Centimani: Enabling Fast AI Accelerator Selection for DNN Training with a Novel Performance Predictor

Zhen Xie, *Binghamton University;* Murali Emani, *Argonne National Laboratory;* Xiaodong Yu, *Stevens Institute of Technology;* Dingwen Tao, *Indiana University;* Xin He, *Xidian University;* Pengfei Su, *University of California, Merced;* Keren Zhou, *George Mason University;* Venkatram Vishwanath, *Argonne National Laboratory*

https://www.usenix.org/conference/atc24/presentation/xie

This paper is included in the Proceedings of the 2024 USENIX Annual Technical Conference.

July 10–12, 2024 • Santa Clara, CA, USA

978-1-939133-41-0

Open access to the Proceedings of the 2024 USENIX Annual Technical Conference is sponsored by

Centimani: Enabling Fast AI Accelerator Selection for DNN Training with a Novel Performance Predictor

Zhen Xie[⋆]✉, *Murali Emani*[†], *Xiaodong Yu*[‡], *Dingwen Tao*[◇], *Xin He*[◁], *Pengfei Su*[▷], *Keren Zhou*[⊙], *and Venkatram Vishwanath*[†]
[⋆]*Binghamton University* [†]*Argonne National Laboratory* [‡]*Stevens Institute of Technology*
[◇]*Indiana University* [◁]*Xidian University* [▷]*University of California, Merced* [⊙]*George Mason University*
✉ *Corresponding Author: Zhen Xie (zxie3@binghamton.edu)*

Abstract

For an extended period, graphics processing units (GPUs) have stood as the exclusive choice for training deep neural network (DNN) models. Over time, to serve the growing demands in a more targeted manner, various artificial intelligence-specific hardware, referred to as *AI accelerators*, have been vigorously developed, aiming to provide more efficient DNN acceleration solutions. However, sufficient solutions are also heterogeneous and thus introduce complexities in accelerator selection. Given a DNN model and a training objective, such as throughput or price-performance ratio, it remains challenging to arrive at the optimal decision among many options due to high reimplementation costs and unexpected performance.

To tackle this challenge, we propose *Centimani*, a performance predictor that accurately and rapidly predicts DNN training throughput on various AI accelerators, thereby facilitating the accelerator selection process. To achieve this goal, we first analyze typical AI accelerators and draw observations that abstract AI accelerator designs and guide our performance modeling approach. In particular, we construct a memory estimation model and decoupled performance models to select the most appropriate batch size and predict the execution time of DNN training. We validate our approach by applying Centimani to six common DNN models on four typical AI accelerators. Results show that Centimani predicts the throughput with an average accuracy of 93.1% on single-device training and 90.4% on multiple-device training, thus the optimal accelerator corresponding to the user's training objective can be obtained.

1 Introduction

The availability of vast public datasets and the rapid advancements of DNNs have led to significant growth in the prevalence of AI-driven applications and services, achieving remarkable outcomes in many tasks [35, 38, 41, 67, 70, 89, 102, 116]. Meanwhile, the increasingly complex DNN models entail significant overhead for model training [122]. Consequently, in recent years, academia and industry have made extensive efforts [30, 90, 91, 104] to offer a broad range of new hardware to optimize DNN training workloads. These new AI-specific hardware are known as *AI accelerators*.

From the trend perspective, AI accelerators have become the mainstream way to sustainably push Moore's Law in the AI field [37, 94]. According to AI Index Report 2023 [2] and Top AI Market Radar [14], the lion's share of private investments in AI, approximately $6 billion or 58%, has been directed towards the development of AI accelerators. In the HPC domain, AI accelerators have been seamlessly integrated into cutting-edge exascale supercomputers, including Aurora [4], Frontier [8], and El Capitan [7]. This integration has significantly contributed to providing sufficient DNN solutions.

From the application perspective, many DNN models [30, 30, 43, 87, 92, 98, 101, 103] have been reimplemented and trained on AI accelerators with substantial performance speedups. For example, the largest biological language models (GenSLMs) [123] have been trained on a new AI accelerator named Cerebras CS-2 system [57]. For GenSLM with 25 billion parameters, only *16* Cerebras CS-2 accelerators can reach convergence in 21.7 hours, which is faster than 560 NVIDIA A100 GPUs [31]. This work has been awarded the 2022 Gordon Bell Special Prize [1].

However, sufficient DNN solutions are also heterogeneous. While having various options provides users with flexibility, it can also lead to the paradox of choice, which leaves practitioners uncertain about which AI accelerator should be used to train their models. What further complicates matters is that no *"one-size-fits-all"* solution works best on all DNN models. Choosing an appropriate AI accelerator involves weighing its performance benefits against the migration cost from the current platform (typically the GPU platform), which makes users have to face a decision-making dilemma.

A natural way to make this decision is to reimplement the models and then measure training performance directly on each AI accelerator, as demonstrated in many existing work [22, 84, 96, 99]. Another common wisdom is to consult existing benchmarks published by vendors and match them with a similar DNN model to obtain a "ballpark estimate" [44, 75, 90, 104]. However, these approaches have their limitations as follows:

Repetitive implementation and direct hardware access: Take GenSLMs [123] as an example, this work requires extensive efforts from researchers to achieve remarkable performance on accelerator selection, because GenSLMs have also been reimplemented and trained on five other AI accelerators, with the selection of the optimal one (such as Cerebras CS-2) contingent on actual performance assessment. This difficulty is due to the unique programming models [40, 46, 82] of AI accelerators and the dependent libraries required to train this DNN model, resulting in a significant expenditure of manpower and cost. Furthermore, measuring performance requires direct access to all AI accelerators considered, which may not always be feasible for all users.

Large deviations from benchmarks: Deriving the training throughput of a particular DNN model through official benchmarks can be inaccurate, primarily because benchmarks typically contain only a limited number of DNN models, such as MLPerf [75] and DAWNBench [32]. Such benchmarks are not effective, especially when the DNN model to be estimated differs significantly from the models in the benchmarks. Additionally, different AI accelerators choose varying hardware-related hyper-parameters [104], such as batch size, for the same model to achieve higher parallelism and better data locality, which can further affect throughput estimation.

Hardware utilization and scalability: Another approach [18, 114] to predict the training throughput of a model is to utilize *hardware utilization*, such as applying GPU training utilization to other AI accelerators. However, this approach may lead to significant errors, because the hardware utilization among accelerators is not a general indicator due to distinct hardware design and software workflow. Moreover, different AI accelerators are connected using unique networks, which presents additional challenges for throughput prediction in a distributed environment.

Based on the above considerations, in this work, we advocate for a hybrid approach *that combines experimental and analytical methods to make direct performance predictions and guide the appropriate selection of accelerators*. We notice that: (1) the hardware design [30, 69] and software execution flow [26, 90] of AI accelerators are starkly noticeable due to distinct performance considerations; and (2) DNN training workloads [50, 60] differ from conventional general programs because they typically consist of routine training stages with required hyper-parameters. Ideally, the throughput prediction of the whole training process can be decomposed into the execution time prediction of multiple training stages for better prediction accuracy, and the execution time of each stage can be predicted on unified accelerator abstractions for more comprehensive coverage.

To enlighten our performance model, we first analyze typical AI accelerators and propose a multi-aspect abstraction that hides the details of accelerator designs; Specifically, the multi-aspect abstraction involves two abstractions from different perspectives to describe AI accelerators: (1) *hardware abstraction* deals with various hardware resources and configurations while hiding the underlying hardware complexity to improve generality to adapt more platforms; and (2) *software abstraction* systematically encapsulates the management of training data and operators. In addition, the hardware abstraction and software abstraction are connected by an *execution modeling*, which builds a connection between multi-aspect abstraction and DNN model training.

We leverage this setting to develop our performance model. Our performance predictor involves two steps: (1) we first estimate the memory consumption of the given DNN model with varying batch sizes and select the optimal one on each AI accelerator; then (2) we predict the execution time of multiple training stages and eliminate overlapping portions among these stages to obtain the final training time.

Specifically, in the first step, we select an appropriate hyper-parameter such as batch size for model training based on the proposed memory estimation model, which traverses the computation graph of DNN models, estimates the memory consumption of each training data, and maps each training data to the actual memory hierarchy of each accelerator. The memory estimation model optimizes data placement by maximizing the allocation of on-chip memory, coordinating specific software workflow strategies, and staying within hardware limitations. Our goal in selecting the maximum batch size is to improve training efficiency.

In the second step, we predict the execution time of multiple stages in DNN training, which include data loading/pre-processing, computation, and communication. We infer to the data loading/pre-processing time from the existing platform (usually GPU). For the computation stage, we traverse different operators in the computation graph to estimate the total computation time. These operators are categorized as either *common* or *uncommon* operators, in which the execution time of common operators is obtained by executing predefined micro-benchmarks, while the execution time of uncommon operators is estimated using a cache-aware roofline model. For the communication stage, we analyze the amount of data that needs to be exchanged and estimate its communication time using communication primitives. Finally, we eliminate the overlapping part among stages to obtain the final prediction.

We implement our model into a Python library that we call *Centimani*, and evaluate its prediction accuracy by comparing predicted throughput with measured performance using six DNN models on four AI accelerators. The average accuracy of Centimani on single-device training and multiple-device training is 93.1% and 90.4% respectively.

In summary, this work makes the following contributions:

- We introduce a new and general performance predictor for DNN training on AI accelerators.
- Centimani introduces model topology and training stages to enable accurate performance prediction on AI accelera-

tors. Its applicability and effectiveness are evaluated using six common DNN models, namely, (i) ResNet-50 v1.5, (ii) U-Net, (iii) CANDLE UNO, (iv) BERT-large, (v) BraggNN [89], and (vi) OpenAI GPT-2, on four AI accelerators (SambaNova, Graphcore, Cerebras, and Habana).

- Centimani uses an automated workflow with high usability. Two training metrics (e.g., throughput and price-performance ratio) are presented to show how Centimani can help users make informed decisions based on their training objectives.

2 Background and Motivations

2.1 Background on DNN Training

A DNN model consists of numerous parameters and can be trained on a single device or across multiple devices. Regardless of the hardware used, the process of DNN training can be broken down into three main stages [28,48,53,115] : (1) *data loading/pre-processing*, which refers to the action required to move and manipulate data samples from a storage location to the memory co-located with the compute units for training; (2) *computation*, which encompasses forward and backward propagation to process a batch of training data through the DNN model and compute the loss function and gradients of each learnable parameter; (3) *communication*, which aggregates all gradients from all devices and synchronizes parameters with a designated optimizer (e.g., SGD [25], Adam [88], etc.). Through iterative refinement of model parameters using these three stages, DNN training continues until the loss function reaches its minimum or a predetermined target.

2.2 Why Select Optimal AI Accelerator?

AI accelerators exhibit varying performance even when executing the same training task. To reveal this situation, we train a common ResNet-50 v1.5 model [55] with the same floating-point precision (half-precision) and dataset (ImageNet [36]) on five platforms, including a conventional NVIDIA A100 GPU platform [31] and four new AI accelerator platforms, namely, SambaNova SN30 [82], Graphcore Bow-IPU [63], Cerebras CS-2 [57], and Habana Gaudi2 [76].

Figure 1(a) demonstrates the training throughput and hardware utilization of each accelerator, where hardware utilization is determined by dividing the achieved computing power by the respective peak performance. Among the five platforms, they exhibit varying training throughputs. Notably, Habana Gaudi2 achieves the highest throughput, surpassing the lowest brought by NVIDIA A100 GPU by a factor of 2.17×. However, such performance differences cannot be solely attributed to disparities in the theoretical peak performance of AI accelerators. For example, Graphcore Bow-IPU

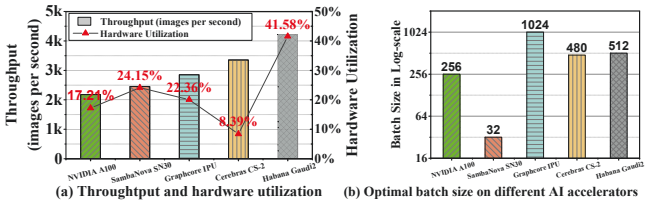

Figure 1: Throughput, hardware utilization, and the optimal batch size of training ResNet-50 on AI accelerators.

has a theoretical peak performance of 350 TFLOPS in the half-precision format, which is 1.21× higher than that of NVIDIA A100 GPU at 312 TFLOPS, yet Graphcore Bow-IPU achieves a speedup of 1.46×, primarily due to its superior hardware utilization of 22.36%, compared to NVIDIA A100 GPU's utilization of 17.21%. The results also show that hardware utilization is not a cross-platform indicator for estimating the actual training performance of AI accelerators.

Furthermore, even for the same task, AI accelerators also require different batch sizes to achieve optimal training throughput. Figure 1(b) lists the recommended batch size for each AI accelerator when training ResNet-50 v1.5 model with ImageNet. Notably, the batch size for Graphcore Bow-IPU is 64× larger than that for SambaNova SN30. The main reason for this discrepancy is that using different batch sizes ensures that the training data can be stored in on-chip memory as much as possible without exceeding the accelerator's memory capacity. Therefore, selecting appropriate batch sizes is important for performance prediction and is part of this work.

2.3 Why Not Measure Memory Consumption on CPUs or GPUs?

If each AI accelerator requests a specific batch size, is it possible to directly measure the memory consumption of different batch sizes on CPUs/GPUs and then choose a suitable one by comparing the memory capacity of AI accelerators? The answer is no. Firstly, GPUs often lack sufficient memory capacity to support runs with larger batch sizes. For example, the commonly used NVIDIA A100 GPU has 40GB/80GB memory, which is significantly less than the 256GB memory on Graphcore BOW-IPU. Moreover, CPUs have adequate memory capacity, but they cannot support certain floating point formats, such as Bfloat16 and FP8. Hence, we will build a memory estimation model to avoid this limitation.

2.4 Why Not Apply Heuristic Algorithms?

An alternative approach to predict the throughput of DNN training is to employ heuristics [39, 51, 79, 108] based on hardware and model characteristics. This approach involves selecting relevant hardware and model features as input, measuring actual performance as output, and statistically analyz-

ing the potential connection between the two using a heuristic algorithm. Then, the resulting model can be used to predict training performance for new hardware and DNN models.

However, a key challenge of this approach is the need for a certain amount of training data to build an accurate heuristic model. Unfortunately, AI accelerator vendors often only demonstrate a limited number of DNN models [64,81], which is insufficient to train a robust model. Furthermore, the heuristic algorithm tends to assume that the training data and testing data have the same distribution. If a heuristic model is trained on a specific set of AI accelerators, it may not be readily applicable to other AI accelerators with significant differences. Therefore, experimental and analytical approaches are more suitable for performance prediction on AI accelerators than heuristic-based approaches.

3 AI Accelerator Analysis and Inspiration

3.1 Micro-architecture of AI Accelerators

AI accelerators are designed with a primary focus on performance enhancements, such as achieving high parallelism, fast memory transactions, and advanced scalability, et al. For example, ❶ SambaNova SN30 [82] revolutionizes training performance by mitigating redundant data movement through the implementation of a Reconfigurable Dataflow Unit (RDU) architecture [83] with a dataflow-based execution model; ❷ Graphcore Bow-IPU [64] optimizes both computational and data exchange aspects by providing more extensive compute parallelism. This is implemented through a Bulk Synchronous Parallel (BSP) model [47], coupled with high memory bandwidth; ❸ Cerebras CS-2 [57] propels DNN training through the utilization of Colossal Tensor Cores (CTCs) [68], expansive on-chip memory capacity, and robust core-core interconnections, which are imposed and powered into a Wafer-Scale Engine (WSE) processor [56]; ❹ Habana Gaudi2 [76] prioritizes versatility and programmability by employing a heterogeneous architecture comprising a Tensor Processing Core (TPC) cluster and Matrix Math Engine (MME) [17]. Each AI accelerator follows distinct design principles and objectives, delivering its unique set of advantages and strengths.

Table 1 lists the hardware specifications of these AI accelerators, comparing them with an NVIDIA A100 GPU as the baseline. These accelerators exhibit striking disparities across various dimensions, encompassing compute core counts, theoretical peak performance, memory configurations, and interconnect networks. Remarkably, the number of computing

Table 1: Hardware specifications of evaluated AI accelerators

Feature	Nvidia A100	SambaNova SN30	Graphcore Bow-IPU	Cerebras CS-2	Habana Gaudi2
Compute Units	6912 CUDA cores 432 Tensor cores	640 PCUs 640 PMUs	1472 cores	850,000 Cores for 8 worker	MME 24 TPCs
Peak Perf. for AI Compute	312 TFLOPS	~300 TFLOPS	350 TFLOPS	320 TFLOPS/worker	~450 TFLOPS
On-Chip Memory	192 KB L1 40 MB L2	320 MB	900 MB	5 GB/worker	48 MB
Off-Chip Memory	40 GB HBM2	12 TB DDR4	256 GB DDR4	256 GB DDR4	96 GB HBM2E
Process	7nm	7nm	7nm	7nm	7nm
Interconnect	NVLink	RDU direct	IPU Link	SR4 link	RoCE2

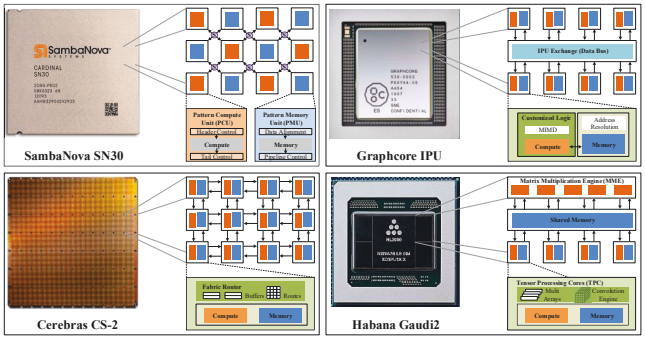

Figure 2: Microarchitecture of SambaNova SN30, Graphcore Bow-IPU, Cerebras CS-2, and Habana Gaudi2.

cores in these accelerators exhibits significant variability. For example, Graphcore Bow-IPU boasts an impressive 1472 cores, whereas Habana Gaudi2 features just one MME and 24 TPCs. Furthermore, substantial distinctions emerge in on-chip and off-chip memory capacities among AI accelerators. For instance, the on-chip memory of each worker in Cerebras CS-2 is 128 times larger than that of NVIDIA A100 GPU, underscoring significant differences in their ability to efficiently localize training data. Moreover, each AI accelerator employs its unique communication network between devices, resulting in distinct distributed performance.

Hardware Analysis of AI Accelerators: Figure 2 outlines the microarchitecture of these accelerators, highlighting significant differences in compute core (e.g., execution units and network-on-chip) and memory hierarchy (e.g., on-chip and off-chip memory) in System-on-Chip (SoC) design. Regarding compute core design, each accelerator employs its own Instruction Set Architecture (ISA), pipeline, and branch prediction. Significantly, *the majority of architectural details have not been publicly disclosed.*

Additionally, all four AI accelerators adopt the commonly used hierarchical memory subsystem, including on-chip and off-chip memory. Concerning on-chip memory design, SambaNova SN30, Graphcore Bow-IPU, and Cerebras CS-2 allocate on-chip memory among compute cores, facilitating fast and local memory access. In contrast, Habana Gaudi2 opts for centralized on-chip memory, such as a shared SDRAM pool, streamlining data transfer and communication among compute cores.

Regarding internetwork design, SambaNova SN30 and Cerebras CS-2 facilitate efficient adjacent communication through on-chip switches and direct core-to-core connection. Graphcore Bow-IPU employs a shared data bus, while Habana Gaudi2 achieves core communication via shared memory.

Software Analysis of AI Accelerators: AI accelerators incorporate specific software execution flows tailored to their hardware architecture. Similar to NVIDIA A100 GPU, Habana Gaudi2 follows a sequential execution in DNN training, where each operation must load the required training

data from off-chip memory to on-chip memory before commencing. In contrast, SambaNova SN30 minimizes off-chip memory access by caching intermediate results within on-chip memory. Graphcore Bow-IPU achieves data locality by employing a memory management technique that allocates training data across different memory hierarchies. Cerebras CS-2 adopts a unique strategy by maintaining all the parameters within on-chip memory at all times to facilitate high bandwidth access.

3.2 Observations and Modeling Trade-offs

Based on these analyses of typical AI accelerators, we have three observations to guide the design of performance model:

Observation 1: These AI accelerators differ significantly in hardware and software implementation, but have some commonalities in the memory subsystem and data management.

These distinguished hardware designs profoundly influence the substantial variations in the performance of these AI accelerators. These unique software designs are intricately linked to the underlying hardware configurations and play a pivotal role in the effectiveness of these accelerators. When attempting to adapt DNN training to a given accelerator, the amalgamation of these specialized designs introduces a higher level of *performance uncertainty*.

While AI accelerators come in various forms, they share similarities in how they organize memory and manage training data. Specifically, aspects such as utilizing on-chip memory and off-chip memory in hierarchical memory subsystem, along with data management methodologies, serve as unifying elements across various AI accelerator designs.

Observation 2: Predicting the execution time of the entire DNN training on AI accelerators is difficult, and hardware simulators offer limited assistance in this regard.

DNN model training is a complex process that involves multiple training stages, which involves a variety of DNN operators and requires different functions from AI accelerators. For example, the computation stage primarily involves compute and memory access units, while the communication stage is more related to network components. As a result, predicting the performance of the entire training process, which comprises distinct stages and operators, can lead to significant deviations.

Traditional hardware simulators, such as Gem5 [24] and ZSim [93], are highly configurable to evaluate different architectures. Real workloads can be executed under full-system mode to collect all details of the runs, including execution time. However, hardware simulators encounter two challenges. (1) The microarchitectures of these accelerators are not made public entirely. (2) Even with open-source hardware, simulators tend to significantly slow down programs [27], typically by a factor of 20× to 40×, making it infeasible to simulate the time-consuming DNN training tasks.

Observation 3: Choosing appropriate batch sizes must consider hardware limits and software optimizations.

The hardware limits of each AI accelerator can constrain the selection of batch size. For instance, SambaNova's on-chip memory is limited to 640 MB, which is designed to store all training data and intermediate results during DNN training, thus greatly limiting the maximum number of samples that can be processed simultaneously.

Based on these observations, we can identify **three trade-offs** that need to be considered in performance modeling. (1) If the execution time of DNN operators can be accurately predicted and appropriately combined, the overall prediction will be more accurate. (2) A uniform hardware abstraction that executes various DNN operators should participate in performance prediction, improving prediction accuracy, and reducing modeling complexity. (3) A uniform software abstraction, which manages training data and affects the selection of batch size under hardware limits, should be included.

4 Centimani: A Performance Predictor

4.1 Centimani Overview

Our performance predictor Centimani is inspired by three key trade-offs. To accommodate various hardware designs and software optimizations and improve the generality and effectiveness of performance modeling, two abstractions are presented: (1) *hardware abstraction* proposes a unified hardware interface that hides fine-grained characterizations of the underlying hardware design and reveals coarse-grained performance behaviors of different resources provided by AI accelerators; (2) *software abstraction* provides a unified mechanism that allows allocation, placement, and accessing of all training data on the hierarchical memory subsystem to meet various computation and transmission demands to enhance the manageability of DNN training on AI accelerators.

In addition, the two abstractions are connected through the *execution modeling*, which maps the DNN training process with associated hyperparameters into an efficient execution on specialized AI accelerators. This mapping is a critical step in revealing the real performance behaviors of each AI accelerator. The modeling process is shown in Figure 3. Firstly, the software abstraction provides control over hardware-related hyperparameters such as batch size (see Section 4.2); secondly, the execution model takes DNN model and optimal hyper-parameters to map three training stages into corresponding training components, and then each training component is further decomposed into a combination of various DNN operators; thirdly, the execution time of all the operators is predicted by decoupled performance models (see Section 4.3), and the overlap of these stages is removed to arrive at a final time prediction (see Section 4.3.4). To sum up, execution modeling provides a bridge between the high-level model description in deep learning frameworks and the low-level

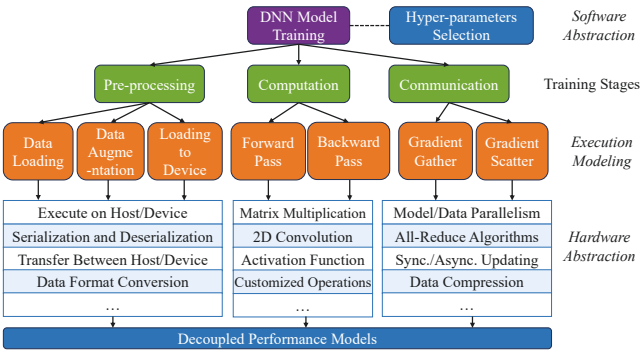

Figure 3: Hardware and software abstractions of AI accelerators, and two corresponding performance modeling components (colored in navy blue) in Centimani.

hardware-specific instructions.

In addition, the predicted execution time can be converted into different performance metrics, such as throughput or price-performance ratio, which can be used by end users (or researchers) to make informed hardware selections.

4.2 Hyper-parameters Selection

Selecting appropriate hyper-parameters is a crucial decision that can significantly impact training throughput, especially when training a DNN model on new AI accelerators. Some default but mismatched hardware-related hyper-parameters can lead to inefficient hardware utilization [33, 74, 118], limited/exceeded parallelism [65, 77, 80], and resource contention [62]. We propose a novel memory estimation model to resolve the hesitancy of choosing the most important hyper-parameter - *batch size*. We first show our preliminary results by comparing training throughputs across multiple batch sizes and then describe the proposed model.

4.2.1 Large Difference across Multiple Batch Sizes

We implement two DNN models, e.g., ResNet-50 v1.5 [55] and Bert-Base [38], on four AI accelerators to study the difference in training throughput of using multiple batch sizes. The training throughput of these two models is collected respectively.

Table 2 summarizes the speedups of using optimal batch sizes compared to default batch sizes used on the GPU platform (NVIDIA A100 40GB) for four AI accelerators. It is obvious that it is unnecessary or even infeasible to tenaciously use default batch sizes on all AI accelerators. For example, training ResNet-50 on SambaNova SN30 or training Bert-Base on SambaNova SN30 and Cerebras CS-2 with default batch sizes will cause out-of-memory (OOM) errors. In addition, the default batch sizes cannot achieve the best training throughput, where the training throughput with optimal batch sizes of the two models can reach on average 4.68× and up to 9.31× higher training throughput than that with default batch sizes. In other words, the result further proved that choosing appropriate batch sizes is critical in using AI accelerators.

4.2.2 Memory Model

Based on Observation 1 in Section 3.2, all AI accelerators employ a hierarchical memory subsystem and manage training data through their memory management mechanisms. The observation inspires our memory model to identify data placement across multilevel memory and differentiate behaviors in their properties. Unlike conventional CPU/GPU systems that put training data in the same memory hierarchy, e.g., placing all training data on DRAM or global memory for CPUs and GPUs, our memory model considers the configuration of multilevel memory and predicts the respective memory consumption of each memory level by introducing *data classification* and *memory estimation model* for a given batch size. From this, our memory model selects the optimal batch size that maximizes memory efficiency and avoids exceeding hardware limits using *batch size selection*.

Data Classification Data classification is designed on top of mainstream training frameworks, such as PyTorch, TensorFlow, and MXNet, which are supported by all AI accelerators and organize the execution of DNN training through a structural representation, known as a *computation graph*.

In a computation graph, each node represents the invocation of a mathematical operator, such as matrix multiplication or concatenation, which takes tensor variables (multidimensional arrays) as input and output. Each mathematical operator incurs a certain memory overhead by math libraries (i.e., cuBLAS and cuDNN for NVIDIA GPUs). Each operator may contain numerical learnable parameters/tensors (i.e., weights and gradients). Additionally, execution dependencies are specified by edges that point from the output of one operator to the input of another. When a batch size for a DNN model is chosen, all tensor shapes involved in the computation graph are fixed. This characteristic is exploited by our memory model.

To fully understand how memory is consumed during DNN training, we classify the allocated data into five categories:

Table 2: Best batch sizes for AI accelerators and speedups over default setting (256 for ResNet-50 and 3 for Bert-Base) on A100 GPU.

AI Accelerators	Models/ Dataset	Best Batch Size	Speedup Over Default Setting
SambaNova SN30	Model: ResNet-50 Dataset: ImageNet	32	Out-of-memory
Graphcore Bow-IPU		1024	4.17x
Cerebras CS-2		480	2.75x
Habana Gaudi2		512	1.94x
SambaNova SN30	Model: Bert-Base Dataset: SQuAD	1	Out-of-memory
Graphcore Bow-IPU		16	9.31x
Cerebras CS-2		2	Out-of-memory
Habana Gaudi2		12	5.24x

• Input/Output Tensors, which include input tensors and output tensors (such as activations). Activations are further computed to forward output and output gradients.
• W&B Tensors, which include weights and biases of operations. They are learnable parameters of training.
• Gradient Tensors, which include gradients, weight gradients, and gradients for momentum. They are computed under backward propagation for updating and calculating weights in the next iteration.
• Algorithm-related Tensors, which include variables used for specific algorithms such as mixed precision training [78] and stabilized SGD [61].
• Ephemeral Tensors, which include temporary variables used in operation implementation such as mathematical library and communication reservation used for multi-device training.

Altogether, such categorization offers a structured classification for comprehending the multifaceted roles and contributions of tensors in the memory consumption of DNN training.

Memory Estimation Model To build the memory estimation model, it is essential to consider both the memory consumption of various training data and their mapping to hierarchical memory system. The workflow of the memory estimation model can be divided into two parts as follows.

The first part is to traverse the computation graph of DNN model and predict the memory consumption of each training data in a fine-grained manner. For example, given a model with N layers, we traverse each model layer in turn and infer the memory consumption of each operator in each model layer. For each model layer (e.g., layer L), the size of the activations of the previous layers (layer $L-1$) and the weights of the current layer (layer L) are estimated according to their dimension and data types in the forward pass. In addition, for the backward part, the size of the gradients of the next layer (layer $L+1$), the weight gradients and the gradients of the current layer (layer L) are inferred. Meanwhile, additional memory consumption of all involved operators (such as communication reservation and temporary variables) is also taken into account. Based on our observation long-live temporary tensors are rare, so we use the peak amount of all allocated tensors to avoid out-of-memory issues.

The second part is to estimate the memory consumption of each memory level. The process consists of two steps: step 1 classifies different training data into various categories using data classification; step 2 maps various data categories to the hierarchical memory where they are located according to the data management policies of AI accelerators. It is worth noting that the matching of different data categories and multilevel memory completely depends on the software execution flow and memory management mechanism of AI accelerators and therefore varies greatly, which can also be obtained from their software design manual and SDK tools [6, 10, 11, 15]. Then, the total memory consumption of on-chip and off-chip memory can be calculated.

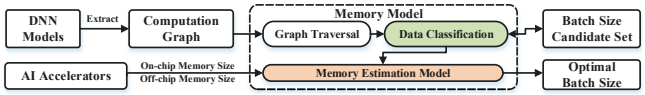

Figure 4: Workflow of memory modeling and selection of batch size.

In summary, the inputs of the memory estimation model are the DNN model and the batch size to be evaluated, and the output is the memory consumption of each memory level on various AI accelerators.

Batch Size Selection Figure 4 depicts the workflow of batch size selection in memory model. Given a DNN model, we extract its computation graph. When evaluating a batch size in a candidate set, we traverse the computation graph, classify different training data, estimate the memory consumption of each training data, and calculate the memory consumption of each memory level using the memory estimation model. Finally, we select the optimal batch size in all candidate sets, which can maximize computational parallelism and stay within the hardware limits of each kind of memory.

4.3 Decoupled Performance Models

To make accurate predictions for DNN training, Centimani introduces the decoupled performance models, which include three models and predict the execution time of multiple training phases separately, including data loading/pre-processing stage, computation stage, and communication stage. In the end, the overlap of these stages is removed to arrive at the final prediction.

4.3.1 Data Loading/Pre-processing Model

Data loading/pre-processing stage of DNN training is responsible for fetching training samples from secondary memory storage and applying additional transformations, such as decoding, augmentation, and batching, to the input data. There is a significant difference between GPU training and AI accelerator training in the pre-processing stage. For GPU training, the pre-processing stage is usually performed on GPU using NVIDIA Data Loading Library [13], while for AI accelerators, the pre-processing stage tends to occur on the host/CPU side [43] due to the lack of special hardware modules.

Therefore, we modify the pre-processing code of GPU training so that it can be executed and measured on the host/CPU side. We approximate the data loading time in AI accelerator training by collecting the data loading time in GPU training and measuring memory bandwidths between host/CPU and GPU/AI accelerators. Additionally, we estimate the pre-processing time in AI accelerator training by collecting the pre-processing time in GPU training and comparing the peak performance of CPU between GPU and AI

accelerator training. In summary, the data loading and preprocessing time are estimated proportionally based on the time of GPU training.

4.3.2 Computation Model

To predict the computation stage of DNN training on a given AI accelerator, we first break down the time that is required for an iteration into the time of individual operators, which can be expressed as follows:

$$T_{batch} = \sum_{i=1}^{L} \sum_{j=1}^{O(i)} E(i,j) \tag{1}$$

where L is the number of layers in a DNN model, $O(i)$ is the number of operators in layer i, and $E(i,j)$ is the batched execution time of $j-th$ operator in layer i.

Therefore, the key to predicting training time lies in accurately predicting the execution time of each operator. Previous work [54, 71, 85] has attempted to predict the execution time of an operator, like $E(i,j)$ in Equation 1. Most work has been based on the assumption that the execution time is linearly related to the number of floating point operations required. However, this assumption is not valid, especially when predicting the execution time of the same operator on different accelerators. For instance, we observe that Graphcore Bow-IPU exhibits 1.91× the performance of NVIDIA A100 GPU on convolution operation-based ResNet50 v1.5 model, despite having only a 10% difference in their floating-point peak performance. As a result, traditional methods are prone to significant prediction errors.

We present an alternative approach that combines experimental and analytical methods to predict the execution time of each operator. Specifically, we categorize operators into two groups: *common* and *uncommon* operators, and predict their execution time using corresponding methods. For common operators, which are those included in the pre-defined micro-benchmark set, we directly measure execution time by constructing synthetic input data with a specific shape from the computation graph. For uncommon operators, which are those not included in the micro-benchmark set or belonging to customized operators, we estimate execution time by collecting the number of floating-point operations and arithmetic intensity (A.I.) of the operator in GPU training and applying cache-aware roofline model [58]. The approach is as follows:

$$E(i,j) \approx \begin{cases} Kernel(E(i,j), Input\ shape), & \text{if } E(i,j) \in Microbenchmark\ Set \\ Roofline(Collected\_FLOPs, A.I.), & \text{otherwise} \end{cases} \tag{2}$$

where $Kernel$ is the operator to $E(i,j)$, $Collected\_FLOPs$ is the collected number of floating-point operations required to process $E(i,j)$, $A.I.$ is the arithmetic intensity of $E(i,j)$.

In addition, we also consider the overhead of kernel launch, although for the accelerator all kernels are offloaded before execution, so this overhead is negligible.

4.3.3 Communication Model

There are two main parallelism modes for distributed training [23]: *model parallelism* and *data parallelism*. The two parallelism modes exhibit distinct patterns. For model parallelism, each device requires results from other devices based on model partitioning. In contrast, data parallelism involves independent computation on each device, and therefore no communication is required during the computation stage.

For model parallelism, we directly collect the communication traffic during the computation stage in multi-GPU training and simulate it as the communication traffic in multi-device training on AI accelerators. We also measure the communication bandwidth among devices to calculate the data transfer time during computation. The computation time can be achieved as follows:

$$T_{device}(d) = \sum_{i=L(d)}^{L(d+1)} \sum_{j=1}^{O(i)} E(i,j) + \frac{Traffic(d,d+1)}{Bandwidth(d,d+1)} \tag{3}$$

where layer $L(d)$ to layer $L(d+1)$ are part of the model assigned to device d by model partition graph, $Traffic(d,d+1)$ is the collected communication traffic from device d to device $d+1$ in multi-GPU training, and $Bandwidth(d,d+1)$ is the measured bandwidth between device d and device $d+1$. The final computation time is the longest path in the model partitioning graph.

Once the computation stage is complete, the device must communicate its local gradient to the global parameters. This communication can be accomplished using either *synchronous* [34] or *asynchronous* [117] learning algorithms. In synchronous learning, every device must wait for all devices to transmit all parameters before the next training iteration. In asynchronous learning, each device is allowed to transmit its gradients once they are calculated, enabling the global model to be updated without waiting for other devices. Therefore, the time required for the communication phase can be modeled as follows:

$$T_{comm}(d) = \begin{cases} T_{device}(d) + \frac{Size(Gradients\ in\ Device\ d)}{Bandwidth(Device\ d, Server)}, & \text{if Async. learning} \\ T_{device}(d) + All\_Reduce(Gradients), & \text{if Sync. learning} \end{cases} \tag{4}$$

where $Bandwidth(Device\ d, Server)$ is the measured bandwidth between device d and parameter server, $Size(Gradients\ in\ Device\ d)$ is the size of gradients on device d, and $All\_Reduce(Gradients)$ is the time required to execution the all-reduce operation.

Finally, we also include data compression techniques [21, 59] that are used to decrease communication traffic in the communication model. Two primary compression methods are quantization [42], which represents data using fewer bits, and sparsification [121], which removes the number of zero elements. Our model takes this communication optimization into account, and the new communication traffic is determined as $Size'(Gradients) = Size(Compress(Gradients))$.

4.3.4 Overlap Removal

Two overlaps are considered in the performance model.
Data Loading/Pre-processing and Computation Overlap: The execution pipelines of GPU training and AI accelerator training differ. For GPU training, data loading can be performed simultaneously with pre-processing and other stages, thereby hiding the overhead of data loading. Conversely, for AI accelerator training, other stages execute simultaneously with data loading and pre-processing stages.
Computation and Communication Overlap: For PS-based distributed DNN training, the computation stage and communication stage can be overlapped [105, 113]. The computation/communication overlap mechanism uses the stale-synchronous parallel synchronization model to overlap the communication of previous iterations with the computation of the current iteration, resulting in the final execution time being the greater of the two stages.

5 Implementation Details

5.1 Memory Model

Formally, the computation graph of a DL model is represented as a directed acyclic graph (DAG), where $DAG = (\overrightarrow{V}, \overrightarrow{E})$. $\overrightarrow{V} = \{u_1, ..., u_n\}$ is the vertex set and each vertex u_i is an operator. $\overrightarrow{E} = \{(u_i, u_j), ...\}$ is the set of directed edges. A directed edge (u_i, u_j) delivers an output tensor of u_i to u_j as input and specifies the dependency between two operators. The DAG is usually a static computation graph or can be converted from a dynamic computation graph.

Let $S = < u_1, ..., u_n >$ be a topological ordering of the operators in DAG that satisfies the condition $\{< u_i, u_j > \notin E \mid i > j\}$. We refer to S as the operator schedule, which represents the actual execution order of the operators. The schedule S can be obtained in GPU training as a reference.

Given a batch size (BS) from the candidate set, our memory model traverses the computation graph DAG sequentially according to the schedule S, estimating the memory consumption of the input/output, weights, gradients, and intermediate tensors used by the math library of each operator. These tensors are then mapped to on-chip and off-chip memory, and the total memory consumption of each type of memory is calculated as $Est_{on-chip}(BS)$ and $Est_{off-chip}(BS)$, respectively.

Finally, we choose the largest batch size that satisfies the hardware limits ($Real_{on-chip}$ and $Real_{off-chip}$) by comparing the estimated memory consumption with the following objective function (W_{onchip} is usually set to 10 in our modeling):

$$\begin{aligned} \min_{BS} \quad & W_{onchip}(Real_{on-chip} - Est_{on-chip}(BS))^2 + (Real_{off-chip} - Est_{off-chip}(BS))^2 \\ \text{s.t.} \quad & Real_{on-chip} > Est_{on-chip}(BS) \\ & Real_{off-chip} > Est_{off-chip}(BS) \\ & BS \in Candidate\ set\ for\ batch\ size \end{aligned} \tag{5}$$

In other words, we seek to find a batch size BS that minimizes the objective function under hardware and candidate constraints. This ensures that the chosen batch size is feasible and that as much on-chip memory as possible can be utilized.

5.2 Micro-benchmarks, Roofline Models, and Communication Primitives

Micro-benchmarks In this study, we define DL micro-benchmarks as the fundamental building blocks of the computation model 4.3.2. We focus on the most commonly used kernels that underlie the majority of DL workloads, including generic matrix multiplication (GEMM), convolution, ReLU, LSTM, and transformer operators. These kernels are designed to accept any input shapes and data types, including single-precision, half-precision formats, and FP8, and are implemented on each AI accelerator. Table 3 selectively presents the performance of two kernels on four AI accelerators and compares them with the NVIDIA A100 GPU. The GEMM kernel involves half-precision multiplication of two square matrices, each of these has a width of 1k. ReLU is applied to 3-D tensors with a batch size of 128 and dimensions of 128×128 for the other two axes.
Roofline Models In addition, a cache-aware roofline model for each AI accelerator is built by collecting peak performance and memory bandwidth of on-chip and off-chip memory.

Figure 5 displays cache-aware roofline models constructed for four AI accelerators. The x-axis represents the arithmetic intensity of operations, and the y-axis represents the achievable performance. The cache-aware roofline model allows us to distinguish where data reside and predict performance with different rooflines. Moreover, Figure 5 depicts three uncommon operators (Batch Normalization, Linear Transformation, and MaxPooling2D) that are not included in micro-benchmarks. Each operator is tested in two precisions (single precision and half precision). Thus, the predicted performance of each operator is the intersection point of the vertical line and the corresponding roofline.
Communication Primitives To investigate the communication cost, we collect the transmission bandwidth between devices in each system. Additionally, Table 3 also demonstrates a communication primitive (all-reduce) across all systems. All-reduce exchanges 240MB of data on each device and calculates its communication bandwidth.

Table 3: Performance results of two DNN operators and one communication primitive.

AI Accelerators	Kernel	TFLOPS	Kernel	TFLOPS	Kernel	GB/s
NVIDIA A100	GEMM	291.83	ReLU	0.62	All-reduce	100.61
SambaNova SN30		272.37		7.43		64.17
Graphcore Bow-IPU		295.32		5.59		103.62
Cerebras CS-2		307.53		5.87		134.75
Habana Gaudi2		430.58		2.27		85.23

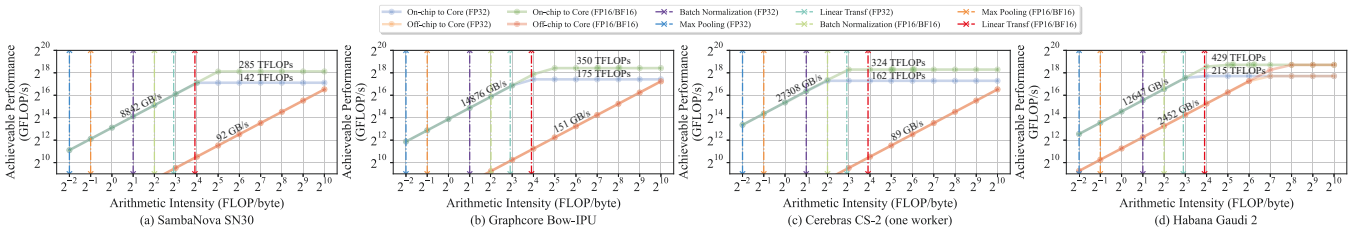

Figure 5: Cache-aware roofline models built on four AI accelerators, and three uncommon operators using single- and half-precision are depicted.

6 Evaluation

6.1 Setup

Platforms and Formats: We conduct experiments on four AI accelerators. (1) SambaNova DataScale SN30-8R rack system, which consists of two DataScale SN30-8 nodes interconnected with an InfiniBand-based fabric. Each SN30-8 node includes eight SambaNova Cardinal SN30 Reconfigurable Dataflow Units (RDUs) and a host module with 1.5TB of memory and 128 cores. (2) Graphcore Bow Pod16 system is powered by four inter-connected Bow-2000s. Each Bow-2000 features four Graphcore Bow-IPU (Intelligence Processing Unit) processors. (3) Cerebras CS-2 cluster is powered by a wafer-scale engine. Each wafer-scale engine features eight worker nodes. (4) Habana Gaudi2 system consists of 16 Gaudi2 accelerators. The Gaudi2 architecture is heterogeneous, with two kinds of engines: Matrix Multiplication Engines (MMEs) and a fully programmable Tensor Processor Core (TPC) cluster.

Models and Dataset: We evaluate Centimani on six different DNN models using a public dataset, with the specifics of each model and dataset provided in Table 4. ResNet-50 v1.5 model [55] is an escalated version of the original ResNet-50 model. U-Net [97] is a convolutional neural network used for biomedical image segmentation. CANDLE-UNO [107] is a pharmacological model that aims to enable precision medicine for cancer treatment. Bert-large [38] is a transformer-based model that is pretrained on a large English language corpus dataset in a semi-supervised manner. BraggNN [72] is a scientific model designed for High Energy X-ray Diffraction Microscopy (HEDM) modeling. OpenAI GPT-2 1.5B [86] is the 1.5B parameters version of GPT-2 and a transformer-based language model created and released by OpenAI.

Implementation and Baselines: This work is implemented based on PyTorch 2.12.1. We implement a memory model and decoupled performance models as extended modules in PyTorch. Micro-benchmarks utilize a combination of PyTorch implementations and vendor-provided benchmarks.

We compare Centimani with three solutions:
❶ Roofline model [106], which predicts the achievable performance relative to hardware limits and application characteristics.
❷ Hardware utilization-based prediction [18], which uses hardware utilization of GPU training and the theoretical performance of AI accelerators to predict execution time.
❸ Similarity-based benchmarking [75, 104], which identifies the most similar model to the target model in the existing benchmarks and uses its performance as the predicted performance of the target model.

Table 4: Evaluated models and dataset

Model	ResNet-50 v1.5 [55]	U-Net [97]	CANDLE-UNO [107]
Dataset	ImageNet	LGG Segmentation	CCLE
Model	Bert-large [38]	BraggNN [72]	OpenAI GPT-2 1.5B [86]
Dataset	Wikipedia	Frames-exp4train	OpenWebText2

6.2 Prediction Accuracy

6.2.1 Single-device Training

Performance Prediction: Figure 6 presents the prediction accuracy of Centimani for single-device training across four different accelerators. Each subfigure compares the predictions of Centimani against three other solutions for all evaluated DNN models.

Centimani achieves an average prediction accuracy of 93.1% (with a range of 98.4% to 82.6%) across all AI accelerators and DNN models, while the roofline model, hardware utilization-based prediction, and similarity-based benchmarking achieve an average prediction accuracy of 28.4%, 37.8%, and 57.2%, respectively. We make the following observations: (1) By leveraging model topology and training stages, Centimani provides a more accurate performance prediction than application-agnostic solutions such as the roofline model and hardware utilization-based prediction. (2) Centimani outperforms similarity-based benchmarking on U-Net, CANDLE-UNO, BraggNN, and OpenAI GPT-2 1.5B models, which are not included in the benchmark set provided by vendors. Similarity-based benchmarking leads to low prediction accuracy due to the significant differences between the models evaluated and the benchmark set. (3) Centimani performs much better than the prediction based on hardware utilization in Graphcore Bow-IPU, because it has distinct hardware

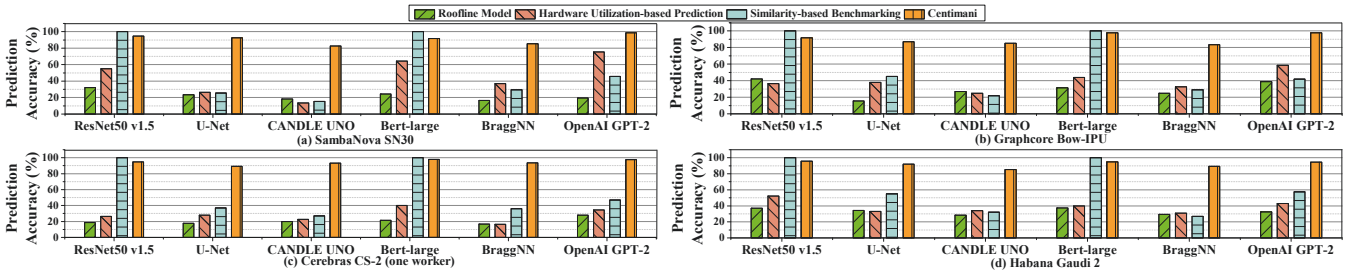

Figure 6: Prediction accuracy of Centimani for single-device training on six DNN models across four AI accelerators.

design and software optimization, resulting in completely different hardware utilization. (4) For models with large memory footprints, such as CANDLE-UNO and BraggNN, Centimani achieves better accuracy on SambaNova SN30 because the memory model accurately predicts memory consumption and selects the most appropriate batch size.

Accelerator Selection: The primary use case of Centimani is to assist deep learning users in making informed decisions when selecting AI accelerators. In the following two scenarios, we demonstrate how Centimani leverages the predicted performance, enabling users to make the correct selection based on their specific training objectives.

One scenario that deep learning users may encounter is deciding whether it is worthwhile to port their DNN models to other AI accelerators or determining which AI accelerator can provide the best training throughput. Figure 7(a) presents Centimani's throughput predictions for three DNN models: ResNet-50 v1.5, Bert-large, and OpenAI GPT-2 1.5B, on four AI accelerators, normalized to the measured throughput on NVIDIA A100 GPU. Notably, both Graphcore Bow-IPU, Cerebras CS-2, and Habana Gaudi2 always perform better than NVIDIA A100 GPU with an average speedup of 1.54×, 1.41× and 1.73×, respectively. In contrast, SambaNova SN30 only provides a marginal throughput improvement with an average speedup of 1.22×. Specifically, Habana Gaudi2 outperforms the other accelerators on ResNet-50 v1.5 model composed of dense matrix multiplication, while Graphcore Bow-IPU achieves superior performance on Bert-large and OpenAI GPT-2 1.5B models, which rely on transformer-based computations.

Another scenario for deep learning users is determining which accelerator offers the best price-performance ratio. This ratio is defined as the throughput divided by the hourly cost of renting the hardware. To calculate the price-to-performance ratio, we collect the hourly rental costs of a single device for each AI accelerator from their cloud platforms [3, 5, 9, 12, 16]. The hourly rental costs[1] are 2.87/device/hr for NVIDIA A100 GPU, 1.74/device/hr for SambaNova SN30, 1.21/device/hr for Graphcore Bow-IPU, 2.89/worker/hr for Cerebras CS-2, and 1.64/device/hr for Habana Gaudi2, respectively. Figure 7(b) displays Centimani's predictions on the price-performance ratio, all normalized to the price-performance ratio of NVIDIA A100 GPU. In particular, Graphcore Bow-IPU offers a significant improvement in the price-performance ratio, owing to its low rental cost and excellent training throughput.

[1] Please note that, over time, the actual price may be different from the price published on the websites.

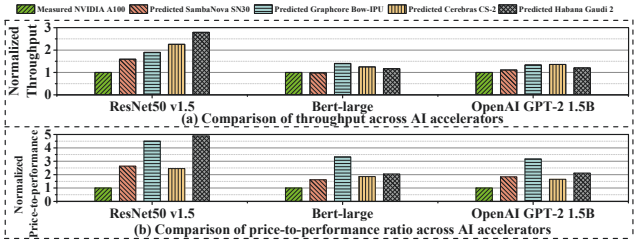

Figure 7: Throughput and price-performance ratio of training three DNN models on different AI accelerators, normalized to the measurement on NVIDIA A100 GPU.

6.2.2 Multiple-device Training

We also evaluated the scaling performance by predicting the training throughput of the U-Net model on two, four, and eight devices for each AI accelerator. To model the amount of communication of each device, we use the communication model in Centimani, and the communication time is simulated using the communication primitive (all-reduce).

Figure 8 displays the results normalized to the throughput of single-device training. As the number of devices increases, all accelerators achieve higher training throughput. Cerebras CS-2 system has slightly stronger scalability than other AI accelerator systems, owing to its higher communication bandwidth between devices. Overall, our model achieves an average prediction accuracy of 90.4% for training on multiple devices.

Summary. Centimani provides highly accurate predictions with an average error of 6.9% and 9.6% for single-training and multiple-device training, respectively. It is noteworthy that even in cases where there are prediction errors, Centimani still correctly predicts the relative ordering of all AI accelerators in terms of their throughput and price-performance ratio. This capability empowers users to make informed decisions based

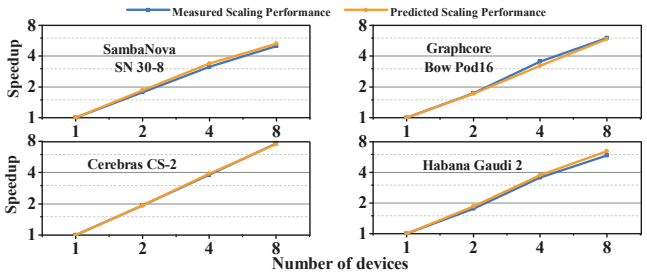

Figure 8: Normalized speedups of training U-Net model on two, four, and eight devices for three accelerators.

on their specific needs for cost, pure throughput, or scalability.

6.3 Breakdown of Performance Prediction

6.3.1 Memory Consumption Prediction

We investigate the impact of memory consumption prediction and batch size selection on Centimani's end-to-end prediction. In our evaluation, Centimani selects the optimal batch size based on memory consumption prediction and hardware limits and then feeds it as input to the decoupled performance models to predict training throughput. In contrast, the ground truth batch sizes are manually selected by enumeration.

Table 5 provides a comparison of the batch sizes selected by Centimani and those manually selected on four AI accelerators. Our results demonstrate that Centimani's memory consumption prediction can accurately determine the appropriate batch sizes for different models, achieving a matching accuracy of $\frac{19}{24}$. Specifically, for models such as ResNet-50 v1.5 and BraggNN, where Centimani predicts smaller batch sizes than those manually selected, the impact on training throughput is minimal, with performance differences of no more than 4.6%. This finding underscores the strength of Centimani's batch size selection: It can automatically predict appropriate batch sizes for most cases and prevent Out-Of-Memory (OOM) errors caused by overly large batch sizes.

Table 5: A comparison between predicted batch sizes and manually selected batch sizes.

	ResNet-50 v1.5		U-Net		CANDLE UNO	
	Selected	Predicted	Selected	Predicted	Selected	Predicted
SambaNova SN30	16	16	2	2	8	8
Graphcore Bow-IPU	1024	512	4	4	512	512
Cerebras CS-2	64	64	2	2	256	256
Habana Gaudi2	64	64	2	2	230	230
	BERT-large		BraggNN		OpenAI GPT-2 1.5B	
	Selected	Predicted	Selected	Predicted	Selected	Predicted
SambaNova SN30	256	256	1024	512	16	16
Graphcore Bow-IPU	16	16	2048	1024	32	32
Cerebras CS-2	16	16	1640	1560	16	16
Habana Gaudi2	16	16	1440	1240	2	2

6.3.2 Computation Prediction

One of the most important aspects in improving prediction accuracy is to accurately predict the execution time of each operator involved in DNN training. In our tests, for the computation model 4.3.2, common operators, which accounted for 58% of all operators, can be predicted by running micro-benchmarks directly, while uncommon operators, which accounted for 42% of all operators, use the predefined cache-aware roofline models. In terms of contribution to the final prediction result, common operators predict 44.6% of the total training time while uncommon operators predict 25.7%, which also shows that these two types of predictions are indispensable to the final result.

6.3.3 Communication Prediction

To evaluate the communication capabilities of various accelerators, we conduct a communication primitive using all-reduce operation to measure the communication bandwidth and time incurred across multiple devices. Specifically, we collected a typical communication pattern from U-Net model consisting of $16,777,216$ floating-point numbers in single precision and ran the communication primitive on eight devices to obtain the communication bandwidths. Our results show that the average communication bandwidth of SambaNova SN30-8R system, Graphcore Bow Pod16 system, Cerebras CS-2 system, and Habana Gaudi2 system is 10.24 GB/s, 54.33 GB/s, 352.37 GB/s, and 62.14 GB/s, respectively.

6.4 Overhead of Performance Modeling

We study the overhead of Centimani. The major overhead includes (1) measuring memory capacity, memory bandwidth, transmission bandwidth between CPU and accelerators, and communication bandwidth between multiple devices, (2) constructing microbenchmarks and roofline model for each accelerator, (3) collecting memory consumption for math library, and (4) traversing computation graph to make memory consumption and performance prediction. It should be noted that the first three components of this overhead occur only once for each accelerator, while the fourth component occurs once for each DNN model. Therefore, the overhead of Centimani is significantly lower than the overhead of porting the DNN model to all accelerators.

7 Related Work

DNN Performance Models for Different Hardware. Previous research has investigated performance models for DNN training on various hardware, including GPUs [66, 73, 95, 100, 109, 111, 112, 120], CPUs [19, 20, 110], and FPGAs [29, 49, 52, 118].

Geoffrey et al. [45] proposed a runtime-based computational performance predictor named Habitat, which helps users make informed and cost-efficient GPU selections for DNN training. However, Habitat uses the performance of one typical GPU to derive the performance of other GPUs, this approach is not suitable for AI accelerators that are very different from GPUs and each other.

To predict the execution time of DNN models, Ruohan et al. [108] proposed a machine learning-based solution that captures low-level hardware-dependent information. However, this approach relies on having sufficient training data, which is not always available for AI accelerators. In contrast, Xiaofan et al. [119] presented an FPGA-based DNN accelerator model that imitates micro-architecture and pipeline structure to discover the drawbacks of existing designs. Nevertheless, this fine-grained performance modeling lacks generality and has substantial overhead.

DNN Benchmark. A significant body of prior work has focused on benchmarks for DNN training [44, 75, 104], providing valuable insights into DNN training performance. However, these studies mainly focus on comparing different algorithms and hardware, rather than performance prediction. In contrast, the microbenchmarks used by Centimani are primarily designed to predict the execution time of each operation in a DNN model. Besides, a cache-aware roofline model is introduced to overcome the shortcomings of the microbenchmarks that cannot cover all operations. By doing so, Centimani provides better performance prediction and assists users in making informed hardware selections.

8 Conclusions

We introduce Centimani, a novel performance predictor that assists deep learning researchers and practitioners in selecting an AI accelerator for training their DNN models. The primary idea behind Centimani is to combine experimental and analytical approaches to predict the execution time of DNN training on each AI accelerator. We evaluate Centimani with six typical DNN models on three AI accelerators and find that it makes execution time predictions with an average accuracy of 93.1% and 90.4% for single-device training and multiple-device training, respectively. Finally, we demonstrate two decision metrics (throughput and price-performance ratio), and Centimani correctly guides which accelerator should be chosen. After that, the introduction of new GPUs and AI accelerators is our future plan.

Acknowledgment

This research was funded in part by and used resources at the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. It was also partially supported by the U.S. National Science Foundation under Contract CNS-2125732.

References

[1] ACM Gordon Bell Special Prize. https://www.acm.org/media-center/2022/november/gordon-bell-special-prize-covid-research-2022.

[2] Ai index report 2023 – artificial intelligence index. https://aiindex.stanford.edu/report/.

[3] Amazon aws. https://aws.amazon.com.

[4] Aurora Supercomputing System. https://www.alcf.anl.gov/aurora.

[5] Cerebras cloud. https://www.cerebras.net/product-cloud/.

[6] Cerebras documents. https://docs.cerebras.net/en/1.8.0/.

[7] El Capitan. https://computing.llnl.gov/about/newsroom/road-el-capitan-1.

[8] Frontier. https://www.olcf.ornl.gov/frontier/.

[9] Graphcloud cloud. https://www.graphcore.ai/ipus-in-the-cloud.

[10] Graphcore documents. https://docs.graphcore.ai/en/latest/.

[11] Habana gaudi documentation. https://docs.habana.ai/en/latest/.

[12] Habana gaudi2. https://habana.ai/.

[13] Nvidia dali. https://docs.nvidia.com/dali/index.html.

[14] Omdia market radar: Top ai hardware startups. https://omdia.tech.informa.com/OM026802/Omdia-Market-Radar-Top-AI-Hardware-Startups.

[15] Sambanova documentation. https://docs.sambanova.ai/home/latest/index.html.

[16] Sambanova pricing. https://sambanova.ai/blog/subscription-pricing/.

[17] Tensor Processing Core and Matrix Math Engines. https://docs.habana.ai/en/latest/Gaudi_Overview/Gaudi_Architecture.html.

[18] Yehia Arafa, Abdel-Hameed Badawy, Gopinath Chennupati, Atanu Barai, Nandakishore Santhi, and Stephan Eidenbenz. Fast, accurate, and scalable memory modeling of gpgpus using reuse profiles. In *Proceedings of the 34th ACM International Conference on Supercomputing*, pages 1–12, 2020.

[19] Ammar Ahmad Awan, Jereon Bédorf, Ching-Hsiang Chu, Hari Subramoni, and Dhabaleswar K Panda. Scalable distributed dnn training using tensorflow and cuda-aware mpi: Characterization, designs, and performance evaluation. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 498–507. IEEE, 2019.

[20] Ammar Ahmad Awan, Hari Subramoni, and Dhabaleswar K Panda. An in-depth performance characterization of cpu-and gpu-based dnn training on modern architectures. In *Proceedings of the Machine Learning on HPC Environments*, pages 1–8. 2017.

[21] Youhui Bai, Cheng Li, Quan Zhou, Jun Yi, Ping Gong, Feng Yan, Ruichuan Chen, and Yinlong Xu. Gradient compression supercharged high-performance data parallel dnn training. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 359–375, 2021.

[22] Jan Balewski, Zhenying Liu, Alexander Tsyplikhin, Manuel Lopez Roland, and Kristofer Bouchard. Time-series ml-regression on graphcore ipu-m2000 and nvidia a100. In *2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 141–146. IEEE, 2022.

[23] Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)*, 52(4):1–43, 2019.

[24] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.

[25] Léon Bottou. Stochastic gradient descent tricks. *Neural Networks: Tricks of the Trade: Second Edition*, pages 421–436, 2012.

[26] Oliver Bringmann, Wolfgang Ecker, Ingo Feldner, Adrian Frischknecht, Christoph Gerum, Timo Hämäläinen, Muhammad Abdullah Hanif, Michael J Klaiber, Daniel Mueller-Gritschneder, Paul Palomero Bernardo, et al. Automated hw/sw co-design for edge ai: state, challenges and steps ahead. In *Proceedings of the 2021 International Conference on Hardware/Software Codesign and System Synthesis*, pages 11–20, 2021.

[27] Anastasiia Butko, Rafael Garibotti, Luciano Ost, and Gilles Sassatelli. Accuracy evaluation of gem5 simulator system. In *7th International workshop on reconfigurable and communication-centric systems-on-chip (ReCoSoC)*, pages 1–7. IEEE, 2012.

[28] Chi-Chung Chen, Chia-Lin Yang, and Hsiang-Yun Cheng. Efficient and robust parallel dnn training through model parallelism on multi-gpu platform. *arXiv preprint arXiv:1809.02839*, 2018.

[29] Yao Chen, Jiong He, Xiaofan Zhang, Cong Hao, and Deming Chen. Cloud-dnn: An open framework for mapping dnn models to cloud fpgas. In *Proceedings of the 2019 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 73–82, 2019.

[30] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, and Tianqi Tang. A survey of accelerator architectures for deep neural networks. *Engineering*, 6(3):264–274, 2020.

[31] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro*, 41(2):29–35, 2021.

[32] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition. *Training*, 100(101):102, 2017.

[33] Henggang Cui, Hao Zhang, Gregory R Ganger, Phillip B Gibbons, and Eric P Xing. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *Proceedings of the eleventh european conference on computer systems*, pages 1–16, 2016.

[34] Dipankar Das, Sasikanth Avancha, Dheevatsa Mudigere, Karthikeyan Vaidynathan, Srinivas Sridharan, Dhiraj Kalamkar, Bharat Kaul, and Pradeep Dubey. Distributed deep learning using synchronous stochastic gradient descent. *arXiv preprint arXiv:1602.06709*, 2016.

[35] Emmanuel De Bézenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124009, 2019.

[36] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[37] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[38] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[39] Diego Didona, Francesco Quaglia, Paolo Romano, and Ennio Torre. Enhancing performance prediction robustness by combining analytical modeling and machine learning. In *Proceedings of the 6th ACM/SPEC international conference on performance engineering*, pages 145–156, 2015.

[40] Jens Domke, Emil Vatai, Aleksandr Drozd, Peng ChenT, Yosuke Oyama, Lingqi Zhang, Shweta Salaria, Daichi Mukunoki, Artur Podobas, Mohamed WahibT, et al. Matrix engines for high performance computing: A paragon of performance or grasping at straws? In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1056–1065. IEEE, 2021.

[41] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655. PMLR, 2014.

[42] Nikoli Dryden, Tim Moon, Sam Ade Jacobs, and Brian Van Essen. Communication quantization for data-parallel training of deep neural networks. In *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)*, pages 1–8. IEEE, 2016.

[43] Murali Emani, Venkatram Vishwanath, Corey Adams, Michael E Papka, Rick Stevens, Laura Florescu, Sumti Jairath, William Liu, Tejas Nama, and Arvind Sujeeth. Accelerating scientific applications with sambanova reconfigurable dataflow architecture. *Computing in Science & Engineering*, 23(2):114–119, 2021.

[44] Wanling Gao, Chunjie Luo, Lei Wang, Xingwang Xiong, Jianan Chen, Tianshu Hao, Zihan Jiang, Fanda Fan, Mengjia Du, Yunyou Huang, et al. Aibench: towards scalable and comprehensive datacenter ai benchmarking. In *Benchmarking, Measuring, and Optimizing: First BenchCouncil International Symposium, Bench 2018, Seattle, WA, USA, December 10-13, 2018, Revised Selected Papers 1*, pages 3–9. Springer, 2019.

[45] X Yu Geoffrey, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. Habitat: A {Runtime-Based} computational performance predictor for deep neural network training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 503–521, 2021.

[46] Pawel Gepner. Machine learning and high-performance computing hybrid systems, a new way of performance acceleration in engineering and scientific applications. In *2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)*, pages 27–36. IEEE, 2021.

[47] Alexandros V Gerbessiotis and Leslie G Valiant. Direct bulk-synchronous parallel algorithms. *Journal of parallel and distributed computing*, 22(2):251–267, 1994.

[48] Hui Guan, Laxmikant Kishor Mokadam, Xipeng Shen, Seung-Hwan Lim, and Robert Patton. Fleet: Flexible efficient ensemble training for heterogeneous deep neural networks. *Proceedings of Machine Learning and Systems*, 2:247–261, 2020.

[49] Yijin Guan, Hao Liang, Ningyi Xu, Wenqiang Wang, Shaoshuai Shi, Xi Chen, Guangyu Sun, Wei Zhang, and Jason Cong. Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 152–159. IEEE, 2017.

[50] Qianyu Guo, Sen Chen, Xiaofei Xie, Lei Ma, Qiang Hu, Hongtao Liu, Yang Liu, Jianjun Zhao, and Xiaohong Li. An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 810–822. IEEE, 2019.

[51] Isabelle Guyon, Amir Reza Saffari Azar Alamdari, Gideon Dror, and Joachim M Buhmann. Performance prediction challenge. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 1649–1656. IEEE, 2006.

[52] Cong Hao, Xiaofan Zhang, Yuhong Li, Sitao Huang, Jinjun Xiong, Kyle Rupnow, Wen-mei Hwu, and Deming Chen. Fpga/dnn co-design: An efficient design methodology for iot intelligence on the edge. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[53] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018.

[54] Sayed Hadi Hashemi, Shadi A Noghabi, William Gropp, and Roy H Campbell. Performance modeling of distributed deep neural networks. *arXiv preprint arXiv:1612.00521*, 2016.

[55] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[56] Andy Hock. Cerebras wafer scale engine. https://www.cerebras.net/product-chip/, August 2023.

[57] Andy Hock. Cerebras wafer scale engine: An introduction. https://www.cerebras.net/blog/introducing-cerebras-systems/, August 2023.

[58] Aleksandar Ilic, Frederico Pratas, and Leonel Sousa. Cache-aware roofline model: Upgrading the loft. *IEEE Computer Architecture Letters*, 13(1):21–24, 2013.

[59] Animesh Jain, Amar Phanishayee, Jason Mars, Lingjia Tang, and Gennady Pekhimenko. Gist: Efficient data encoding for deep neural network training. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 776–789. IEEE, 2018.

[60] Arpan Jain, Ammar Ahmad Awan, Quentin Anthony, Hari Subramoni, and Dhableswar K DK Panda. Performance characterization of dnn training using tensorflow and pytorch on modern clusters. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11. IEEE, 2019.

[61] Stanisław Jastrzębski, Zachary Kenton, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. On the relation between the sharpest directions of dnn loss and the sgd step length. *arXiv preprint arXiv:1807.05031*, 2018.

[62] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of {Large-Scale}{Multi-Tenant}{GPU} clusters for {DNN} training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 947–960, 2019.

[63] Zhe Jia, Blake Tillman, Marco Maggioni, and Daniele Paolo Scarpazza. Dissecting the graphcore ipu architecture via microbenchmarking. *arXiv preprint arXiv:1912.03413*, 2019.

[64] Zhe Jia, Blake Tillman, Marco Maggioni, and Daniele Paolo Scarpazza. Dissecting the graphcore ipu architecture via microbenchmarking. *arXiv preprint arXiv:1912.03413*, 2019.

[65] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1:1–13, 2019.

[66] Heehoon Kim, Hyoungwook Nam, Wookeun Jung, and Jaejin Lee. Performance analysis of cnn frameworks for gpus. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 55–64. IEEE, 2017.

[67] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[68] Gary Lauterbach. The path to successful wafer-scale integration: The cerebras story. *IEEE Micro*, 41(6):52–57, 2021.

[69] Kyuho Jason Lee, Jinmook Lee, Sungpill Choi, and Hoi-Jun Yoo. The development of silicon for ai: Different design approaches. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(12):4719–4732, 2020.

[70] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International journal of robotics research*, 37(4-5):421–436, 2018.

[71] Xiaqing Li, Guangyan Zhang, H Howie Huang, Zhufan Wang, and Weimin Zheng. Performance analysis of gpu-based convolutional neural networks. In *2016 45th International conference on parallel processing (ICPP)*, pages 67–76. IEEE, 2016.

[72] Zhengchun Liu, Hemant Sharma, J-S Park, Peter Kenesei, Antonino Miceli, Jonathan Almer, Rajkumar Kettimuthu, and Ian Foster. Braggnn: fast x-ray bragg peak analysis using deep learning. *IUCrJ*, 9(1):104–113, 2022.

[73] Souley Madougou, Ana Varbanescu, Cees de Laat, and Rob van Nieuwpoort. The landscape of gpgpu performance modeling tools. *Parallel Computing*, 56:18–33, 2016.

[74] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

[75] Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. Mlperf training benchmark. *Proceedings of Machine Learning and Systems*, 2:336–349, 2020.

[76] Eitan Medina and Eran Dagan. Habana labs purpose-built ai inference and training processor architectures: Scaling ai training systems using standard ethernet with gaudi processor. *IEEE Micro*, 40(2):17–24, 2020.

[77] Xupeng Miao, Yujie Wang, Youhe Jiang, Chunan Shi, Xiaonan Nie, Hailin Zhang, and Bin Cui. Galvatron: Efficient transformer training over multiple gpus using automatic parallelism. *arXiv preprint arXiv:2211.13878*, 2022.

[78] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[79] Ibomoiye Domor Mienye, Yanxia Sun, and Zenghui Wang. Prediction performance of improved decision tree-based algorithms: a review. *Procedia Manufacturing*, 35:698–703, 2019.

[80] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.

[81] Hongwu Peng, Caiwen Ding, Tong Geng, Sutanay Choudhury, Kevin Barker, and Ang Li. Evaluating emerging ai/ml accelerators: Ipu, rdu, and nvidia/amd gpus. *arXiv preprint arXiv:2311.04417*, 2023.

[82] Raghu Prabhakar and Sumti Jairath. Sambanova sn10 rdu: Accelerating software 2.0 with dataflow. In *2021 IEEE Hot Chips 33 Symposium (HCS)*, pages 1–37. IEEE, 2021.

[83] Raghu Prabhakar, Sumti Jairath, and Jinuk Luke Shin. Sambanova sn10 rdu: A 7nm dataflow architecture to accelerate software 2.0. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 350–352. IEEE, 2022.

[84] Saket Pradhan, Raj Shah, Ranveer Shah, and Anuj Goenka. Identifying deepfake faces with resnet50-keras using amazon ec2 dl1 instances powered by gaudi accelerators from habana labs. In *2022 IEEE Region 10 Symposium (TENSYMP)*, pages 1–6. IEEE, 2022.

[85] Hang Qi, Evan R Sparks, and Ameet Talwalkar. Paleo: A performance model for deep neural networks. In *International Conference on Learning Representations*, 2017.

[86] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[87] Nikil Ravi, Pranshu Chaturvedi, EA Huerta, Zhengchun Liu, Ryan Chard, Aristana Scourtas, KJ Schmidt, Kyle Chard, Ben Blaiszik, and Ian Foster. Fair principles for ai models with a practical application for accelerated high energy diffraction microscopy. *Scientific Data*, 9(1):657, 2022.

[88] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[89] Markus Reichstein, Gustau Camps-Valls, Bjorn Stevens, Martin Jung, Joachim Denzler, and Nuno Carvalhais. Deep learning and process understanding for data-driven earth system science. *Nature*, 566(7743):195–204, 2019.

[90] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey and benchmarking of machine learning accelerators. In *2019 IEEE high performance extreme computing conference (HPEC)*, pages 1–9. IEEE, 2019.

[91] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey of machine learning accelerators. In *2020 IEEE high performance extreme computing conference (HPEC)*, pages 1–12. IEEE, 2020.

[92] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Ai and ml accelerator survey and trends. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–10. IEEE, 2022.

[93] Daniel Sanchez and Christos Kozyrakis. Zsim: Fast and accurate microarchitectural simulation of thousand-core systems. *ACM SIGARCH Computer architecture news*, 41(3):475–486, 2013.

[94] John Shalf. The future of computing beyond moore's law. *Philosophical Transactions of the Royal Society A*, 378(2166):20190061, 2020.

[95] Shaohuai Shi, Qiang Wang, and Xiaowen Chu. Performance modeling and evaluation of distributed deep learning frameworks on gpus. In *2018 IEEE 16th Intl*

Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), pages 949–957. IEEE, 2018.

[96] Shaswot Shresthamali, Yuan He, and Masaaki Kondo. Faws: Fault-aware weight scheduler for dnn computations in heterogeneous and faulty hardware. In *2022 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 204–212. IEEE, 2022.

[97] Nahian Siddique, Sidike Paheding, Colin P Elkin, and Vijay Devabhaktuni. U-net and its variants for medical image segmentation: A review of theory and applications. *Ieee Access*, 9:82031–82057, 2021.

[98] Venkat Srinivasan, Darshan Gandhi, Urmish Thakker, and Raghu Prabhakar. Training large language models efficiently with sparsity and dataflow. *arXiv preprint arXiv:2304.05511*, 2023.

[99] Nupur Sumeet, Karan Rawat, and Manoj Nambiar. Performance evaluation of graphcore ipu-m2000 accelerator for text detection application. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering*, pages 145–152, 2022.

[100] Yifan Sun, Trinayan Baruah, Saiful A Mojumder, Shi Dong, Xiang Gong, Shane Treadway, Yuhui Bao, Spencer Hance, Carter McCardwell, Vincent Zhao, et al. Mgpusim: enabling multi-gpu performance modeling and optimization. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 197–209, 2019.

[101] Cheng Tan, Chenhao Xie, Tong Geng, Andres Marquez, Antonino Tumeo, Kevin Barker, and Ang Li. Arena: Asynchronous reconfigurable accelerator ring to enable data-centric parallel computing. *IEEE Transactions on Parallel and Distributed Systems*, 32(12):2880–2892, 2021.

[102] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[103] Chao Wang, Lei Gong, Qi Yu, Xi Li, Yuan Xie, and Xuehai Zhou. Dlau: A scalable deep learning accelerator unit on fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(3):513–517, 2016.

[104] Yuxin Wang, Qiang Wang, Shaohuai Shi, Xin He, Zhenheng Tang, Kaiyong Zhao, and Xiaowen Chu. Benchmarking the performance and energy efficiency of ai accelerators for ai training. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 744–751. IEEE, 2020.

[105] Jinliang Wei, Wei Dai, Aurick Qiao, Qirong Ho, Henggang Cui, Gregory R Ganger, Phillip B Gibbons, Garth A Gibson, and Eric P Xing. Managed communication and consistency for fast data-parallel iterative analytics. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 381–394, 2015.

[106] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[107] Justin M Wozniak, Rajeev Jain, Prasanna Balaprakash, Jonathan Ozik, Nicholson T Collier, John Bauer, Fangfang Xia, Thomas Brettin, Rick Stevens, Jamaludin Mohd-Yusof, et al. Candle/supervisor: A workflow framework for machine learning applied to cancer research. *BMC bioinformatics*, 19(18):59–69, 2018.

[108] Ruohan Wu, Mingfan Li, Hanxi Li, Tianxiang Chen, Xinghui Tian, Xiaoxin Xu, Bin Zhou, Junshi Chen, and Hong An. Machine learning-enabled performance model for dnn applications and ai accelerator. In *2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pages 25–34. IEEE, 2022.

[109] Zhen Xie, Wenqian Dong, Jiawen Liu, Hang Liu, and Dong Li. Tahoe: tree structure-aware high performance inference engine for decision tree ensemble on gpu. In *Proceedings of the Sixteenth European Conference on Computer Systems*, pages 426–440, 2021.

[110] Zhen Xie, Wenqian Dong, Jie Liu, Ivy Peng, Yanbao Ma, and Dong Li. Md-hm: memoization-based molecular dynamics simulations on big memory system. In *Proceedings of the ACM International Conference on Supercomputing*, pages 215–226, 2021.

[111] Zhen Xie, Siddhisanket Raskar, Murali Emani, and Venkatram Vishwanath. Trainbf: High-performance dnn training engine using bfloat16 on ai accelerators. In *European Conference on Parallel Processing*, pages 458–473. Springer, 2023.

[112] Zhen Xie, Guangming Tan, Weifeng Liu, and Ninghui Sun. Ia-spgemm: An input-aware auto-tuning framework for parallel sparse matrix-matrix multiplication. In *Proceedings of the ACM International Conference on Supercomputing*, pages 94–105, 2019.

[113] Eric P Xing, Qirong Ho, Wei Dai, Jin-Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: A new platform for distributed machine learning on big data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1335–1344, 2015.

[114] Chengru Yang, Zhehao Li, Chaoyi Ruan, Guanbin Xu, Cheng Li, Ruichuan Chen, and Feng Yan. Perfestimator: A generic and extensible performance estimator for data parallel dnn training. In *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*, pages 13–18. IEEE, 2021.

[115] Chih-Chieh Yang and Guojing Cong. Accelerating data loading in deep neural network training. In *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pages 235–245. IEEE, 2019.

[116] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014.

[117] Shanshan Zhang, Ce Zhang, Zhao You, Rong Zheng, and Bo Xu. Asynchronous stochastic gradient descent for dnn training. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6660–6663. IEEE, 2013.

[118] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. ACM, 2018.

[119] Xiaofan Zhang, Hanchen Ye, Junsong Wang, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. Dnnexplorer: a framework for modeling and exploring a novel paradigm of fpga-based dnn accelerator. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[120] Yao Zhang and John D Owens. A quantitative performance analysis model for gpu architectures. In *2011 IEEE 17th international symposium on high performance computer architecture*, pages 382–393. IEEE, 2011.

[121] Zhaorui Zhang and Choli Wang. Mipd: An adaptive gradient sparsification framework for distributed dnns training. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):3053–3066, 2022.

[122] Xun Zhou, A Kai Qin, Maoguo Gong, and Kay Chen Tan. A survey on evolutionary construction of deep neural networks. *IEEE Transactions on Evolutionary Computation*, 25(5):894–912, 2021.

[123] Maxim Zvyagin, Alexander Brace, Kyle Hippe, Yuntian Deng, Bin Zhang, Cindy Orozco Bohorquez, Austin Clyde, Bharat Kale, Danilo Perez-Rivera, Heng Ma, et al. Genslms: Genome-scale language models reveal sars-cov-2 evolutionary dynamics. *bioRxiv*, pages 2022–10, 2022.