-dimensional matrix adds/multiplies constant	$O(n)$
Two-dimensional matrix adds/multiplies constant	$O(n^2)$
Three-dimensional matrix adds/multiplies constant	$O(n^3)$
One-dimensional matrix multiplication	$O(n)$
Two-dimensional matrix multiplication	$O(n^3)$
Three-dimensional matrix multiplication	$O(n^4)$

Gem5 provides a variety of CPU models, system models, storage models, and instruction set architectures. Gem5 provides four CPU models, AtomicSimpleCPU, TimingSimpleCPU, MinorCPU (In Order), and O3CPU (Out of Order). AtomicSimpleCPU is a single IPC (that is, one clock cycle completes one instruction) CPU model that uses atomic operation to access memory. TimingSimpleCPU is similar to AtomicSimpleCPU but uses a sequential memory access model. Minor CPU is a fixed pipeline, but data structure and execution behavior can be changed. The configured in-order CPU. O3 CPU is an out-of-order CPU that is not strictly based on Alpha 21264, and unlike most simulators, Gem5 uses a model that actually executes instructions during the execution phase with high time accuracy. Gem5 also provides two system modes, system call mode (SE) and full system emulation mode (FS). The system call mode can simulate most system calls without simulating the operating system. The full system emulation mode can simulate the complete system, including the operating system, network connection, peripherals, etc. The user needs to provide the compiled Linux kernel and disk image, and the system call mode requires a longer simulation time than required. In addition, Gem5 provides two storage systems, classic mode, and ruby mode. The classic mode inherited from M5 provides a fast and easy-to-configure storage system, while the ruby mode inherited from GEM can accurately simulate storage systems that support different cache coherence protocols. At the same time, Gem5 also supports a variety of instruction set architectures, including ARM, MIPS, Power, SPARC, x86, RISC-V, etc. [22,25].

4.2. Experiment settings

We evaluate IoTBench based on Gem5 Simulator. We compare two common instruction set architectures (ISA) in IoT systems, ARM and RISC-V. In the AArch64 execution state, the A64 instruction set is used, which is a fixed-length 32-bit instruction set. We use RV64GC, an instruction set that includes compressed instructions and general-purpose instructions. We also compare in-order (Minor CPU) processors and out-of-order (O3 CPU) processors according to the way the processor executes instructions. Moreover, we evaluate various L1 and L2 Cache settings using IoTBench. The configuration of the evaluated architectures is shown in Table 4.

Table 4
Configuration of simulator.

Parameter	Value					
ISA	ARM	RISC-V				
CPU MODEL	Minor CPU	O3 CPU				
L1 ICache size	64 kB	32 kB	16 kB	8 kB	4 kB	2 kB
L1 DCache size	64 kB	32 kB	16 kB	8 kB	4 kB	2 kB
L2 Cache size	1024 kB	512 kB	0 kB			

We chose ARM and RISC-V because they are mainstream ISAs used in IoT. Also, in-order and out-of-order are two typical architectures of processors. In addition, we set the cache size according to some commercial processor manufacturers like SiFive. These settings are implemented through the command line according to the documentation of Gem5.

We use the data types INT and FLOAT in the C language; the data dimension is divided into 1 to 3 dimensions; considering that the data scale processed by the microprocessor is generally small, the data is set to two scales, namely 6144 and 12288. By modifying the DATA_SIZE, DATA_TYPE, and DATA_DIM in the macro definition, 12 evaluation subspaces are obtained. Table 5 shows the setting of the evaluation subspace in the experiment.

The cross-compilers used are aarch64-linux-gnu-gcc and riscv64-linux-gnu-gcc. ARM instruction set is Arm64, RISC-V instruction set is RV64GC; Gem5 version is 21.2.1.0. In the Gem5 directory, use SE mode to run the experiments.

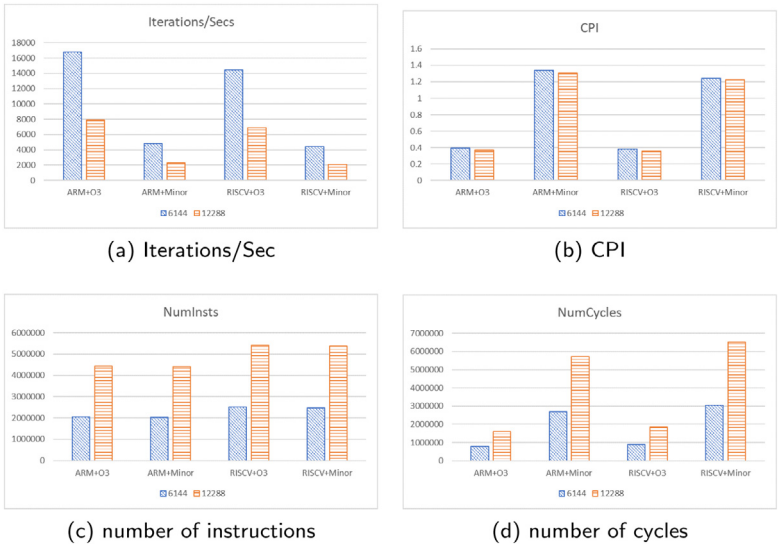

Fig. 2. Results obtained with different data scales.

Table 5
The data format of each evaluation subspace.

Evaluation subspace	DATA_SIZE/Bytes	DATA_DIM	DATA_TYPE
A	6144	1	INT_TYPE
B	6144	2	INT_TYPE
C	6144	2	FP32_TYPE
D	6144	1	FP32_TYPE
E	12288	1	INT_TYPE
F	12288	2	INT_TYPE
G	12288	2	FP32_TYPE
H	12288	1	FP32_TYPE
I	12288	3	FP32_TYPE
J	12288	3	INT_TYPE
K	6144	3	INT_TYPE
L	6144	3	FP32_TYPE

5. Results

In this section, we first analyze the impact of different data types, data dimensions, and data scales on processor performance. The results show that data type, data dimension, and data scale affect the performance distinctly. That is to say, the data features are important factors for IoT benchmarking. We then compare ARM and RISC-V with MinorCPU and O3CPU using IoTBench. We find that the ARM ISA is more efficient with the same micro-architecture configuration than RISC-V. We explore the variation of evaluation subspaces with different architecture configurations and find the different optimal architectures of different evaluation subspaces.

5.1. The impact of data feature

This subsection compares the Iterations/Sec (Iterations/Sec represents how many IoTBench iterations the processor can run per second), CPI (Cycles Per Instructions), number of instructions, and number of cycles for processors when processing data with different scale, type, and dimension, and analyzes the possible causes.

5.1.1. Data scale

As shown in Fig. 2(a), with fixed data dimensions and data types, Iterations/Sec is approximately inversely proportional to the data scale. As shown in Fig. 2(c), the number of instructions is roughly proportional to the data scale. From Fig. 2(b), CPI changes insignificantly with the data scale. It is slightly larger when the data scale is smaller.

5.1.2. Data type

As is known, with the same data size, floating-point operations are slower than integer operations. According to Fig. 3, the value of Iterations/Sec is slightly higher when the data type is int than float32. By analyzing the log of Gem5, the number of floating-point instructions accounts for less than 2% of the total instructions when the data type is float32, while integer instruction account for more than 40% and memory read/write types account for more than 50%.

5.1.3. Data dimension

As shown in Fig. 4(a), the performance is significantly better when the data dimension is one-dimensional than when the data is two-dimensional and three-dimensional. The main reason for this result is that the number of instructions is significantly lower when the data dimension is one-dimensional than when the data dimension is two-dimensional and three-dimensional (Fig. 4(c)). Through Analyzing the log of Gem5, we found that when the data is two-dimensional or three-dimensional, the integer type operations and memory read operations are about three times that of one-dimensional, and the integer multiplication operations are about six times. By analyzing the code, we found that when the data is two-dimensional or three-dimensional, it takes a lot of integer addition and multiplication operations to calculate the array index, resulting in an increase in the number of instructions. As a result, when data is one-dimensional, although the CPI is higher (Fig. 4(b)), the Iterations/Sec is still larger.

5.2. Comparison of ISAs and processor models

From Fig. 5(a), we can see that the performance of the ARM architecture is better than that of the RISC-V architecture overall with the same processor frequency; the performance of the out-of-order processor is significantly better than that of the in-order processor. From Fig. 5(c), the number of instructions of ARM architecture is less than that of RISC-V architecture. The main reason is that the ARM instruction set uses many complex instructions, such as SIMD (Single Instruction Multiple Data). The SIMD computing mode improves the computing performance but also makes the instruction set complex. And RISC-V instruction function is more simple and more basic, so the number of instructions under the RISC-V architecture will be more. Because the RISC-V architecture has more instructions, even though the CPI of the RISC-V architecture is lower than that of the ARM architecture, as shown in Fig. 5(d), the program execution cycle of the processor of the ARM architecture is still less than that of the RISC-V architecture.

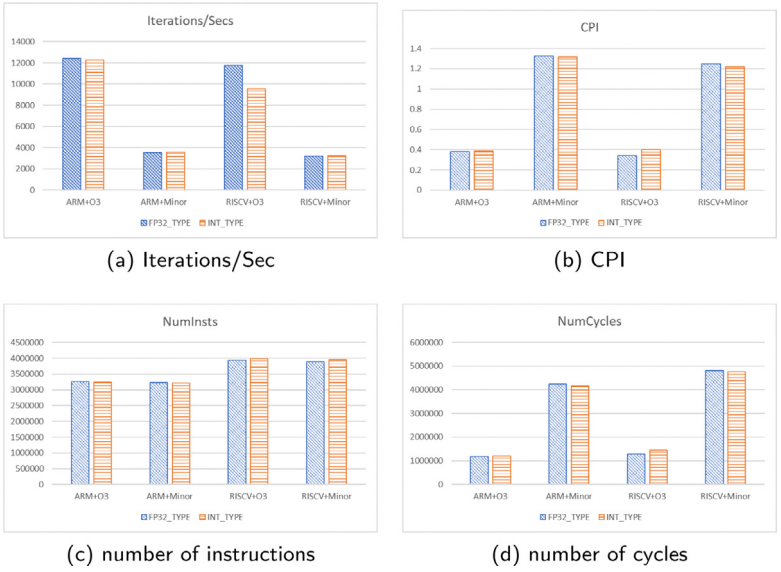

Fig. 3. Results obtained with different data types.

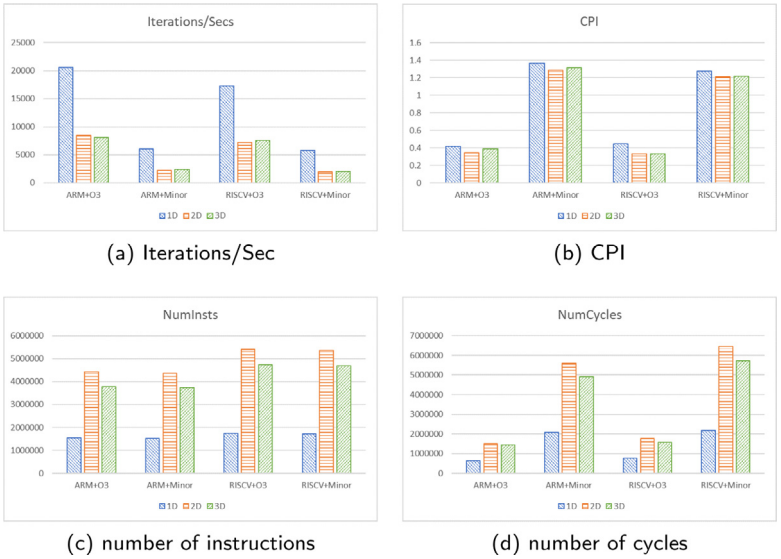

Fig. 4. Results obtained with different data dimensions.

5.3. Analysis and optimization in different evaluation subspace

This subsection analyzes the variation of processor performance with cache sizes under different CPU models and ISAs and finds the optimal cache size configuration for different subspaces.

5.3.1. Subspace A

Taking subspace A as an example, we analyze the impact of cache size on processor performance. The optimal configuration is selected based on Iterations/Sec and cache size under four configurations: ARM instruction set architecture with an out-of-order processor (ARM+O3), ARM instruction set architecture with an in-order processor (ARM+Minor), RISC-V instruction set architecture with an out-of-order processor (RISC-V+O3), and RISC-V instruction set architecture with an in-order processor (RISC-V+Minor).

When the L1 DCache is set to 16 kB, and the L1 ICache is set to 8 kB, the processor shows the same performance as both caches are set to 64 kB. L2 Cache is not set in the optimal configuration. Because the L2 Cache size is larger than the data size and the test time is short, if L2 Cache is used, the cold start will take up part of the time, making the performance drop.

Table 6 shows the configuration corresponding to the horizontal coordinate numbers in the figures below. As shown in Figs. 6(a) and 6(b), Iterations/Sec and CPI show roughly opposite trends with cache size. The increase in CPI can be attributed to the increase in miss rate due to the decrease in L1 Cache size, which causes more processor stalls. With a stable instruction count, a higher CPI implies an increase in instruction execution time, which leads to a decrease in the Iterations/Sec value. The Iterations/Sec value at numbers 5–7 increases slightly and then decreases, opposite to the L2 Cache miss rate trend. After the L1 ICache is reduced to 16 kB, the L1 ICache miss rate increases, and the number of L2 Cache access increases, resulting in a decrease in the percentage of L2 Cache cold misses and a decrease in the average miss rate. Similarly, when the L1 Cache is increased to 64 kB, the number of accesses to the L2 Cache decreases, and the average miss rate increases, causing a performance loss. Performance improves at configuration number 14 because the L2 Cache setting is eliminated here, and there is no time consumption caused by L2 Cache miss, so the performance improves. The performance drops from number 19 to

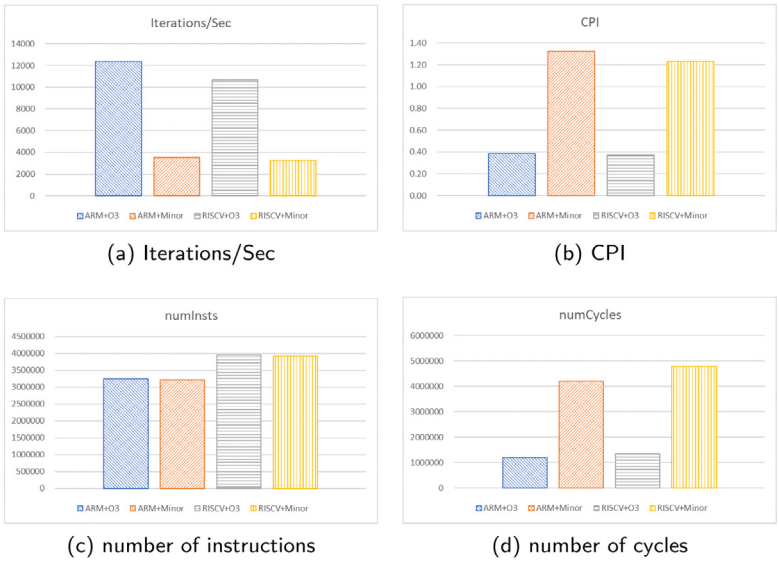

Fig. 5. Results obtained with different ISAs and CPU models.

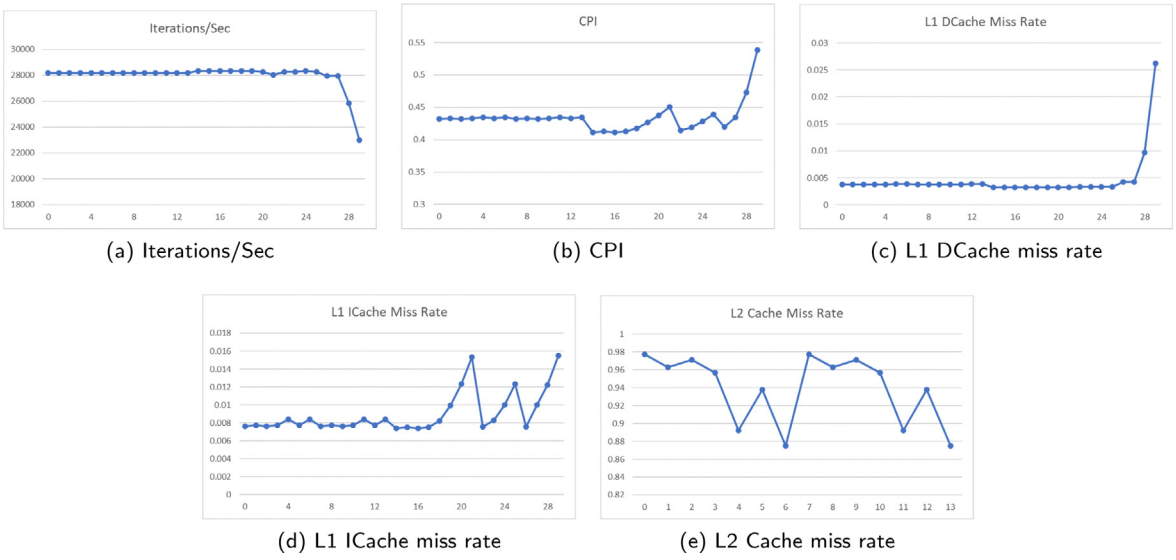

Fig. 6. Result of ARM+O3 in subspace A.

number 20 because the L1 ICache drops at this point. Combined with Fig. 6(d), we can see that the L1 ICache miss rate rises significantly here, and the same is true for numbers 23 to 25. The performance drops significantly from number 27 to number 29, when both the L1 DCache and L1 ICache drop from 8 kB to 2 kB, and both the L1 ICache miss rate and L1 DCache miss rate increase significantly.

According to Fig. 6(c), when the L1 DCache is set to 8 kB and above, its size change does not greatly affect the L1 DCache miss rate. However, below 8 kB, the miss rate varies significantly with the size of the L1 DCache because the tested data size is 6144 bytes. As shown in Fig. 6(d), when the L1 ICache is set to 16 kB and above, its size variation does not have much effect on the miss rate. However, below 16 kB, the miss rate varies significantly with size. Analyzing the first 13 sets of data with L2 Cache to get Fig. 6(e), the L2 miss rate decreases significantly at numbers 4/6/11/13, which are all configurations with L1 ICache of 16 kB. Combined with Fig. 6(d), it can be seen that the L1 ICache miss rate increases, and the number of accesses to the L2 Cache increases from the time the L1 ICache drops to 16 kB.

The performance is optimal when L1 DCache is set to 16 kB, and L1 ICache is set to 8 kB. According to Figs. 6 and 7, the results for each configuration under ARM+Minor are roughly in line with the trend of the results under ARM+O3 with cache size. However, the average value of Iterations/Sec is lower than when using the out-of-order processor, the CPI is higher, the L1 DCache miss rate and L1 ICache miss rate is significantly lower, and the L2 Cache miss rate does not change much.

When the L1 DCache is set to 16 kB and the L1 ICache is set to 8 kB, the processor achieves the best performance, the same as when both caches are set to 64 kB. According to Figs. 8 and 6, the trend of each test result with cache size under RISC-V+O3 is similar to that under ARM+O3. The average value of Iterations/Sec is lower than that of ARM architecture, CPI is higher, and the cache miss rate does not change significantly. Comparing Figs. 8(a) and 6(a) with Figs. 8(b) and 6(b), we can find that when the L1 Cache size decreases from 8 kB to 4 kB and from 4 kB to 2 kB, the performance degradation of the RISC-V group slows down, but the performance degradation of the ARM group intensifies. According to Fig. 8(c) and 6(c), the rate of increase of L1 DCache miss rate in the above interval is significantly smaller for RISC-V architecture than for ARM architecture, which may be the reason for the above phenomenon.

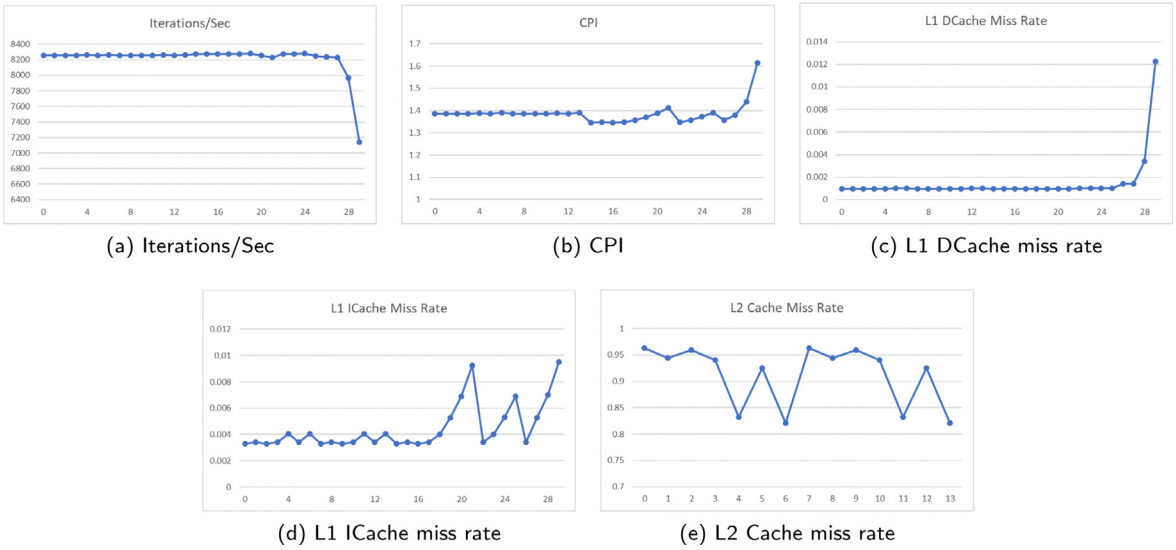

Fig. 7. Result of ARM+Minor in subspace A.

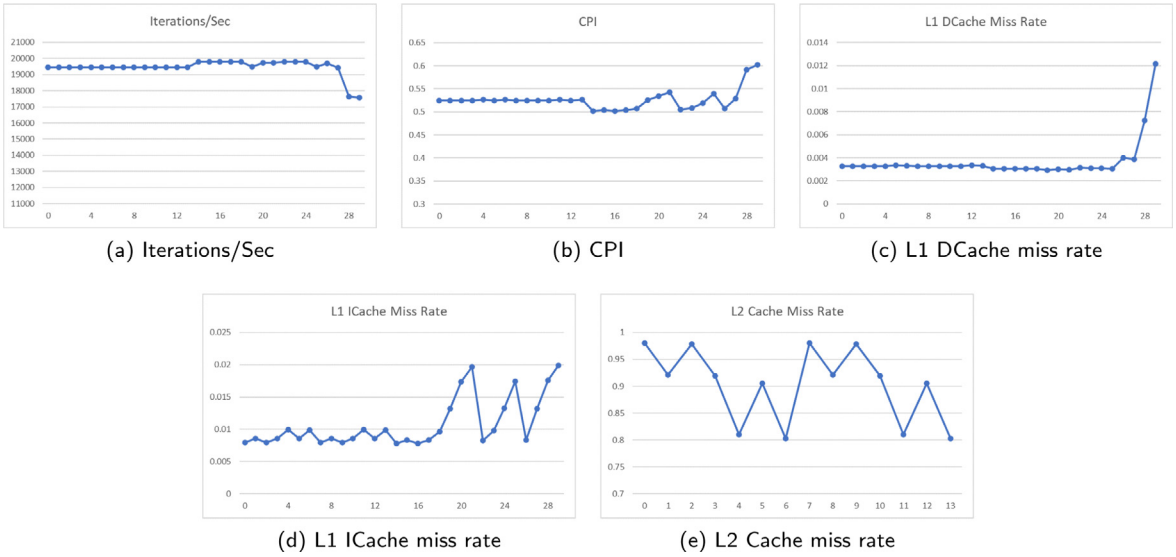

Fig. 8. Result of RISC-V+O3 in subspace A.

The performance is optimal when both caches are set to 16 kB. Comparing Figs. 9 and 8, we can see that the Iterations/Sec value decreases significantly, the CPI increases, the miss rate of both L1 DCache and L1 ICache decreases significantly, and the L2 Cache miss rate does not change significantly. Comparing Fig. 9 with Fig. 7, Iterations/Sec is relatively lower in RISC-V architecture, and CPI is also lower than the ARM architecture. The variation of performance with cache size is smaller in RISC-V architecture than in ARM architecture.

5.3.2. Other subspaces

Tables 7 summarize the optimal configurations for the 12 evaluation subspaces respectively.

The trend of processor performance with cache size varies in different evaluation subspaces, and the configuration with the best performance also varies in each evaluation subspace. Existing benchmarks such as CoreMark are tested under a fixed data size, data type, and data dimension and give a single performance score. However, IoTBench can give the final performance score under different data sizes, types, and dimensions. Users can modify the above parameters to test under what data characteristics the processor will get better performance. Users can optimize the processor for a given data space, taking into account the needs of a particular application area. It is also possible to obtain the impact of the optimization of a certain configuration on the processing of certain characteristic data. This is useful for manufacturers to produce processors for specific application areas and for users to select processors that are better suited to their data processing needs.

6. Conclusion

This paper constructs a benchmark (IoTBench) for evaluating the performance of processors in IoT scenarios. The benchmark divides the data space into multiple evaluation subspaces according to data scale, type and dimension, which aligns with IoT applications' fragmented nature. We use the Gem5 simulator to simulate processors with various configurations and use IoTBench to test the performance of each processor in different evaluation subspaces. We analyze the impact of different data types, data dimensions, and data scales on processor performance. The results show that data type, data dimension, and data scale affect the performance distinctly. The comparison shows that the ARM ISA is generally more efficient than RISC-V. The 12 evaluation

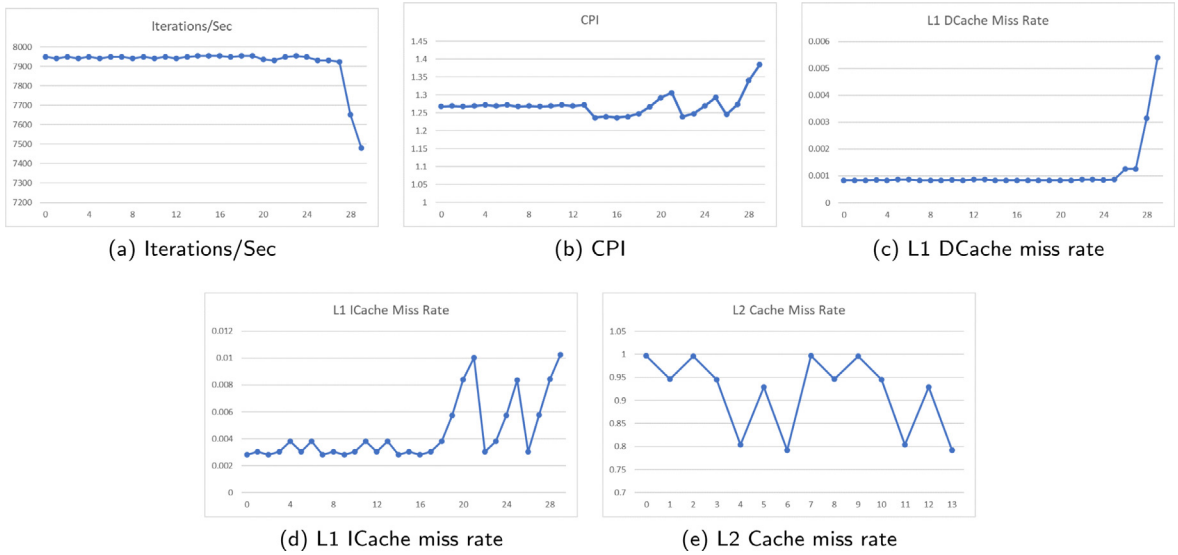

Fig. 9. Result of RISC-V+Minor in subspace A.

Table 6
The configuration corresponding to the number.

Number	L2 Cache/kB	L1 DCache/kB	L1 ICache/kB
0	1024	64	64
1	1024	64	32
2	1024	32	64
3	1024	32	32
4	1024	32	16
5	1024	16	32
6	1024	16	16
7	512	64	64
8	512	64	32
9	512	32	64
10	512	32	32
11	512	32	16
12	512	16	32
13	512	16	16
14	0	64	64
15	0	64	32
16	0	32	64
17	0	32	32
18	0	32	16
19	0	32	8
20	0	32	4
21	0	32	2
22	0	16	32
23	0	16	16
24	0	16	8
25	0	16	4
26	0	8	32
27	0	8	8
28	0	4	4
29	0	2	2

Table 7
Optimal configuration for subspace A-L.

Subspace	ISA	CPU Model	L1 DCache/kB	L1 ICache/kB	Iterations/s
A	ARM	O3	16	8	28328.61
B	ARM	O3	16	16	11695.91
C	ARM	O3	16	8	12121.21
D	ARM	O3	16	16	28571.43
E	ARM	O3	32	32	13386.88
F	ARM	O3	32	8	5173.31
G	ARM	O3	32	16	5181.35
H	ARM	O3	32	8	13458.95
I	ARM	O3	32	32	5837.71
J	ARM	O3	32	8	5621.14
K	ARM	O3	16	32	10548.52
L	ARM	O3	16	16	10964.91
A	RISC-V	O3	16	8	19801.98
B	RISC-V	O3	32	16	10718.11
C	RISC-V	O3	16	8	11013.22
D	RISC-V	O3	16	16	27247.96
E	RISC-V	O3	32	8	10070.49
F	RISC-V	O3	32	8	3915.43
G	RISC-V	O3	32	64	4738.13
H	RISC-V	O3	32	32	13192.61
I	RISC-V	O3	32	8	5208.33
J	RISC-V	O3	32	32	5130.84
K	RISC-V	O3	16	8	10152.28
L	RISC-V	O3	32	16	10214.50

subspaces obtained through IoTBench show that the same processor configuration performs differently in different evaluation subspaces, and the processor configurations corresponding to the optimal performance in different evaluation subspaces are also different. Users can set the data dimension, type, and scale of IoTBench to test different processors according to their needs to obtain processor optimization that better meets their requirements.

There are several improvements that would be made in future works. First, the experiments in this paper are conducted in the system call mode of gem5, and more experiments could be conducted in the full-system simulation mode. Second, more modules can be added to the processor configuration to test the impact of different configurations on the processors' performance. Third, More representative workloads in IoT scenarios can be selected. Fourth, the relationship between different configurations and indicators in different evaluation subspaces can be further explored.

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.

Acknowledgments

This work is supported by the Strategic Priority Research Program of the Chinese Academy of Sciences, Grant No. XDA0320000 and XDA0320300. The authors are very grateful to anonymous reviewers for their insightful feedback.

References

[1] A.R. Weiss, Dhrystone benchmark, in: History, Analysis, "Scores" and Recommendations, White Paper, ECL/LLC, 2002.

[2] R.P. Weicker, Dhrystone: a synthetic systems programming benchmark, Commun. ACM 27 (10) (1984) 1013–1030.

[3] E. Consortium, et al., Coremark, 2009.

[4] J. Bucek, K.-D. Lange, J. v. Kistowski, SPEC CPU2017: Next-generation compute benchmark, in: Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, 2018, pp. 41–42.

[5] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, et al., Bigdatabench: A big data benchmark suite from internet services, in: 2014 IEEE 20th International Symposium on High Performance Computer Architecture, HPCA, IEEE, 2014, pp. 488–499.

[6] T.P.P. Council, TPC benchmark c standard specification revision 5.2, 1996, http://www.tpc.org/tpcc/spec/tpcc_current.pdf.

[7] W. Kohler, A. Shah, F. Raab, Overview of TPC benchmark c: The order-entry benchmark, in: Transaction Processing Performance Council, Technical Report, 1991.

[8] G. Kang, L. Wang, W. Gao, F. Tang, J. Zhan, OLxPBench: Real-time, semantically consistent, and domain-specific are essential in benchmarking, designing, and implementing HTAP systems, in: 2022 IEEE 38th International Conference on Data Engineering, ICDE, IEEE, 2022, pp. 1822–1834.

[9] T. Tomesh, P. Gokhale, V. Omole, G.S. Ravi, K.N. Smith, J. Viszlai, X.-C. Wu, N. Hardavellas, M.R. Martonosi, F.T. Chong, Supermarq: A scalable quantum benchmark suite, in: 2022 IEEE International Symposium on High-Performance Computer Architecture, HPCA, IEEE, 2022, pp. 587–603.

[10] J. Paparrizos, Y. Kang, P. Boniol, R.S. Tsay, T. Palpanas, M.J. Franklin, TSB-UAD: an end-to-end benchmark suite for univariate time-series anomaly detection, Proc. VLDB Endow. 15 (8) (2022) 1697–1711.

[11] A. Galli, E. Masciari, V. Moscato, G. Sperlí, A comprehensive Benchmark for fake news detection, J. Intell. Inf. Syst. 59 (1) (2022) 237–261.

[12] C. Luo, F. Zhang, C. Huang, X. Xiong, J. Chen, L. Wang, W. Gao, H. Ye, T. Wu, R. Zhou, et al., AIoT bench: towards comprehensive benchmarking mobile and embedded device intelligence, in: International Symposium on Benchmarking, Measuring and Optimization, Springer, 2018, pp. 31–35.

[13] C. Luo, X. He, J. Zhan, L. Wang, W. Gao, J. Dai, Comparison and benchmarking of ai models and frameworks on mobile devices, 2020, arXiv preprint arXiv:2005.05085.

[14] 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.

[15] V. Janapa Reddi, D. Kanter, P. Mattson, J. Duke, T. Nguyen, R. Chukka, K. Shiring, K.-S. Tan, M. Charlebois, W. Chou, et al., MLPerf mobile inference benchmark: An industry-standard open-source machine learning benchmark for on-device AI, Proc. Mach. Learn. Syst. 4 (2022) 352–369.

[16] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, L. Van Gool, Ai benchmark: Running deep neural networks on android smartphones, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018.

[17] A. Ignatov, R. Timofte, A. Kulik, S. soo Yang, K. Wang, F. Baum, M. Wu, L. Xu, L.V. Gool, AI benchmark: All about deep learning on smartphones in 2019, in: 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), 2019, pp. 3617–3635.

[18] PrimateLabs, geekbench, https://www.geekbench.com/.

[19] Y. Huang, Z. Zha, M. Chen, L. Zhang, Moby: A mobile benchmark suite for architectural simulators, in: 2014 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, IEEE, 2014, pp. 45–54.

[20] J. Zhan, Call for establishing benchmark science and engineering, BenchCouncil Trans. Benchmarks Stand. Eval. 1 (1) (2021) 100012, http://dx.doi.org/10.1016/j.tbench.2021.100012, URL https://www.sciencedirect.com/science/article/pii/S2772485921000120.

[21] R. York, Benchmarking in context: Dhrystone, 2002, ARM, March.

[22] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, et al., The gem5 simulator, ACM SIGARCH Comput. Archit. News 39 (2) (2011) 1–7.

[23] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, S.K. Reinhardt, The M5 simulator: Modeling networked systems, Ieee Micro 26 (4) (2006) 52–60.

[24] M.M. Martin, D.J. Sorin, B.M. Beckmann, M.R. Marty, M. Xu, A.R. Alameldeen, K.E. Moore, M.D. Hill, D.A. Wood, Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset, ACM SIGARCH Comput. Archit. News 33 (4) (2005) 92–99.

[25] A. Butko, R. Garibotti, L. Ost, G. Sassatelli, Accuracy evaluation of gem5 simulator system, in: 7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), IEEE, 2012, pp. 1–7.