Cambricon-Q: A Hybrid Architecture for Efficient Training

Yongwei Zhao[1,2,3], Chang Liu[1,2,3], Zidong Du[1,3], Qi Guo[1], Xing Hu[1], Yimin Zhuang[1,2,3], Zhenxing Zhang[1,2,3], Xinkai Song[1,2,3], Wei Li[1], Xishan Zhang[1,3], Ling Li[4], Zhiwei Xu[1,2], and Tianshi Chen[3]

[1]SKL of Computer Architecture, Institute of Computing Technology, CAS [2]University of Chinese Academy of Sciences
[3]Cambricon Tech. Ltd [4]Institute of Software, CAS

Abstract—**Deep neural network (DNN) training is notoriously time-consuming, and quantization is promising to improve the training efficiency with reduced bandwidth/storage requirements and computation costs. However, state-of-the-art quantized algorithms with negligible training accuracy loss, which require on-the-fly statistic-based quantization over a great amount of data (e.g., neurons and weights) and high-precision weight update, cannot be effectively deployed on existing DNN accelerators. To address this problem, we propose the first customized architecture for efficient quantized training with negligible accuracy loss, which is named as Cambricon-Q. Cambricon-Q features a hybrid architecture consisting of an ASIC acceleration core and a near-data-processing (NDP) engine. The acceleration core mainly targets at improving the efficiency of statistic-based quantization with specialized computing units for both statistical analysis (e.g., determining maximum) and data reformating, while the NDP engine avoids transferring the high-precision weights from the off-chip memory to the acceleration core. Experimental results show that on the evaluated benchmarks, Cambricon-Q improves the energy efficiency of DNN training by 6.41× and 1.62×, performance by 4.20× and 1.70× compared to GPU and TPU, respectively, with only $\leqslant 0.4\%$ accuracy degradation compared with full precision training.**

I. Introduction

DNN training is notoriously tedious and time-consuming. A distinct example is the most recent GPT-3 language model from OpenAI [7], which has 175 billion parameters with 3.14E23 FLOPS for training, and thus theoretically it requires 355 years to train on a single NVIDIA V100 with 28TFLOPS peak [41]. As the size of models continues to increase, both industry and academy cry out for new hardware architecture with high training efficiency.

Quantization is a promising technique for improving training efficiency [44]. By converting the full-precision floating-point value (e.g., 32-bit floating-point, FP32) to low bit-width data (e.g., 8-bit fixed-point, INT8) [33], it provides the potential of highly efficient hardware through computation and data access with reduced energy and latency, see Table I.

State-of-the-art quantized training algorithms already achieve negligible training accuracy loss (e.g., only 0.02%) with INT8 data on a wide range of DNN models, including AlexNet, ResNet50, and SSD [62]. However, they cannot be effectively deployed on any existing DNN accelerators, e.g. GPU [12], [45] and TPU [33], **even though such accelerators have already integrated INT8 or INT16 arithmetic units to accelerate DNN inference**. For example, on the NVIDIA V100 GPU with Tensor Core, the performance of quantized training algorithms only achieves 78% of the traditional training with FP32 on the evaluated benchmarks. Another example is the FloatPIM accelerator, which integrates various functional units including INT16, INT32, BF16, and FP32. By using INT16 for training, FloatPIM results in significant accuracy degradation (i.e., 5.2%) on VGGNet [30], let alone lower bit-width data such as INT8.

The inefficiency of quantized training algorithms on existing accelerators is caused by two reasons. The first reason is that quantized training typically employs on-the-fly statistic-based quantization over a great amount of data such as neurons and weights. For example, Yang et al. count the maximum value of a layer to quantize synaptic weights to 8-bit floating-point data (FP8) [61]. Zhu et al. compute the cosine similarity between the quantized weights and original weights for obtaining INT8 data [65]. Zhang et al. propose to use the vector distance and maximum value of weights for quantizing weights and neurons to INT8 and INT16 respectively [62]. The second reason is that while most of the data are quantized to low bit-width data, the weight update still requires memory access and computation on high-precision data such as FP32. Actually, on the NVIDIA V100 GPU, the execution time of statistic-based quantization and high-precision weight update is 38% of the computation time for training VGGNet.

To address these bottlenecks, in this paper, we propose the first customized architecture called as Cambricon-Q for efficient quantized training with negligible training accuracy loss. Cambricon-Q features a hybrid architecture consisting

TABLE I
Efficiency comparison of different bit-width data (45nm) [27] (* for our results).

Bit-width	Operation	Energy	Relative costs
32-bit	Floating-point **ADD**	0.9pJ	30
	Floating-point **MUL**	3.7pJ	123.33
	Fixed-point **ADD**	0.1pJ	3.33
	Fixed-point **MUL**	3.1pJ	103.33
	DRAM access (Average)	0.65∼1.3nJ	21667∼43333
16-bit	Floating-point **ADD**	0.4pJ	13.33
	Floating-point **MUL**	1.1pJ	36.67
	*Fixed-point **ADD**	0.05pJ	1.67
	*Fixed-point **MUL**	1.55pJ	51.67
	DRAM access (Average)	0.33∼0.65nJ	10000∼21667
8-bit	Fixed-point **ADD**	0.03pJ	1
	Fixed-point **MUL**	0.2pJ	6.67
	DRAM access (Average)	0.16∼0.33nJ	5333∼10000

TABLE II
EXISTING HARDWARE FOR DNN TRAINING.

Hardware supports	V100 [12]	TPU [20]	FloatPIM [30]	SIGMA [51]	This paper
low bit-width units	✔	✔	✔	✔	✔
statistical analysis	✗	✗	✗	✗	✔
Reformating	✔	✗	✗	✔	✔
In-place weight update	✗	✗	✔	✗	✔

of an ASIC acceleration core and a near-data-processing (NDP) engine for accelerating statistic-based quantization and high-precision weight update, respectively. In contrast to conventional statistic-based quantization containing separately processed *statistical analysis* (e.g., determining maximum) and *reformating* over all data, which inevitably incurs extra data accesses, the acceleration core integrates a specialized computing unit for conducting statistical analysis and reformating consecutively over each partitioned slice of the entire data. In contrast to conventional weight update requiring transferring high-precision weights from the off-chip memory to the on-chip computation logics, the NDP engine is implemented by integrating a configurable optimizer (e.g., AdaGrad [15] and RMSProp [25]) into the DRAM, allowing *in-place weight update* within the memory. Table II compares Cambricon-Q and representative DNN accelerators in terms of hardware supports for key operations in training.

We conduct experiments on a broad range of network models and compare the experimental results against edge GPU and TPU. Concretely, compared to the edge-side GPU (i.e., Jetson TX2), the energy efficiency gains of DNN training 6.41×, performance 4.20×, respectively. Compared to the TPU, the energy efficiency gain of DNN training is 1.62×, performance 1.70×. Moreover, compared to full-precision training on GPU, the accuracy loss is $\leqslant$0.4%.

This paper makes the following contributions.

- We conduct a thorough analysis on state-of-the-art quantized training algorithms and observe that major bottlenecks stem from on-the-fly statistic-based quantization and full-precision weight update.
- To our best knowledge, we propose the first architecture for efficient quantized training with negligible training accuracy loss.
- We propose an acceleration core that performs statistic-based quantization locally over sliced data instead of entire data, which not only improves computing efficiency but also significantly reduces the amount of data accesses.
- We propose a near-data-processing engine that integrates a configurable optimizer into the DRAM for avoiding the costly data transferring in weight update.

The rest of the paper is organized as follows: Section II discusses state-of-the-art quantized training techniques and their inefficiency on existing training hardware. Section III introduces our proposed hardware-friendly quantization technique. Section IV describes Cambricon-Q architecture. Section V describes our methodology. Section VI describes the physical

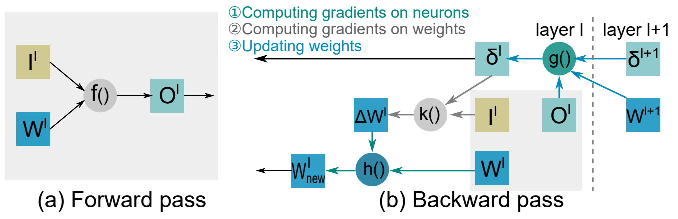

Fig. 1. DNN training. (a) Forward pass. (b) Backward pass. (I are the input neurons, W are the synaptic weights, O are the output neurons, δ are the gradients on neurons, ΔW are the gradients on weights), and W_{new} are the updated weights. Note that $*^l$ and $*^{l+1}$ are the data in layer l and $l+1$, respectively.

implementation and hardware costs, evaluates the performance of Cambricon-Q against the baselines. Section VIII discusses the related works and Sec IX concludes this paper.

II. Background and Motivation

A. Quantized Training

This section introduces quantization techniques applied in the training stage.

1) DNN training: Backpropagation (BP) is the mainstream DNN training algorithm. Roughly, BP is a two-pass algorithm, including *forward pass* and *backward pass*. In forward pass, DNN computes neuron outputs for input samples. The neuron outputs of layer l are computed as $O^l = f(I^l, W^l)$, where I^l and W^l are the inputs and synaptic weights of layer l, $f()$ is the layer function, see Figure 1 (a). The backward pass consists of three stages: *computing gradients on neurons*, *computing gradients on weights*, and *updating weights* (Figure 1 (a)).

- In *computing gradients on neurons* stage, each layer computes its local gradients on neurons based on the passed-back gradients of loss function, which measures the errors between the desired network outputs and actual network outputs. Gradients in layer i is computed as $\delta^l = g(O^l, \delta^{l+1}, W^{l+1})$ based on the gradients in layer $l+1$, where δ^{l+1} is the gradients on neurons in layer $i+1$, W^{l+1} is the weights of layer $l+1$, and $g()$ is the computing function (① in Figure 1 (a)).
- In *computing gradients on weights* stage, each layer computes synaptic weight corrections with regard to gradients from all connected output neurons. Gradients on weights in layer l is computed as $\Delta W^l = k(I^l, \delta^l)$, where $k()$ is the computing function (② in Figure 1 (a)).
- In *Updating weights* stage, each synapse update its weight value according to the delta rule or optimization methods. It can be formulated as $W^l = h(W^l, \Delta W^l)$, where h is the updating function (③ in Figure 1 (a)).

2) Statistic-based quantization: Quantization is a technique that uses fewer bits to represent data so as to reduce the hardware costs of computation, storage, and data transfer. It can be formulated as $X_q = round(\frac{X-\alpha}{\beta})$, where X is the full-precision data, X_q is the quantized data, α and β are the quantization parameters with regard to $scale$ and $offset$.

Quantization techniques have been applied in the inference stage for a long time. However, it is challenging to apply

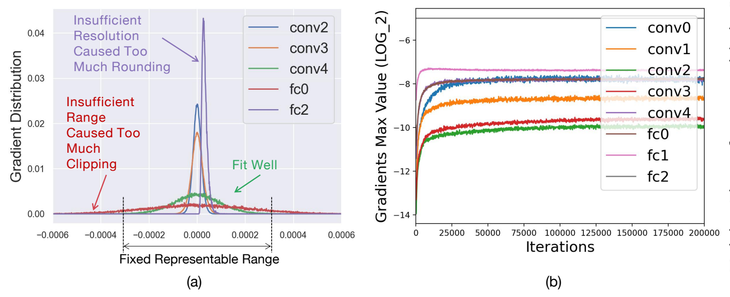

Fig. 2. (a) The data distributions of gradients in different layers when training the AlexNet [37]. Conventional quantization cannot fit for all the data in a DNN model due to variety. (b) Maximum value of gradients in different iterations when training the AlexNet.

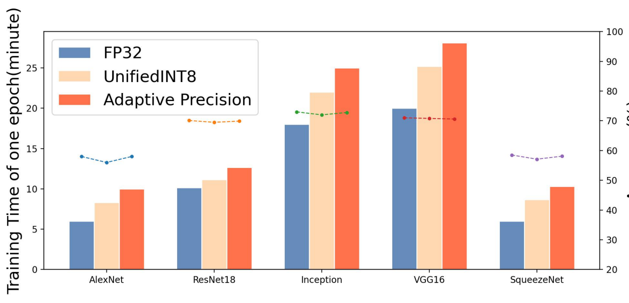

Fig. 3. DNN training with/without quantization on CPU+GPU platform.

them in the training stage because training data are much more precision-sensitive. Statistic-based quantization techniques are proposed to address this issue, which achieved 8 bit-width DNN training with negligible accuracy loss. As listed in Table III, they reply on two key factors, *on-the-fly statistic-based quantization* and *high precision weights update*.

Regarding *on-the-fly statistic-based quantization*, the key difference from previous quantization methods in DNN inference is using statistical analysis to on-the-fly determine the quantization parameters, including maximum value [60]–[62], [64], [65], cosine distances [65], and vector distances [62]. The main reason is that the data distribution of gradients in backward pass varies drastically across different layers and training epochs, where the *static quantization* methods worked in DNN inference could introduce too much quantization errors on gradients. As the data distributions depicted in Figure 2 when training AlexNet [37], the maximum absolute value of gradients show a variance of two orders of magnitudes between layers (0.00035∼0.0312), and three orders of magnitudes between epochs (7.01E-14∼3.13E-2). If applying the static quantization methods to gradients, for a representation data range of [-0.0003,0.0003], we can easily notice that only the conv4 layer in AlexNet can be well quantized; conv2 layer, conv3 layer, and fc2 layer will cause too much rounding errors due to its narrower value range; fc0 layer will cause overflow errors due to its larger value range. Similarly, other methods like sampling or clipping to fast estimate the statistic results have the same drawback of introducing non-negligible quantization errors that affecting the training accuracy significantly. For example, it is confirmed that the statistic max absolute value should be accurate in quantized training [31], [48], [63]. In [63], an inaccurate statistic max absolute value used in quantized training algorithms would cause 0.4∼4.6% overall accuracy performance drop on VGG, 0.3%∼3.1% on densenet, and 0.2%∼2.2% on Inception. Therefore, dynamic quantization methods, which perform on-the-fly statistic counting and quantization, is essential to retain training accuracy by taking care of unstable and variable data distribution in the training stage.

Regarding *high precision weights update*, instead of quantized into low bit-width data, state-of-the-art quantization methods use high precision data in the stage of *Updating weights*. The necessity of high precision data for weights may lie in both the smaller numbers and the larger numbers [44]. The former could cause 5% of the gradients on weights becoming zeros as they have exponents smaller than -24, slowing the training process. The latter could invalidate the weight update by overflow to zeros if the weights are much layer than the gradients on weights, causing errors that can not be recovered. To avoid the slow convergence and severe errors from accumulative numerical inaccuracy, the *updating weights* stage is still performed with high precision data, including FP16 [60], FP24 [61] and FP32 [62], [64], [65].

TABLE III
Low Bit-width Training Algorithms (X: original weights; X': quantized weights).

Algorithm	Data Format	Statistic	Special Cases
Wang et al. 2018 [60]	FP8	$\max\|X\|$	Weight update (FP16)
Zhu et al. 2019 [65]	INT8	$\max\|X\|$, $\cos(X,X')$	Weight update (FP32) Learned clipping range
Yang et al. 2020 [61]	INT8	$\max\|X\|$	Weight update (FP24)
Zhong et al. 2020 [64]	Shiftable INT8	$\max\|X\|$	Weight update (FP32) Quantized in groups
Zhang et al. 2020 [62]	INT8/INT16	$\max\|X\|$, $\overline{X}-\overline{X'}$	Weight update (FP32) Adaptive precision

B. Motivation

Though DNN training can be quantized to as less as 8 bit-width data, the two key factors of state-of-the-art quantization methods for DNN training prevent existing training hardware, including GPUs and accelerators, from fully leveraging benefits of data quantization. Comparing to the training without quantization (Figure 4 (a)), quantized training requires more interactions with CPU, and thus could even lead to a slowdown. For example, when perform quantized training on an Nvidia V100 GPU, with comparable training accuracy, it is even $1.09\times \sim 1.78\times$ worse than the normal training without quantization, averaging on five representative DNNs, see Figure 3. The reason is two-fold.

Lacking of hardware support. Initially, GPU and accelerators lack efficient hardware support. For GPUs, they lack

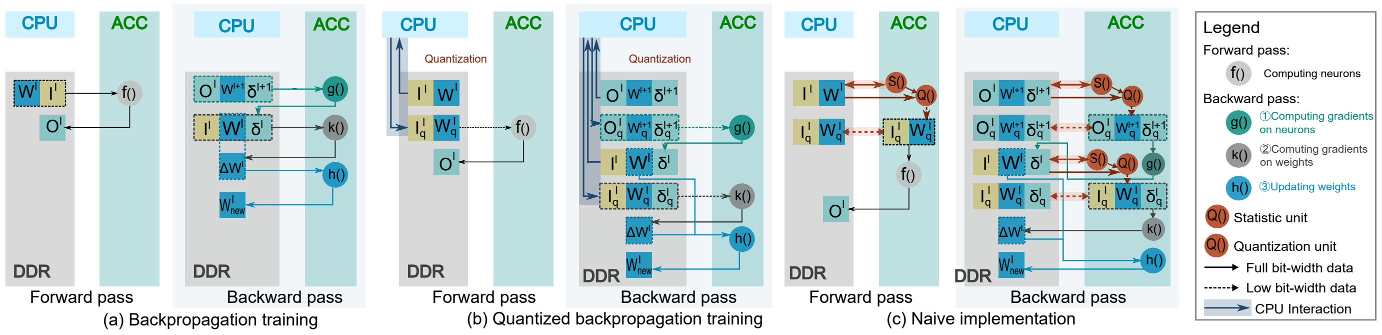
Fig. 4. Processing DNNs on CPU+ACC platform. (a) Backpropagation training. (b) Quantized backpropagation training. (c) Naive implementation, where Quantization units are placed in the ACC. ($*_q$: quantized data; $*$: unquantized full bit-width data.)

hardware support for *on-the-fly statistic-based quantization*, including both statistic analysis and hardware quantization. For existing quantized accelerators [3], [9], [16], [22], [23], [34], [38], [42] which usually quantize data offline, it could be even worse if they lack of high precision units for *high precision weights update*. Therefore, as shown in Figure 4 (b), the host CPU must be invoked to help with such processes, which is slow and costly. For example, when training AlexNet on V100, the total data transaction between CPU memory and GPU is $2.55\times$ more in quantized training than normal training.

Critical data movements in *Updating weights*. Moreover, as *high precision weights update* requires high precision data (e.g.,W, ΔW, and W_{new}), their movements between memory and accelerator become more critical in quantized training. It is simply because other data in quantized training have been quantized to low bit-width, comparing the backward pass in quantized training against that in normal training (Figure 4 (b) vs. (a)). Analytically, when training AlexNet with other data quantized to 8-bit, high precision data movements become $1.80\times$ more in quantized training (53.5%) than that in normal training (29.8%).

Our solution. One may think that the naive and intuitive solution of adding quantization support to the ACC could address above issues, as shown in Figure 4 (c) where specialized statistic units ($S()$) and quantization units ($Q()$) are equipped. However, they are still far from efficiently processing state-of-the-art quantized DNN training. The main reason is that statistic-based quantization is not free lunch and introduces extra data accesses during statistic counting and quantization. Specifically, a two-pass data access is required for each data: one for statistic analysis and one for quantization. As a result, for each epoch, ACC has to access at least $2\times$ more data than training without quantization, leading to at least $2\times$ more memory energy costs. Even worse, when ACC has limited on-chip storage whose size is less than a layer in DNNs, such as in IoT devices particularly, ACC has to exchange intermediate results back to main memory, which inevitably incurs more data access.

In summary, a high energy-efficient hardware for quantized DNN training should not only efficiently support *on-the-fly statistic-based quantization* and *high precision weights update*, but also reduce the number of *extra* data access and *high precision* data access.

In this paper, we propose a novel hybrid architecture of an ASIC acceleration core and a NDP Engine called Cambricon-Q. Initially, we propose a Hardware-friendly Quantization Technique (HQT), which can perform statistic analysis and quantization with one-pass data access (detailed in Section III). Together with the Statistic Quantization Units (SQU) and Quantization Buffer Controller (QBC), Cambricon-Q addresses the efficient hardware support *on-the-fly statistic-based quantization* issue as well as reducing the *extra* data access. Cambricon-Q features the NDP Engine to address the issue of hardware support for *high precision weights update* as well as reducing the *high precision* data access (detailed in Section IV).

III. Hardware-friendly Quantization Technique

In this section, we introduce our hardware-friendly quantization technique (HQT) for quantized DNN training. HQT could avoid extra data access during quantization with its Local Dynamic Quantization (LDQ), and could reduce quantization errors for the long-tail-distributed data with its component of Error-estimation-based Quantization Multiplexing (E^{2}BQM).

A. *Local Dynamic Quantization (LDQ)*

Previous layer-wise statistic-based quantization techniques need to perform global statistics before quantization, which leads to data dependency as a “bottleneck”. Each element in the quantized result X_q depends on the statistic θ, which itself depends on every element in the original data X. θ is available only after a full scan of X, thus a full scan of X is inevitable to obtain θ. Moreover, until when θ is available, no value in X_q can be computed, thus a second full scan of X to compute X_q is also inevitable. The “bottleneck” phenomenon forbids the hardware leveraging data locality and causes at least $2\times$ data access.

More fine-grained Local Dynamic Quantization (LDQ) can solve the “bottleneck” phenomenon. LDQ slices the data into blocks of fixed size. The statistic is performed within each block, constrained the data dependencies to local scope. The block can fit in the on-chip buffer size, hence avoid the excessive memory accesses.

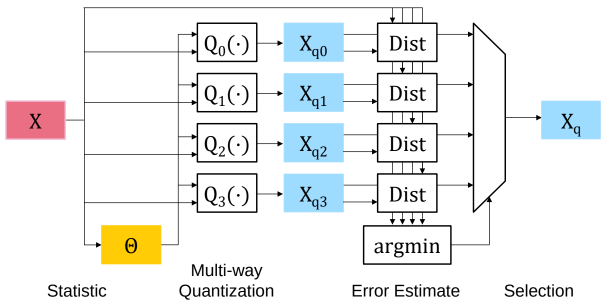

Fig. 5. Error-estimation-based Quantization Multiplexing.

Notice that the error with LDQ is guaranteed to be less or equal to the error with layer-wise statistic-based quantization. The statistic θ is the max absolute value ($\max|X|$) in the data for every state-of-the-art algorithms. Trivially, the max absolute value of a local data block (θ_i) never exceeds the max absolute value of all the data, hence $\theta_i \leqslant \theta$ always holds. Dynamic quantization forbids the occurrence of data clipping, so the representation with narrower range (i.e. smaller θ) will have a less rounding error. In summary, LDQ always has smaller or equal rounding error compared to the layer-wise dynamic quantization. We also validate this proposition with experiments, on 10 sets of trained models and training algorithms, LDQ achieved $+0.02\%$ final accuracy on average, compared to the original quantization method.

Besides, the compression ratio of LDQ is close to layer-wise dynamic quantization (denote as DQ), if the block size K is large enough. Suppose data is quantized into 1 byte, and the statistic θ of each block is stored in 2 bytes. The compression ratios of LDQ and DQ are calculated as

$$C_{LDQ} = \frac{4N}{\frac{N}{K}(K+2)} = \frac{4}{1+2/K}$$
$$C_{DQ} = \frac{4N}{N+2} = \frac{4}{1+2/N}$$

where N is the number of data, and K is the block size. If K is greater than 200 or 4000, the compression efficiency loss is less than 1% or 0.05%.

B. *Error-estimation-based Quantization Multiplexing (E^2BQM)*

Researches found that the long tail distribution of data in DNN models exaggerated the rounding errors in fixed point representations [32], [62], [64], [65]. Current algorithms cover the long tail distribution with various techniques, mainly dynamically switched quantization. Zhong et al. [64] proposed *Shiftable Fixed-Point Data Format*, which encodes data two different fixed-point value ranges with an additional bit, covering both the representable range and resolution. Jain et al. [32] proposed a very similar data representation *BiScaled-FxP* for the long tail distribution, encoding two scales (namely *Scale-Fine* for most data and *Scale-Wide* for the long-tail). Zhang et al. [62] use dynamically selected data format according to estimated quantization error between INT8 and INT16, to cover different distributions. Zhu et al. [65] proposed *Direction Sensitive Gradient Clipping* to clip the long tail with a minimal precision penalty. This technique optimizes the best clipping range setting with backpropagation, using the cosine similarity between the dequantized and original data as the loss function.

The divergence of techniques is unfriendly to the design of efficient hardware. The key observation here is that all these techniques are choosing the best quantization function $Q_i(\cdot)$ among several candidates $Q_0(\cdot), Q_1(\cdot), \ldots, Q_N(\cdot)$ according to the estimation of quantization error (the distance between the original data X and the dequantized data X_q). E.g. *Shiftable Fixed-Point Data Format* can be seen as choosing the best fit value range between two candidates according to any error estimations, and *Direction Sensitive Gradient Clipping* can be seen as choosing the best clipping range among many possible settings according to the distance defined in the inner-product space. We unified these methods into the Error-estimation-based Quantization Multiplexing (E^2BQM) technique, shown in Figure 5. This technique consists of four steps: (1) performs statistic on the original data X; (2) quantizes the data X into multiple candidates X_{q_i} via different quantization function $Q_i(\cdot)$; (3) calculate the distance between X and each dequantized data $X'_i = Q_i^{-1}(X_{q\_i})$ as the estimation to quantization errors; (4) select the best candidate as the final result X_q, according to the error estimation.

As an example, we use a 4-way E^2BQM with the rectilinear distance estimation ($\sum|x_i - x'_i|$) to simulate the *Direction Sensitive Gradient Clipping* [65]. Experimental results on AlexNet and ResNet-18 show that the accuracy difference is +0.1%/-0.2% respectively, which are not significantly affected. We also validate a 4-way E^2BQM to simulate the *Shiftable Fixed-Point Data Format* on ResNet-18[1], the accuracy difference is +1.1% showing an significant improvement.

IV. CAMBRICON-Q ARCHITECTURE

A. *Overview*

Figure 6 shows the overall architecture of Cambricon-Q, which consists of two major parts: an ASIC Acceleration Core and a practical near-data-processing (NDP) Engine. The Acceleration Core is designed to quantize data with HQT and perform the major computations in quantized DNN training. It consists of a PE Array for matrix/vector computing, a scalar functional unit (SFU) for scalar operations, and three on-chip buffers (for input neurons (NBin), output neurons (NBout), and synapses (SB) respectively). Besides, it includes three specially designed modules to efficiently support HQT: a Statistic Quantization Unit (SQU) that performs the on-the-fly statistic and LDQ, two Quantization Buffer Controllers (QBCs) coupled with NBin and SB to manage data quantized with different parameters (e.g., scales and offsets). The NDP Engine is designed to perform in-place updating of high-precision weights in memory. It consists of two specialized modules: an SQU and a NDP optimizer (NDPO) to update weights with various optimizers.

[1]Zhong et al. [64] with *Shiftable Fixed-Point Data Format* is not directly applicable on AlexNet, so we only report the result on ResNet-18.

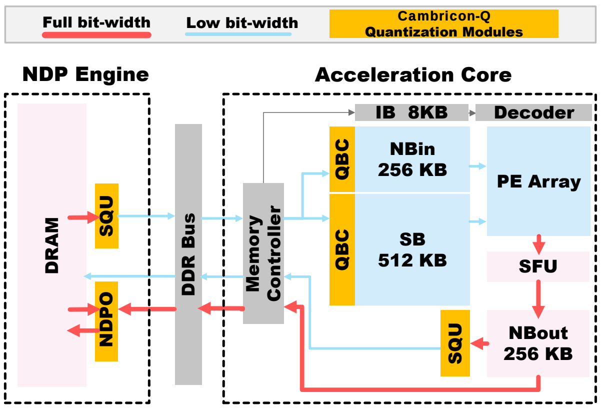

Fig. 6. Architecture Overview of Cambricon-Q. Cambricon-Q reduces computation, storage, and transfer overheads with low bit-with data (blue blocks and arrows) based on quantization modules (yellow blocks): SQU, QBC, and PIMO.

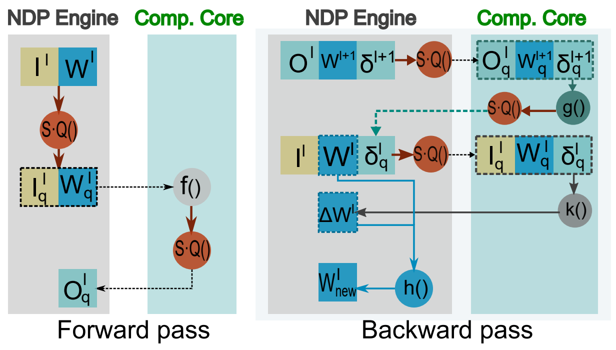

Fig. 7. Processing quantized forward and backward pass on Cambricon-Q. $S \cdot Q()$ is the fused statistic and quantization units, which is performed by SQU. $f()$, $g()$, and $k()$ are all performed by the PE Array and SFU. $h()$ is performed by the NDP Engine.

To perform the *forward pass* of DNN training, as shown in Figure 7, Cambricon-Q quantizes the input neurons and synaptic weights through the SQU inside NDP Engine directly (i.e., $S{\cdot}Q()$) and then loads the quantized data to NBin and SB in the Acceleration Core, respectively. The Acceleration Core manages the data with different quantization parameters in NBin and SB through the two coupled QBCs. The acceleration core performs computation on the PE Array and SFU (i.e., $f()$). After computation, the PE Array sends its final results to SFU for activation function and stores to NBout. The final results are written back to memory through on-the-fly quantization with SQU (i.e., $S \cdot Q()$). It can be noticed that only quantized data are transferred between NDP Engine and Acceleration Core.

To perform the *backward pass* of DNN training, as shown in Figure 7, Cambricon-Q follows the principle that eliminates full bit-width data traffic as many as possible for the three stages of the backward pass. For the first stage, i.e., *computing gradients on neurons*, Cambricon-Q loads the output neurons of layer l, gradients of layer $l + 1$, weights of layer $l + 1$ to NBin, NBin, and SB, respectively, while streaming the data through SQU in NDP (i.e., $S \cdot Q()$) for

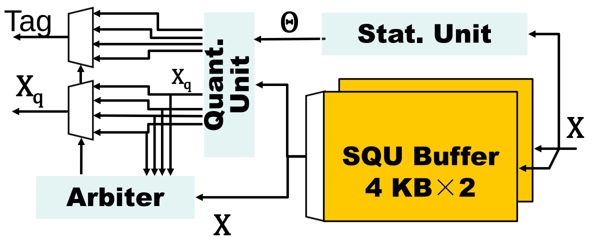

Fig. 8. Statistic Quantization Unit. (X: unquantized data; Θ: quantization parameters; X_q: quantized data.)

quantization. Cambricon-Q then computes the gradients of layer l through the PE Array (i.e, $g()$) and writes only the quantized gradients to the memory after performing on-the-fly quantization through SQU in the Computing Core. For the second stage, i.e.,*computing gradients on weights*, similarly, Cambricon-Q loads only quantized data and computes the results through the PE Array (i.e., $k()$). But slightly different from the previous stage, Cambricon-Q writes back the full bit-width results of weight corrections instead of quantization. For the third stage, i.e., *updating weights*, Cambricon-Q performs in-place updating weights (i.e., $h()$) through the NDPO in NDPO Engine directly instead of passing to Acceleration Core, which eliminates huge data traffic. It can be noticed that most data transferred between NDP Engine and Acceleration Core are quantized data as well as eliminating extra data access with on-the-fly HQT.

In the rest of this section, we first introduce the specialized modules designed for supporting statistic-based quantization. Then we introduce the instruction set used in Cambricon-Q and how the controller works. At last, we introduce the functional units for computation.

B. Statistic-based Quantization support

Overall, Cambricon-Q addresses the challenges in Section II-B with the help of HQT and the specialized modules, including SQU, QBC, and NDPO. First, together with HQT, SQU supports on-the-fly statistic counting and quantization that eliminates non-trivial extra data access. Second, QBC, a cache-like buffer controller, manages the data that is quantized with different parameters in hardware. Third, NDP Engine supports in-place synaptic weight update, avoiding costly data transfer between memory and on-chip computing units.

1) SQU: The SQU is designed for on-the-fly statistic counting and quantization (i.e., $S \cdot Q()$), to support HQT efficiently. As shown in Figure 8, a SQU consists of two 4 KB buffers, a Statistic Unit (Stat. Unit), a Quantization Unit (Quant. Unit), and an Arbiter. At first, the unquantized data (i.e., X) are stored to the SQU buffers, where the two SQU buffers are worked in a double-buffering manner to improve the throughput. Simultaneously, the unquantized data (X) is sent to the Statistic Unit, which computes the statistic results to decide the quantization parameters (i.e., Θ). Then, based on the quantization parameters, the Quantization Unit performs the quantization, which converts full bit-width data

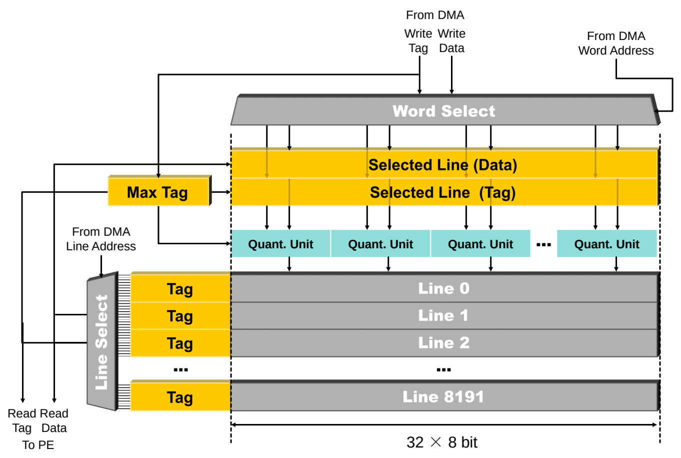

Fig. 9. NBin coupled with QBC.

into low bit-width data (i.e., X_q). Moreover, to support the E^{2}BQM technique, SQU performs a time-multiplexing 4-way quantization with different parameters so as to find the best quantization parameters. The Arbiter compares the quality of quantization with different quantization parameters, and selects the quantized data and corresponding tag as the final quantization results.

2) QBC: In HQT, even neighboring data may be split into two independent quantization process with different parameters. To manage such data, we design the QBC, a cache-like on-chip buffer controller, coupling with NBin and SB. Figure 9 shows the architecture of on-chip buffer with QBC, using NBin as an example. Similarly to cache, QBC manages the NBin in buffer lines, where data in a buffer line share the same data format (i.e., bit-width and quantization parameters) and each buffer line has a tag to record that data format. When data is read out from the NBin, both data and tag are provided to perform the computations correctly according to their data format. In the implementation in this paper, each buffer line in Cambricon-Q has of 32 words and each word is 8 bits.

QBC maintains the data in one buffer line having the same data format. For most cases, data in DNNs are tensors that are read/written from/to on-chip buffers in lines with the same data format, where QBC only need to record the tag for each line. But in few cases such as matrix transposition, data to store in one buffer line have different formats. To maintain the integrity of buffer lines, QBC performs requantization for the data in the buffer line, when data are indexed through a byte-addressing manner, see Figure 9. Specifically, when writing data to a buffer line that has different tag, data and their corresponding tags are saved in the *Selected Line* firstly. Meanwhile, the *Max Tag* is updated to the maximum one of the data and the buffer line. Then the *Selected Line* is re-quantized based on the *Max Tag* and flushed back to buffer line, and the tag of the corresponding buffer line is set to *Max Tag*. In such a manner, data in buffer line are maintained with the same quantization parameters, i.e., same tag. Please note NBout is not coupled with QBC for it contains only full-precision data.

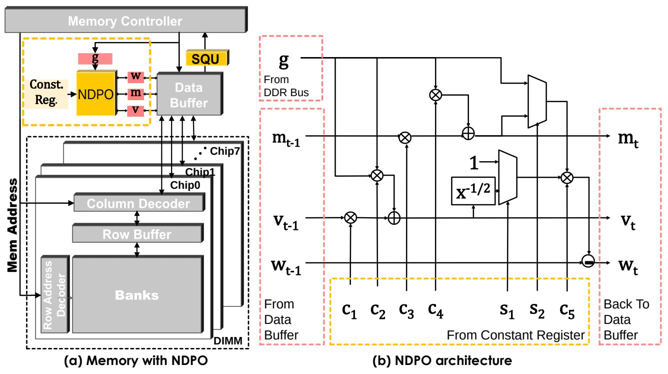

Fig. 10. Architecture of NDP Engine.

3) NDPO: Table IV lists the commonly used optimizer (i.e., $h()$) in *Updating weights*, including AdaGrad [15], RMSProp [25] and Adam [36], where η denotes the learning rate, g denotes the gradient on weights, m and v are parameters maintained by optimizers, β, β_1 and β_2 are decay rates. It can be noticed that these optimizers introduce full-precision parameters that has at least the same number of weights, due to the element-wise functions of $h()$. To avoid the full bit-width data traffic, we propose a Near-Data-Processing Optimizer (NDPO) aside the memory to leverage the in-place computation. As the functionality required for *Updating weights* is relatively simple, the NDP Engine, which is put aside the original memory controller, introduces only small overhead. In our evaluation, the cost of entire NDP engine is 0.49mm^2 and NDPO cost is only 0.07mm^2.

Figure 10 (a) shows the architecture of memory in the NDP Engine. The NDP Engine enhances the memory with a NDPO for computation and several registers for buffering weights and optimizer parameters. The NDPO is designed to support all the optimizers listed in Table IV. As all the optimizers can be summarized by the following formulas:

$$\begin{aligned} m_t &= c_1 \times m_{t-1} + c_2 \times g & v_t &= c_3 \times v_{t-1} + c_4 \times g^2 \\ t_1 &= m_t \textbf{ or } g & t_2 &= v^{-\frac{1}{2}} \textbf{ or } 1 \\ w_t &= w_{t-1} - c_5 \times t_1 \times t_2, \end{aligned} \tag{1}$$

we design the NDPO to perform above formulas, see Figure 10 (b). For example, for Adam optimization, Formula 1 has $c_1 = \beta_1$, $c_2 = 1 - \beta_1$, $c_3 = \beta_2$, $c_4 = 1 - \beta_2$, $c_5 = \eta\frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$, $s_1 = 1$ and $s_2 = 1$. NDP Engine first sets the corresponding parameter registers according to the parameters from controller in the Acceleration Core. The memory controller sends three successive ACTIVATE signals to the memory, where the rows storing weight value and corresponding m and v values are activated from the cell array. Then the memory controller in Cambricon-Q sends WRITE signals with a gradient value g and a destination column address through the DDR bus, where three specific values from data buffer, namely w_{t-1}, m_{t-1} and v_{t-1}. The updated values w_t, m_t and v_t are computed by the NDPO bases on Optimizer (i.e., $h()$), and sent back to the data

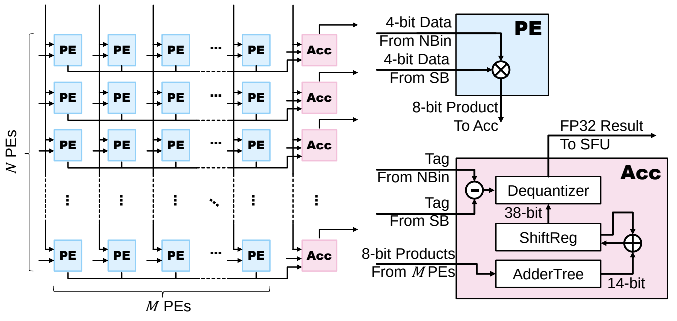

Fig. 11. The PE Array consists of low bit-width PEs and high bit-width accumulators (ACC).

buffer. After the current row is finished, the memory controller sends three successive PRECHARGE signals to write back the updated weights and optimizer parameters to the cell array.

C. Instruction Set

Cambricon-Q adopts a tensor-based instruction set, which supports high-level operations, such as convolution (CONV), matrix multiply (MM), vector operations (e.g. VMUL), vector-scalar operations (e.g. VFMUL), horizontal vector operations (e.g. HMUL), vector load/store (VLOAD/VSTORE) and stripe load/store (SLOAD/SSTORE). Especially, to efficiently support quantized DNN training as well as manage the NDP Engine, we also include several specially designed instructions, as listed in Table V. Particularly, QSTORE/QLOAD/QMOVE performs data storing/loading/moving with on-the-fly quantization on Acceleration Core/NDP Engine/Acceleration Core, CROSET performs NDP configuration, and WGSTORE perform in-place weight update in NDP Engine.

D. Function Units

Cambricon-Q consists of two computing units, the PE Array and the SFU. The PE Array is designed to perform high parallel tensor operations and the SFU is designed to perform scalar functions including non-linear operations.

TABLE IV
COMMONLY-USED OPTIMIZERS FOR DNN TRAINING.

Optimizer	Parameters Maintained	Computations Performed
SGD		$w_t = w_{t-1} - \eta \times g$
AdaGrad [15]	m	$m_t = m_{t-1} + g^2$ $w_t = w_{t-1} - \eta \times g \times m_t^{-\frac{1}{2}}$
RMSProp [25]	m	$m_t = \beta \times m_{t-1} + (1-\beta) \times g^2$ $w_t = w_{t-1} - \eta \times g \times m_t^{-\frac{1}{2}}$
Adam [36]	m, v	$m_t = \beta_1 \times m_{t-1} + (1-\beta_1) \times g$ $v_t = \beta_2 \times v_{t-1} + (1-\beta_2) \times g^2$ $\hat{m}_t = m_t/(1-\beta_1^t)$ $\hat{v}_t = v_t/(1-\beta_2^t)$ $w_t = w_{t-1} - \eta \times \hat{m}_t \times \hat{v}_t^{-\frac{1}{2}}$

TABLE V
THE PROPOSED ISA IN CAMBRICON-Q.

Type	Operation	Example
Control	Set DDR Constants	CROSET creg_id, imm
Data I/O	Vector I/O Stripe I/O On-the-fly Quantized I/O Store and 0ptimize	VLOAD dest, src, size SLOAD dest, src, dest_str, src_str, size, n QSTORE dest, src, size WGSTORE dest, dest2, dest3, src, size
Compute	Matrix Multiply 2D Convolution Vector Operations	MM dest, lsrc, rsrc, m, n, k CONV dest, weight, src, ... VMUL, VFMUL, HMul, ...

Figure 11 shows the organization of PE Array, which consists of $N \times M$ PEs, and N Accumulators. Each PE is capable of processing a 4-bit multiplication, where a 8-bit output is generated and sent to Accumulator. Each Accumulator consists of an adder-tree, a shift-adder, and a dequantizer, see Figure 11. In this paper, we choose $M = N = 64$ to fit in edge devices. The adder-tree accumulates the results of its M connected PEs, receiving 8-bit inputs from PEs and generating 14-bit summation result without loss of numerical precision. The shift-adder is used to realize computation of wider bit-width inputs in a time-serial manner, therefore PE Array could perform 4-bit, 8-bit, 12-bit and 16-bit quantization with 4-bit operators. The dequantizer is used to dequantize the final 38-bit result into 32-bit floating-point format for later operations such as full bit-width weight update precisely. Please note that instead of dequantizing inputs ahead of computation, Cambricon-Q only perform dequantization on the accumulated results in Accumulators. Thus, it not only reduces the required dequantization logic from $M \times (N+1)$ shifters to N adders and N shifters, but also reduces the hardware overhead of PEs and Accumulator with 4-bit operators.

V. METHODOLOGY

In this section, we first introduce the experimental setup including the benchmarks and the hardware platform configurations for comparison.

A. Benchmarks

Our benchmark set (Table VI) covers a wide spectrum of efficiency-optimized NN models including both CNNs and recurrent networks. We test the state-of-the-art dynamic quantization algorithms (Zhu [65] and Zhang [62]), and the tailored version with optimization of HQT on them (Zhu [65]+HQT

TABLE VI
BENCHMARKS.

Model	Dataset	Batchsize
AlexNet [37]	ImageNet [52]	32
ResNet-18 [24]	ImageNet	32
GoogLeNet [57]	ImageNet	32
SqueezeNet-V1 [29]	ImageNet	32
Transformer-Base [59]	WMT17 [5]	260
PTB-LSTM-Medium [26]	PennTreeBank [43]	1000

and Zhang [62]+HQT). We report the average execution time and energy consumption per minibatch of the whole training process. Considering the resource-constrained characteristic of our experimenting platforms, the minibatch size is set to best fit the memory space. Data is quantized into 8-bit format whenever possible.

B. Hardware Configurations

We compare the training performance and hardware cost with the most commonly-used architectures: GPU and TPU.

a) Cambricon-Q: We implement Cambricon-Q in Verilog RTL. To obtain the area and power, we synthesize and place&route the RTL code with Synopsys toolchains under TSMC 45 nm technology. We use CACTI 7 [4] and DESTINY [50] to model the DRAM memory and on-chip SRAM buffers. Due to the unbearable long duration of silicon simulation, we also implement a cycle-accurate performance simulator to evaluate the total execution latency (cycles). The simulator also reports the exact memory traces (Ramulator [35] is integrated) and module activities, which is then used to calculate dynamic energy consumptions. The Cambricon-Q has the 64×64 4-bit PE Array working at 1 GHz, providing a peak performance of 8 Tops @ INT4 / 2 Tops @ INT8, with the memory bandwidth of 17.06 GB/s. Please note Cambricon-Q performs the two algorithms in a same manner but with different parameters, and thus we have the same performance and energy numbers.

b) GPU: We choose NVIDIA Jetson TX2 as the GPU baseline platform since the comparable hardware configuration. The GPU has 256 CUDA cores (each able to perform two FP16 FMA operations in one cycle) running at 1302 MHz max frequency, providing a peak performance of 1.33 TFlops @ FP16 [18], with the memory bandwidth of 59.7 GB/s. We deploy our benchmarks on PyTorch 1.5 [49], CUDA 10.0 and cuDNN 7.6.3. We measure the execution time with `nvprof` and the energy comsumption with an Aitek AWE1611 power analyzer. We run the training in mixed data format to enable the best possible performance, since the GPU do not have hardware INT8 support. To support mixed precision in training we use Apex [46], a PyTorch extension provided by NVIDIA.

c) TPU: We re-implement TPU architecture as a simulator, based on SCALE-Sim [53]. We expand SCALE-Sim with necessary features to support quantized DNN training, including mixed precision, the backward process, statistic function units and quantization units. The overall architecture is organized as Figure 4(c) shown. For a fair comparison, we align the hardware configuration with Cambricon-Q. More specifically, it has the 32×32 8-bit PE Array running at 1 GHz, providing a peak performance of 2 Tops @ INT8, 256 KB SRAM NBin, 512 KB SRAM SB and 256 KB SRAM NBout, with the memory bandwidth limited at 17.06 GB/s. Additionally, for a fair comparison, we run the HQT quantized DNN training on TPU, too, to collect the performance and energy numbers.

VI. Experimental Results

In this section we first evaluate the power/area overhead of Cambricon-Q, then we compare the performance of Cambricon-Q, TPU, and GPU. Finally, we analyze the insights of Cambricon-Q design.

TABLE VII
HARDWARE CHARACTERISTICS.

-	Area (mm^2)	(%)	Power (mW)	(%)
Acceleration Core	8.69	100	891.37	100
SQU	0.42	4.88	122.67	13.76
QBC	0.09	0.99	1.69	0.19
FU	2.11	24.28	483.88	54.29
NBin	1.31	15.11	6.28	0.70
SB	1.52	17.45	9.65	1.08
NBout	0.72	8.29	4.43	0.50
Decode	0.11	1.23	50.04	5.61
IB	0.36	4.14	25.28	2.84
MC	0.23	2.65	83.00	9.31
PHY	1.83	21.00	104.45	11.72
NDP Engine	0.49	100	138.94	100
SQU	0.42	86.70	122.67	88.29
NDPO	0.07	13.30	16.27	11.71

A. Hardware Characteristics

The detailed hardware characteristics are listed in Table VII. The Acceleration Core in Cambricon-Q occupies 8.69 mm^2 area, consuming a power of 891.37 mW, at the technology of 45 nm. It can be observed that Cambricon-Q can efficiently process on-the-fly statistic-based quantized DNNs training at low hardware costs, only 5.87% ($0.51\,mm^2$) extra area and 13.95% ($124.36\,mW$) extra power consumption. We also evaluate the additional components introduced in the NDP Engine (i.e. SQU and NDPO). They occupy 0.49 mm^2 area and consuming a power of 138.94 mW, also low hardware costs when considering the large dram.

B. Performance

We compare the performance of Cambricon-Q against TPU and GPU on 6 network models based on two state-of-the-art statistic quantization algorithms.

a) Accuracy: Table VIII compares the training accuracy results on our Cambricon-Q against two state-of-the-art quantization algorithm, as well as the original FP32 unquantized training. Zhu [65] works well on 4 CNN benchmarks, where Cambricon-Q achieves $\leqslant 0.2\%$ accuracy loss, -0.1% on average compared with its original version. However, with Zhang [62] Cambricon-Q achieves the same or even better accuracy on 5 out of 6 benchmarks (+0.1% on average

TABLE VIII
TRAINING ACCURACY RESULTS.

Model	FP32	Zhu [65]	+HQT	Zhang [62]	+HQT
AlexNet	58.0	57.7	57.6	**58.0**	**58.0**
ResNet-18	70.1	69.6	69.5	69.6	**70.0**
GoogLeNet	72.8	72.0	72.0	**72.8**	**72.8**
SqueezeNet	58.5	57.3	57.1	**58.1**	**58.1**
Transformer (BLEU)	25.6	-	-	24.9	**25.0**
LSTM (Perplexity[*])	115.29	423.39	425.71	**115.25**	116.04

[*]Lower is better.

of CNNs and +0.4% BLEU on Transformer), only slightly underperforms on LSTM (+0.7% perplexity). The reason is because HQT could adjust quantization parameters in a finer-grained manner, leading to a better quantization quality.

b) Speedup: Figure 12(a) shows the performance improvement of Cambricon-Q over the GPU and TPU baseline. Overall, Cambricon-Q outperforms GPU by a factor of 4.20×, TPU 1.70×, averaging on the two quantization algorithms and six DNN models. The results show that Cambricon-Q can boost performance of statistic quantization algorithms by moving the precision-sensitive data movement out of critical path.

In the further step, we breakdown the training epoch into six parts, including forward pass (FW), backward pass (computing gradients on neurons (NG), computing gradients on weights (WG), updating weights (WU)), statistic analysis (S), and quantization (Q), as shown in Figure 12(b). Cambricon-Q achieves least speedup on AlexNet (2.09×), but most speedup over TPU on AlexNet (2.07×). The main reason is because AlexNet has the most number of weights, therefore costly in Updating weights. But Cambricon-Q performs such operation on the NDP Engine, which saves the time of data transfer from memory. As the most difference between TPU and Cambricon-Q is the HQT, we can observe that most of benefits are from the on-the-fly statistic and quantization, where TPU and GPU hardly benefit from the dynamic quantization techniques. It can also confirm that backward pass is much more costly than forward on both TPU and Cambricon-Q.

C. Energy

We report the energy comparison of TPU, GPU, and Cambricon-Q across the six neural networks and the result is shown in Figure 12(c). Overall, Cambricon-Q achieves 1.62×, and 6.41× better energy efficiency compared to TPU and GPU, respectively. To clearly show the energy efficiency sources, we further breakdown the energy into different components, including functional modules in Acceleration Core (ACC), on-chip buffer (BUF), memory standby (DDR-SB), and memory dynamic (DDR-DY), as shown in Figure 12(d). Energy efficiency of Cambricon-Q comes from two parts: 1) the elimination of memory traffic, including both high precision data movements and extra data access, resulting 1.54× energy reduction on memory side; 2) the computation units with lower bit-width, which is less significant.

VII. Discussion

A. Performance scalability of Cambricon-Q

To show the efficiency of Cambricon-Q, we compare Cambricon-Q against two high-end GPUs: desktop-level Nvidia 1080Ti [11] and server-level Nvidia V100 [12]. With 17.64% and 1.6% peak performance of 1080Ti and V100, Cambricon-Q achieves 33.29% and 8.55% performance, 2.16× and 2.27× better efficiency, respectively, when training the ResNet-18.

Moreover, to achieve high throughput, we scale Cambricon-Q to Cambricon-Q-T and Cambricon-Q-V, which have comparable peak performance as 1080Ti and V100, respectively. Cambricon-Q-T and Cambricon-Q-V are organized in a similar way as Tangram [19]. For Cambricon-Q-T, we increase the number of PE array in Cambricon-Q to eight, thus a total 16 Tops@INT8 perk performance (141.09% of 1080Ti's 11.34 Tflops), where each PE array has its SB for private weights and all the PE arrays share the NBin for broadcast neurons. We scale the memory bandwidth for Cambricon-Q-T 4× up to 68.24 GB/s (14.10% of 1080Ti's 484 GB/s bandwidth). For Cambricon-Q-V, we increase the one PE array in Cambricon-Q to an 8×8 2D mesh, thus a 128 Tops@INT8 peak performance (102.4% of V100's 125 Tflops), where each column in the 2D mesh shares the same weights from SB and each row shares the same neurons from NBin for batch parallelism. We scale the memory bandwidth for Cambricon-Q-V 16× up to 272.96 GB/s (30.33% of V100's 900 GB/s bandwidth).

Figure 13 reports the performance of Cambricon-Q, Cambricon-Q-T, and Cambricon-Q-V, when compared against Jetson TX2, 1080Ti, and V100, respectively. We choose the best minibatch size settings on each hardware. Cambricon-Q-T and Cambricon-Q-V runs faster on both ResNet-18 and LSTM than corresponding GPUs, showing good performance scalability.

B. Generalized to other quantization methods

Cambricon-Q has been demonstrated to well support a bunch of state-of-the-art statistic-based quantization algorithms with different data formats, statistical methods and quantization policies, see Section III and IV. The flexibility of Cambricon-Q is achieved by the proposed configurable architecture: (1) the Quant Unit in SQU / QBC can support different formats such as INT4/INT8/INT12/INT16; (2) the Arbiter/State Unit can support various statistical methods such as Max Absolute Value, Rectilinear Distance, and Mean Bias.

Actually, Cambricon-Q can efficiently support all statistic-based quantization algorithms with two common and general characteristics via HQT III: (1) the scale statistic (i.e., θ) only depends on the original data X; (2) the error estimation statistic depends on X and the dequantized data X'. As long as future statistic-based quantization algorithms fit in the two characteristics, Cambricon-Q can be efficiently applied. Moreover, Cambricon-Q can easily support non-statistic-based quantization algorithms by simply bypassing the Stat unit.

C. Generalized to other low-bitwidth PEs

Cambricon-Q supports other low-bitwidth data such as INT4 directly with 4-bit operators in PE. The reason is two-fold. First, Cambricon-Q with 4-bit PE and serial computation can be flexible for supporting more data precisions, including INT8, INT12, INT16, and other precisions that are multiples of 4. Therefore, Cambricon-Q with 4-bit PE can directly support models that are capable of 4-bit precision (e.g., inference) and efficiently support models that have mixed precision. For 4-bit models, Cambricon-Q can further increase the performance/energy efficiency by 2.33x/2.35x if switched to 4-bit. Second, Cambricon-Q with 4-bit PE is able to maintain

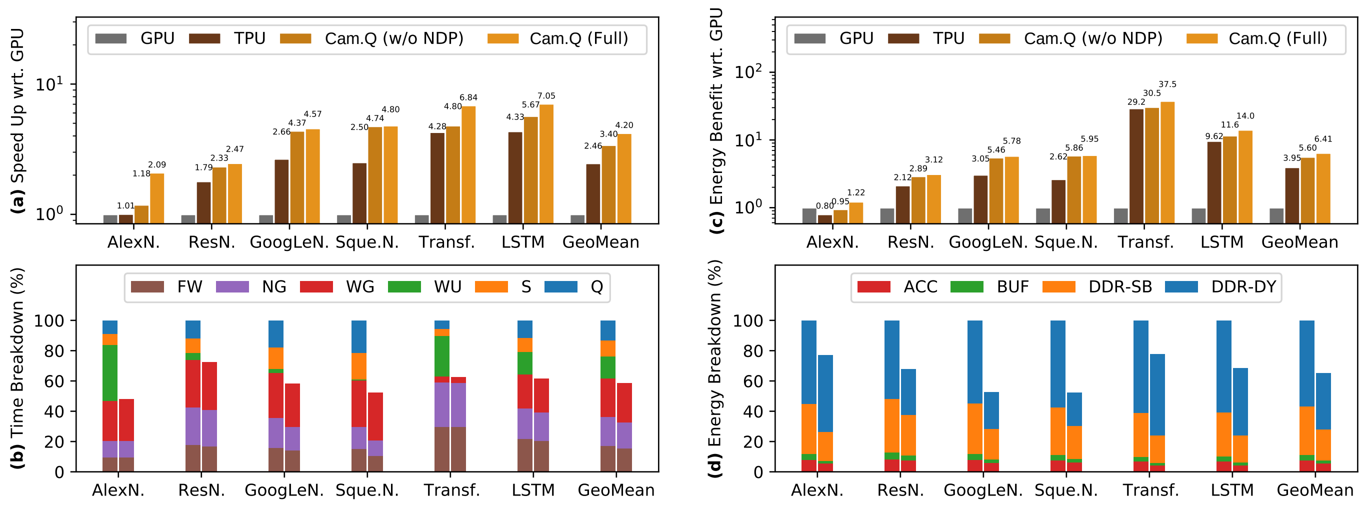
Fig. 12. Performance/energy comparison and breakdown.

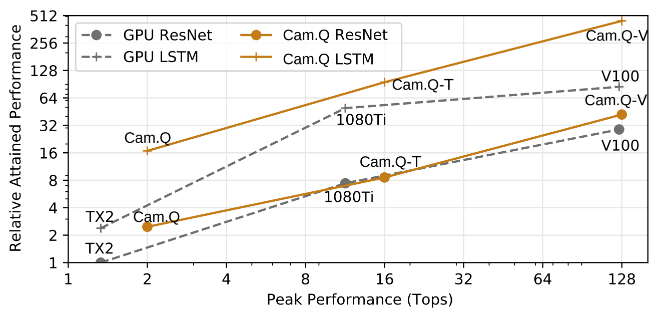
Fig. 13. Performance when scaling Cambricon-Q to Cambricon-Q-T and Cambricon-Q-V.

high throughput easily with regard to bit-serial computing. Bit-serial computing is commonly introduced to address the varying bitwidth issue, but it requires multiples of PEs to deliver high performance and high parallelism in data to avoid low utilization of PEs [34], [54]. As a result, Cambricon-Q adopts 4-bit as the basic operator but leveraging time-serial methods to support various precision.

D. *Cambricon-Q without NDP*

As analyzed in Section II-B, data movements in *weight updating (WU)* can also be critical. Therefore NDPO is required to achieve the most efficient training. Figure 12 also report the speed up and energy benefit of Cambricon-Q wihout NDP Engine. On several models which involve heavy WU processes (i.e. AlexNet, Transformer), Cambricon-Q without NDP can only achieve negligible improvements on both performance and energy. On the other hand, on models whose WU take marginal proportions (i.e. GoogleNet, SqueezeNet), Cambricon-Q without NDP achieves similar improvements with Cambricon-Q with NDP. It can be seen that NDP Engine is critical to support arbitrary DNN models efficiently, and achieves non-negligible improvements over the baseline on average (38% performance / 42% energy efficiency improvements over TPU).

E. *Comparison to approximate PIM architecture*

The near-data-processing scheme adopted by Cambricon-Q allows the acceleration core only performing low-precision computation for high efficiency. Unlike previous PIM designs that usually allocate the low-precision or approximate computations in PIM architecture for better efficiency, Cambricon-Q keep the essential full-precision of weight update on the memory side. Specifically, we propose a rank-level NDP design for the full-precision weight update, our NDP engine can be configured to support various training optimizers including SGD, AdaGrad, RMSProp, and Adam.

VIII. Related Work

A. *Quantization Algorithms*

Quantization techniques are commonly used to reduce the computation, storage, and communication overhead by using lower bit-width to represent data. Although quantization techniques can be applied in both inference and training processes, precision sensitivity of inference and training is different. Previous works empirically demonstrated that 16-bit fixed-point data was able to handle most of image detection inference tasks [9], [10]. More studies were spawned to push the limit to the border of 8 bit-width, 4 bit-width, or even lower bit-width [28]. Training stage, however, is much more challenging to apply quantization with retained accuracy. Training data, especially gradients, are more sensitive to the quantization methodology, since the lower bit-width may damage the model accuracy. Recently, statistic-based quantization techniques [8] have been proposed, which successfully achieved 8 bit-width on most of training data. The key idea was to determine the quantization parameters separately for weights, neuron activations, and gradients according to the run-time statistic of data distribution. Our work envisions the inefficiency of

TABLE IX
RECENT QUANTIZED-TRAINING-AWARE ACCELERATORS.

	Cambricon-Q	Agrawal 2021 [1]	Oh 2020 [47]	Lee 2019 [40]	Wang 2018 [60]	Fleischer 2018 [17]
Supported data format	FxP, INT	HFP8 [56], FP16	DLFloat16 [2]	FGMP [40]	FP8 [60]	FP16
Supported training bit-width	4/8/12/16	8/16	16	8/16	8	16
Dynamic quantization support	✔	✗	✗	Threshold-based	✗	✗
Extra cost in weight update	-	Round-off Residual†	-	-	Stochastic Rounding‡	-
ResNet-18 accuracy @ Bit-width	70.0% @ 8/16	69.39% @ 8	-	68.19% @ 8/16	65.74% @ 8	-
Technology	45 nm	7 nm	14 nm	65 nm	-	14 nm
TOPS/W	2.24 @ INT8	1.9 @ FP8	1.1 @ FP16	1.63 @ FP8	-	-

† Introduces additional processing pass and 16-bit data on weights. ‡ Requires random number generation, not implemented in the proposed hardware.

deploying such smart quantization techniques on existing deep learning architectures and proposes Cambricon-Q architecture to reduce the overhead of run-time quantization and full-precision computing in optimizers during statistic-based quantization.

B. DNN Architectures for Quantization

Most existing DNN accelerator architectures focus on leveraging quantization to accelerate DNN inference [34], [39], [48], [54], [58]. Stripes applied quantization to each layer of CNN networks and got 2∼13 bit-width for inference [34]. BISMO [58] and BitFusion [54] proposed bit-serial multiplication designs for reconfigurably executing DNN models with variable quantized bit-width. OLAccel utilized quantization with both 4-bit and 16-bit MACs [48]. ShapeShifter [39] proposed loss-less memory compression techniques for width adaption. Boroumand *et al.* proposed the PIM design to quantize data near memory for data movement reduction [6]. DRQ tried dynamic region-based quantization for 4-bit or 8-bit data during DNN inference [55]. BiScaled-DNN proposes the new data format for quantification [32]. HBFP [14] leverage the quantized data format to increase the hardware density of the PE arrays with the BFP data format, but can not ensure the training accuracy and benefit from quantization in data transmission and storage for high energy efficiency and performance. Existing architectures focus on leveraging quantization to accelerate the inference. During the training stage, the variability and precision-sensitivity of data distribution makes it much more challenging to efficiently apply quantization techniques [13], [21]. Cambricon-Q innovatively proposes the composite design of ASIC acceleration and NDP-based optimizers with local dynamic quantization algorithms to efficiently accelerate DNN training with retained accuracy.

Table IX compares Cambricon-Q with recently proposed quantized-training-aware ASIC accelerators. Regarding the supported training bit-width, Cambricon-Q supports 4/8/12/16-bit fixed point arithmetics, while B. Fleischer *et al.* [17] and J. Oh *et al.* [47] only quantize to 16-bit which is much less efficient than other 8-bit and 4-bit implementations. Regarding the support of dynamic quantization, Cambricon-Q provided the on-the-fly statistic and quantization support to dynamically, efficiently and properly re-scale the data, while all the previous accelerators except J. Lee *et al.* [40] lack architectural support thus require a proper scaling factor set manually to converge. Regarding the extra overhead in the weight update process, Cambricon-Q proposed the NDP Engine to minimize the overhead of the weight update process, while A. Agrawal *et al.* [1] (based on HFP8 [56] format) introduces round-off residuals which costs 13.7% more training time. Regarding the accuracy retained, Cambricon-Q retained the highest accuracy with $\leqslant 0.1\%$ loss compared to the FP32 baseline (70.1%), while J. Lee *et al.* [40] and N. Wang *et al.* [60] suffer from non-negligible accuracy loss ($> 1\%$). Cambricon-Q also achieves the highest performance efficiency (2.24 TOPS/W @ INT8 mode, 45 nm). Therefore, Cambricon-Q is the first architecture that efficiently supports quantized training with negligible accuracy loss.

IX. CONCLUSION

In this paper, we propose a novel hybrid architecture, called Cambricon-Q, to efficiently process quantized DNN training. Roughly, Cambricon-Q leverages our proposed hardware-friendly quantization technique as well as the hybrid AISC acceleration core and NDP Engine hardware to address on-the-fly statistic quantization DNN training. Cambricon-Q is able to reduce the *extra* data access as well as *high precision* data access, achieving 4.20× and 1.70× better performance, 6.41× and 1.62× better energy efficiency over the GPU and TPU, respectively. To the best of our knowledge, Cambricon-Q is the first architecture that can perform quantized DNN training with only negligible accuracy loss.

ACKNOWLEDGEMENT

This work is partially supported by the NSF of China (under Grants 61925208, 61732007, 61732002, 61906179, U19B2019, U20A20227), the National Key Research and Development Program of China (under Grant 2020AAA0103802, 2018AAA0103300, 2017YFA0700900), Beijing Natural Science Foundation (JQ18013), Strategic Priority Research Program of Chinese Academy of Science (XDB32050200), Youth Innovation Promotion Association CAS, Beijing Academy of Artificial Intelligence (BAAI) and Beijing Nova Program of Science and Technology (Z191100001119093) ,Youth Innovation Promotion Association CAS and Xplore Prize.

This work is initiated from an internal project of Cambricon Technologies.

References

[1] A. Agrawal, S. K. Lee, J. Silberman, M. Ziegler, M. Kang, S. Venkataramani, N. Cao, B. Fleischer, M. Guillorn, M. Cohen, S. Mueller, J. Oh, M. Lutz, J. Jung, S. Koswatta, C. Zhou, V. Zalani, J. Bonanno, R. Casatuta, C. Y. Chen, J. Choi, H. Haynie, A. Herbert, R. Jain, M. Kar, K. H. Kim, Y. Li, Z. Ren, S. Rider, M. Schaal, K. Schelm, M. Scheuermann, X. Sun, H. Tran, N. Wang, W. Wang, X. Zhang, V. Shah, B. Curran, V. Srinivasan, P. F. Lu, S. Shukla, L. Chang, and K. Gopalakrishnan, "A 7nm 4-core ai chip with 25.6tflops hybrid fp8 training, 102.4tops int4 inference and workload-aware throttling," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 144–146.

[2] A. Agrawal, S. M. Mueller, B. M. Fleischer, X. Sun, N. Wang, J. Choi, and K. Gopalakrishnan, "Dlfloat: A 16-b floating point format designed for deep learning training and inference," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, 2019, pp. 92–95.

[3] J. Albericio, P. Judd, A. D. Lascorz, S. Sharify, and A. Moshovos, "Bit-Pragmatic Deep Neural Network Computing," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 1–16.

[4] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, Jun. 2017.

[5] O. Bojar, R. Chatterjee, C. Federmann, Y. Graham, B. Haddow, S. Huang, M. Huck, P. Koehn, Q. Liu, V. Logacheva, C. Monz, M. Negri, M. Post, R. Rubino, L. Specia, and M. Turchi, "Findings of the 2017 conference on machine translation (wmt17)," in *Proceedings of the Second Conference on Machine Translation, Volume 2: Shared Task Papers*. Copenhagen, Denmark: Association for Computational Linguistics, September 2017, pp. 169–214.

[6] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 316–331.

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[8] C. Chen, J. Choi, K. Gopalakrishnan, V. Srinivasan, and S. Venkataramani, "Exploiting approximate computing for deep learning acceleration," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 821–826.

[9] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems (ASPLOS)*, Salt Lake City, UT, USA, 2014, pp. 269–284.

[10] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 367–379.

[11] N. Corporation, "NVIDIA GTX 1080Ti User Guide," https://www.nvidia.com/content/geforce-gtx/GTX_1080_Ti_User_Guide.pdf, 2017.

[12] N. Corporation, "NVIDIA Tesla V100 GPU Architecture," https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf, 2018.

[13] D. Das, N. Mellempudi, D. Mudigere, D. Kalamkar, S. Avancha, K. Banerjee, S. Sridharan, K. Vaidyanathan, B. Kaul, E. Georganas, A. Heinecke, P. Dubey, J. Corbal, N. Shustrov, R. Dubtsov, E. Fomenko, and V. Pirogov, "Mixed precision training of convolutional neural networks using integer operations," in *International Conference on Learning Representations*, 2018.

[14] M. Drumond, T. Lin, M. Jaggi, and B. Falsafi, "Training dnns with hybrid block floating point," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 451–461.

[15] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. null, p. 2121–2159, Jul. 2011.

[16] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, 2018.

[17] B. Fleischer, S. Shukla, M. Ziegler, J. Silberman, J. Oh, V. Srinivasan, J. Choi, S. Mueller, A. Agrawal, T. Babinsky, N. Cao, C. Chen, P. Chuang, T. Fox, G. Gristede, M. Guillorn, H. Haynie, M. Klaiber, D. Lee, S. Lo, G. Maier, M. Scheuermann, S. Venkataramani, C. Vezyrtzis, N. Wang, F. Yee, C. Zhou, P. Lu, B. Curran, L. Chang, and K. Gopalakrishnan, "A scalable multi- teraops deep learning processor core for ai trainina and inference," in *2018 IEEE Symposium on VLSI Circuits*, 2018, pp. 35–36.

[18] Franklin, Dustin, "NVIDIA Jetson TX2 delivers twice the intelligence to the edge," https://developer.nvidia.com/blog/jetson-tx2-delivers-twice-intelligence-edge/, 2017.

[19] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "Tangram: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019.

[20] Google, "Cloud TPU documentation," https://cloud.google.com/tpu/docs/tpus.

[21] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," *CoRR*, vol. abs/1502.02551, 2015.

[22] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 243–254.

[23] S. Han, H. Mao, and W. J. Dally, "Deep Compression - Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *Iclr*, pp. 1–13, 2016.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[25] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning: Overview of mini-batch gradient descent," https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, 2020.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997.

[27] M. Horowitz, "Computing' s Energy Problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 10–14.

[28] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, p. 6869–6898, Jan. 2017.

[29] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: AlexNet-level accuracy with 50x fewer parameters and $<$1mb model size," *CoRR*, vol. abs/1602.07360, 2016.

[30] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "FloatPIM : In-Memory Acceleration of Deep Neural Network Training with High Precision," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019.

[31] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[32] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, K. Gopalakrishnan, and L. Chang, "Biscaled-dnn: Quantizing long-tailed datastructures with two scale factors for deep neural networks," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019.

[33] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-L. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. Mackean, A. Maggiore,

M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA'17)*, 2017, pp. 1–17.

[34] P. Judd, J. Albericio, and A. Moshovos, "Stripes: Bit-Serial Deep Neural Network Computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, vol. 6056, no. c, 2016, pp. 1–1.

[35] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[37] A. Krizhevsky, G. E. Hinton, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances In Neural Information Processing Systems*, pp. 1–9, 2012.

[38] A. D. Lascorz, P. Judd, D. M. Stuart, M. Mahmoud, K. Siu, and A. Moshovos, "Bit-Tactical : A Software / Hardware Approach to Exploiting Value and Bit Sparsity in Neural Networks," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 749–763.

[39] A. D. Lascorz, S. Sharify, I. Edo, D. M. Stuart, O. M. Awad, P. Judd, M. Mahmoud, M. Nikolic, K. Siu, Z. Poulos, and A. Moshovos, "Shapeshifter: Enabling fine-grain data width adaptation in deep learning," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 28–41.

[40] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H. Yoo, "7.7 lnpu: A 25.3tflops/w sparse deep-neural-network learning processor with fine-grained mixed precision of fp8-fp16," in *2019 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2019, pp. 142–144.

[41] C. Li, "OpenAI's GPT-3 Language Model: A Technical Overview," https://lambdalabs.com/blog/demystifying-gpt-3/, 2020.

[42] S. Li, D. Niu, K. T. Malladi, B. Brennan, and H. Zheng, "DRISA : A DRAM-based Reconfigurable In-Situ Accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, vol. 14, 2017, pp. 288–301.

[43] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Comput. Linguist.*, vol. 19, no. 2, p. 313–330, Jun. 1993.

[44] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," *arXiv preprint arXiv:1710.03740*, 2017.

[45] Nvidia, "NVIDIA A100 TENSOR CORE GPU," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet.pdf.

[46] NVIDIA, "Apex API Documentation," https://nvidia.github.io/apex/, 2019.

[47] J. Oh, S. K. Lee, M. Kang, M. Ziegler, J. Silberman, A. Agrawal, S. Venkataramani, B. Fleischer, M. Guillorn, J. Choi, W. Wang, S. Mueller, S. Ben-Yehuda, J. Bonanno, N. Cao, R. Casatuta, C. Y. Chen, M. Cohen, O. Erez, T. Fox, G. Gristede, H. Haynie, V. Ivanov, S. Koswatta, S. H. Lo, M. Lutz, G. Maier, A. Mesh, Y. Nustov, S. Rider, M. Schaal, M. Scheuermann, X. Sun, N. Wang, F. Yee, C. Zhou, V. Shah, B. Curran, V. Srinivasan, P. F. Lu, S. Shukla, K. Gopalakrishnan, and L. Chang, "A 3.0 tflops 0.62v scalable processor core for high compute utilization ai training and inference," in *2020 IEEE Symposium on VLSI Circuits*, 2020, pp. 1–2.

[48] E. Park, D. Kim, and S. Yoo, "Energy-efficient Neural Network Accelerator Based on Outlier-aware Low-precision Computation," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, 2018.

[49] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8026–8037.

[50] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "Destiny: A tool for modeling emerging 3d nvm and edram caches," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 1543–1546.

[51] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 58–70.

[52] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[53] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[54] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 764–775.

[55] Z. Song, B. Fu, F. Wu, Z. Jiang, L. Jiang, N. Jing, and X. Liang, "DRQ : Dynamic Region-based Quantization for Deep Neural Network Acceleration," in *Proceedings of the 47th International Symposium on Computer Architecture (ISCA)*, 2020, pp. 1010–1021.

[56] X. Sun, J. Choi, C.-Y. Chen, N. Wang, S. Venkataramani, V. V. Srinivasan, X. Cui, W. Zhang, and K. Gopalakrishnan, "Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[57] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[58] Y. Umuroglu, L. Rasnayake, and M. Själander, "Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 307–3077.

[59] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[60] N. Wang, J. Choi, D. Brand, C. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *NeurIPS*, 2018.

[61] Y. Yang, S. Wu, L. Deng, T. Yan, Y. Xie, and G. Li, "Training high-performance and large-scale deep neural networks with full 8-bit integers," *Neural Networks*, 2020.

[62] X. Zhang, S. Liu, R. Zhang, C. Liu, D. Huang, S. Zhou, J. Guo, Y. Kang, Q. Guo, Z. Du *et al.*, "Fixed-point back-propagation training," in *CVPR*, 2020.

[63] R. Zhao, Y. Hu, J. Dotzel, C. De Sa, and Z. Zhang, "Improving neural network quantization without retraining using outlier channel splitting," *arXiv preprint arXiv:1901.09504*, 2019.

[64] K. Zhong, T. Zhao, X. Ning, S. Zeng, K. Guo, Y. Wang, and H. Yang, "Towards lower bit multiplication for convolutional neural network training," *arXiv preprint arXiv:2006.02804*, 2020.

[65] F. Zhu, R. Gong, F. Yu, X. Liu, Y. Wang, Z. Li, X. Yang, and J. Yan, "Towards unified int8 training for convolutional neural network," *arXiv preprint arXiv:1912.12607*, 2019.