Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

3CBench: A Unified Benchmarking Framework for the Computing Capacity of Heterogeneous AI Clusters

Weixing Zhang
Zhejiang Lab
Xizhi Wang
Zhejiang Lab
Jun Yan
Zhejiang Lab
Jiasun Feng
Zhejiang Lab
Yiying Liu
Zhejiang Lab
Haiyan Li
Zhejiang Lab
Qun Chen
Zhejiang Lab
Zhe Tang
Zhejiang Lab
Xin Cui
Zhejiang Lab
Fei Yang
yangf@zhejianglab.org

Zhejiang Lab

Research Article

Keywords: Benchmarking Framework, Evaluation Metrics, Automated Evaluation, Scalability, Computing Capacity Evaluation, Metric Calculation

Posted Date: October 9th, 2025

DOI: https://doi.org/10.21203/rs.3.rs-7602328/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

3CBench: A Unified Benchmarking Framework for the Computing Capacity of Heterogeneous AI Clusters

Weixing Zhang[1†], Xizhi Wang[1†], Jun Yan[1], Jiasun Feng[1], Yiying Liu[1], Haiyan Li[1], Qun Chen[1], Zhe Tang[1], Xin Cui[1], Fei Yang[1*]

[1] Zhejiang Lab, Hangzhou, 100190, Zhejiang, China.

*Corresponding author(s). E-mail(s): yangf@zhejianglab.org;
†These authors contributed equally to this work.

Abstract

The rapid evolution of Artificial Intelligence has driven the demand for extensive computational resources and the deployment of AI tasks across heterogeneous computing platforms. However, existing benchmarking systems face several challenges, including limited compatibility with diverse hardware, insufficient support for varied deep learning frameworks and tasks, and a lack of comprehensive evaluation metrics for the computing capacities. To address these issues, we propose 3CBench, a unified benchmarking framework designed for heterogeneous AI clusters. Featuring a modular architecture encompassing environment management, task execution, and metrics analysis, 3CBench provides automated workflows and ensures seamless compatibility with diverse GPU architectures and deep learning frameworks. It provides a comprehensive evaluation metrics system to rigorously assess computational performance and stability across both transformer-based large language models and convolutional neural networks, thereby covering dominant deep learning architectures. Extensive experiments demonstrate 3CBench's scalability on heterogeneous AI clusters, compatibility with various deep learning frameworks and tasks, and the support for a wide range of applications. Additionally, 3CBench aids in problem diagnosis during the development process of GPU vendors. These features establish 3CBench as a robust tool for benchmarking, optimization, and system-level evaluation in heterogeneous AI clusters.

Keywords: Benchmarking Framework, Evaluation Metrics, Automated Evaluation, Scalability, Computing Capacity Evaluation, Metric Calculation

1 Introduction

Recently, Artificial Intelligence (AI) has undergone significant progress, notably characterized by the exponential growth in model sizes; for instance, Llama 3 [1] has 405 billion parameters and Qwen [2] has 72 billion parameters. This trend aligns with scaling laws, indicating that larger models tend to deliver superior performance when trained on adequately sized datasets [3, 4]. Consequently, the demand for extensive data and substantial computational resources has surged. For example, pre-training Llama 65B model [5] requires 2048 NVIDIA A100 GPUs to take approximately 21 days. Such extensive compute demand necessitates the deployment of AI models on heterogeneous computing platforms [6]. These platforms are designed to leverage diverse hardware to optimize performance and resource allocation for large-scale training and inference tasks. In addition, specific frameworks (e.g. DeepSpeed [7] and Megatron [8]) and optimization strategies are implemented specifically with respect to GPU architectures. Therefore, establishing an accurate and fair computing benchmarking is crucial for assessing computational costs and identifying bottlenecks in model training and inference, thereby facilitating the configuration of particular AI tasks to maximize the utilization of computing resources.

Developing fair and accurate computing capacity benchmarking is still an under explored problem. Prior works like NVIDIA Nsight [9] focus on profiling without providing comprehensive statistical metrics for deeper analysis. There are benchmark suites such as PyTorch profiler [10] which are tightly integrated within the frameworks, and are not applicable to others. Furthermore, existing benchmarking often lacks compatibility with various heterogeneous computing environments [11]. It poses challenges to develop an unified computing benchmarking system. One primary obstacle is the diverse architectures of the underlying AI clusters. While NVIDIA GPUs are widely recognized as the standard for deep learning tasks, alternative hardware solutions from vendors such as Huawei Ascend [12], Cambricon [13], Kunlunxin [14] and Moore Threads [15] are gaining traction. It is crucial for benchmarking to achieve compatibility across diverse hardware platforms. Secondly, the diversity of AI tasks introduces additional complexity to the software stack, as various tasks are built on various frameworks that accommodate different optimization algorithms and parallelization strategies. This introduces additional variables into benchmarking, thus complicating the benchmarking process. Thirdly, determining the most relevant and comprehensive set of computing evaluation metrics that effectively reflect the bottlenecks of AI tasks is also critical. Without a standardized set of metrics, comparisons between diverse GPU architectures can become ambiguous.

In this paper, we present an unified **C**omputing **C**apacity benchmarking system of heterogeneous AI **C**lusters, **3CBench**, specifically designed to tackle the aforementioned key challenges. Our goal is to provide an end-to-end, scalable workflow for a diverse array of AI tasks across heterogeneous computing environments, accompanied by reliable metrics that facilitate automated configurations for optimal efficiency. As shown in Figure 1, 3CBench is structured into three core components: environment management, task execution, and metrics analysis. Such modular architecture not only streamlines the management of the distinct stages of AI workflows but also ensures consistency and scalability throughout the benchmarking process. Through

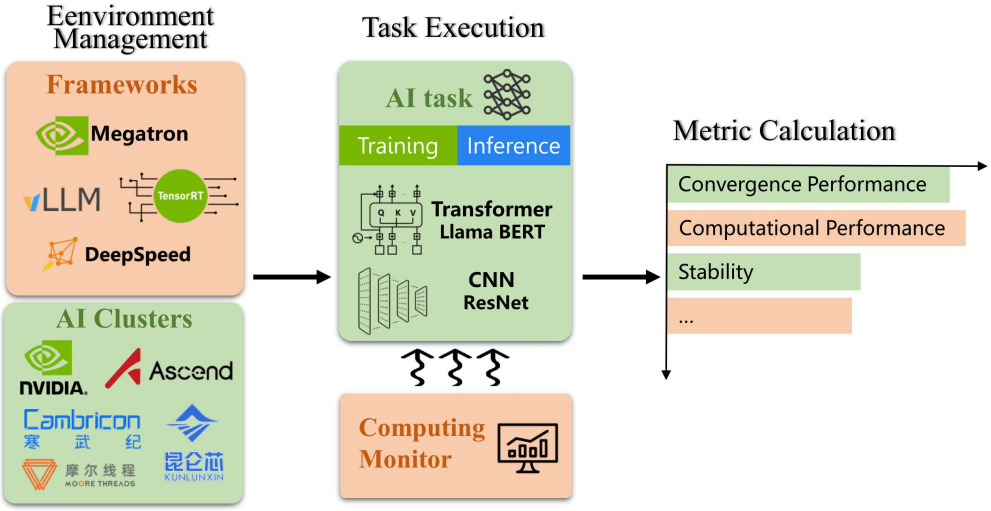


Fig. 1: 3CBench benchmarking framework for the computing capacity of heterogeneous AI clusters. 3CBench features a modular framework that enables streamlined workflows across diverse GPU architectures, machine learning frameworks, and AI tasks. This benchmarking suite also accommodates a comprehensive metrics system for the computing capacity, facilitating reliable evaluation on varied hardware and software configurations.

strategic collaborations with GPU vendors, 3CBench offers pre-configured runtime environments optimized for specific AI frameworks, significantly minimizing deployment overheads. By standardizing these environments, we enhance the reproducibility of benchmarking results, allowing users to reliably assess performance across different tasks and frameworks. Moreover, 3CBench is versatile, enabling cross-domain benchmarking for large language models (LLMs) based on transformer architectures as well as convolutional neural networks (CNNs) for vision applications. Additionally, we have developed a comprehensive evaluation metrics system that rigorously examines the interplay between task types, hyperparameters and underlying configurations. 3CBench not only aids in identifying performance bottlenecks but also provides a standardized methodology for optimizing resource utilization, ultimately enhancing the productivity of machine learning workflows. We performed extensive experiments with 3CBench to ensure its compatibility across heterogeneous hardware and AI tasks, demonstrating its adaptability to diverse computational environments. 3CBench provides core computing evaluation metrics offering a comprehensive assessment for optimizing model deployment and resource allocation on large-scale systems.

Our contribution are summarized as follows:

- Development of an unified computing capacity benchmarking for heterogeneous environments, enabling computing evaluation under diverse AI clusters.
- Design of a modular architecture for the benchmark suite, providing an end-to-end, scalable workflow for assessing a wide range of AI tasks and frameworks.

- Proposing a comprehensive system of computing evaluation metrics, facilitating a fair and holistic evaluation of both training and inference.
- The framework has been tested and demonstrated on a variety of AI clusters, including anonymous devices other than NVIDIA GPUs, highlighting its capability to address real-world challenges.

2 Background and Related Work

2.1 Chips for AI computing

The AI chip ecosystem remains GPU-centric but is rapidly diversifying through heterogeneous architectures and specialized accelerators. General-purpose GPUs reduce graphics-oriented functions to maximize parallel computation, making them indispensable for model training and inference. NVIDIA [16] dominates this domain through its CUDA ecosystem and successive architectures—Volta (V100), Ampere (A100), and Hopper (H100)—which continuously improve tensor operations, memory bandwidth, and scalability. At the same time, competitors are advancing: AMD's MI GPUs [17], built on CDNA architectures, rival NVIDIA through high memory bandwidth and performance-per-watt efficiency; Intel's Xe GPUs [18] target cloud-based AI workloads; Google's TPU [19] clusters exemplify domain-specific design for large-scale tensor operations. Meanwhile, Chinese vendors are emerging as influential players: Huawei's Ascend series [12] emphasizes throughput and efficiency across data centers and edge devices, while Cambricon [13] and Kunlunxin [14] develop accelerators that extend AI capabilities across both cloud and localized applications. This growing diversity of architectures highlights a fundamental challenge: the heterogeneity of AI chips makes it increasingly difficult to establish consistent and fair evaluation across platforms. Addressing this challenge through unified compute assessment is the central objective of this work.

2.2 Performance Metrics for AI Chips

The evaluation of AI accelerators has attracted considerable attention, with researchers proposing a wide variety of metrics to capture different aspects of performance. Conventional measures such as TFLOPS (Tera Floating Point Operations Per Second) provide peak computational throughput, while memory bandwidth, scalability, and parallel processing capacity reflect critical hardware constraints [20, 21]. Derived indicators, including Model FLOPs Utilization (MFU) and Hardware FLOPs Utilization (HFU), offer finer-grained insights into the effective use of compute resources, and energy efficiency metrics such as performance per watt address sustainability concerns. In addition, profiling tools and frameworks such as HPCToolkit [22] and Nsight Systems [9] emphasize different perspectives, ranging from instruction-level throughput to trace-based temporal bottlenecks [23].

Despite this diversity, current evaluation remains fragmented. Existing approaches often focus on isolated dimensions, failing to provide a unified methodology that integrates computational efficiency, communication overhead, and power-performance trade-offs into a coherent framework. The challenges encountered in the development

of large language models, exemplified by works like PaLM [24], critically underscore the absence of consistent categorization across performance metrics. This fragmentation severely impedes fair comparisons between heterogeneous accelerators and limits their applicability in guiding large-scale AI deployments. Therefore, while many metrics exist, what is still missing is a comprehensive evaluation system that organizes and unifies these measures into a systematic framework. Establishing such a framework is the central motivation of this paper.

2.3 Benchmarking for AI Workloads

The rapid proliferation of AI accelerators underscores the need for standardized frameworks capable of evaluating computational performance across heterogeneous platforms. Existing efforts such as MLPerf [11] and FlagPerf [25] have established a widely accepted foundation for benchmarking both training and inference, enabling fair comparisons across hardware and software ecosystems. Recent updates have expanded to cover large language models and broader AI workloads, reflecting the shift toward generative AI. Nevertheless, these benchmarks often remain constrained by vendor-specific configurations and limited adaptability when applied to diverse or emerging architectures. Traditional tools, such as GEMM-based frameworks [26], primarily measure peak matrix multiplication performance, thereby capturing only theoretical computational potential and failing to adequately reflect the performance characteristics of server-grade accelerators that dominate large-scale AI training and inference.

To address these limitations, we propose 3CBench, an application-driven benchmarking suite designed around multi-vendor hardware platforms, mainstream training frameworks [27], and a diverse set of deep learning models. Unlike prior efforts, 3CBench is explicitly tailored for large-scale heterogeneous clusters, ensuring full compatibility with the computational and communication demands of modern AI training and inference. The framework decouples benchmarking procedures from vendor-specific tuning, automates log collection and report generation, and incorporates both well-established architectures such as ResNet [28] and BERT [29] as well as emerging large-scale models like Llama [5]. By emphasizing real-world performance, computational efficiency, and scalability across heterogeneous environments, 3CBench delivers a unified and extensible methodology for consistent, reproducible, and fair evaluation of contemporary AI accelerators.

3 Methodology

3.1 Overview of Our Framework

We have meticulously designed a comprehensive evaluation framework, as illustrated in Figure 2. This framework encompasses a holistic set of evaluation metrics, an easily extensible benchmark framework, automated evaluation tool-kits, and extensive support for various computing environments.

Complete Evaluation Metrics: To ensure that the evaluation results of computing clusters are comprehensive, objective, and fair, we have constructed a holistic

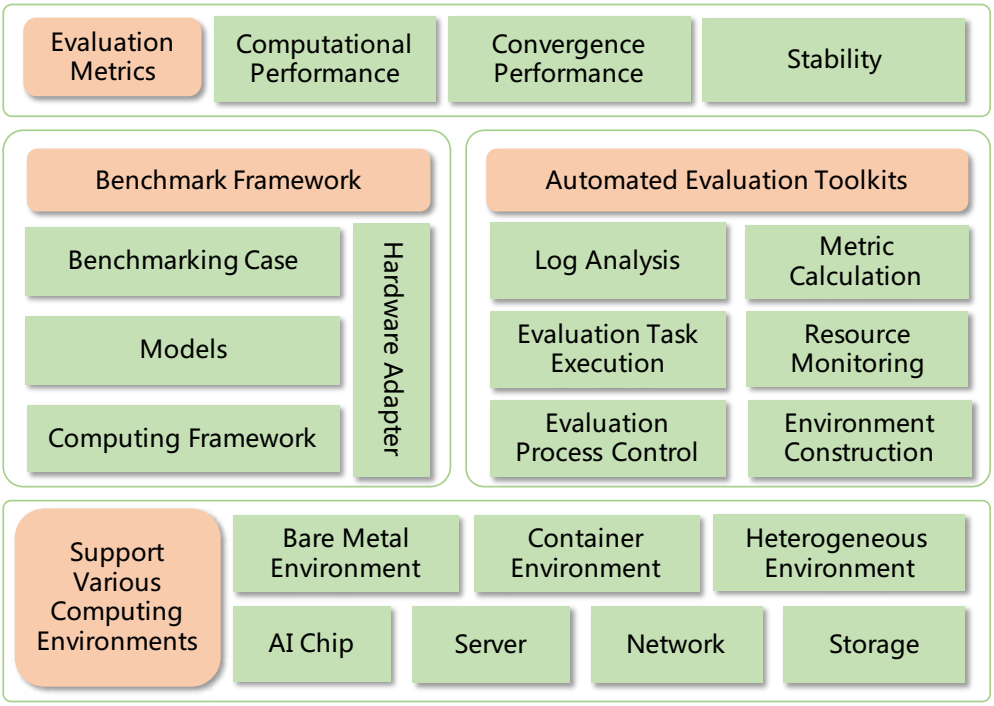


Fig. 2: Comprehensive Overview Framework of 3CBench.

set of evaluation metrics. This set covers multiple aspects, including computational performance metrics, convergence performance metrics, and stability metrics, thereby comprehensively reflecting the overall performance of computing clusters.

Easily Extensible Benchmark Framework: In consideration of the diversity of actual evaluation scenarios, the benchmark framework we have designed possesses high flexibility and extensibility, encompassing computing frameworks, models, parameter configurations, and adapters for AI chips. Through this framework, users can effortlessly add new evaluation tasks. We have already implemented evaluation tasks for representative models in mainstream domains, providing abundant test resources for evaluations.

Automated Evaluation Tool-kits: To improve the efficiency of computing cluster evaluations and reduce the uncertainty associated with manual operations, we provide a set of lightweight automated evaluation tool-kits. This suite covers multiple functional modules, including environment construction, evaluation process control, resource monitoring, evaluation task execution engines, log analysis, and metric calculation, enabling automated execution of the evaluation process.

Extensive Support for Various Computing Environments: 3CBench supports the evaluation of computing clusters of various forms and scales. In terms of cluster forms, we support both bare metal environments and container environments. In terms of cluster scales, we support clusters ranging from single-node single-card to single-node multi-card, multi-node multi-card, and even larger scales, including

hundreds, thousands, and tens of thousands of cards. Additionally, we support computing clusters with multiple AI accelerator chip types, meeting the evaluation needs of different scenarios.

3.2 Benchmarking Metrics for Computing Capability Evaluation

To enable fair and consistent comparisons across diverse AI chip architectures and model types, 3CBench incorporates a robust and comprehensive suite of performance metrics specifically tailored to the unique requirements of various models. The framework supports performance evaluation for two primary categories of models: **general-purpose typical architecture deep models**, such as ResNet50 and BERT-large, and **open-source general-purpose large language models (LLMs)**, including the LLaMA and GPT-2 series. By providing a scalable and systematic evaluation environment, 3CBench facilitates detailed analyses of AI chip performance across these distinct categories. The performance metrics employed by 3CBench are organized into three core categories: **Computational Performance**, **Convergence Performance**, and **Stability**. Each category encompasses a set of key metrics designed to capture critical aspects of an AI chip's ability to meet the computational demands of various models. Through this structured approach, 3CBench delivers a comprehensive and comparative assessment, offering insights into the efficiency and reliability of AI chips in supporting heterogeneous model architectures. This ensures an in-depth evaluation of AI chips' capabilities, fostering advancements in both hardware and algorithm design. The overall metric framework is illustrated in Figure 3.

3.2.1 Metric Categories

The following categories represent the key performance dimensions assessed in 3CBench. Each category evaluates a different aspect of the chip's performance and efficiency across various model types.

1. **Computational Performance**: Measures the chip's efficiency and resource utilization during training and inference. This includes subcategories like general computational performance, inference-specific computational performance, resource utilization, and power efficiency.
2. **Convergence Performance**: Evaluates how well the model converges during training, focusing on accuracy, convergence speed, and error metrics.
3. **Stability**: Assesses the robustness of the training process over extended periods, including recovery from failure and continuous training duration.

3.2.2 Metric Definitions

To maintain consistency, 3CBench provides rigorous definitions for each performance metric, requiring strict adherence to these definitions for all chips. This approach ensures that all metrics are uniformly scaled, enabling precise cross-platform comparisons across various heterogeneous chip environments. The detailed definitions and

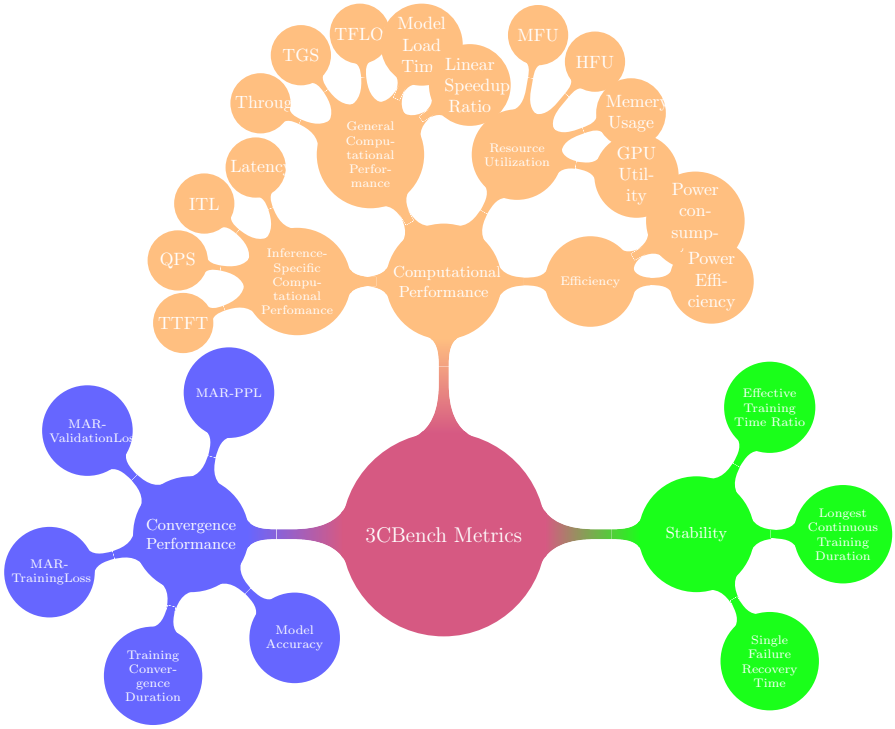


Fig. 3: The overall metric framework of 3CBench.

formulas for all metrics are summarized in Table 1. For metrics that require additional clarification, we provide further explanations through formulas, while for those with clear and unambiguous definitions, no additional formulas are necessary.

3.2.3 Formulas for Detailed Metrics

The following are the formulas for the metrics that require further explanation.

$$\text{Throughput} = \frac{N_{\text{samples}}}{T_{\text{total}}} \quad , \tag{1}$$

where:

- N_{samples} is the number of samples processed, and
- T_{total} is the total processing (training or inference) time (in seconds).

$$\text{TGS} = \frac{N_{\text{tokens}}}{T_{\text{total}}} \quad , \tag{2}$$

where:

- N_{tokens} is the number of tokens processed, and
- T_{total} is the total processing (training or inference) time (in seconds).

Table 1: Definitions of 3CBench Metrics

Metric	Description	Formula
General Computational Performance		
Throughput	The number of samples processed per unit of time (seconds) by a single accelerator card during training.	Eq. (1)
TGS (Tokens Generated per Second)	The number of tokens generated per second during training.	Eq. (2)
TFLOPS	The number of floating-point operations per second during training.	Eq. (3)
Linear Speedup Ratio	The ratio of throughput with multiple cards to the throughput of a single card under baseline conditions, multiplied by the number of cards.	Eq. (4)
Inference-Specific Computational Performance		
TTFT (Total Time to First Token)	The total time taken from the start of an inference request until the first token is generated.	-
QPS (Queries per Second)	The number of queries the model can process per second.	Eq. (5)
ITL (Inter-Token Latency)	The average time between consecutive token outputs.	Eq. (6)
Latency	The average response time for all requests.	Eq. (7)
Resource Utilization		
MFU (Model FLOPs Utilization)	Measures the efficiency with which a model utilizes the available floating-point operations per second (FLOPs) of the hardware during its training or inference process.	Eq. (8)
HFU (Hardware FLOPs Utilization)	Measures the efficiency with which the hardware's floating-point operations per second (FLOPs) are utilized during the execution of a model's training or inference process.	Eq. (9)
Memory Usage	The max or mean amount of memory utilized by a large model during its training or inference process.	-
GPU Utilization	The utilization of the GPU's computing cores during training or inference process.	Eq. (10)
Efficiency		
Power Consumption P_{avg}	The average power consumption of the card during training or inference process.	Eq. (11)
Power Efficiency R_{train}	The ratio of TFLOPS to average power consumption.Quantifies the computational performance achieved for each unit of power consumed.	Eq. (12)
Convergence Performance		
Model Accuracy	Final accuracy of the fully trained model.	-
Training Convergence Duration	Total training time required for the model to converge.	-
MAR-TrainingLoss	The average relative error of the training loss compared to the baseline across all training steps.	Eq. (13)
MAR-ValidationLoss	The average relative error of validation loss on different datasets compared to the baseline.	Eq. (14)
MAR-PPL	The average relative error of perplexity (PPL) values compared to the baseline during training.	Eq. (15)
Stability		
Effective Training Time Ratio $R_{\text{effective}}$	The ratio of effective training time to total training time.	Eq. (16)
Longest Continuous Training Duration	Longest continuous training time with and without checkpoint recovery.	-
Single Failure Recovery Time (SFRT)	The duration required to recover a large model training or inference process from a single failure event.	-

$$\text{TFLOPS} = r \times \frac{\sum_{i=1}^{N_{\text{iterations}}} \frac{B \times L \times \text{FLOPs}_l}{T_i}}{N_{\text{iterations}} \times 10^{12}} \ , \tag{3}$$

where:

- FLOPs_l is the number of floating-point operations for layer l, which is calculated as:

$$\text{FLOPs}_l = \begin{cases} 24 \times s \times d_{\text{model}}^2 + 4 \times s^2 \times d_{\text{model}}, & \text{for Transformer model,} \\ 2 \times C_{\text{in}} \times C_{\text{out}} \times H_l \times W_l \times K^2, & \text{for CNN model (e.g., ResNet).} \end{cases}$$

- B is the global batch size in an iteration,
- L is the total number of layers in the model,
- s is the length of sequence,
- d_{model} is the model dimension for Transformer models,
- C_{in} and C_{out} are the input and output channels for CNN models,
- H_l and W_l are the height and width of the output feature map at layer l for CNN models,
- K is the size of the convolution kernel,
- T_i is the processing time of the i-th iteration,
- $N_{\text{iterations}}$ is the total number of iterations in the training process, and
- r is a scaling factor, which is set to 3 during training and to 1 during inference.

$$\text{Linear Speedup Ratio} = \frac{\text{Throughput}_{\text{multi-card}}}{\text{Throughput}_{\text{single-card}} \times N_{\text{cards}}} \ , \tag{4}$$

where:

- $\text{Throughput}_{\text{multi-card}}$ is the throughput achieved using multiple cards,
- $\text{Throughput}_{\text{single-card}}$ is the throughput of a single card (baseline), and
- N_{cards} is the number of cards used.

$$\text{QPS} = \frac{N_{\text{queries}}}{T_{\text{total}}} \ , \tag{5}$$

where:

- N_{queries} is the total number of queries handled during inference, and
- T_{total} is the total time taken to process all queries (in seconds).

$$\text{ITL} = \frac{1}{N_{\text{tokens}} - 1} \sum_{i=2}^{N_{\text{tokens}}} (TS_i - TS_{i-1}) \ , \tag{6}$$

where:

- TS_i is the timestamp for the i-th token output, and
- N_{tokens} is the total number of tokens generated.

$$\text{Latency} = \frac{1}{N_{\text{requests}}} \sum_{i=1}^{N_{\text{requests}}} T_i \ , \tag{7}$$

where:

- T_i is the response time for the i-th request, and
- N_{requests} is the total number of requests.

$$\text{MFU} = \frac{\text{TFLOPS}}{\text{GPU single-card FLOPs} \times N_{\text{cards}} \times 10^{12}} \ , \tag{8}$$

where:

- TFLOPS is calculated as formula (3),
- GPU single-card FLOPs is the theoretical FLOPs of a single GPU, and
- N_{cards} is the number of cards used.

$$\text{HFU} = \frac{\text{Hardware FLOPs per iteration}}{\text{GPU single-card FLOPs} \times N_{\text{cards}}} \ , \tag{9}$$

where:

- Hardware FLOPs per iteration is the actual floating-point operations performed by the hardware during each training iteration. It is affected by factors such as memory load, data access latency, and re-computation operations due to memory bottlenecks. During training, the need to reload data or handle cache misses can cause additional computation, increasing the number of floating-point operations beyond the ideal case, and
- GPU single-card FLOPs, and N_{cards} are the same as in the MFU equation.

$$\text{GPU Utilization } (\%) = \left(\frac{T_{\text{active}}}{T_{\text{total}}} \right) \times 100 \ , \tag{10}$$

where:

- T_{active} is the time during which the GPU was actively computing, and
- T_{total} is the total measurement period.

$$P_{\text{avg}} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{N_{\text{cards}}} P_{ij} \cdot T_{ij}}{N_{\text{cards}} \cdot T_{\text{total}}} \ , \tag{11}$$

where:

- P_{ij} is the power consumed by the j-th card at the i-th measurement point (in watts),
- T_{ij} is the duration for which power P_{ij} is sustained for the j-th GPU during the i-th measurement period (in seconds),
- N_{cards} is the number of cards used,
- T_{total} is the total runtime of the program (in seconds).

$$R_{train} = \frac{\text{TFLOPS}}{P_{avg} \cdot N_{\text{cards}}} \ , \tag{12}$$

where:

- TFLOPS is the average training TFLOPS of each iteration, calculated as formula (3),
- P_{avg} is the average power consumption during training (in watts), and

- N_{cards} is the number of cards used.

$$\text{MAR-TrainingLoss} = \frac{1}{S}\sum_{i=1}^{S}\left|\frac{\text{Loss}_{\text{A-card},i} - \text{Loss}_{\text{N-card},i}}{\text{Loss}_{\text{N-card},i}}\right| \quad , \tag{13}$$

where:

- S is the number of training steps, and
- $\text{Loss}_{\text{A-card},i}$ and $\text{Loss}_{\text{N-card},i}$ are the training losses for A-card and N-card at step i.

$$\text{MAR-ValidationLoss} = \frac{1}{S}\sum_{i=1}^{S}\left|\frac{\text{ValLoss}_{\text{A-card},i} - \text{ValLoss}_{\text{N-card},i}}{\text{ValLoss}_{\text{N-card},i}}\right| \quad , \tag{14}$$

where:

- S is the number of validation steps, and
- $\text{ValLoss}_{\text{A-card},i}$ and $\text{ValLoss}_{\text{N-card},i}$ are the validation losses for A-card and N-card at step i.

$$\text{MAR-PPL} = \frac{1}{S}\sum_{i=1}^{S}\left|\frac{\text{PPL}_{\text{A-card},i} - \text{PPL}_{\text{N-card},i}}{\text{PPL}_{\text{N-card},i}}\right| \quad , \tag{15}$$

where:

- S is the number of steps, and
- $\text{PPL}_{\text{A-card},i}$ and $\text{PPL}_{\text{N-card},i}$ are the perplexity values for A-card and N-card at step i.

$$R_{\text{effective}} = \frac{\sum_{i=1}^{N_{\text{iterations}}} T_i}{T_{\text{total}}} \quad , \tag{16}$$

where:

- $N_{\text{iterations}}$ is the total number of iterations in the training process,
- T_i is the **effective training time** for the i-th iteration, and
- T_{total} is the total time from the start to the end of training, which includes both the effective training time and **non-training time**.
- In practice, we primarily record the start and end timestamps of each training iteration to calculate the effective training time for each iteration.

As described in Equation (16), the effective training time specifically refers to the period when the model is actively engaged in the training process. This includes key activities such as loading data from main memory to GPU memory, exchanging data between nodes in distributed parallel computing, executing computational tasks, and handling GPU underutilization due to scheduling inefficiencies. Non-training time, on the other hand, is excluded from the total training time. Non-training time consists of various overhead activities, including initialization time T_{init}(e.g., dataset loading, model setup), training failure time T_{fail} , fault recovery time T_{rec} (e.g., system failure handling, rollback operations), reload checkpoint time T_{reload} , checkpointing time

T_{chech} (e.g., saving model states and resuming training), as well as other system-level delays (e.g., communication setup and synchronization). The complete timeline of the training process, illustrating the distribution of these time segments, is shown in Figure 4.

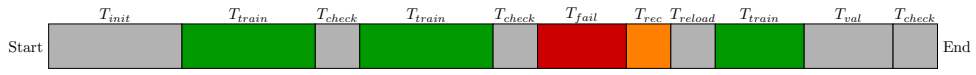


Fig. 4: The overall timeline of the training process.

3.3 Scalable Benchmark Framework

In practical applications, benchmarking scenarios are diverse and complex, with the number of benchmarking tasks potentially reaching hundreds. 3CBench aims to support benchmarking of computing clusters for AI accelerators from different vendors, covering representative models in various fields such as LLM (Large Language Models), NLP (Natural Language Processing), CV (Computer Vision), multimodal, and scientific computing, while also supporting multiple training frameworks or inference engines. For instance, a typical benchmarking task involves training an Llama-2-7B large model using the Megatron framework on an NVIDIA A100 cluster. To flexibly and efficiently expand benchmarking tasks, 3CBench has designed a scalable Benchmark framework that includes a base framework layer, a model layer, a benchmarking case layer, and a hardware adapter.

Base Framework Layer: This layer refers to the training framework or inference engine, requiring the provision of corresponding Docker images and task launch scripts. These images have been adapted and optimized for AI accelerators from different vendors to ensure efficient operation. Common frameworks include PyTorch, TensorFlow, Megatron (for large model training), and TensorRT and vLLM (for inference optimization).

Model Layer: Building upon the base framework, specific model implementations are required. The computational load of model implementations across different frameworks and vendors must be consistent. Typical models include ResNet, BERT, GPT, Llama, etc.

Benchmarking Case Layer: This layer encompasses commonly used parameter settings for benchmarking cases, such as resource parameters (number of nodes, number of GPUs), model parameters (model structure, model size, etc.), and training/inference parameters (parallelization strategy, batch size, etc.). Based on these parameters, multiple instances of the same benchmarking task can be evaluated, providing comprehensive and detailed benchmarking results.

Hardware Adapter: The hardware adapter is used to adapt frameworks and models to AI accelerators from different vendors. This not only enhances the reusability of code across different AI accelerators, reducing development efforts, but also fully leverages the characteristics of AI accelerators to optimize the performance of computing clusters, ensuring the objectivity and fairness of benchmarking results.

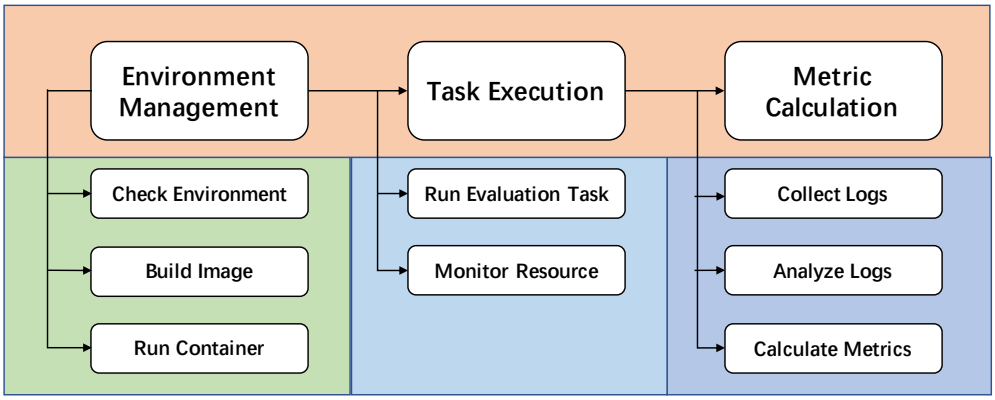


Fig. 5: Flowchart for Automated Evaluation and Metric Calculation.

3.4 Automated Evaluation and Metric Calculation

To significantly enhance evaluation efficiency and reduce the uncertainty of evaluation results, 3CBench incorporates a robust automated evaluation capability. After specifying the cluster environment and evaluation benchmark cases, 3CBench can autonomously conduct performance evaluations and accurately calculate various metrics. The entire process, as detailed in Figure 5, primarily comprises three core components: environment management, task execution, and metric calculation. The control logic of the automated process, outlined in Pseudo-Code 1, flexibly schedules each module based on actual demands and monitors the running status of tasks in real-time to ensure the smooth progression of the evaluation.

In terms of environment management, 3CBench is capable of adapting to different cluster environments. It initially performs a series of preliminary checks, including the connectivity between cluster machines and the compliance of data and code on each node. Subsequently, it constructs a matching execution environment tailored to the specific requirements of the evaluation task and cluster environment. For instance, in a bare-metal cluster environment, it utilizes a Dockerfile that matches the task to build a Docker image. Finally, on each node, it initiates the execution environment for evaluation jobs based on preset resource configurations and automatically installs necessary libraries and dependencies, providing comprehensive support for the evaluation tasks.

The task execution phase focuses on two primary tasks: resource monitoring and evaluation jobs. Resource monitoring tasks closely track the usage status of key resources such as AI chips, CPUs, memory, and power consumption. Through periodic data collection, these tasks meticulously record the data in resource monitoring logs, providing a solid foundation for subsequent in-depth analysis. The evaluation task execution engine, characterized by high framework compatibility, parses the hyperparameters of the benchmark and initiates distributed evaluation tasks. Simultaneously, the engine monitors the running status of the tasks, promptly collects relevant logs, and stores them in evaluation job logs, ensuring the integrity and accuracy of the evaluation data.

Pseudo-Code 1 Pseudo-Code for Automated Evaluation Process

Input: `nodeList`, `benchmarkCases` ▷ Cluster node information and evaluation task configuration
Output: reportFile ▷ A report file containing cluster evaluation metrics
checkCluster(`nodeList`) ▷ Verify cluster connectivity
`validCases` = validateConfig(`benchmarkCases`) ▷ Verify the existence of data and code paths for each case
for `case` in `validCases` **do** ▷ The case includes information such as vendor, framework, and model
    buildImage(`case`, `dockerfileDir`) ▷ Build environment image
    startContainer(`case`, `nodeList`) ▷ Launch container on cluster
    startResourceMonitor(`nodeList`) ▷ Start resource monitor on cluster
    runBenchmarkTask(`case`) ▷ Run benchmark test case
    waitForTaskToEnd(`case`) ▷ Waiting for the benchmark task process to end
    stopResourceMonitor(`nodeList`) ▷ Stop resource monitor on cluster
    stopContainer(`case`, `nodeList`) ▷ Stop container on cluster
    collectLogs(`nodeList`, `logDir`)
end for
`detailInfo` = analyzeLogs(`validCases`, `logDir`) ▷ Parse and align log information
`reportFile` = calculateMetrics(`detailInfo`)

In the metric calculation stage, 3CBench first comprehensively collects logs from each node, encompassing both resource monitoring logs and evaluation job logs. Subsequently, it conducts detailed parsing and alignment of the basic logs, including key indicators such as iteration time, throughput, and GPU memory usage. Finally, following a unified standard for sampling, it accurately calculates various performance metrics based on the indicator system defined in Section 3.2, including trends in indicators, average values, maximum values, and distribution situations. These comprehensive and accurate metric data provide a true reflection of the actual performance level of the computing cluster during training and inference tasks.

3.5 Extensive Support for Various Computing Environments

3CBench is capable of conducting comprehensive evaluations on computing clusters of various sizes and configurations. It supports computing clusters ranging from single-node single-GPU, single-node multi-GPU, and multi-node multi-GPU setups, to clusters with hundreds, thousands, and even tens of thousands of GPUs. We offer two operation modes: bare-metal environment and container environment, and ensure compatibility with AI acceleration chip computing clusters from multiple vendors. Additionally, 3CBench possesses the capability to evaluate heterogeneous computing clusters with mixed chips.

In terms of cluster scale, 3CBench is highly compatible with various distributed computing frameworks, enabling it to execute evaluation tasks in single-node single-GPU, single-node multi-GPU, and multi-node multi-GPU environments. For large-scale model training, our evaluation framework can easily handle the demands of evaluating computing clusters with hundreds to thousands, even tens of thousands of GPUs, ensuring that evaluations across different scales can be satisfactorily conducted.

In the bare-metal environment, 3CBench integrates multiple functions such as cluster management, image management, and container management, standardizing the environment preparation process. For specific evaluation cases, the system matches the corresponding Dockerfile based on the evaluation task, automatically builds the evaluation task image, and starts containers on each node as needed, including configurations such as resource allocation and path mapping. In the container environment, there is no need to build images and start containers; 3CBench can be directly deployed and run within containers. We provide convenient container environment management tools to facilitate users in synchronizing evaluation codes and data, and also allow for the synchronized use of tools like apt and pip across multiple nodes to uniformly configure the environment, ensuring environmental consistency.

Furthermore, 3CBench supports the evaluation of AI acceleration chips from multiple vendors, and our benchmark framework can flexibly adapt to new vendor AI chips. For heterogeneous computing clusters that include AI chips of different models or even from different vendors, we possess comprehensive evaluation capabilities. 3CBench can configure exclusive images, commands, and parameters for different types of nodes, and evaluate the computing power of the entire heterogeneous cluster through the heterogeneous computing framework.

4 Experiments and Analysis

4.1 Experimental Setup

3CBench accommodates both single-node and multi-node configurations, allowing for scalable testing across heterogeneous AI clusters. Our GPU setup includes industry-standard NVIDIA A100 and V100 GPUs, which are commonly recognized for their high-performance capabilities in training large neural networks. Additionally, we extend our platform's compatibility through collaboration with other GPU vendors, referred to here as anonymous devices, who provide in-house GPUs for evaluation purposes. This collaboration involves in-depth integration efforts to ensure that our benchmarking system effectively utilizes these alternative GPU architectures, enabling a broader range of hardware configurations and thus providing a more comprehensive evaluation landscape. To streamline the deployment of various machine learning frameworks, we utilize Docker containers to manage the software environment consistently across different hardware setups. Each Docker image is tailored to support specific GPU architectures and contains optimized versions of deep learning frameworks. This includes GPU-accelerated versions of Megatron-LM and DeepSpeed, which are essential for handling large-scale models and parallelized training strategies. By encapsulating these frameworks within Docker containers, we ensure that each experiment

runs in an isolated environment, reducing variability in results due to environmental differences and improving reproducibility across hardware platforms.

For the benchmarking tasks, we select the Llama series as the primary model. As a representative of open-source LLMs, Llama models leverage advanced transformer architectures to handle vast amounts of training data and are available in multiple parameter scales, making them ideal for testing the computational limits across different configurations, providing insights into how various system architectures perform when subjected to the computationally intensive demands of transformer networks. The pre-training dataset for Llama is based on the OSCAR [30] corpus, a large-scale multilingual dataset that provides a diverse and comprehensive set of textual data. 3CBench also incorporates diverse corpus databases [31] to facilitate evaluating language models during training by monitoring loss convergence compared to baselines, which helps promptly identify potential deviations in custom hardware or algorithm implementations. In addition to Llama, we incorporate ResNet [32] as a benchmark model to evaluate CNN performance. ResNet is regarded as the CNN counterpart to transformer models, representing a foundational architecture with comparable significance in computer vision area. Thus we ensure that 3CBench can comprehensively assess performance across both transformer-based language models and CNN-based visual models.

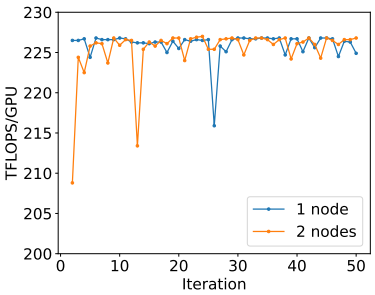
The experiments primarily focus on the pre-training stage of both Llama and ResNet models, which is particularly computationally demanding and resource-intensive, making it an essential phase for evaluating the efficiency and scalability of hardware platforms. 3CBench deliberately skips the initial few training cycles in order to focus on the more stable segments of training, which are more representative of the sustained computational demands of deep learning models and provide a clearer picture of hardware performance under realistic workloads.

4.2 Comprehensive Benchmarking results of Computing Capacity

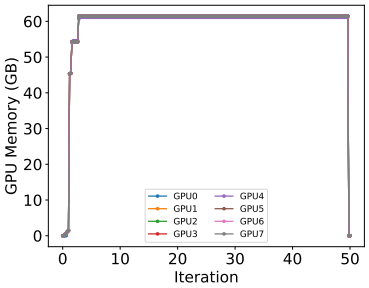
To demonstrate the capacities of 3CBench, extensive experiments are conducted using both single-node (8 GPUs) and dual-node (16 GPUs) configurations with NVIDIA A100 GPUs. These evaluations encompass the pre-training phase of ResNet and transformer-based language models, leveraging two prominent frameworks: Megatron and DeepSpeed. The results, detailed in Tables 2, include comprehensive computing metrics such as throughput, floating-point operations per second (FLOPS), Model Flop Utilization (MFU), average and peak GPU memory usage, average power consumption, power efficiency, and model loading time. By covering a broad spectrum of critical metrics, 3CBench demonstrates a comprehensive assessment capacity, accurately reflecting the computational demand across various AI tasks. These experiments also illustrate adaptability of 3CBench to distributed training in diverse cluster environments, and its support for a wide range of AI tasks. The experiments also highlight the modular design's ability to adapt to miscellaneous configurations, with notable differences in performance across single-node and multi-node setups depending on the specific workloads and metrics. 3CBench also extends its capabilities to include real-time monitoring across the temporal dimension, as demonstrated in Figure 6.

Table 2: Evaluation results of 3CBench for multiple AI pre-training tasks on single-node 8 (1×8) and dual-node 16 (2×8) NVIDIA A100 GPUs. 3CBench provides comprehensive metrics for computing capacities, including throughput, teraFLOPS(TFLOPS), average GPU memory usage (Avg. Mem), peak GPU memory usage (Max. Mem), power efficiency(Power Eff.), and the loading time of AI models.

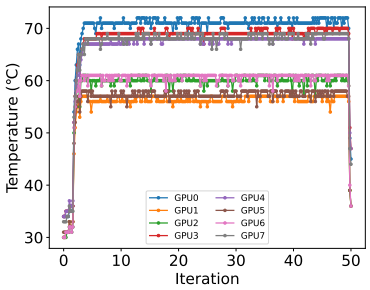
		Throughput (samples/s)	TFLOPS /GPU	MFU (%)	Avg. Mem (GB)	Max. Mem (GB)	Avg. Power (W)	Power Eff. (TFLOPS/W)	Loading Time (s)
ResNet-50	1×8	3894	48.28	15.47	21.09	28.16	196.97	0.25	0.33
	2×8	3884	48.16	15.44	19.22	25.95	194.71	0.25	0.41
BERT	1×8	49299	103.72	33.24	31.58	34.26	282.45	0.37	0.15
	2×8	42225	88.84	28.47	29.11	31.29	260.37	0.34	0.18
Llama2-7B-Megatron	1×8	5477	225.95	72.42	59.15	61.43	382.12	0.59	0.61
	2×8	5468	225.57	72.30	53.09	56.21	373.93	0.60	0.86
Llama2-7B-Deepspeed	1×8	4631	191.04	61.23	61.47	63.63	385.65	0.50	26.71
	2×8	4563	188.23	60.33	56.08	58.69	386.63	0.49	23.11
Llama3-8B-Megatron	1×8	3759	217.66	69.76	58.12	61.84	396.12	0.55	1.17
	2×8	3662	212.09	67.98	53.02	55.08	394.04	0.54	1.50



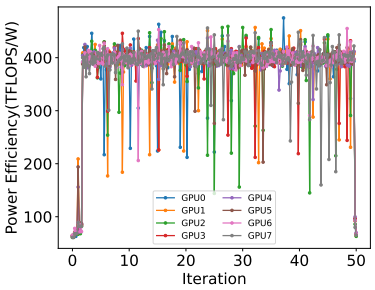
(a) TFLOPS/GPU for single/dual-node.



(b) GPU memory during single node training.



(c) Temperature during single node training.



(d) Power efficiency during single node training.

Fig. 6: Timeline of 3CBench metrics on single- and dual-node 8 NVIDIA A100 GPUs. Single-node training follows a data parallelization strategy, ensuring consistent distribution of (b) GPU Memory and (d) Power Efficiency across GPUs.

The timeline feature exhibits temporal fluctuations as the complement for statistical metrics.

3CBench integrates diverse datasets to ensure comprehensive evaluation and precision alignment with partner vendors. As illustrated in Figure 7 and Figure 8, it provides loss convergence data from a baseline platform for LLM pre-training, offering a benchmark against which hardware-specific algorithmic implementations can be rigorously assessed. 3CBench also contains domain-specific test datasets, such as C4-EN, Dolma Books, and WikiText-103, alongside relative error metrics (Figure 8c, to evaluate the consistency and stability of algorithmic convergence across varying configurations. Discrepancies in loss trends are promptly detected, enabling root-cause analysis and targeted refinements. This systematic validation ensures robust alignment between device-specific frameworks and baseline expectations, promoting reliable and efficient LLM deployment at scale.

To further validate 3CBench's capability to support large-scale distributed training, experiments focus on pre-training Llama2-7B-Megatron across multi-node NVIDIA A100 clusters with varying GPU and node counts. Figure 9 illustrates the

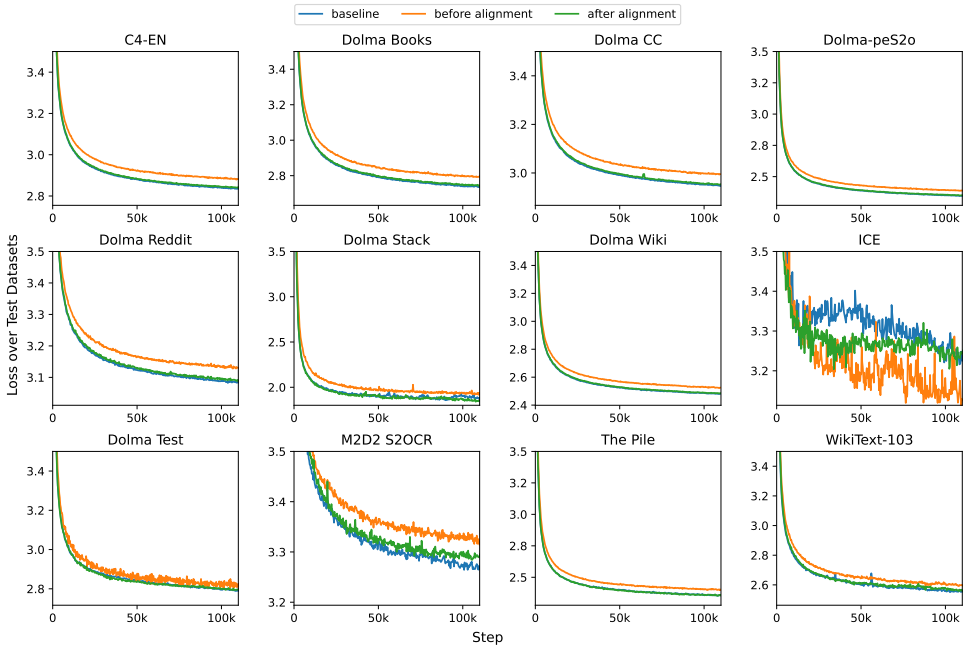


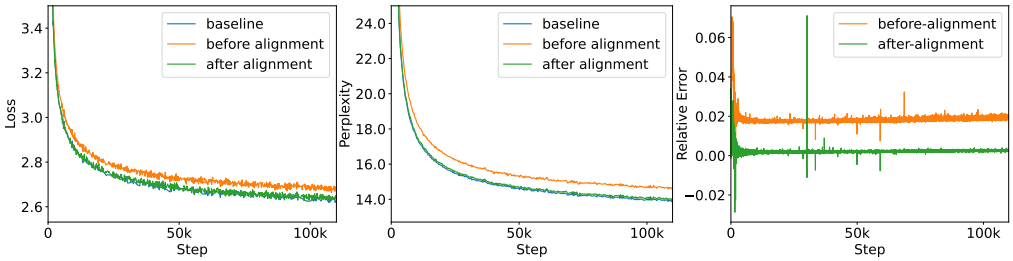
Fig. 7: Loss convergence across multiple test datasets during LLM pre-training, comparing the baseline platform with the pre-alignment and post-alignment performance of an anonymous device.

scaling behavior of per-GPU TFLOPS, accumulated TFLOPS, and the ideal accumulated TFLOPS. The configuration with 8 GPUs achieves the highest per-GPU TFLOPS of 226.0, which gradually decreases as GPUs and nodes scales up, reaching 189.6 at 512 GPUs. Meanwhile, the accumulated TFLOPS shows a consistent growth trend, closely approaching the ideal linear scaling but slightly lagging due to communication and synchronization losses among multiple nodes. This experiment highlights the importance of optimizing the partitioning strategy for distributed large-model training to mitigate communication and synchronization bottlenecks.

3CBench effectively benchmarks heterogeneous AI clusters by leveraging collaborations with anonymous vendors to implement kernel optimization, algorithm alignment, and custom Docker environments tailored to specific GPU architectures. Table 3 illustrates performance metrics for Llama2-7B pre-training across GPUs, including Device-01, Device-02, Device-03, Device-04, Device-05, and NVIDIA's A100 and V100. The NVIDIA A100 achieves a balanced performance profile, maintaining consistent TFLOPS and throughput across configurations, alongside competitive power efficiency. In contrast, the NVIDIA V100 shows lower TFLOPS and throughput, highlighting the advancements in newer architectures. Device-01 stands out for achieving the highest single-node performance with 227.43 TFLOPS/GPU and a throughput of 5513 samples/s, surpassing even the NVIDIA A100 in both metrics. Device-02 exhibits

Table 3: TFLOPS, average GPU memory (Avg.Mem),average power (Avg.Power), throughput and power efficiency(Power Eff.) evaluated by 3CBench for Llama2-7B-Megatron pre-training across heterogeneous GPU clusters. "1×8" represents 1 node with 8 GPUs per node, while "2×8" denotes 2 nodes with 8 GPUs per node. To preserve confidentiality, certain values of anonymous GPUs have been proportionally degraded while retaining comparative trends.

	TFLOPS /GPU		Avg.Mem (GB)		Avg.Power (W)		Throughput (samples/s)		Power Eff. (TFLOPS/W)	
	1×8	2×8	1×8	2×8	1×8	2×8	1×8	2×8	1×8	2×8
NVIDIA A100	225.95	225.58	59.15	53.09	382.12	373.93	5477	5468	0.591	0.603
NVIDIA V100	69.04	56.69	28.06	20.33	243.97	224.28	1674	1374	0.283	0.253
Device-01	227.43	218.16	67.81	62.35	304.40	290.80	5513	5288	0.747	0.750
Device-02	58.47	50.58	37.22	32.72	382.63	404.37	1417	1226	0.138	0.113
Device-03	135.94	132.79	19.74	17.50	194.33	191.51	3295	3219	0.630	0.624
Device-04	146.13	141.84	55.32	49.49	279.13	298.14	3542	3438	0.472	0.428
Device-05	113.04	111.34	45.97	44.42	487.48	482.23	2740	2699	0.209	0.208



(a) Training loss. (b) Perplexity validation. (c) Relative error.

Fig. 8: Comparison of LLM pre-training performance metrics for the baseline platform with the pre-alignment and post-alignment performance of an anonymous GPU configuration.

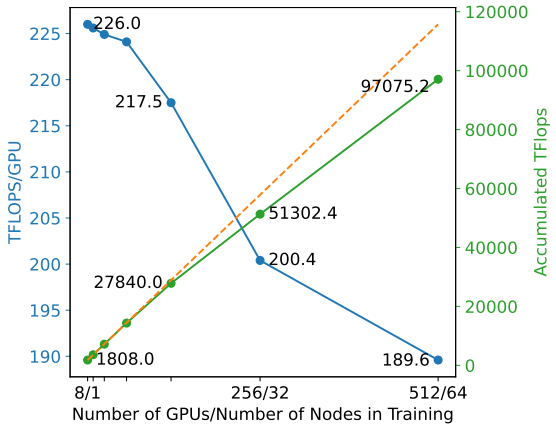


Fig. 9: Evaluation of TFLOPS/GPU and accumulated TFLOPS for Llama2-7B pre-training across NVIDIA A100 clusters with varying configurations, including 1, 2, 4, 8, 16, 32, and 64 nodes with 8 GPUs/node. The red dotted line represents the ideal accumulated TFLOPS.

the lowest power efficiency among all devices, with only 0.138 TFLOPS/W in single-node and 0.113 TFLOPS/W in dual-node setups, coupled with a significant decline in performance metrics. Meanwhile, Device-03 demonstrates steady TFLOPS/GPU performance, maintaining 135.94 in single-node and 132.79 in dual-node configurations, reflecting strong scalability. Its memory usage and power efficiency remain stable as well, with only marginal changes, indicating effective workload balancing. These results illustrate 3CBench's ability to evaluate critical performance metrics, identify architectural trade-offs, and assess scalability across heterogeneous GPU clusters. It

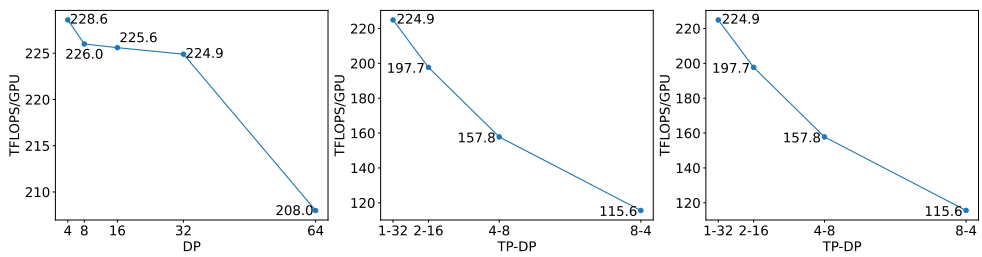
provides insights for optimizing resource allocation, memory distribution, and inter-node communication, supporting large-scale deep learning workflows and aiding in system design improvements.

4.3 Performance Analysis

The evaluation metrics provided by 3CBench offer critical insights into the performance of various AI tasks across heterogeneous clusters. By analyzing these metrics, significant differences in system utilization, throughput, and power efficiency are observed among different cluster configurations and AI workloads. These findings emphasize the importance of tailoring resource allocation and computational parameters to match the specific demands of large-scale distributed training. The flexibility and scalability of 3CBench enable it to optimize GPU cluster configurations, ensuring balanced utilization and maximizing training efficiency. Such analyses serve as a practical guideline for selecting and optimizing resource distributions, achieving optimal performance in large-scale AI training pipelines.

We conducted experiments with various distributed training configurations for Llama2-7B, including data parallelization (DP), tensor parallelization (TP), and pipeline parallelization (PP). Figure 10 summarizes the performance of these configurations. Figure 10a demonstrates that when GPUs have sufficient memory to store the full forward and backward passes of the network, data parallelization achieves higher efficiency compared to pipeline and tensor parallelism. In this setup, the highest performance is observed with DP at the minimal value, achieving FLOPS efficiency of 228.6, and it monotonically decreases as DP increases. Figure 10b and Figure 10c explore scenarios where the world size is fixed at 32 GPUs. Both figures highlight the trend of decreasing FLOPS as the degree of parallelization increases due to increased communication overhead between GPUs. However, the performance degradation in PP outpaces that of TP, indicating that excessive pipeline stages lead to pronounced inefficiencies. These findings suggest that while DP is the most effective for memory-sufficient scenarios, a balanced approach to TP and PP configurations is critical for maintaining efficiency in large-scale distributed training. Additionally, combining TP and PP strategically with DP could further enhance scalability.

Based on the relative FLOPS/GPU ratio to NVIDIA A100 presented in Figure 11, key observations highlight significant variations for ResNet, Llama2-7B, and BERT, offering critical insights into optimizing deep learning workloads across heterogeneous hardware architectures. Device-01 notably outperforms the A100 for Llama2-7B, which emphasizes its superior performance for transformer-based computations. This result highlights the architectural optimizations within Device-01, particularly for matrix-intensive operations central to attention mechanisms. In contrast, ResNet's FLOPS/GPU ratios remain lower across all devices compared to Llama2-7B and BERT, underscoring the challenge of convolution-dominated workloads on modern GPUs, which are increasingly optimized for transformer models. For BERT, Device-04 emerges as the leader with the highest ratio of 1.75, showcasing exceptional efficiency for this specific transformer-based model, while Device-02 lags significantly at 0.22, indicating poor compatibility with BERT's computational characteristics. Although



(a) DP configurations. (b) TP configurations. (c) PP configurations.

Fig. 10: FLOPS for various parallelization configurations of Llama-7B-Megatron on NVIDIA A100 clusters. (a) Evaluation with fixed tensor and pipeline parallelization (TP=1, PP=1). (b)(c) Evaluation with a fixed world size of 32 GPUs.

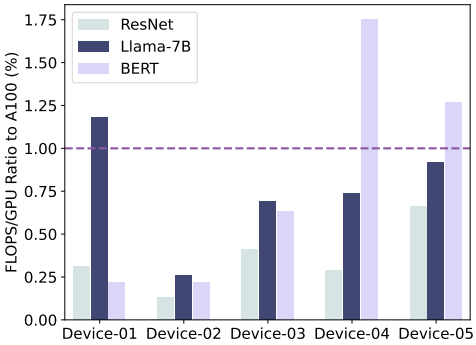


Fig. 11: FLOPS comparison for ResNet, Llama2-7B, and BERT across different GPU architectures against NVIDIA A100.

the A100 demonstrates strong, consistent performance across all workloads, it is evident that task-specific architectural differences influence performance. These findings underline the importance of task-aware optimization strategies for deploying AI workloads on heterogeneous GPU clusters. While transformer models, such as Llama2-7B and BERT, benefit from robust compatibility across diverse hardware, convolutional networks like ResNet require tailored configurations to unlock the full computational potential of the underlying hardware.

5 Conclusion

We propose 3CBench, a scalable benchmarking framework tailored to assess the computing capacity of AI clusters in diverse deep learning tasks under single-node and distributed environments. By unifying computational benchmarking across heterogeneous hardware and complex workloads, 3CBench addresses the challenges of measuring system throughput, scalability, and efficiency in real-world AI applications.

Through extensive experimentation, we demonstrated its ability to evaluate end-to-end pipeline performance automatically with comprehensive metrics, offering insights into algorithmic alignment and optimizing task scheduling across nodes. The framework's modular design supports seamless integration with various AI frameworks and tasks, ensuring adaptability to evolving deep learning trends.

In the future, we plan to extend 3CBench by collaborating with industry partners to incorporate emerging AI technologies. Enhancements will focus on profiling tasks with complex resource requirements, such as transformer-based large-scale models and foundation model training, to improve workload characterization. Furthermore, we aim to integrate a visualization interface to facilitate intuitive analysis of key performance indicators, enabling practitioners to optimize system design and workload management effectively. As a comprehensive and extensible tool, 3CBench is poised to standardize performance evaluation and drive innovation in AI system design.

References

[1] Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al.: The llama 3 herd of models. arXiv preprint arXiv:2407.21783 (2024)

[2] Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al.: Qwen technical report. arXiv preprint arXiv:2309.16609 (2023)

[3] Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., Amodei, D.: Scaling laws for neural language models. arXiv preprint arXiv:2001.08361 (2020)

[4] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D.d.L., Hendricks, L.A., Welbl, J., Clark, A., et al.: Training compute-optimal large language models. arXiv preprint arXiv:2203.15556 (2022)

[5] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al.: Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)

[6] Lu, Z., Du, C., Jiang, Y., Xie, X., Li, T., Yang, F.: Quantitative evaluation of deep learning frameworks in heterogeneous computing environment. CCF Transactions on High Performance Computing **6**(1), 94–111 (2024)

[7] Rasley, J., Rajbhandari, S., Ruwase, O., He, Y.: 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, pp. 3505–3506 (2020)

[8] Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., *et al.*: Efficient large-scale language model training on gpu clusters using megatron-lm. In:

Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–15 (2021)

[9] Developer, N.: NVIDIA Nsight Systems (2024). https://developer.nvidia.com/nsight-systems

[10] Profiler, P.: Pytorch (2024). https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html

[11] Mattson, P., Cheng, C., Diamos, G., Coleman, C., Micikevicius, P., Patterson, D., Tang, H., Wei, G.-Y., Bailis, P., Bittorf, V., *et al.*: Mlperf training benchmark. Proceedings of Machine Learning and Systems **2**, 336–349 (2020)

[12] Ascend, H.: Huawei Ascend (2024). https://e.huawei.com/en/products/computing/ascend

[13] Cambricon: Cambricon (2024). https://www.cambricon.com/

[14] Kunlunxin: Kunlunxin (2024). https://www.kunlunxin.com/

[15] Threads, M.: Moore Threads (2024). https://www.mthreads.com/

[16] NVIDIA: NVIDIA GPUs (2024). https://developer.nvidia.com/

[17] AMD: AMD Chip (2024). https://www.amd.com/en/products/accelerators/instinct/mi300.html

[18] Intel: Intel Chip (2024). https://www.intel.com/

[19] Google: Google Chip TPU (2024). https://cloud.google.com/tpu?hl=en

[20] Zhou, K., Adhianto, L., Anderson, J., Cherian, A., Grubisic, D., Krentel, M., Liu, Y., Meng, X., Mellor-Crummey, J.: Measurement and analysis of gpu-accelerated applications with hpctoolkit. Parallel Computing **108**, 102837 (2021)

[21] Gilman, G., Walls, R.J.: Characterizing concurrency mechanisms for nvidia gpus under deep learning workloads. ACM SIGMETRICS Performance Evaluation Review **49**(3), 32–34 (2022)

[22] Chabbi, M., Murthy, K., Fagan, M., Mellor-Crummey, J.: Effective sampling-driven performance tools for gpu-accelerated supercomputers. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 1–12 (2013)

[23] Sridharan, S., Heo, T., Feng, L., Wang, Z., Bergeron, M., Fu, W., Zheng, S., Coutinho, B., Rashidi, S., Man, C., et al.: Chakra: Advancing performance benchmarking and co-design using standardized execution traces. arXiv preprint arXiv:2305.14516 (2023)

[24] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H.W., Sutton, C., Gehrmann, S., *et al.*: Palm: Scaling language modeling with pathways. Journal of Machine Learning Research **24**(240), 1–113 (2023)

[25] BAAI: FlagPerf (2024). https://flagopen.baai.ac.cn/

[26] Allada, V., Benjegerdes, T., Bode, B.: Performance analysis of memory transfers and gemm subroutines on nvidia tesla gpu cluster. In: 2009 IEEE International Conference on Cluster Computing and Workshops, pp. 1–9 (2009). IEEE

[27] Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., Catanzaro, B.: Megatron-lm: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053 (2019)

[28] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)

[29] Devlin, J.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)

[30] OSCAR: Open Super-large Crawled Aggregated Corpus (2024). https://oscar-project.org/

[31] Soldaini, L., Kinney, R., Bhagia, A., Schwenk, D., Atkinson, D., Authur, R., Bogin, B., Chandu, K., Dumas, J., Elazar, Y., et al.: Dolma: An open corpus of three trillion tokens for language model pretraining research. arXiv preprint arXiv:2402.00159 (2024)

[32] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)