Design Automation for Embedded Systems (2024) 28:91–110
https://doi.org/10.1007/s10617-024-09282-2

# Novel adaptive quantization methodology for 8-bit floating-point DNN training

**Mohammad Hassani Sadi**[1] · **Chirag Sudarshan**[2] · **Norbert Wehn**[1]

Received: 29 June 2023 / Accepted: 18 January 2024 / Published online: 16 February 2024


**Abstract**
There is a high energy cost associated with training Deep Neural Networks (DNNs). Off-chip memory access contributes a major portion to the overall energy consumption. Reduction in the number of off-chip memory transactions can be achieved by quantizing the data words to low data bit-width (E.g., 8-bit). However, low-bit-width data formats suffer from a limited dynamic range, resulting in reduced accuracy. In this paper, a novel 8-bit Floating Point (FP8) data format quantized DNN training methodology is presented, which adapts to the required dynamic range on-the-fly. Our methodology relies on varying the bias values of FP8 format to fit the dynamic range to the required range of DNN parameters and input feature maps. The range fitting during the training is adaptively performed by an online statistical analysis hardware unit without stalling the computation units or its data accesses. Our approach is compatible with any DNN compute cores without any major modifications to the architecture. We propose to integrate the new FP8 quantization unit in the memory controller. The FP32 data from the compute core are converted to FP8 in the memory controller before writing to the DRAM and converted back after reading the data from DRAM. Our results show that the DRAM access energy is reduced by 3.07× while using an 8-bit data format instead of using 32-bit. The accuracy loss of the proposed methodology with 8-bit quantized training is ≈ 1% for various networks with image and natural language processing datasets.

**Keywords** Floating-point · Deep neural network training · DRAM · Adaptive number system

## 1 Introduction

Deep Neural Network (DNN) models are rapidly evolving since the last decade to enable complex context-sensitive learning capabilities. The training phase of DNN models involves large datasets with GBs of memory footprint and high computation requirements [1]. DNN

✉ Mohammad Hassani Sadi
m.sadi@rptu.de

[1] Microelectronic Systems Design Research Group, University of Kaiserslautern-Landau, Kaiserslautern, Germany

[2] Forschungszentrum Jülich GmbH, Peter Grünberg Institute (PGI-14), Jülich, Germany

training on compute platforms such as GPU, TPU, and other hardware accelerators consumes extremely high energy. For instance, training the GPT-3 model consumes 1287 megawatt-hours (MWh) [2]. Many prior research works [3, 4] focused on reducing the DNN training energy by optimizing the computation logic, but less attention is given to data access energy. However, a major portion of the total energy is consumed by off-chip memory like DRAM on the aforementioned compute platforms due to the memory-intensive nature of DNN tasks. For example, a single 64bit GDDR6-DRAM transaction consumes 350–480 pJ while a 32-bit Floating-Point (FP32) multiplication only needs 1.31 pJ [5] in the latest 7nm technology. Similar memory-computation energy gaps were observed for other technology nodes [5]. Neural network accelerators such as DaDINO [6], Cambricon-X [7], eBrain-II [8], and NeuroCube [9] published the memory energy consumption to be in the range of 50–90% of the total energy. This memory energy overhead directly impacts the overall energy efficiency, especially during training due to its high memory-intensive nature. Hence, this work focuses on reducing the DRAM data access energy for the DNN training. One solution to reduce data access energy and increase overall energy efficiency is Processing-in-Memory (PIM). However, PIM accelerators are mainly focused on the inference part and DNN training is predominantly confined to conventional platforms. Hence it is important to reduce the data access energy in conventional Non-PIM platforms.

One of the recent emerging trends to reduce the data access energy and increase the overall energy efficiency is to adapt the 8-bit Floating-Point (FP8) data format for DNN training rather than the traditional FP32 format. The FP8 data format enables the packing of more data words within a DRAM bus and thereby reduces the total number of transactions by 4× in comparison to FP32. Therefore, it leads to reduced total data access energy and increased computation throughput. The main challenge of using the FP8 data format is the limited dynamic range compared to FP32. In many instances, FP8 format's dynamic range is not sufficient for training DNN models and results in accuracy loss. For example, the data range required in the backward pass of Resnet18 (excluding the extreme outliers that are less than $10^{-40}$ = almost zero) is between $10^{-9}$ and $10^{-1}$. This range is only sufficed by FP32, BFloat16, AMD's FP24, and TinyFloat, while 16-bit Floating-Point (FP16) and FP8 does not. Table 1 compares and summarizes the various data formats and their respective range. To compensate for this limitation, there are two approaches in state-of-the-art. The first approach [10, 11] involves multiplying the data by a scaling factor to shift the data to the dynamic range of FP8. The second approach [12, 13] is to use a floating-point data format with a variable bias.[1] This approach shifts the dynamic range of FP8 to the desired data range by changing the bias value. The advantage of the variable bias approach is it only requires an additional INT8 adder for range shifting as compared to floating-point multiplications in the case of the scaling factor approach. As of now, all the aforementioned publications [10–14] have only demonstrated the possibility of training DNN models with FP8 format by relying on an offline statistical analysis of the DNN parameters and feature map data which is in FP32 format. In other words, the training is first computed in FP32 data format to log all the DNN model-related data, and then by assessing this data a suitable scaling factor or bias value is identified. This methodology is not suitable for real-world applications where the data sets and DNN models are evolving continuously. Furthermore, the requirement of performing training first with FP32 data format in offline-based approaches will effectively not yield any net energy reductions. Hence online scaling factor or bias identification would be very promising.

[1] floating-point bias, not DNN bias.

**Table 1** Comparison of Bit-width, Exponent, Mantissa, and Dynamic Range of various data format

| Format | Bit Width | Exponent | Mantissa | Dynamic range (±) |
|---|---|---|---|---|
| FP32 | 32 | 8 | 23 | $1.40e^{-45}$ - $3.4e^{38}$ |
| FP16 | 16 | 5 | 10 | $5.96e^{-5}$ - $6.5e^{4}$ |
| BFlot16 | 16 | 8 | 7 | $1.18e^{-38}$ - $3.4e^{38}$ |
| AMD FP24 | 24 | 7 | 16 | $3.30e^{-24}$ - $9.22e^{18}$ |
| TinyFloat | 12 | 7 | 4 | $1.35e^{-20}$ - $8.64e^{18}$ |
| FP8(1,5,2) | 8 | 5 | 2 | $3.05e^{-5} - 1.13e^{5}$ |
| FP8(1,4,3) | 8 | 4 | 3 | $7.81e^{-3} - 4.7e^{2}$ |

In September 2022 NVIDIA introduced the first GPU [1] (H100 Hopper) that is capable of performing the DNN training in FP8 format with builtin module for detecting the appropriate scaling factor using an online statistical unit. NVIDIA's statistical unit is capable of online computing the scaling factor at a tensor level (i.e. one scaling factor per tensor) based on the history of maximum values of previous iterations. However, there are some open issues in this context: (1) In H100 GPU the DRAM data access energy reduction advantage is only applicable if the DNN training is computed on the specialized FP8 tensor core and is not compatible with other cores. (2) NVIDIA neither present the hardware design of their statistical unit nor the results such as area, latency, and energy overhead, which limits the research and open source communities to incorporate the advantages of FP8 data type to their DNN training accelerator architectures. Furthermore, the exact methodology for online range analysis is also not made publicly available by NVIDIA, except for a brief description. (3) None of the state-of-the-art has investigated variable bias approach in combination with online statistical analysis for FP8 training.

In this work, we present a new online statistical analysis-based methodology to reduce the DRAM data access energy with FP8 data format and variable bias. In order to make our approach generic and compatible with any compute core without major architectural modifications, we propose to integrate the online statistical analysis unit in the DRAM memory controller. During the first few initial epochs of training all data is stored in DRAM in original format(i.e. 32-bit or 16-bit). These transactions are sampled by statistical analysis unit to compute the suitable bias value. The identified bias value is then employed in the rest epochs to store the DRAM data in FP8 format during the write operation and reconvert to the original format (i.e. 32-bit or 16-bit) during the read operation. Thereby reducing the overall DRAM data access energy. It is highly possible that DNN data contain extreme outlier samples. This problem is mitigated in our methodology by computing the bias value using the median as a statistical metric, which is robust to outliers in comparison to other statistical metrics like minimum, maximum, and mean values. The statistical unit in the memory controller should also ensure that no added latency is introduced to the memory transactions. There is energy overhead for bias calculation and conversion of data to FP8 and backward. This is especially a concern for the online calculation of the median value, which requires large buffers and many cycles. To maintain a low area and energy overheads, we present a novel approximate online median calculation unit. This unit is designed to operate in the background without stalling the compute core or memory data accesses (i.e. no additional latency). Furthermore, our approximate unit does not require any additional DRAM memory accesses for statistical analysis or large internal buffers. It has an area overhead of 6.3% compared to the memory

controller area in a 22 nm FD-SOI technology and a low power of 0.864 mW. Our methodology reduces the total DRAM access energy by 3.07× compared to FP32, while the training accuracy loss is ≈ 1%. All our evaluation results are based on a large number of experiments with various datasets, DNN networks, and DRAM types.

Summarized new contributions of this work are:

- A new online statistical analysis based methodology to reduce the DRAM data access energy with FP8 data format and variable bias in the DNN training phase. Our approach is generic and compatible with any compute core as we propose to integrate the online statistical unit in the DRAM memory controller.
- A novel approximate, low area, and low power online median calculation unit for FP8 bias identification that does not introduce added latency to the memory transactions.
- Through investigation of our methodology with various datasets and networks. We also present the detailed design of our statistical unit and present all the hardware results in addition to the accuracy results.

The rest of the paper is organized as follows: Sect. 2 presents the proposed new methodology for online adoption of FP8 data format for reduced data accesses energy during training. The hardware implementation is presented in Sect. 3. The experimental setup and all the associated results are presented in Sect. 4 and Sect. 5, respectively.Sect. 6 presents the state-of-the-art works. The paper is finally concluded in Sect. 7.

## 2 Methodology

State-of-the-art DNN training is computed using floating-point format. A standard floating-point number is represented using Eq. 1. The bias value is given by Eq. 2. The state-of-the-art format for DNN training is IEEE-754 FP32. Recently, DNN training with 16-bit (BFloat16) and 12-bit (TinyFloat12) formats [15, 16] also achieved similar accuracy as 32-bit. The current trend is to target 8-bit floating point number system for DNN training. There are two popular FP8 configurations i.e. (1,4,3) and (1,5,2), where (1,4,3) means 1-bit sign, 4-bit exponent and 3-bit mantissa [10, 11]. In [10, 11], (1,4,3) format is used for forwards pass of the training and (1,5,2) format for backward pass. Furthermore, [10] also showed that it is not feasible to use (1,4,3) format for the backward pass. In this work, our aim is to employ the same data format for forward and backward pass. Hence, we adopted (1,5,2) format for our FP8.

As mentioned in Sect. 1, this paper targets adaptively adjusting FP8 range by statistical analysis. As per Eq. 2, for a standard case, the bias value is 15 and the dynamic range of FP8 is $\pm 3.05 \times 10^{-5}$ to $\pm 1.13 \times 10^{5}$. This range is not sufficient for all the training tasks. Varying the bias is one way to adjust the FP8 data format range to the required range. The key question is how to calculate Bias without causing range errors, especially for adaptively adjusting the FP8 range using online statistical analysis, which we answer in this section.

$$FP\ Value = (-1^{Sign}) * 1.Mantissa * 2^{(Exponent - Bias)} \tag{1}$$

$$Bias = 2^{(ExponentSize-1)} - 1 \tag{2}$$

Our methodology is focused on DRAM memory, but the methodology is applicable to any level in the memory hierarchy. The details of our methodology are as follows. The data access from the DRAM in the initial 'e' epochs of training out of the total 'E' epochs, where e<<E, are performed in FP32. The FP32 data that are read and written to the DRAM are sampled by a statistical analysis unit to identify a suitable bias value by analyzing the data

distribution. After identification of a suitable bias value, DRAM write requests from the compute core are quantized to FP8. This enables a linear reduction of the total number of DRAM accesses since more data words can be packed in each transaction. Similarly, the read data are dequantized to FP32 in the memory controller before forwarding it to the core. The procedure for Quantization and Dequantization of FP32 number to/from FP8 is detailed in Algorithm 1. In this algorithm, we ignore subnormal and special numbers of FP32 and only consider normalized numbers. For more details, please refer to chapter 17 of [17].

**Algorithm 1** Quantization and Dequantization of FP32 number to/from FP8

```
procedure QUANTIZATION ()
    input:FP32_number, selected_FP8_bias
    output:FP8_number (1,5,2)
    FP32_sign = FP32_number[31]
    FP32_exponent = FP32_number[30:23]
    FP32_mantissa = FP32_number[22:0]
    FP32_bias = 127
    new_exponent = FP32_exponent - FP32_bias
    if − selected_FP8_bias <= new_exponent <= 32 − selected_FP8_bias then
        5bit_FP8_exponent = new_exponent + selected_FP8_bias
    else
        if new_exponent < −selected_FP8_bias then
            5bit_FP8_exponent = 0
        else
            5bit_FP8_exponent = 31
        end if
    end if
    2bit_FP8_mantissa = Round(FP32_mantissa)
    1bit_FP8_sign = FP32_sign
end procedure

procedure DEQUANTIZATION ()
    input:FP8_number, selected_FP8_bias
    output:FP32_number (1,8,23)
    FP8_sign = FP32_number[7]
    FP8_exponent = FP32_number[6:2]
    FP8_mantissa = FP32_number[1:0]
    FP8_bias = selected_FP8_bias
    FP32_bias = 127
    new_exponent = FP8_exponent - FP8_bias
    8bit_FP8_exponent = AppendZerosToLeft(new_exponent + FP32_bias)
    23bit_FP8_mantissa = AppendZerosToRight(FP8_mantissa)
    1bit_FP32_sign = FP8_sign
end procedure
```

In our methodology, we calculate four bias values, one for each type of training data: forward pass activations, backward pass error, weight gradients, and weights (refer Figure 2 of [10] for definitions). This is because the data range for each phase of training varies. Please note that in residual networks the residual connections have the same dynamic range as activations as a result we have utilized the same bias value for both. Fig. 1 shows the top-level architectural view of our approach. The online statistical analysis unit is integrated in the memory controller to enable our approach to be compatible with any DNN compute core without major modifications. The online statistical analysis unit includes three key blocks, i.e., quantizer, dequantizer, and bias estimator units. At a memory controller level, it is not

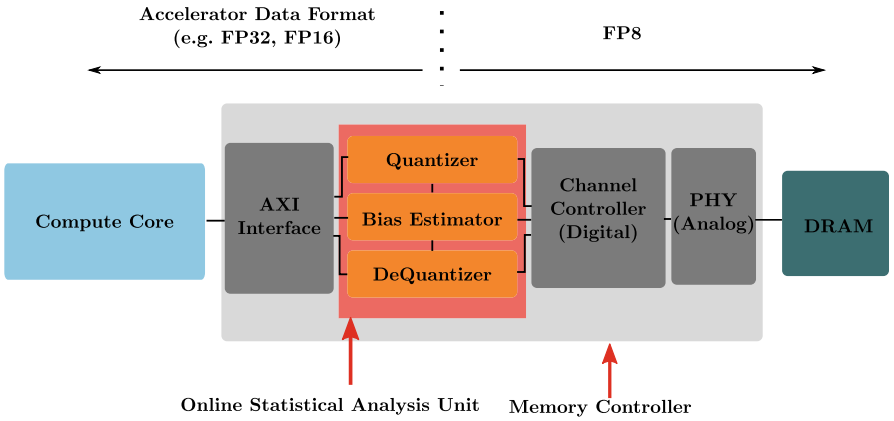



**Fig. 1** High-level architecture

possible to differentiate the type of data, (i.e. forward pass activations or backward pass errors or weights or gradient), which is required to select the appropriate bias value among the four. Hence, our approach requires the compute core to indicate the type of accessed data via the user signals of the AXI interface along with the data read/write requests. The user signals of AXI interface are also used to differentiate between the DNN data and other compute-related instructions. These compute-related instruction accesses are bypassed and retained in their original format. The additional information that is to be sent via AXI user interface can be extracted by the memory management unit with minimal modifications if each of the four types of data and instructions are separated based on address space.

## 2.1 Bias estimator

The online statistical analysis unit has to fulfill the following requirements (1) capability to compute the statistical results in the background, (2) streamline processing of the data without stalling or increasing the latency of the compute core or memory transactions, (3) avoiding any additional DRAM accesses for the statistical analysis, and (4) minimal area/energy overhead. The bias estimator is the most important block and should suffice the aforementioned requirements, while the quantizer and dequantizer blocks have low complexity once the bias value is known. One possiable metric can be the minimum (min) and maximum (max) values. Calculating the min-max value requires only two FP32 comparators and can be calculated in a streamline fashion. Mean is another low-complexity metric that is also suitable for streamline processing of the incoming data and can be implemented with a set of accumulators. However, neither mean, nor min–max are robust statistics since a single outlier can drastically impact the value of these metrics. For example, an extreme sample at $\pm 10^{-42}$ in a single epoch during bias calculation will result in wrong bias selection. Similarly, a long tail of data samples on the histogram would also shift the mean. Other complex statistics like standard deviation and probability distribution of data introduce the large area and energy overhead with long stalling of computation.

We used median value in our statistical analysis to maintain a balance between numerical robustness and implementation complexity. Median is known for its robustness against outliers [18] and it has a low complexity compared to standard deviation or probability distribution. Median represents the middle number within the given data samples after sorting

**Table 2** Reference table for mapping median value to Bias

| Reference median value | Bias | Reference median | Bias | Reference median | Bias |
|---|---|---|---|---|---|
| 64 | 10 | 2 | 15 | 0.0625 | 20 |
| 32 | 11 | 1 | 16 | – | – |
| 16 | 12 | 0.5 | 17 | – | – |
| 8 | 13 | 0.25 | 18 | – | – |
| 4 | 14 | 0.125 | 19 | 5.960E-8 | 40 |

(refer Sect. 3 for implementation details). Our approach calculates the median of DNN data which is in FP32 format. Afterward, we find the nearest reference median value from Table 2 to which the computed median value matches. The reference median values of the FP8 format listed in Table 2 is the middle number of the FP8 data range for a given bias value which is calculated using use Eq. 3. The selected bias as a function of the median is then used for quantizing memory access to FP8.

$$Reference\ Median\ Value = 1.0 * 2^{(16-Bias)} \tag{3}$$

Please note, this reference table is stored as a look-up-table in the bias estimator hardware unit. Figure 2 shows an example of adjusting FP8 range based on the median of DNN data. The histogram of backward pass error data for epoch 2 of MobileNet-V2 network trained with TinyImagenet dataset is shown in the top graph of this figure. The median of this histogram is marked as a red dashed line, which is at value $4.18 * 10^{-5}$. This median value is used to find the nearest median value within the reference table (i.e. Table 2), thereby selecting the bias value as 31. As shown in the middle graph of Fig. 2, the data range of the FP8 format with bias 31 fits most of the data samples of the distribution with an exception of a few extreme samples that are less than $\pm 10^{-40}$ and are almost considered as zero by the network. We will show in the results section (i.e. Sect. 5) that the impact of this on the DNN training accuracy of various networks and datasets is negligible. Finally, we also present the data range of FP8 data format with the standard bias value 15 in the bottom graph of Fig. 2, which results in a large data range error.

### 2.2 Median variation and 'e' identification

As already stated, our methodology uses the initial 'e' epochs with FP32 data transaction for bias value identification, which is used for the data quantization in the remaining epochs. Hence, it is important to analyze the variation of the median value across the epochs, as the bias is calculated using the median. Figure 3 shows the median variation across the epochs of MobileNet-V2 and GoogleNet. Respective bias boundaries computed using Table 2 are also presented. As we can observe, the median variation is nominal and the maximum change in the bias value is two. For example, the required bias value in epoch 1 is 24 and in epoch 28 is 26 for gradient data of MobileNet training. This error of 2 in the bias value has a low impact on the range of FP8 where the data range is from $\pm 5.9 \times 10^{-8}$ to $\pm 2.24 \times 10^{2}$ for bias 24 and from $\pm 1.49 \times 10^{-8}$ to $\pm 5.6 \times 10^{1}$ for a bias value of 26. Due to the high overlapping of range, the impact on DNN accuracy is minimal (refer Sect. 5). We validated the nominal variation of median for other networks and other datasets as well, whose results are shown in Table 3. A similar trend is also observed in the histograms presented in Figure 3 of [19]

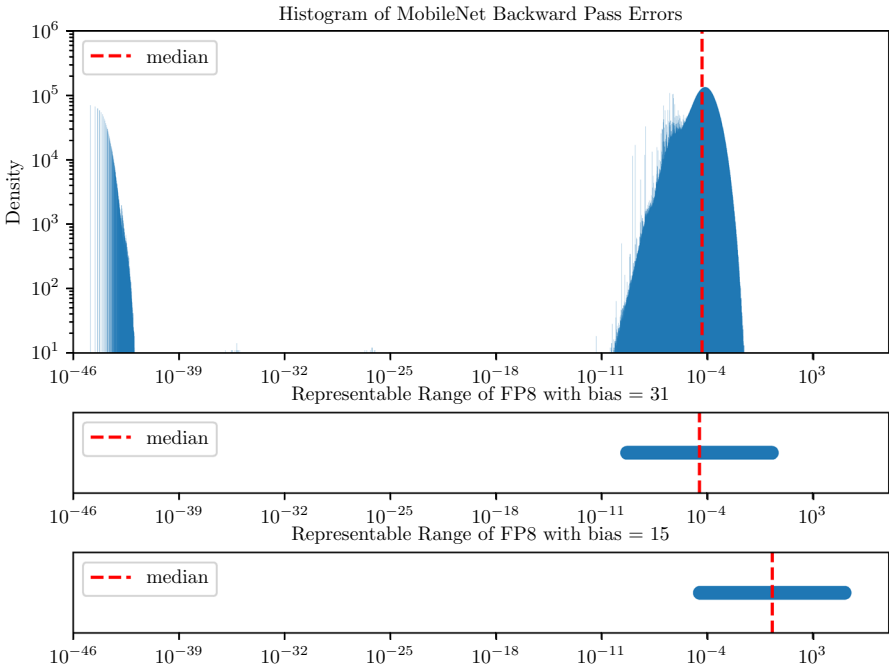
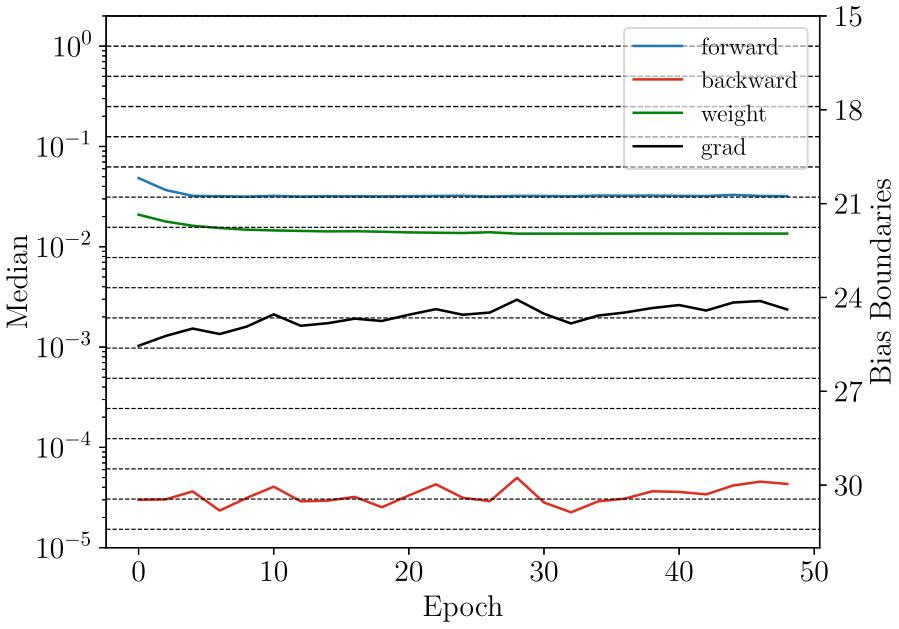



**Fig. 2** Fitting the FP8 data format range to the MobileNet-V2 backward pass errors histogram. The data points shown in this histogram are excluding the sign

and Figure 2 of [20], which show the distribution of DNN parameters to be consistent across the epochs and thereby the resulting median has nominal variation. Furthermore, we also validated if a similar trend is observed when the weight initialization is set to all zeros, refer Table 3 Zero initialization. In all the cases, the median and resulting bias variation are very minimal. Based on these experimental results, the 'e' in our methodology can be set to two, i.e. the DRAM transactions in the first two epochs are in FP32 to identify the bias value, which is used for the rest of the epochs to quantize and dequantize the DRAM transactions to and from FP8 format. However, we will show in Sect. 3 that our approximate median calculator will require an additional 2 epochs, making 'e' a total of 4.

The bias estimator that is integrated in the memory controller has low median variation only if all the data are passed through the DRAM memory controller at least once. In a typical DNN training compute platform for large networks, most of the forward pass activation and backward pass error data are stored in DRAM due to batch normalization and their requirement for later stage computations like gradient calculation. Furthermore, the weights and the gradients are also stored in the DRAM due to their large parameter size. We also evaluated the median calculation with a limited number of samples to verify the case where the whole data samples of forward, backward, weights, or gradient are not stored. We calculated the median with 30% of the total samples, and the median variation resulted in a maximum bias change of one.



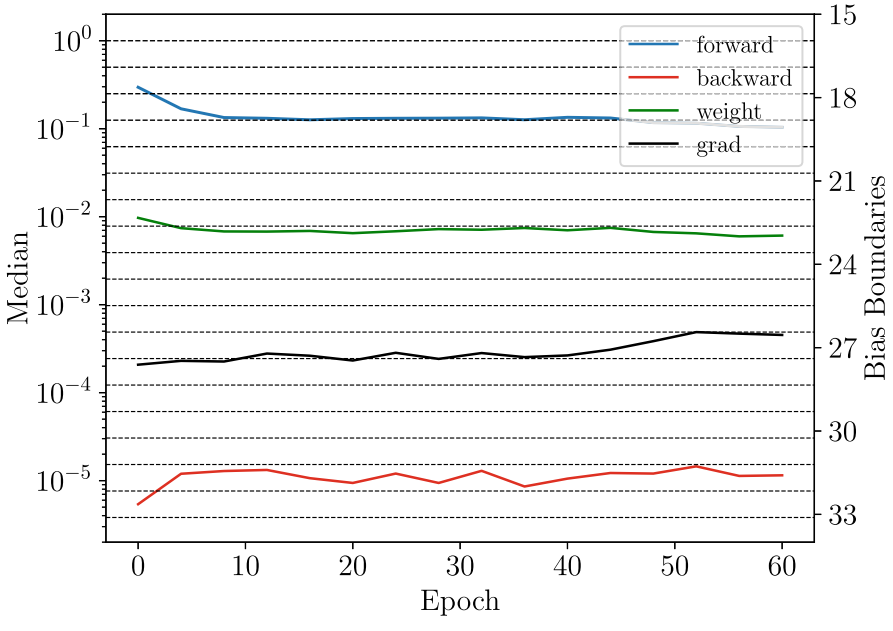




**Fig. 3** Median variation across epochs for training data of **a** MobileNet-V2 with TinyImagenet dataset and **b** GoogleNet with Cifar100 dataset. The graph also shows the nearest bias value for the given median

**Table 3** Median/Bias variation vs epoch for various weights initialization strategies, networks and datasets

| Model | Type | Glorot initialization [21] | | Zero initialization | |
|---|---|---|---|---|---|
| | | 2nd Epoch | 50th Epoch | 2nd Epoch | 50th Epoch |
| VGG16 Cifar100 | F | 0.31(18) | 0.252(18) | 0.291(18) | 0.248(18) |
| | B | 0.701E-06(31) | 1.10E-06(31) | 0.521E-06(31) | 1.43E-06(31) |
| | W | 0.401E-02(24) | 0.0871E-02(26) | 0.42E-02(24) | 0.26E-02(25) |
| | G | 0.20E-03(28) | 0.32E-03(28) | 0.355E-03(28) | 0.381E-03(27) |
| ResNet18 Cifar100 | F | 0.556(17) | 0.404(17) | 0.71(17) | 0.40(17) |
| | B | 1.23E-05(31) | 0.91E-05(31) | 0.98E-05(31) | 1.08E-05(31) |
| | W | 0.46E-02(24) | 0.39E-02(24) | 0.62E-02(23) | 0.501E-02(24) |
| | G | 0.63E-03(27) | 0.91E-03(26) | 0.71E-03(26) | 0.11E-02(25) |
| ResNet18 TinyImageNet | F | 0.821(16) | 0.409(17) | 0.61(17) | 0.36(17) |
| | B | 7.09E-06(31) | 9.16E-06(31) | 3.01E-06(31) | 7.42E-06(31) |
| | W | 0.0162(22) | 0.0204(22) | 0.0158(22) | 0.0190(22) |
| | G | 0.601E-03(27) | 0.904E-03(26) | 0.504E-03(27) | 0.606E-03(27) |

Representation = Median(Bias)
$F$ = Forward pass activation, $B$ Backward pass error, $W$ = Weights, $G$ = Gradients

**Table 4** Median/Bias variation for different networks and TinyImage dataset

| | Type | Real data samples Actual | Proposed approximate | Real samples with synthetic outliers Proposed approximate |
|---|---|---|---|---|
| MobileNet-V2 TinyImageNet | F | 0.053(20) | 0.31(18) | 0.32(18) |
| | B | 3.18E-05(31) | 9.009E-05(31) | 4.0E-05(31) |
| | W | 0.018(22) | 0.041(21) | 0.0409(21) |
| | G | 0.11E-02(26) | 0.84E-03(26) | 0.41E-02(24) |
| ResNet18 Cifar100 | F | 0.594(17) | 0.34(18) | 0.350(18) |
| | B | 0.13E-05(31) | 0.62E-05(31) | 0.642E-05(31) |
| | W | 0.802E-02(23) | 0.402E-02(24) | 0.49E-02(24) |
| | G | 0.53E-03(27) | 0.58E-03(27) | 0.400E-03(27) |

Actual median @ epoch 4 versus approximate median with epoch 1–4 data
Representation = Median(Bias)

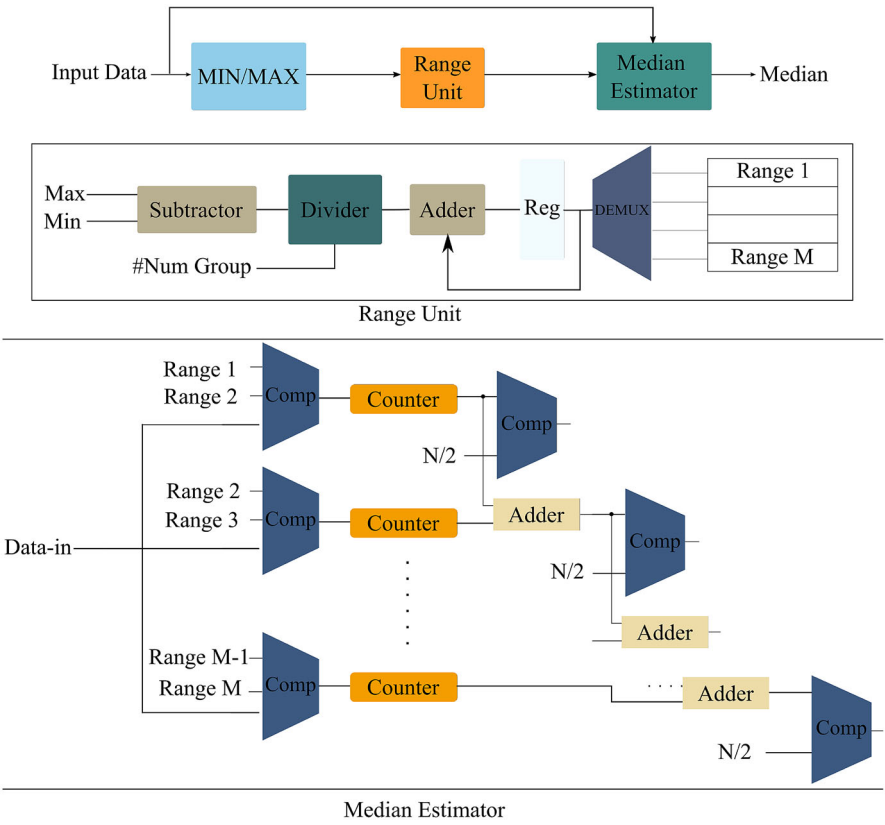



**Fig. 4** Hardware architecture for median calculation

## 3 Hardware implementation

In this section, we present our novel hardware architecture for online median calculation, which is the key function of the bias estimator. The architecture processes the incoming data in a streamlined fashion, without stalling the compute core or requiring any additional DRAM accesses. The proposed hardware architecture is shown in Fig. 4. It requires four iterations to compute the median, where each iteration is equal to an epoch in our approach. This approximate median calculation is based on the premise that the median variation between the epochs is minimal (as shown in Fig. 3). In the first epoch, the Min/Max block compares incoming FP32 data with the previous value to identify the minimum and maximum (min-max) values. At the end of the first epoch, the min-max values are used to identify the data range and divide the range into 'M' bins by the range unit. In the second epoch, each incoming FP32 data is compared against the bin ranges and the bin counters are incremented if the data is within its range. At the end of the second epoch, the bin counter values are added sequentially from lowest to highest to identify the bin at which the 'N/2' crossing over occurs (referred to as half-bin henceforth), where 'N' is the total number of samples in that epoch. The median unit uses a separate counter for counting 'N' in the given epoch. The bin at which this cross-over occurs is the bin with the median value, and the average of its range is approximately equal to the median. For increased accuracy of the median value, the second

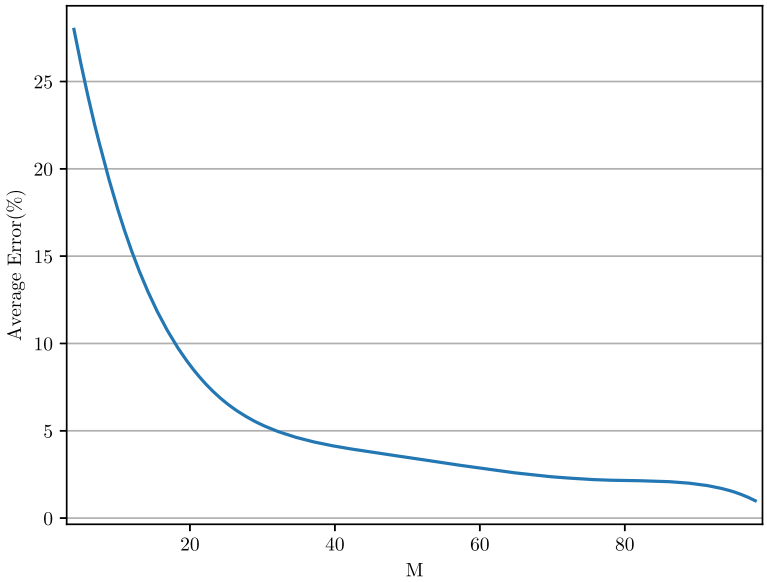



**Fig. 5** Error of proposed approximate median unit in comparison to the accurate median value for various number of bins (M)

epoch's half-bin range is further divided into 'M' bins by range unit. In the third epoch, the bin ranges are configured to the new values and the bin counting is repeated to fine-tune the median value. Our methodology employs four epochs in total to identify the median with low error.

The optimum value of M is obtained based on error analysis of the proposed approximate median hardware. Figure 5 shows the error of the approximate median hardware unit in comparison to the accurate median for differed values of M. As shown, the error is reduced by utilizing more number of bins. In this work, we aim to have the error value less than 10%, as it will result in the same bias value as accurate median. Therefore, we set the value of M to 20 which has an error value less than 10%. Higher values of M shows will improve the accuracy of the median but at the cost of a higher hardware footprint. All calculations related to median value in the memory controller are performed in the background and adds a maximum of 40 cycles delay (i.e. 40 ns for 1 GHz clock frequency) for range division and half-bin identification at the end of an epoch. The time period of one epoch computed on a standard DNN accelerator of GPU varies from seconds to minutes depending on the network, dataset, and computation speed. This additional latency per epoch is very minimal when compared to the 360 ns stall introduced by the memory controller during each DRAM refresh operation that occurs at an interval of 7.8 $\mu$s. As already mentioned, the bias estimator calculates four bias values, one for each type of training data i.e. forward pass activation, backward pass error, weight gradient, and weights. Bias estimator has individual min-max block, range register, and bin counter registers for each training phase. The incoming FP32 data is separated and accordingly processed based on the user signal of AXI interface bits that indicate the type of data.

Table 4 compares the actual median vs the approximate median at the end of the 4th epoch. The resulting bias variation due to the approximate median is a maximum of two. The 4th epoch approximate median in comparison to the 50th epoch's actual median (refer Table 3)

also resulted in a bias variation of maximum two, which results in a low accuracy loss (refer Fig. 5). Additionally, we synthetically introduced extreme data samples in the first epoch (i.e. $10^{-40}$ and $10^{10}$) to adversely impact the min-max value and compare the median error after four epochs. The results shows that even with extreme outliers, bias variation maximum is two.

## 4 Experimental setup

For evaluating the accuracy results, we use PyTorch and QPyTorch [22] libraries. We modified the QPyTorch library to support Floating-Point data format with variable bias. After the bias estimation epochs, the modified library is used to quantize the DNN data to FP8 based on the calculated bias. The median calculation is performed using a simulation model of the proposed architecture. The hardware implementation of all the blocks of the online statistical analysis unit i.e., bias estimator, quantizer, and deqauntizer, were designed in Verilog and Synthesized in Synopsis' Design Compiler using Global Foundry, FD-SOI, 22 nm technology. In order to calculate the DRAM energy and latency reduction of the FP8 data format, we employ the following open-source tools. Firsts, SCALE-Sim [23], an open-source systolic array cycle-accurate simulator, generates the data access requests to the DRAM memory controller for various types of training data. SCALE-Sim's topology configuration is configured as per the training phase (i.e. forward pass or backpropagation) to obtain data access requests of each type. Second, DRAMSys [24] and DRAMPower [25], a DRAM subsystem (memory controller + DRAM) exploration framework, to obtain the energy and latency of DRAM accesses. Our evaluations are performed for DDR3 and DDR4 DRAM memories.

## 5 Results

This section presents DNN accuracy results, area and power results of hardware units, and DRAM data access energy.

### 5.1 Accuracy

This section presents DNN training accuracy results of our methodology and compares it against other state-of-the-art data formats such as IEEE-754 FP32, FP16, and BFloat16 [15]. Table 5 summarizes the accuracy results. Our evaluations are conducted on a wide range of well-known DNN models such as LSTMs, Transformer, VGG-16, ResNet18, ResNet101, MobileNet-V2, DenseNet, and GoogleNet. Furthermore, our evaluations consider six datasets (i.e. Multi30K for English to German translation, IMDB review, PennTreeBank, Cifar-10, Cifar-100, and TinyImageNet) that cover image classification and natural language processing applications. We consider a wide range of training epochs between 15 and 250 to demonstrate the feasibility of our methodology for a wide range of cases. For all the networks our methodology resulted in minimal DNN training accuracy loss. For the image classification applications, the resulting accuracy reduction of the FP8 format is on average 1% lower compared to the reference FP32 format. The same is the case for LSTMs with the IMDB review dataset. The accuracy of the LSTMs for PennTreeBank dataset is measured in perplexity (PPL) score (lower value is better), and Transformer-base [26] for Multi30K dataset is measured in BLEU score (higher value is better). In both the cases the accuracy of

FP8 is almost close to FP32. For example, a BLEU score of 30-40 is considered "understandable to good translations", which the FP8, BFloat16, and FP32 data format achieve for the Transformer-base [26] model considered in our experiments. We were unable to compare the accuracy of our methodology vs NVIDIA's FP8 methodology due to the non-availability of the methodologies in the public platform. In the future, we will conduct our experiments on the H100 board to obtain accuracy results and comparison to our methodology. It is important to note that our approach has an edge over NVIDIA's approach as it is compatible with any compute core by integrating the statistical unit in the memory controller.

### 5.2 Online statistical analysis unit area and power

Table 6 shows the hardware implementation results of varies the blocks of our online statistical analysis unit. The total area is 0.019 $mm^2$ in 22 nm FD-SOI technology. The area overhead of this unit in comparison to the memory controller digital logic area is 6.3%. The area of memory controller digital logic is 0.890 $mm^2$ in 65 nm technology [27] that is linearly scaled to 22 nm technology (0.301 $mm^2$). All the blocks are designed to operate at 1 GHz frequency to meet the compute core data access request frequency without adding additional stalls. The bias estimator consumes the highest power of 10.8 mW among the three blocks. However, this unit is only active during initial epochs(e.g. 4 epochs) and for the rest of training it is deactivated via clock gating. Hence, the average power of the bias estimator is 0.864 mW for 50 epochs, which is very low in comparison to the typical power consumption of a DDR4 DRAM device (i.e. 1–2W [28])

### 5.3 DRAM access analysis

Table 7 presents the DRAM energy consumption and elapsed time for the following number of requests i.e. Forward = 13925992, Backward = 13862080, and Gradient = 13926181, in different data widths. The data requests are generated from SCALE-Sim tool for one single input image in ResNet18. The systolic array and the on-chip memory configurations of SCALE-Sim are set as per the two well-known hardware accelerators, i.e., Google TPUv4 [5] and Qualcomm Cloud Al 100 [29]. The energy consumption of DRAM is first-order linearly reduced with data width. For instance, in TPU-like accelerator with DDR4 DRAM the forward pass energy is reduced from 5.35 to 1.53 mJ. Our methodology with four epochs in FP32 and 46 epochs in FP8 reduces the total DRAM energy by 3.07× compared to conventional FP32 data format training for 50 epochs. In the future, we will extend our experiments to other DRAM types like LPDDR4, LPDD5, DDR5, HBM2, and GDDR5.

## 6 Related works

The success of using low bit-width data format for inference of DNN [30–32], has increased interest in utilizing low-precision data format for training. 16-bit data formats have gained popularity as an energy-efficient alternative to FP32 for DNN training. IEEE FP16 data format is one of the common 16-bit formats for training, which allocates 5-bits for the exponent and 10-bits for the mantissa. However, using FP16 for DNN training poses a challenge as the reduced dynamic range as shown in table 1 due to the 5-bit exponent size can lead to accuracy degradation. This is especially true for the gradients values in the backward pass of DNN training which often requires a larger range. To overcome this challenge, NVIDIA proposed

**Table 5** DNN training accuracy results comparison

| Application | DataSet | DNN Model | Accuracy Measured in | Number of Training Epochs | FP32 | FP16 | BFloat16 | FP8 (This Work) |
|---|---|---|---|---|---|---|---|---|
| Natural Language Processing | Multi30k | Transformer-base | BLEU | 250 | 33.2 | 29.6 | 32.8 | 30.8 |
| | IMDB review | LSTM | % | 15 | 88.20 | 85.10 | 88 | 87.31 |
| | PennTreeBank | LSTM | PPL | 35 | 104.4 | 107.7 | 104.6 | 109.2 |
| Image Classification | Cifar100 | DenseNet | % | 65 | 80.20 | 32 | 80 | 79.10 |
| | | ResNet18 | % | 65 | 71.60 | 41 | 71.50 | 71 |
| | | ResNet101 | % | 65 | 79.78 | 68.10 | 79.70 | 78.80 |
| | | VGG16 | % | 65 | 68.60 | 24 | 67.70 | 67 |
| | | GoogleNet | % | 65 | 78.10 | 75 | 78 | 77.03 |
| | Cifar10 | ResNet18 | % | 100 | 93.02 | 91.80 | 93 | 93 |
| | | VGG16 | % | 100 | 93.64 | 92.10 | 93.60 | 93.25 |
| | | GoogleNet | % | 100 | 95 | 91 | 95 | 94.60 |
| | | ResNet101 | % | 100 | 95.50 | 89.40 | 95.50 | 94.80 |
| | TinyImageNet | MobileNet-V2 | % | 50 | 52.30 | 37.20 | 52 | 51.10 |
| | | VGG16 | % | 50 | 52.10 | 45.40 | 52 | 52 |
| | | ResNet18 | % | 50 | 45 | 35 | 45 | 44.60 |
| | | GoogleNet | % | 50 | 50.70 | 12 | 50.40 | 50 |

**Table 6** Hardware implementation results of online statistical analysis unit (post-synthesis)

| Block | OP | Area | Freq | Power |
|---|---|---|---|---|
| Quantizer | FP32toFP8 | 66 $\mu$m2 | 1 GHz | 0.0033 mW |
| DeQuantizer | FP8toFP32 | 31 $\mu$m2 | 1 GHz | 0.003 mW |
| Bias Estimator | Median Calc | 18996 $\mu$m2 | 1 GHz | 10.8 mW |

multiplying the gradient values to a scaling factor to shift the values into a dynamic range of FP16. This approach achieves the same accuracy as the baseline FP32. Another well-known data format for DNN training is BFloat16, which has an 8-bit exponent and a 7-bit mantissa part. Since the BFloat exponent size is the same as that of FP32, all DNN training phases(i.e. forward, backward,...) can be performed using BFloat16 without the need for scaling since the dynamic range is almost covers similar range to FP32, as shown in Table 1. Recently, researchers have attempted to train DNNs with sub 16-bit data formats. The authors of [16] proposed a 12-bit data format called TinyFloat, with 7-bit exponent and 4-bit mantissa. This data format achieves almost same accuracy as FP32 since the number of exponent bits is close to the FP32 format. However, training DNNs with less than 12-bit width, such as 8-bit, poses a significant challenge as it is not possible to provide sufficient range.

In 2019, IBM [10, 14] proposed the first hybrid FP8 based DNN training, which relied on scaling factors to fit backward pass data within the dynamic range of FP8. Additionally, they allocated different exponent sizes for forward and backward passes, resulting in different computation units. NVIDIA, ARM, and Intel published a white paper [11] on employing FP8 for a variety of DNN training tasks. Similar to IBM, their work also relied on the scaling factor. Finding the scaling factor in [11] is based on offline heuristics experiments with model parameters and intermediate layer data output. In [12] an FP8 format with variable bias is presented to adjust the data format range to the application desired range. However, the procedure for finding bias is offline, similar to other previous publications. NVIDIA recently introduced Hopper 100 GPU [1] which supports FP8 data format with online identification of scaling factor capability. NVIDIAs GPU by performing statistical analysis on each tensor and the history of that tensor in the previous iteration detect the scaling factors. NVIDIA does not reveal the details such as the hardware design of the statistical analysis unit which limits the researchers utilizing the FP8 data type with online scaling factor detection to their DNN training accelerators. NVIDIA's approach is only compatible with their own custom FP8 format and can not incorporate into other accelerators with data formats such as FP32 or FP16. None of the previous works have presented an online methodology so far to detect the bias value for variable bias floating point format.

## 7 Conclusion

This paper presents a novel 8-bit quantization methodology to reduce DRAM energy consumption of DNN training. FP8 can not provide sufficient range for DNN training. By varying the bias value of FP8 format, the range can be shifted to the application-required range. We proposed an online statistical analysis approach to detect the suitable bias value for a given DNN model without stalling training. The statistical metric employed in this work is median due to its robustness towards outliers. notation Our experiments showed that the median value is consistent across the epochs. This enabled the identification of the bias value within the first

**Table 7** DRAM energy results for processing data requests from different training tasks of ResNet18

| SCALE-Sim Config | Data Width | Type | DDR3-DIMM DQ = 64, DataRate=1866 | | DDR4-DIMM DQ = 64, DataRate=2133 | |
|---|---|---|---|---|---|---|
| | | | Time(ms) | Energy(mJ) | Time(ms) | Energy(mJ) |
| TPU-like #MACs:128×128 on-chip:36MB | 8-bit | F | 1.16 | 2.14 | 0.68 | 1.53 |
| | | B | 1.18 | 2.17 | 0.69 | 1.58 |
| | | G | 1.03 | 1.95 | 0.63 | 1.23 |
| | 32-bit | F | 4.13 | 7.32 | 2.49 | 5.35 |
| | | B | 4.13 | 7.35 | 2.51 | 5.47 |
| | | G | 4.44 | 8.96 | 2.83 | 5.38 |
| Qualcomm-like #MACs:64×64 on-chip:9MB | 8-bit | F | 1.01 | 1.93 | 0.6 | 1.44 |
| | | B | 1 | 1.94 | 0.61 | 1.47 |
| | | G | 1.07 | 2.02 | 0.63 | 1.31 |
| | 32-bit | F | 4.16 | 5.13 | 2.39 | 7.11 |
| | | B | 4.15 | 5.21 | 2.4 | 7.09 |
| | | G | 4.35 | 4.77 | 2.68 | 8.49 |

The notation F, B, and G contain all the required data for computing each respective phase; for instance, F includes both activations and weights in forward pass

four epochs using FP32 data transactions, while the remaining 46 epochs' data transactions are quantized to FP8. The paper also presented a new hardware-efficient approximate median calculation unit that has low area, power, and latency overhead. In order to be compatible with any DNN compute core without any major architectural modifications, the new quantization unit is integrated in the memory controller. Overall, our methodology reduces the total DRAM energy by 3.07× in comparison to the conventional FP32 transactions, while having a minor accuracy degradation of 1%.

**Funding** Open Access funding enabled and organized by Projekt DEAL.

**Data availibility** The data which are used in this work are openly available in the following URLs. https://www.cs.toronto.edu/\protect\unhbox\voidb@x\penalty\@M\kriz/cifar.html https://www.kaggle.com/competitions/tiny-imagenet/data https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews https://github.com/multi30k/dataset https://www.kaggle.com/datasets/nltkdata/penn-tree-bank



## References

1. Choquette J (2022) Nvidia hopper gpu: scaling performance. In: 2022 IEEE Hot Chips 34 Symposium (HCS), IEEE Computer Society, (pp. 1–46)
2. Patterson D, Gonzalez J, Le Q, Liang C, Munguia L-M, Rothchild D, So D, Texier M, Dean J (2021) Carbon emissions and large neural network training. arXiv preprint arXiv:2104.10350
3. He X, Liu J, Xie Z, Chen H, Chen G, Zhang W, Li D (2021) Enabling energy-efficient DNN training on hybrid GPU-FPGA accelerators. In: Proceedings of the ACM International Conference on Supercomputing, (pp. 227–241)
4. You J, Chung J-W, Chowdhury M (2022) Zeus: Understanding and Optimizing GPU Energy Consumption of DNN Training. arXiv preprint arXiv:2208.06102
5. Jouppi NP, Yoon DH, Ashcraft M, Gottscho M, Jablin TB, Kurian G, Laudon J, Li S, Ma P, Ma X(2021) Ten lessons from three generations shaped Google's TPUv4i: Industrial product. In: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), (pp. 1–14). IEEE
6. Chen T, Du Z, Sun N, Wang J, Wu C, Chen Y, Temam O (2014) Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. ACM SIGARCH Comput Archit News 42(1):269–284
7. Zhang S, Du Z, Zhang L, Lan H, Liu S, Li L, Guo Q, Chen T, Chen Y (2016) Cambricon-X: an accelerator for sparse neural networks. In: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), (pp. 1–12). IEEE
8. Stathis D, Sudarshan C, Yang Y, Jung M, Weis C, Hemani A, Lansner A, Wehn N (2020) eBrainII: a 3 kW realtime custom 3D DRAM integrated ASIC implementation of a biologically plausible model of a human scale cortex. J Signal Process Syst 92(11):1323–1343
9. Kim D, Kung J, Chai S, Yalamanchili S, Mukhopadhyay S (2016) Neurocube: a programmable digital neuromorphic architecture with high-density 3D memory. ACM SIGARCH Comput Archit News 44(3):380–392
10. Sun X, Choi J, Chen C-Y, Wang N, Venkataramani S, Srinivasan VV, Cui X, Zhang W, Gopalakrishnan K (2019) Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks. In: Advances in Neural Information Processing Systems **32**
11. Micikevicius P, Stosic D, Burgess N, Cornea M, Dubey P, Grisenthwaite R, Ha S, Heinecke A, Judd P, Kamalu J, et al (2022) FP8 Formats for Deep Learning. arXiv preprint arXiv:2209.05433
12. Sudarshan C, Sadi MH, Steiner L, Weis C, Wehn N (2022) A Critical Assessment of DRAM-PIM Architectures-Trends, Challenges and Solutions. In: International Conference on Embedded Computer Systems, (pp. 362–379). Springer

13. Park J, Lee S, Jeon D (2021) A neural network training processor with 8-bit shared exponent bias floating point and multiple-way fused multiply-add trees. IEEE J Solid-State Circuits 57(3):965–977
14. Lee SK, Agrawal A, Silberman J, Ziegler M, Kang M, Venkataramani S, Cao N, Fleischer B, Guillorn M, Cohen M (2021) A 7-nm four-core mixed-precision AI chip with 26.2-tflops hybrid-fp8 training, 10.49-tops int4 inference, and workload-aware throttling. IEEE J Solid-State Circuits 57(1):182–197
15. Kalamkar D, Mudigere D, Mellempudi N, Das D, Banerjee K, Avancha S, Vooturi DT, Jammalamadaka N, Huang J, Yuen H, et al (2019) A study of BFLOAT16 for deep learning training. arXiv preprint arXiv:1905.12322
16. Sudarshan C, Sadi MH, Weis C, Wehn N (2022) Optimization of DRAM based PIM architecture for energy-efficient deep neural network training. In: 2022 IEEE International Symposium on Circuits and Systems (ISCAS), (pp. 1472–1476). IEEE
17. Behrooz P (2000) Computer arithmetic: algorithms and hardware designs. Oxford Univ Press 19:512583–512585
18. Rousseeuw PJ, Hubert M (2018) Anomaly detection by robust statistics. Wiley Interdiscip Rev Data Min Knowl Discov 8(2):1236
19. Liu L, Jiang H, He P, Chen W, Liu X, Gao J, Han J (2019) On the variance of the adaptive learning rate and beyond. arXiv preprint arXiv:1908.03265
20. Wen W, Xu C, Yan F, Wu C, Wang Y, Chen Y, Li H (2017) Terngrad: Ternary gradients to reduce communication in distributed deep learning. In: Advances in Neural Information Processing Systems **30**
21. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings, (pp. 249–256)
22. Zhang T, Lin Z, Yang G, Sa CD (2019) QPyTorch: a low-precision arithmetic simulation framework
23. Samajdar A, Joseph JM, Zhu Y, Whatmough P, Mattina M, Krishna T (2020) A systematic methodology for characterizing scalability of dnn accelerators using scale-sim. In: 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, (pp. 58–68)
24. Steiner L, Jung M, Prado FS, Bykov K, Wehn N (2020) DRAMSys4.0: a fast and cycle-accurate systemC/TLM-based DRAM simulator. In: International Conference on Embedded Computer Systems. Springer, (pp. 110–126)
25. Chandrasekar K, Weis C, Li Y, Goossens S, Jung M, Naji O, Akesson B, Wehn N, Goossens K. Drampower: open-source dram power and energy estimation tool
26. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. In: Advances in Neural Information Processing Systems **30**
27. Sudarshan C, Lappas J, Weis C, Mathew DM, Jung M, Wehn N (2019) A lean, low power, low latency DRAM memory controller for transprecision computing. In: International Conference on Embedded Computer Systems. Springer, (pp. 429–441)
28. tomsHardware: measuring DDR4 power consumption. Accessed (2014). https://www.tomshardware.com/reviews/intel-core-i7-5960x-haswell-e-cpu,3918-13.html
29. Chatha K (2021) Qualcomm® Cloud Al 100: 12TOPS/W scalable, high performance and low latency deep learning inference accelerator. In: 2021 IEEE Hot Chips 33 Symposium (HCS). IEEE, (pp. 1–19)
30. Keller B, Venkatesan R, Dai S, Tell SG, Zimmer B, Dally WJ, Gray CT, Khailany B (2022) A 17–95.6 TOPS/W deep learning inference accelerator with per-vector scaled 4-bit quantization for transformers in 5nm. In: 2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits). IEEE, (pp. 16–17)
31. Keller B, Venkatesan R, Dai S, Tell SG, Zimmer B, Sakr C, Dally WJ, Gray CT, Khailany B (2023) A 95.6-tops/w deep learning inference accelerator with per-vector scaled 4-bit quantization in 5 nm. IEEE J Solid-State Circuits 58(4):1129–1141
32. Sadi MH, Mahani A (2021) Accelerating deep convolutional neural network base on stochastic computing. Integration 76:113–121