PL-NPU: An Energy-Efficient Edge-Device DNN Training Processor With Posit-Based Logarithm-Domain Computing

Yang Wang, Dazheng Deng, Leibo Liu, *Senior Member, IEEE*, Shaojun Wei, *Fellow, IEEE*, and Shouyi Yin, *Member, IEEE*

***Abstract*— Edge device deep neural network (DNN) training is practical to improve model adaptivity for unfamiliar datasets while avoiding privacy disclosure and huge communication cost. Nevertheless, apart from feed-forward (FF) as inference, DNN training still requires back-propagation (BP) and weight gradient (WG), introducing power-consuming floating-point computing requirements, hardware underutilization, and energy bottleneck from excessive memory access. This paper proposes a DNN training processor named PL-NPU to solve the above challenges with three innovations. First, a posit-based logarithm-domain processing element (PE) adapts to various training data requirements with a low bit-width format and reduces energy by transferring complicated arithmetics into simple logarithm domain operation. Second, a reconfigurable inter-intra-channel-reuse dataflow dynamically adjusts the PE mapping with a regrouping omega network to improve the operands reuse for higher hardware utilization. Third, a pointed-stake-shaped codec unit adaptively compresses small values to variable-length data format while compressing large values to fixed-length 8b posit format, reducing the memory access for breaking the training energy bottleneck. Simulated with 28nm CMOS technology, the proposed PL-NPU achieves a maximum frequency of 1040MHz with 343mW and 5.28mm^2. The peak energy efficiency is 3.87TFLOPS/W for 0.6V at 60MHz. Compared with the state-of-the-art training processor, PL-NPU reaches 3.75× higher energy efficiency and offers 1.68× speedup when training ResNet18.**

***Index Terms*— DNN training processor, edge-devices, reconfigurable dataflow, posit, logarithm-domain computing.**

I. Introduction

DEEP neural networks (DNN) have achieved remarkable successes on many recognition tasks [1], such as image segmentation [2], object detection [3], significantly promoting the development of artificial intelligence. Despite the outstanding performance of DNNs, off-the-shelf DNN models pre-trained with a public dataset suffer from severe accuracy degradation for customized applications since the user-specific dataset from a variety of environments would have distinct features with the training used public dataset [4], as shown in Fig. 1. Retraining the DNN models on cloud servers with user-specific data can recover the performance degradation. However, it consumes enormous data transmission energy and has a risk of privacy disclosure. Edge-device training is a practical solution to these issues [4]–[9], dynamically adapting DNN models with local datasets. Nevertheless, apart from feed-forward (FF) as inference, DNN training still requires back-propagation (BP) and weight gradient (WG), which have different properties as FF, introducing three challenges for energy-efficient computing on edge devices.

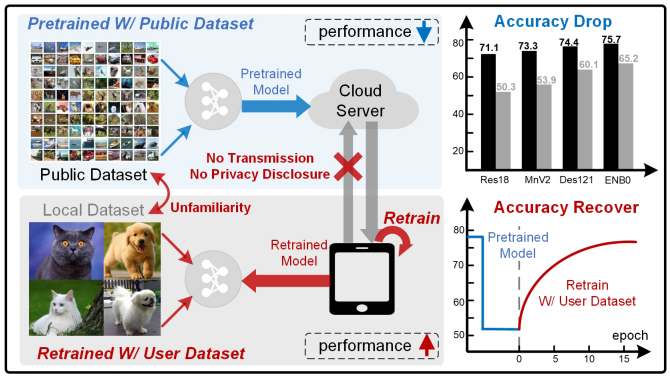


Fig. 1. The pretrained model suffers from accuracy degradation due to the unfamiliarity between domain-specific local dataset and public dataset.

First, compared with FF, the error maps (Emap) and the gradients in BP and WG need a wider range [10] and a higher precision [11]. It forces high bit-width floating-point (FP) computing units, which incurs large multiply-and-accumulate (MAC) energy. Recent low precision training works [11]–[16] tried to reduce the bit-width into 8-bit for MAC. However, they suffer from both precision loss for dynamic training data distribution due to fixed exponent size, and high power consumption with the large FP unit to mitigate the accuracy degradation for low-precision FP format. Second, unlike FF and BP, which only have single-dimension large intra-channel reuse, where one weight channel slides on only one feature map (Fmap)

Manuscript received 27 February 2022; revised 24 May 2022; accepted 14 June 2022. Date of publication 22 June 2022; date of current version 29 September 2022. This work was supported in part by NSFC under Grant 62125403, Grant U19B2041, and Grant 92164301; in part by the National Key Research and Development Program under Grant 2018YFB2202600; in part by the Beijing Science and Technology Project under Grant Z191100007519016; and in part by the Beijing Advanced Innovation Center. This article was recommended by Associate Editor H. Zhang. *(Corresponding author: Shouyi Yin.)*

The authors are with the Beijing Innovation Center for Future Chip and the Beijing National Research Center for Information Science and Technology, School of Integrated Circuits, Tsinghua University, Beijing 100084, China (e-mail: yinsy@tsinghua.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2022.3184115.

Digital Object Identifier 10.1109/TCSI.2022.3184115

channel, WG has concurrent dual-dimension intra-channel and inter-channel reuse, where one Emap channel slides on all Fmap channels and vice versa. Current accelerators [5], [8], [9], [17], [18] only adopt fixed intra-channel-reuse dataflow for all three phases. However, it mismatches with the small intra-channel reuse of large-window convolution in WG under the same bandwidth as FF and BP [9], leading to severe 80% hardware underutilization. Third, instead of using each layer's Fmap and weight only once as FF, the BP and WG phases repetitively retrieve the Fmaps and Emaps from off-chip DRAM. Since 32-bit DRAM access energy is 426.7× higher than FP16 MAC, the enormous memory access introduces a bottleneck for energy-efficient training. Current accelerators adopt compression methods to solve it, which are mainly for static-and-sparse data distribution [19]–[21]. However, they are either afflicted by performance degradation for dynamic training data distribution of recent high-accuracy networks with non-sparsity activations or high hardware complexity and large codec latency.

This paper proposes a processor named PL-NPU, solving the above challenges with low-power logarithmic computing, dynamic reuse dataflow, and value adaptive compression. Compared with training a ResNet18 model with the state-of-the-art standard training processor [5]–[8], [22], the proposed PL-NPU achieves 3.75× higher energy efficiency and reduces 1.68× training time. The following three innovations contribute to the superior performance of PL-NPU:

1) A posit-based logarithm-domain PE (PLPE) ensures high precision data format and low-cost MAC computation. The posit format with dynamic-exponent-width quantization dynamically adapts to ever-changing DNN training data distribution in three phases, and the logarithm domain computation transfers complicated linear domain multiplication into simple logarithmic domain addition for MAC energy reduction. It improves energy efficiency by 1.97× and speed by 1.68×.
2) An inter-intra-channel-reuse dataflow dynamically reconfigures the operand mapping pattern to match the data reuse in the different training phases. It performs intra-channel reuse in FF and BP while reconfiguring to dual-dimension inter-intra-channel reuse in WG. It increases hardware utilization by 1.41×.
3) A pointed-stake-shaped codec unit (PSCU) compresses the massive small operands into a variable-length taper-shaped exponential-golomb format but the rare large operands into fixed-length posit format, leading to a pointed-stake-shaped distribution of the compressed operands. The data-value adaptive compression improves the compression ratio while reducing the codec overhead. It reduces DRAM access by 65.7%.

The rest of this article is organized as follows. Section II introduces the background and motivations. Section III shows the overall architecture of the proposed PL-NPU. Section IV describes PLPE. Sections V and VI detail the convolution core with the reconfigurable ICRD and PSCU, respectively. Section VII presents the evaluation results. Finally, section VIII concludes this paper.

II. Background And Motivations

A. DNN Training

The most common DNN models always consist of blocks comprising convolution layers and batch normalization (BN) layers. Within each block, one or more convolution layers are typically followed by a BN layer. DNN training iteratively performs FF, BP and WG in a mini-batch granularity. We detail training phases of $i-th$ layer as follows with $k\times k$ kernel size, C_{in} input channels and C_{out} output channels.

1) FF and BP Phase: In FF, input Fmap x is convoluted with a set of weight kernels w to obtain the output Fmap y. The pixel $y_{h,w}$ of $j-th$ channel can by computed by:

$$y^{i,j}_{h,w} = \sum_{n\in[-k,k]^2}\sum_{C_{in}} x^{i}_{(h,w)+n}\times w^{i,j}_{n} + b^{i,j}$$

In BP, Emap $(\partial\mathcal{L}/\partial x)$ in the $i-th$ layer is computed by convolving the $i+1-th$ layer's Emap $(\partial\mathcal{L}/\partial y)$ with $i-th$ layer's transposed w^T as follows:

$$(\frac{\partial\mathcal{L}}{\partial x})^{i,j}_{h,w} = \sum_{n\in[-k,k]^2}\sum_{C_{out}} (\frac{\partial\mathcal{L}}{\partial y})^{i+1}_{(h,w)+n}\times (w^{i+1,j}_{n})^{T} + (\frac{\partial\mathcal{L}}{\partial b})^{i+1,j}$$

Though the procedure in BP seems similar to FF, there are two different features between them. First, error propagation of BP is a progressive stochastic process due to the stochastic mini-batch gradient descent, leading to a wider data range compared with FF. It comes from the reason that the output of the model would have large difference with ground truth at the early training iterations while the difference would be small at the end of training [10], [11]. Second, the BN in BP requires retrieving Emaps multiple times to calculate the data-dependent parameters by traversing Emaps' batch, introducing several times higher memory access than FF, which can merge BN into convolution [23].

2) WG Phase: Unlike FF and BP, which are small window intra-channel sliding convolution, WG performs large window inter-intra-channel sliding convolution. For the intra-channel dimension, Emaps $(\partial\mathcal{L}/\partial y)$ conduct large window sliding [9], [24] with few sliding numbers since the Emaps have nearly the same size as Fmaps (x). In conclusion, the weight gradient of $j-th$ filter can be computed by:

$$(\frac{\partial\mathcal{L}}{\partial w})^{i,j}_{r,s} = \sum_{n\in[0,H-k][0,W-k]} (\frac{\partial\mathcal{L}}{\partial y})^{i,j}_{n}\times x^{i}_{(r,s)+n}$$

Apart from the intra-channel sliding convolution, WG still contains inter-channel sliding convolution, where each Emap channel should slide on all Fmap channels. The weight gradients calculated by WG are needed to be accumulated across the mini-batches to update the model. These gradients would be extremely small when the model is converging, indicating a high precision data representation requirement.

B. Three Challenges of DNN Training

1) Large MAC Power Consumption: The high bit-width FP data format requirement in BP and WG results in intolerable MAC power consumption. Unlike FF as inference, BP and WG empirically require a high bit-width FP data format, like FP32

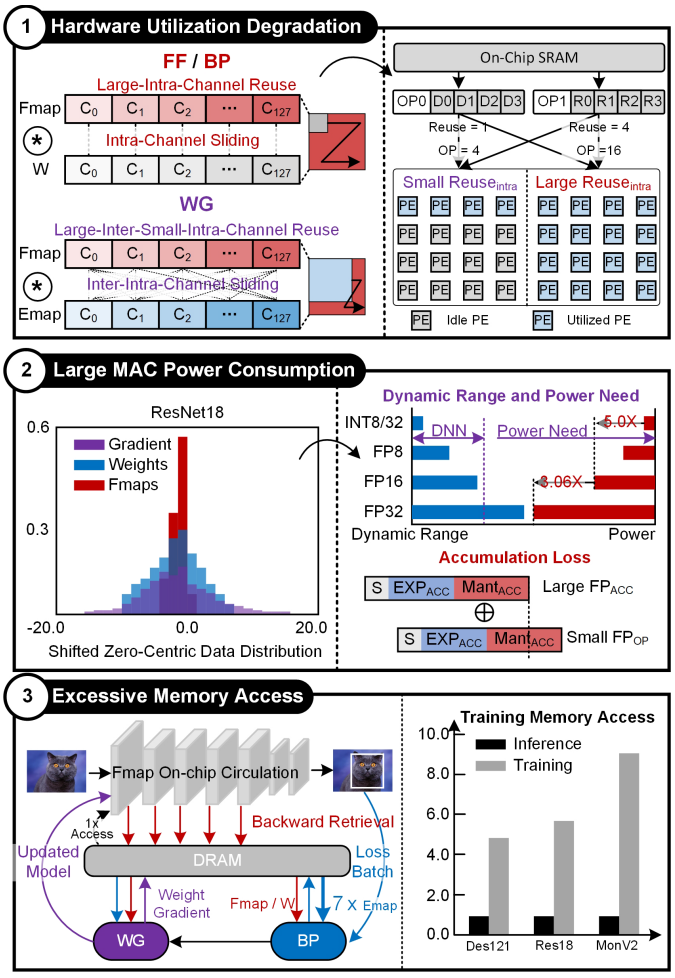


Fig. 2. Three challenges of local training introduced by different characteristics of DNN training phases.

and denormal FP16, to satisfy the wide data range in BP and high data precision in WG, which even reach 2^{40} and 2^{-20} in ResNet18 [10], respectively. It leads to massive memory access and huge MAC power consumption. Recent low-precision training works proposed low-bit-width and fixed-bin-width quantization to solve it but confront the low resolution for cluster center data in the sharp DNN distribution. What is worse, it still requires preserving high precision intermediary values like FP30 with actual hardware implementation [25] for higher accuracy, resulting in a large FP MAC unit. For instance, WAGE [12] confronts severe accuracy degradation with its linear fixed-bin-width quantization, and some works [13], [15], [16], [26] need FP32 general matrix multiply operation for high precision convolution. However, the power consumption of FP32 MAC is 15.3× and FP16 MAC is 5.0× compared with that of INT8 multiplication and INT32 accumulation used in FF, introducing intolerable energy for edge-device training.

2) Hardware Utilization Degradation: When facing small intra-channel data reuse in WG due to the large window convolution, existing dataflows [17], [18], [27], mainly designed for FF and BP of large intra-channel reuse, suffer from hardware utilization degradation under the same bandwidth. For instance, assuming we have 16 PEs with a bandwidth of 4×2 operands, each cycle would have 16 operations if the reuse time is 4. In this case, each of the 16 PEs can receive an operation, resulting in 100% utilization. However, if the reuse time is only 1, the operation becomes 4 and 12 PEs have to keep idle. Therefore, the utilization degrades to 25%. The reuse difference between FF/BP and WG is extremely large. Specifically, in FF, the 3×3 small window weight in the second convolution layer of ResNet18 can be reused 56×56 times when sliding on 56×56 input Fmap. On the contrary, for the same layer, the intra-channel operands reuse times of large-window convolution in WG is only 9. Thus, the utilization is 12.5% for NVDLA-liked output stationary dataflow [17] and 14.1% for ShiDianNao-liked output stationary dataflow [18]. Some works [9], [24] proposed heterogeneous core to optimize the intra-channel reuse of WG, but the WG core keeps idle in FF and BP, leading to nearly 55% underutilization in FF and BP phases. The severely degraded utilization limits the training energy efficiency of WG, which has nearly the same computation amount as FF and BP.

3) Excessive Memory Access: Different from FF, WG and BP need to retrieve Fmaps and Emaps from off-chip memory multiple times, which consumes enormous data access energy, introducing a bottleneck for energy-efficient training. Specifically, the largest Fmap in ResNet18 is of size $64 \times 56\times56$, which is 392KB with 16b data format. In FF, since most processors are usually integrated with SRAM larger than 392KB, it can store the former layer's Fmap for the latter layers' reuse, avoiding off-chip data transmission. However, BP and WG require all layers' Fmaps generated in FF, which is too large to be stored on-chip, introducing at least 2× extra off-chip Fmap retrieval. Besides, the BN layer in BP requires 6× repetitive off-chip Emaps retrieval across batch granularity due to the data-dependent computation [23]. For instance, with a batch size of 32, the memory access of the corresponding BN layer in ResNet18 is 73.5MB, which is equal to 8.21G FP16 MAC operations. Compression is a prevalent method to mitigate it, but they mostly are based on the static weight in inference [19] or high sparsity with ReLU [5], [20], making them hard to be applied to dynamic training data distribution with non-sparsity activation. Works with adaptivity often come with hardware overhead or codec latency. For instance, apart from code table overhead, adaptive huffman coding [28] requires traversing a leaf to root to generate one encoding word and reconstructing the coding tree for every encoding and decoding. Thus, the massive training memory access requirement is power-and-time-consuming for edge devices.

III. Overall Architecture

Fig. 3 shows the overall architecture of PL-NPU. It consists of a top controller, three SRAM buffers, a parameter predictor, a pointed-stake-shaped compression unit (PSCU), four cores with inter-intra-channel-reuse dataflow (ICRD) consisting of a total of 256 posit-based logarithm-domain PEs (PLPE). Top controller controls the execution of PL-NPU. A 128KB input memory stores the Fmaps and weights in FF, Emaps and weights in BP and Fmaps and Emaps in WG, respectively. The 288KB output buffer distributed in four cores stores

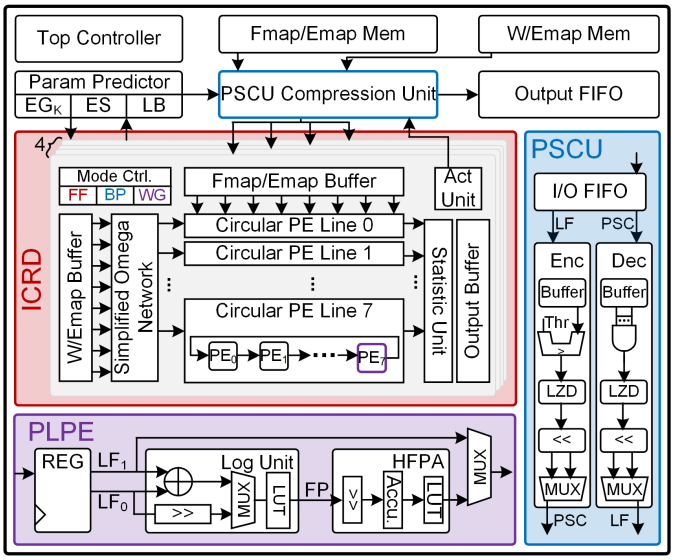


Fig. 3. Overall Architecture of PL-NPU processor.

the computation output. The parameter predictor collects the run-time information from cores and calculates the control code (layer bias (LB), EG_K and ES) for PSCU. In each core with 64 PLPEs, 8 PLPEs is grouped to be a PE lines, which works under ICRD with reduced-case omega network. In FF or BP phase, core is configured to be weight stationary dataflow. Each PE line computes one output Fmap channel while input Fmaps are shared across 8 PE lines. While in WG, core is reconfigured based on the Emap's channel with reduced-case omega network, computing corresponding weight gradient. PLPE works with logarithm-domain multiplication and linear-domain accumulation. The former achieved by the energy-saving log unit, while the latter guarantees the accumulation precision by hybrid-floating-fixed-point accumulator. The input and output of the core are compressed with PSCU, which greatly reduces the memory access consumption. The encoder compresses the small operands into short-and-variable-length exponential-golomb format while the large operands into long-and-fixed-length posit format. The decompression unit performs the reverse process.

IV. Posit-Based Logarithm-Domain PE

Low bit-width training is practical to reduce the MAC Power. However, the previous low-bit-width methods suffer from severe training accuracy degradation since their fixed data format cannot satisfy the concurrent data range and precision requirement. We propose posit-based logarithm-domain PE. It performs dynamic-exponent-width quantization to adapt to the various data range and precision requirements for considerable training accuracy and transfers complicated linear domain multiplication into logarithmic domain addition for MAC energy reduction.

A. Posit-Based Data Format

Posit number system is an encoding scheme with tapered precision. The general format of the posit consists of four parts [29], [30], and the detail structure is shown as Table I.

TABLE I
THE FP AND POSIT FORMATS' COMPARISON

float	Sign	Exponent (Size ES)		Fraction (Size F)
	s	$e_1e_2e_3\ldots e_{es}$		$f_1f_2f_3\ldots f_F$
posit	Sign	Regime (Run Length K)	Exponent (if any) (Size ES)	Fraction (if any)
	s	$r_1r_2r_3\ldots\overline{r_k}$	$e_1e_2e_3\ldots$	$f_1f_2f_3\ldots$

TABLE II
EXAMPLE OF LOGARITHM POSIT FORMAT

Posit	Regime	Exponent	Fraction	Value
01001110	10	0	1110	$4^0 \times 2^{0.1110}$
11111101	111110	1	None	$-4^5 \times 2^{1.0}$

A posit format (N, ES) is defined by its total bit-width N and its exponent size ES. The regime bits of posit, whose length K is encoded with the run-length method, include $K-1$ identical bits $r_1r_2\ldots r_{k-1}$ and ends with one opposite bit $\overline{r_k}$. Its value V_k is $K-1$ if $r_1=1$, and $-K$ with $r_1=0$. The scaling factor $ussed^{V_K}=(2^{2^{ES}})^{V_K}$ calculated by ES and K, determines the dynamic range of a posit number. Apart from the sign bit and regime bits, the ES-bit exponent $e_1e_2\ldots$ and fraction $f_1f_2\ldots$ compose the remaining part. Since the length of regime bits is dynamic, which at most extends to $N-1$, if there is no room for exponent and fraction, they are assumed to be zero. Regime part and exponent part act as the exponent function as standard FP by $ussed^{V_K}\times 2^e$. The logarithm posit format transfers the fraction value $1.f_1f_2\ldots$ into the $2^{0.f_1f_2\cdots}$. Thus, the value of a logarithm posit number is given by Eq. (1) and Table II gives two examples for the posit (8, 1).

$$value=\begin{cases}0, & 000\ldots0\\ \pm\infty, & 100\ldots0\\ (-1)^s\times ussed^{V_k}\times 2^{e.f}, & O.W.\end{cases} \quad (1)$$

It is very appropriate for DNN training [31], which has different data distributions in different phases. Concretely, the Fmap in the FF phase has a relatively smaller range, and we can use (8, 1) to represent the data. In the BP phase, Emap has a wider range, where we utilize (8, 2) or (8, 3) posit to confirm the coverage. Due to the run-length encoding manner of regime part, posit with low magnitude becomes more precise for it has a wider fraction part. Since DNN has a sharp-bell-curve log-normal data distribution, a large portion of zero-centric data after being shifted from the original DNN distribution will have higher precision, leading to higher network accuracy.

B. Logarithm-Domain Computing

1) Log Unit: PLPE adopts log unit, as shown in Fig. 4, to achieve simple logarithm-domain computation with an adder or a shifter, and transfer intermediate result to linear domain FP with a LUT. Log unit mainly consists of a log adder/shifter, a layer bias adder and a logarithm to float (LogToF) unit. It receives two 10b logarithm-domain

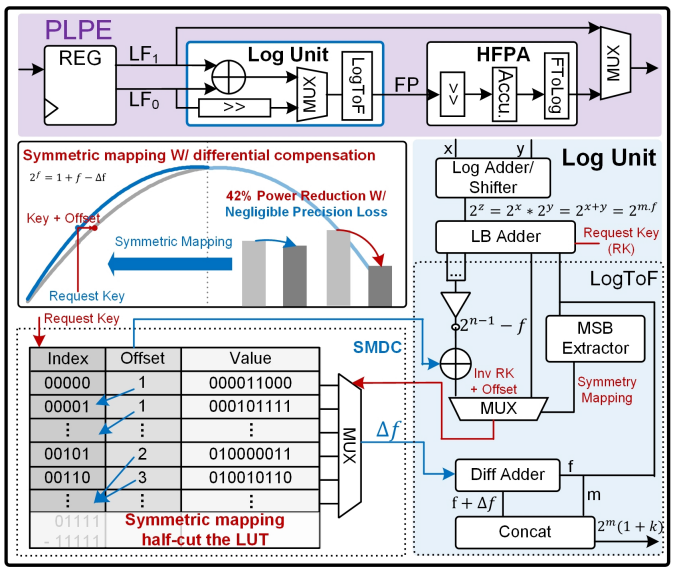


Fig. 4. Proposed log unit with logarithm domain computation and the symmetric mapping with differential compensation LUT.

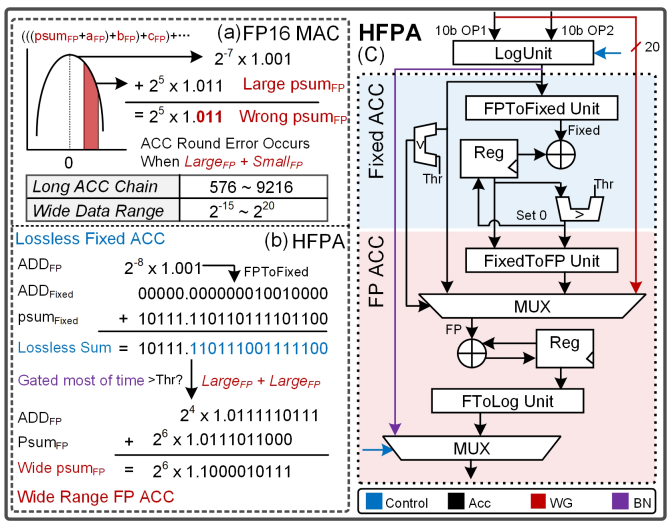


Fig. 5. Proposed HFPA with two-stage accumulator, consisting of a fixed-point accumulator and a floating-point accumulator.

fixed-point input operands, containing 1b linear-domain sign and 9b logarithm-domain fixed-point (LF), whose integer bit-width is determined by parameter ES. For instance, it is a 5b integer and 4b fraction with $ES = 1$ while 6b and 3b with $ES = 2$. In convolution, the log adder achieves multiplication with an adder by performing addition of log domain input operands. Then, the layer bias adder adds an exponent bias to the obtained products, resulting in a zero-centric data distribution, which facilitates further accumulation in Hybrid-Floating-Fixed-Point Accumulator (HFPA). Also, the logarithm-domain products are transferred into linear domain by the LogToF unit before accumulation, since it is costly to perform addition/subtraction in logarithm domain compared with linear one. Additionally, achieved with an adder and a shifter, log unit supports BN process, which contains square, square root and division, equally to 1b shift and subtraction in logarithm domain.

However, the above-mentioned procedure introduces extra logarithm domain to linear domain transformation with large LUT to maintain high precision, leading to area deficiency and severe latency. We tackle this with the commonly used approximate technique [32], which is based on two features with symmetric mapping with a differential compensation (SMDC) mechanism. First, the range of difference function $\Delta f = 1 + f - 2^f$, is small, where f represents the fraction part of LF, reducing the bit-width of slots by representing Δf rather than $1 + f$. Second, the difference function is approximately symmetric, enable it to half-cut the LUT. However, direct symmetric mapping leads to serve transformation error. Instead, as shown in Fig. 4, a link list LUT storing the low bit-width offset is introduced as a compensation to guarantee transformation precision. We use a multiplexer to choose the one's complement of fraction with offset or fraction as key to request the LUT, and then access the value Δf to calculate the $1 + f$ with a diff adder. With SMDC, we can achieve at most 42% LUT power reduction with negligible precision loss.

The proposed PLPE only involves simple addition or shifter operation. Therefore, the PLPE is more power-and-area-saving than high-bit FP multiplication.

2) Hybrid-Floating-Fixed-Point Accumulator: Since the input operand has been transferred to the linear domain, we use a HFPA to perform linear-domain accumulation. The FP32 accumulator is a prevalent but costly accumulator for edge device training. Though FP16 accumulator is an energy-efficient alternative to FP32 accumulator, it incurs severe summation precision loss, when two input operands have a large exponent difference [33], [34], as shown in the Fig. 5(a). Posit quire [30] is a large enough fixed-point accumulator proposed to solve this problem and used in some works [35]–[38]. It should be large enough to cover all possible partial products to avoid summation precision loss. Besides, guard bits are used in it to avoid overflow [39]. However, it is unpractical for DNN training. First, the data range in DNN training is extremely wide, which can be up to 2^{40} with some DNNs [10], accounting for at most 2^{80} partial sum range. Second, the accumulation chain can be exceedingly long, which is 9216 in second convolution layers in block 5 of ResNet18, accounting for 13b guard bits.

Thus, instead of adopting quire or FP accumulator, we use the HFPA to effectively solve this issue, inspired by [11], [40]. As shown in Fig. 5(c), one is a fixed-point accumulator, which is lossless but with a small data range, the other is an FP accumulator, which has a large data range but will cause summation precision loss when adding large FP and small FP. We take the advantage of both the fixed-point accumulator and FP accumulator based on the feature of DNN data distribution. Fig. 5(b) demonstrates the accumulation process. First, since FP partial products usually have a small value around zero due to the layer bias adder, they are transferred to the fixed-point and summed up with a lossless fixed-point accumulator. If the lossless partial sum or the input operand exceeds the preset threshold, it will be transferred to large FP and sent to be added with large FP in the FP accumulator, guaranteeing the wide data range. Throughout the whole computation in

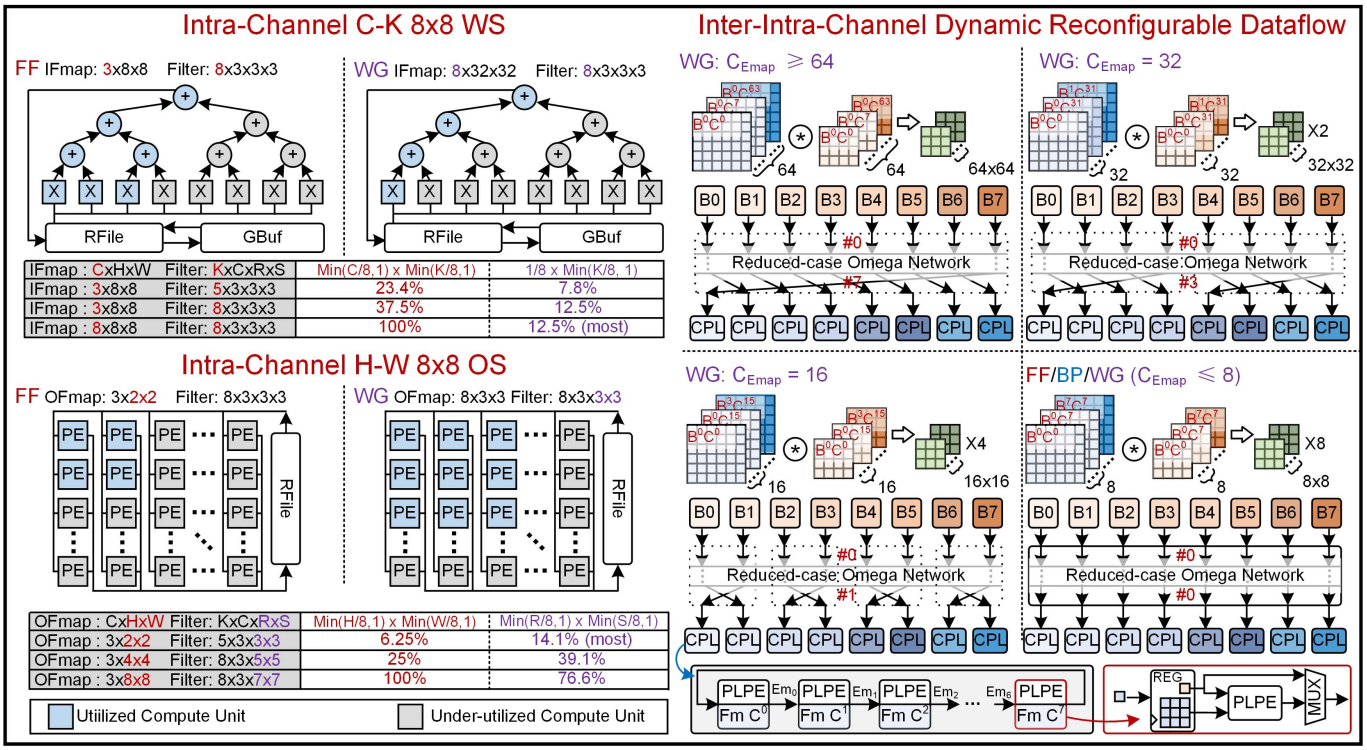


Fig. 6. Left subfigure shows comparison among the WS, OS and proposed ICRD, and the right subfigure demonstrates the detailed mapping of ICRD with RCON and CPL. The blue map and the orange map are Fmaps and Emaps, respectively. The green map is weight gradient, the result of the large window convolution.

ResNet18, nearly 72.3% summation is worked at the energy-saving and lossless fixed-point accumulator thanks to the offset zero-centric partial product distribution, largely reducing the power consumption while maintaining high precision. Finally, we transfer the FP result back to the LF with FToLog, which is designed nearly the same as LogToF. As a result, HFPA can reduce 35.6% accumulation power during training.

V. Inter-Intra-Channel-Reuse Dataflow

A practical dataflow, which effectively reuses data, is essential for training energy efficiency improvement. However, existing dataflows focus on the intra-channel reuse in FF and BP, but they suffer from hardware underutilization for WG because the large-window convolution in WG lacks intra-channel reuse. We propose a dataflow, leveraging the inter-and-intra-channel reuse with intra-PE sliding for intra-channel reuse and inter-PE cyclic operands exchange for inter-channel reuse, which enhances hardware utilization for higher training energy efficiency.

A. Bandwidth Requirement Analysis for Dataflow

For a specific dataflow, we can analyze its PE utilization and bandwidth (BW) with the following equation:

$$PE_{utilized} = BW \times RF = BW \times RF_{intra} \times RF_{inter}$$

Here, $PE_{utilized}$ is the average utilized PE number of a PE array per computation cycle, and RF, denoting the reuse factor representing MAC numbers generated by a pair of input operands per cycle, is the product of RF_{intra} and RF_{inter}. RF is the key for BW reduction. Given constant BW, the larger the RF, the larger the $PE_{utilized}$. Nevertheless, most of dataflows are designed based on the intra-channel reuse, which is large in FF and BP while extremely small in WG, while neglecting the inter-channel reuse, which is 1 in FF and BP while extremely large in WG. The BW is fixed in practical implementation, thus the reuse mismatch between FF, BP and WG leads to severe hardware underutilization. We illustrate this based on two commonly used dataflow, OS dataflow and WS dataflow, as shown in Fig. 6. For instance, WS dataflow of 8×8 PE array has a utilization of 0.78%-12.5% for it only leverages the intra-channel reuse and performs inter-channel accumulation, which is incompatible with WG's inter-channel reuse characteristic and intra-channel accumulation pattern. The inter-channel accumulation pattern limited its utilization to 12.5% and the fixed bandwidth as that in FF and BP further aggregates the underutilization to the minimum of 0.78%. The OS dataflow with only intra-channel reuse also confronts the same dilemma.

B. Inter-Intra-Channel-Reuse Dataflow

This work proposes a dataflow, named inter-intra-channel-reuse dataflow (ICRD), achieving high PE utilization by introducing RF_{inter} in WG with fixed BW at the expense of little hardware overhead, which enables WG to have as high RF as FF and BP. Furthermore, it reduces memory access because its RF in WG is as large as that in FF and BP. Fig. 6 demonstrates the mapping of ICRD, and Fig. 7 shows the

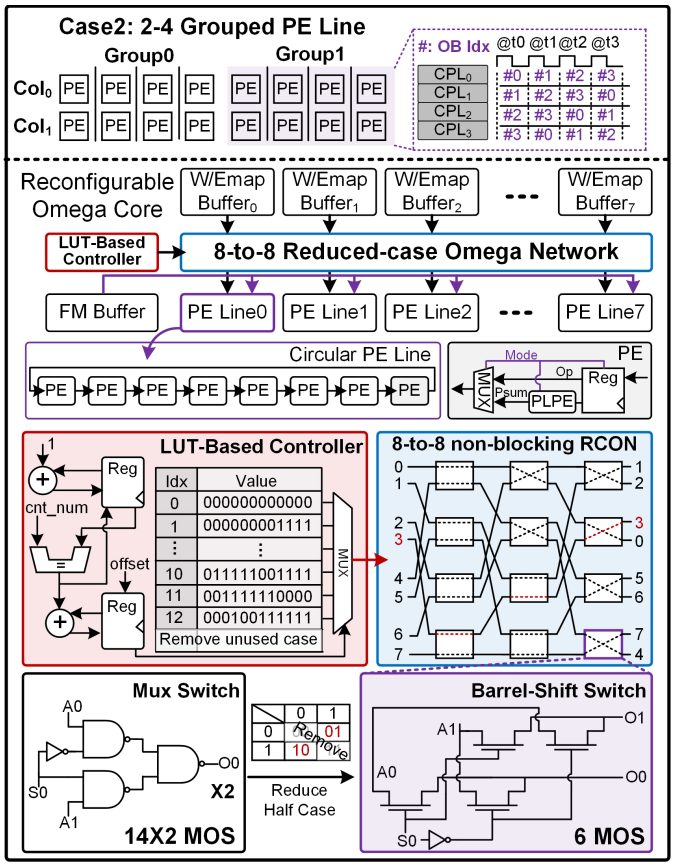


Fig. 7. Detailed hardware of reconfigurable core with reduced-case omega network and circular PE line.

reduced-case omega network and circular PE line to implement it. The ICRD outperforms the previous methods, especially for energy-efficient training.

1) Dynamic Mapping Based on C_{Emap}: ICRD is optimized based on the WS, which is acknowledged as one of the most energy-efficient dataflow utilizing the intra-channel reuse in FF and BP [17]. It can be configured during run time based on the top controller, which sends the configuration and stores it in the configuration buffer of the ICRD core before next layer execution. In FF and BP, we preload weights and keep partial sums in the buffers and the PE registers to reduce the BW. Each PE line is responsible for one output Fmap channel. As shown in the bottom part of Fig. 6, the related input Fmap channels are evenly split into 8 parts for 8 PEs, and each PE performs intra-channel accumulation for corresponding input Fmap independently. Then, 8 partial sums are accumulated to obtain one output Fmap channel convolution result. In this case, ICRD is nearly the same as WS except for the in-place accumulation in FF and BP, adapting to FC, depth-wise, point-wise convolution layers, and different strides. Note that, we adopt the in-place accumulation, rather than adder tree accumulation as NVDLA [17], to facilitate the WG computation, which we will illustrate below.

In WG, each PE computes weight gradients of one filter, and one PE group is responsible for one batch weight gradients. Therefore, the total mapping PE number within one batch is C_{Emap}. We introduce the batch dimension to enhance the utilization and divide the mapping into four categories based on the C_{Emap}. For example, if $C_{Emap} = 32$, as shown in the Fig. 6, we divide the circular PE lines (CPL) into two groups, and map one batch weight gradients to a CPL group containing 4 CPL, and the other batch to the other group. Eight channel Emaps are stored in one buffer, thus the buffer0 (B0) to buffer3 store a batch of Emaps with 32 channels. Each PE is also responsible for one channel Fmaps. Since each channel Emap should convolute with all channel Fmaps, Emaps in 4 buffers should be sent to all PEs within one group. We achieved it by reduced-case omega network and CPL, and illustrate them in detial in next subsection.

2) Reconfigurable ICRD Core: To achieve the above-mentioned mapping, we design a reconfigurable core with reduced-case omega network (RCON), CPL and in-situ reuse scheme. First, we maximize the intra-channel reuse based on the in-situ reuse scheme with operand buffers. In FF and BP, the W/Emap buffer preloads weights to avoid repetitive access as WS dataflow. In WG, W/Emap buffers store Fmaps as sliding windows, and we keep overlapping Fmap pixels between two sliding windows of two adjacent Emap pixels in-situ, reducing 2.8× BW requirement.

We fully leverage the inter-channel reuse in WG with inter-PE cyclic operands exchange by CPL and RCON. As shown in middle part of Fig. 7, controlled by *mode* signal, CPL right streams partial sums stored in the registers to accumulate them in the right-most PE in FF and BP, while performing intra-line-cycle to exchange the Emaps stored in registers of each PE in WG, accounting for 8× BW reduction in WG. Thus, each Emap convolutes all Fmaps within one CPL. Besides, instead of adopting naive 8-to-8 MUXs to broadcast Emaps in 8 buffers to each CPL, RCON is introduced to perform inter-line-cycle, which can be configured to be a circular shifter within the group. For instance, with C_{Emap}=32, CPL0 sequently receives Emaps from B0 to B3, which means that one Fmap channel convolutes with 32 Emap channels, further reducing nearly 8× BW requirement. The same is simultaneously executed for the other group. As a total, with ResNet18, ICRD can contribute to at most 48.3× PE utilization enhancement in WG compared with WS. When it comes to overall training, ICRD accounts for 1.41× utilization improvement and 2.68× SRAM energy reduction.

To reduce the hardware overhead, we simplify the omega network with the LUT-based controller and the barrel-shifter switch. First, with our special designed grouping manner, we remove the unused case of original omega network, avoiding the complex control unit tackling blocking with a LUT-based controller and making highly congested networks non-blocking. Second, we remove the redundant propagation cases of switch, 00 and 11, in broadcast-enable omega network [41] by only preserving straight and cross, and further optimize the MUX switch to the barrel-shifter switch, reducing the MOS transistors number from 28 to 6. Third, we cut the LUT in half, and eliminate the extra MUXs and the critical path introduced by cascade propagation destination index. The destination index acts as the switch's selecting signal and needs to propagate along the multi-layer omega

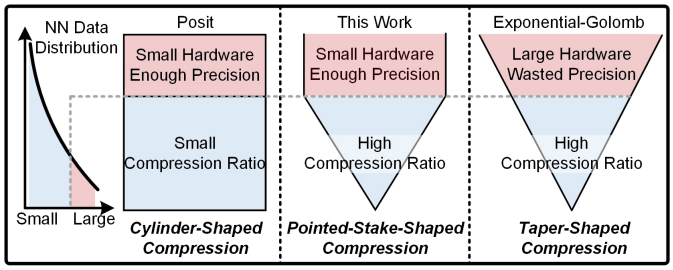


Fig. 8. Comparison of three compression formats.

TABLE III
EXPONENTIAL GOLOMB CODING RULE TABLE

v	Signed	$EG_K = 0$	$EG_K = 1$	$EG_K = 2$	$EG_K = 3$
0	0	1	10	100	1000
1	1	010	11	101	1001
2	-1	011	0100	110	1010
3	2	00100	0101	111	1011
4	-2	00101	0110	01000	1100
5	3	00110	0111	01001	1101
6	-3	00111	001000	01001	1110

network. Instead, we only store the control signals rather than destination indexes, reducing the hardware overhead and latency.

VI. POINTED-STAKE-SHAPED-COMPRESSION

Compression is a promising method to tackle excessive training memory access, which is extremely power-consuming. However, the prior approaches suffer from either sub-optimal compression ratio or large overhead since they cannot adapt to the dynamic training data distribution. We propose a pointed-stake-shaped compression unit, taking the advantage of exponential-golomb and posit format by compressing the small operands into variable-length to achieve a high compression ratio and encoding large operands into fixed-length to reduce hardware overhead while maintaining considerable accuracy. We further extract the shared hardware of them to integrate them, addressing the training bottleneck with little extra hardware cost.

A. Data Distribution and Compression Format Analysis

DNN data distribution is narrow, and small numbers take up the majority. Taper-shaped variable-length coding is a prevalent technique to handle this bell-curve-liked distribution with an appreciable compression ratio. However, it pays costly hardware overhead for rare numbers with little return. For instance, with exponential-golobm (EG) format, nearly 81% operands in ResNet18 take up less than 8b length, while the rest forces 19b hardware to adapt to all possible value, resulting in more than 2× hardware overhead.

EG is a universal extension of golomb coding [42], [43] for optimal geometric distribution compression, which is appropriate for DNN compression. It consists of three parts, including quotient (q), remainder (r) and tunable parameter EG_K. The input value v with offset $+2^{EG_K}$ can be represented by

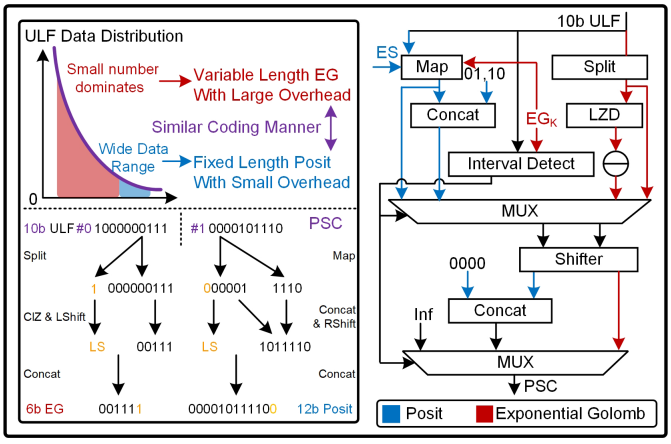


Fig. 9. Detailed datapath of PSCU encoder. This encoder is configured with $EG_K = 0$ and $ES = 1$.

$2^{q+EG_K} + r$, where q is encoded with unary prefix and can be represented as q one followed by a zero. R is encoded by truncating the $q + EG_k$ LSBs. With the signed pattern, EG further maps the signed positive number pv to $2 \times pv - 1$ while the signed negative numbers nv to $-2 \times nv$ before applying the unsigned encoding. Therefore, the EG coding rule with parameter EG_K is demonstrated in Table III. As mentioned above, EG code length is based on the operand's value, thus the representable maximal number leads to maximal EG coding length, which is rare in DNN distribution. However, to adapt to all possible values in training, the EG codec unit forces a circuit with up to 2× bit-width compared with original data format, introducing seldom used but expensive extra hardware for rare operands.

Unliked variable-length compression formats, posit can roughly be seen as a cylinder-shaped fixed-length compression format for higher bit-width FP format. It has a variable-length exponent size, enabling it to have a wider representable range needed in BP and WG. Thus, it is proved efficient in DNN training. However, it fails to remove the redundant 0s in MSB of small operands, which is the majority of DNN data, leading to a limited compression ratio.

B. Shared Codec Unit

Based on the above analysis, an effective solution is to combine these two formats. However, simply combing EG and posit fails to work due to following two reasons. First, q of EG is encoded with leading zero manner while regime of posit is encoded with leading zero or leading one manner, resulting in mis-decoding. We tackle this by prefixing posit with 0s to distinguish it and EG. Second, their coding ranges overlap, leading to a waste of bits. Adding an offset to posit seems to solve this problem. However, they are both zero-centric signed format and the offset coding only covers the positive or the negative half axis, forcing asymmetric coding that makes it difficult to design hardware. Thus, we introduce a novel offset unsigned posit to solve it. We follow the similar signed to unsigned mapping rule as EG for the regime of the posit (map signed positive regime pk to $2 \times pk$ while the negative

regime nk to $-2\times nk-1$) to make unsigned posit more precise with relative small number. For instance, since 8b posit with $ES = 1$ occupies an integer range from [-14, 12] and the maximal unsigned logarithm-domain fixed-point (ULF) for 8b EG with $EG_K = 0$ is 0.1111, we relocate posit's integer part to unsigned [1, 27]. Finally, we term the combination of them as the pointed-stake-shaped (PSC) format for it is variable-length for small data and fixed-length for large data.

Furthermore, we achieve a high compression ratio with small extra hardware overhead compared with posit-only codec unit by exploiting the common coding pattern in EG and posit. First, in the encoding stage, posit and EG needs to right shift and left shift the operand to encode the regime and q, respectively. Second, in the decoding stage, they both need a leading zero detector to calculate the regime and total bit-width, and a shifter to move the fraction part and next decoded operand away, respectively. Thus, by sharing the common part, we integrate them into one codec unit.

1) PSCU Encoder: The proposed encoder, transferring the ULF to the PSC, is demonstrated in Fig. 9. It first determines the coding manner with interval detect, and then select the pre-processed data based on it. With EG format, the split unit extracts the linear sign bit (LS), and then the leading zero detector (LZD) finds out the 0 MSBs to be left shifted away. With posit format, based on ES and EG_K, it first extracts the integer of ULF and performs offset and signed to unsigned mapping of integer to get the regime and ES. Then, the ES and fraction would be concatenated with 01 or 10 before being right shifted. We reverse the intermedia result to be left shifted in EG data path and reverse it again after shifting to avoid an extra shifter. Finally, a multiplexer chooses the EG or posit as the output. For example, the first ULF is within the range $[0, 0.1111]$, which means it should be encoded as generally used EG. The second input should be encoded with posit format. Thus, we first subtract integer by $EG_K + 1$ to get the coded integer 1, and then obtain the regime 0 and ES 1. Concatenated with 10, ES and fraction part should be right shifted based on regime, and we can get the final output 000010111100 with the prefix 0000 and suffix linear sign bit.

2) PSCU Decoder: The proposed decoder, transferring the PSC data into ULF, is depicted in Fig. 10. First, the NOR gate detects whether there is prefix (PF) 0000 to determine the decoding manner, and the leading zero detector calculates the regime and the total bit-width of posit and EG, respectively. Based on the prefix, the multiplexer selects the intermediate result to be decoded, and then the shifter left shifts to remove away ES and fraction parts for posit or next decoded data for EG. Same as encoder, the EG operand to be right shifted is reversed before and after left shifting to prevent an extra shifter. After shifting, MuxConcat and mapping unit post-processes the posit before posit and EG are selected by a multiplexer. The upper subfigure of Fig. 10 shows the comparison between EG and posit. It demonstrates that by combing them, we can achieve variable code-length for small data with small extra hardware cost, which is depicted by red lines for EG data path in right subfigure. The lower subfigure shows the decoding process of the PSC. Since the first input does not have prefix, it should be decoded with EG. The second input with prefix is posit, and should be decoded by a reverse process as encoder.

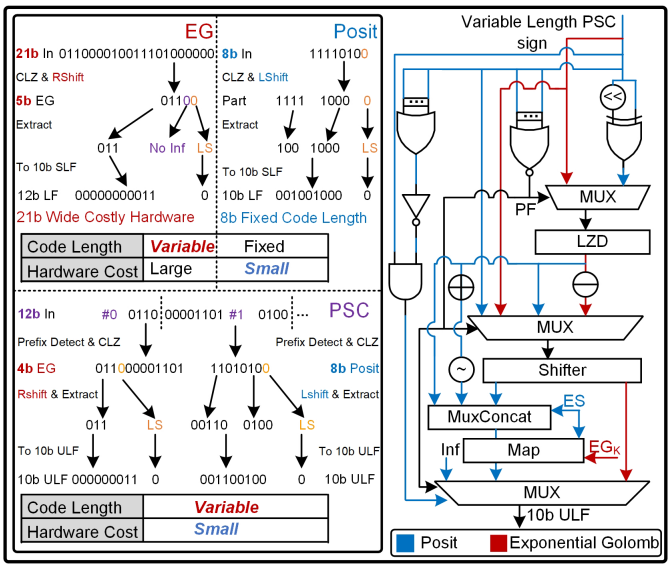
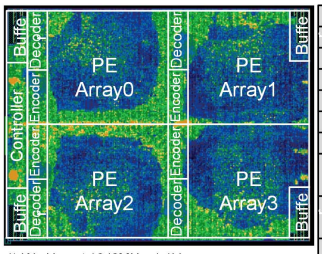


Fig. 10. Detailed datapath of PSCU decoder. The decoder is configured with $EG_K = 0$ and $ES = 1$.



1) Working at 1040MHz, 1.1V
2) Working at 95MHz, 0.6V

Chip Summary

	Specification
Technology [nm]	28
Area [mm^2]	5.28
Data Precision	Posit8
Supply Voltage [V]	0.6 - 1.1
Frequency [MHz]	60 - 1040
Power [mW]	11 - 343
Peak Performance [TFLOPS]	0.532[1)]
Area Efficiency [TFLOPS/mm^2]	0.101[1)]
Energy Efficiency [TFLOPS/W]	3.87[2)]

Fig. 11. Backend graph and summary of PL-NPU.

VII. Measurement Results

A. Chip Implementation Results

Fig. 11 depicts the backend micrograph and provides a summary of measurement results. Synthesized with 28nm TSMC technology, PL-NPU occupies a die area of 5.28mm^2. It contains 256 PLPEs, performing logarithm-domain multiplication and hybrid-floating-fixed-point accumulation. Here, one MAC accounts for two operations. Working under 0.6-to-1.1V, its corresponding frequency is of 60-to-1040MHz, resulting in a peak performance of 0.532TFLOPS and a peak area efficiency of 0.101TFLOPS/mm^2 at 1.1V. The power consumption of PL-NPU is 11-to-343mV, enabling it to achieve a peak energy efficiency of 3.87TFLOPS/W when evaluated at 0.6V with 95MHz. Since local retraining is the main target of PL-NPU, Table IV shows accuracies of 4 commonly used networks with negligible accuracy degradation when retrained with our PL-NPU on ImageNet [44] and CIFAR-100 [45] with the algorithmically generated corruptions following the set of [46], compared with FP32 baseline (the models trained

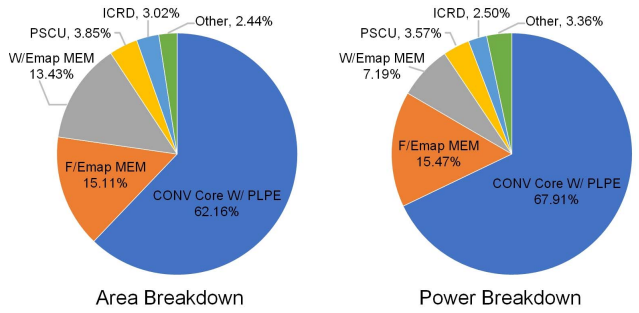


Fig. 12. The area and power breakdown of PL-NPU.

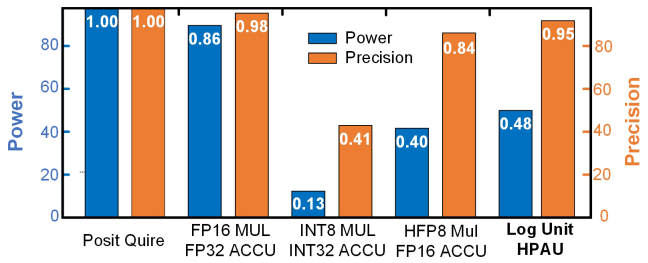


Fig. 13. Comparison among different PEs in terms of precision and power.

TABLE IV
RETRAINING ACCURACY

Dataset	Model	FP32 Baseline Top-1 (%)	Posit Top-1 (%)
ImageNet	ResNet18	70.9	70.1
	VGG11	70.7	70.0
	DenseNet121	74.7	73.9
	GoogLeNet	70.1	69.5
CIFAR-100	ResNet18	75.8	75.3
	VGG11	69.1	68.7
	DenseNet121	77.6	77.1
	GoogLeNet	78.3	77.9

TABLE V
TRAINING-FROM-SCRATCH ACCURACY OF RESNET18 ON IMAGENET

Network	Acc.	Δ	Format
ICLR'18 [12]	66.92	-2.68	INT8
NIPS'18 [11]	67.34	-2.26	FP8
ICML'19 [16]	65.8	-3.8	INT8
ICLR'19 [13]	69.6	0.0	S2FP8
JSSC'21 [25]	69.0	-0.6	FP8
Ours	**69.4**	**-0.2**	**8-bit Log Posit**

on CIFAR-100 is modified to adapt to the 32 × 32 input). We further compare PL-NPU with other low-precision training works [11]–[13], [16], [25] for a train-from-scratch task with ResNet18 on ImageNet, as shown in Table V.

Fig. 12 depicts the area and power breakdown of PL-NPU, which is evaluated based on the standard library from TSMC. Convolution core with PLPEs takes an area proportion of 62.16%, mostly from PLPEs and output buffers. It dominates the power consumption with the 67.91% proportion, which is contributed by the high utilization of PLPEs due to ICRD. PSCU accounts for 3.85% area and 3.57% power proportion since extremely high reuse brought by ICRD results in less workload and corresponding working time of PSCU. Since we achieve ICRD with RCON and CPL, the cost of wiring and switch are lower compared with commonly used MUXs or benes network, reducing the area proportion to 3.02% and power proportion to 2.50%.

B. Evaluation on Chip Feature

This section gives a detailed evaluation on the proposed techniques based on 4 commonly used networks, DenseNet121, ResNet18, GoogLeNet, VGG11 on ImageNet. We first get the energy efficiency baseline by adopting the weight stationary dataflow with in-situ accumulation, the prototype of our dataflow, to demonstrate the corresponding improvement brought by each technique. Thus, the peak energy efficiency for our baseline is 1.03TFLOPS/W@FP16 for ResNet18.

1) PLPE Evaluation: Since we target our PE to operate DNN training, Fig. 13 provides a comparison among 5 PEs in terms of average MAC precision and power when training ResNet18 on ImageNet, and they are normalized based on the posit quire MAC unit. Thanks to our log unit and HFPA, our PLPE achieves significantly low power consumption and comparable precision compared with FP unit. The former is gained because we perform logarithm-domain addition instead of linear-domain multiplication, and most accumulation in HFPA is executed in the fixed-point accumulator, accounting for 48% relative power consumption. The latter is gained due to the high precision transformation with our low-cost SMDC LUT and two-stage accumulation in HFPA, avoiding the large transformation error and accumulation loss, contributing to 95% relative precision. The other 4 MAC units do not utilize the characteristic of DNN data distribution, thus incurs the precision or power issues. Posit quire achieves the highest precision while consuming maximal power for it performs multiplication in linear domain and preserve its intermedia result in 128b fixed-point with 16b posit input [30]. The FP16 multiplier and FP32 accumulator confront the same dilemma for high power-consuming multiplier and accumulator, leading to 86% relative power and 98% relative precision. Accumulation error sometimes occurs, accounting for slightly precision loss compared with posit quire when large operands is added with small operands. The INT8 multiplier and INT32 accumulator have the lowest power consumption, 13%, but also results in extremely low precision, 41%, which fails to work in training. The HFP8 multiplier and FP16 accumulator have comparable power consumption with PLPE but lower precision, leading to accuracy degradation. Apart from hardware overhead reduction brought by PLPE, its data format, 8b posit, also contributes memory access reduction. It further reduces 58.2% memory access compared with FP16. PLPE thereby contributes to 1.97× energy efficiency improvement, reaching 2.03TFLOPS/W@Posit8.

2) ICRD Evaluation: Fig. 14 shows the performance improvement with ICRD. It fully utilizes the intra-channel reuse in FF/BP while inter-intra-channel reuse in WG, enabling PL-NPU to achieve high utilization, which is 92.4% for DenseNet121 and 98.3% for ResNet18, and decreasing the 10.46× SRAM access energy for DenseNet121 and 2.68× for ResNet18. We especially evaluate the computa-

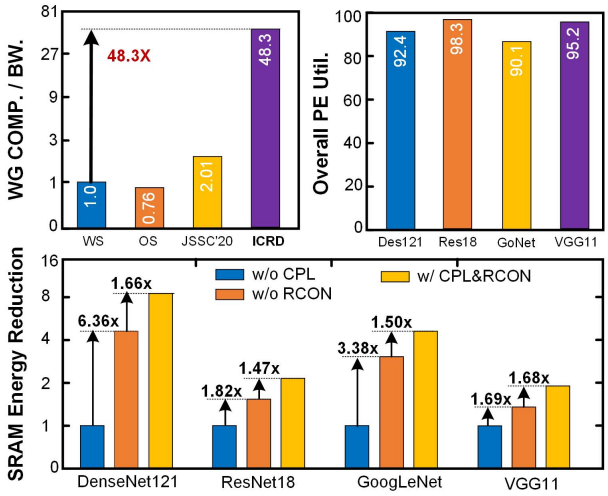


Fig. 14. PE utilization enhancement and memory access reduction brought by ICRD core.

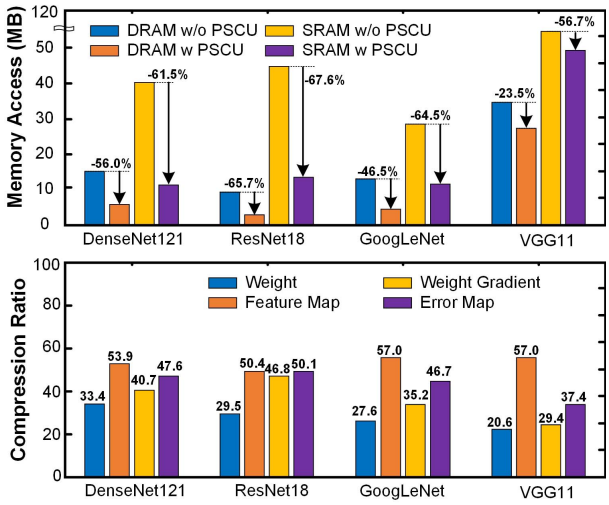


Fig. 15. Memory access reduction and compression ratio with PSCU.

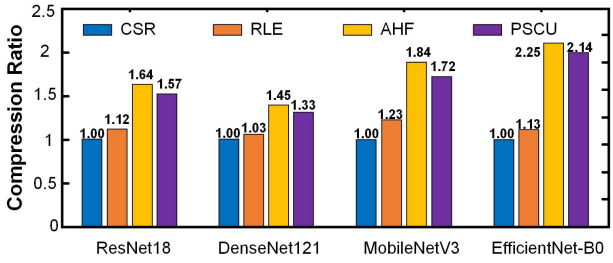


Fig. 16. Memory compression ratio comparison among CSR, AHF, RLE and PSCU with 4 networks on ImageNet.

tion/bandwidth ratio of the 4 dataflows, NVDLA-liked WS with in-situ accumulation, ShiDiannao-liked OS, the WG core of JSSC'20 [9], and ICRD, to show our bandwidth requirement reduction in WG. Then, we demonstrate our priority in terms of the utilization enhancement and SRAM access reduction in the overall training process with ICRD. Compared with WS dataflow, our ICRD achieves at most 48.3× increment and even outperforms the JSSC'20 [9], which is specially optimized for WG, since we fully leverage the inter-channel reuse in WG. The bandwidth requirement reduction further translates to our utilization enhancement. The utilization of 4 networks reaches up to 92.4%, 98.3% 90.1%, 95.2%, which is 1.41× improvement when training ResNet18 compared with WS dataflow. The utilizations of DenseNet121 and GoogLeNet are lower than ResNet18 and VGG11 for they have a point-wise convolution, which does not exist overlapping sliding window, and thus need higher bandwidth for its limited intra-channel reuse potential, leading to lower utilization. Though pointwise convolution lacks intra-channel reuse, it allows not repetitive access within intra-channel dimension, resulting in higher SRAM access reduction. With ICRD, DenseNet121 and GoogLeNet have the SRAM access reduction ratio up to 10.56× and 5.07×. As a result, ICRD achieves 1.41× utilization enhancement and 2.68× SRAM energy reduction, further improving energy efficiency from 2.03TFLOPS/W@Posit8 to 2.72TFLOPS/W@Posit8, 1.34× higher than only using PLPE.

3) PSCU Evaluation: To analyze our compression approach, we first get the baseline without compression. Fig. 15 depicts the compression ratio and the corresponding memory access reduction across 4 networks. The lower subfigure details the compression ratio of 4 types of elements when retraining. Weight and weight gradients have the highest compression ratio across all networks since weights' distribution is narrow, and the gradients of some weights between two iterations are relatively small, such as 2^{-30}, becoming zeros after low-bit quantization. With PSCU, as shown in upper subfigure, we can reduce overall DRAM access up to 56.0%, 65.7%, 46.5% and 23.5% in DenseNet121, ResNet18, GoogLeNet, and VGG11, respectively, when retraining on ImageNet. When it comes to SRAM access, the PSCU can reduce up to 61.5%, 67.6%, 64.5% and 56.7%. We further compare it with 3 commonly used compression methods, including compressed sparse row (CSR), adaptive Huffman coding (AHF), run-length encoding (RLE) with 4 networks on ImageNet, and normalize their performance with CSR in Fig. 16. Since PL-NPU targets edge device local training, we measure the compression ratio during retraining of initial networks with 50% weight sparsity, which allows a higher energy efficiency. ResNet18 and DenseNet121 are ReLU-based networks while MobileNetV3 and EfficientNet-B0 are non-sparsity or few-sparsity swish-based networks, which eliminate the most of input and output sparsity. As shown in Fig. 16, unlike CSR and RLE, which have strict requirements for data distribution to achieve a high compression ratio, AHF and PSCU have the adaptivity to various dynamic DNN training data distribution, allowing them to outperform CSR and RLE, especially for non-sparsity-activation networks. Though AHF's compression ratio is slightly higher than PSCU, as analyzed in Section I, it incurs large hardware overhead and codec latency. On the contrary, our hardware overhead is fairly low for it can be simply integrated into the original posit codec unit with extra 15.1% power consumption and 14.7% area consumption while achieving an extremely high compression ratio compared

TABLE VI
COMPARISON WITH THE PREVIOUS DNN PROCESSORS

	A100 [47][1)]	ISSCC'19 [5][1)]	VLSI'20 [7]	ISSCC'20 [8][1)]	ISSCC'21 [6]	This Work
Technology (nm)	7	65	14	65	7	**28**
Die Area (mm^2)	826	16	9.8	32.4	19.6	**5.28**
Compression Support	CSR	Run-Length	N	N	N	**PSC**
WG Optimization	N	N	N	N	N	**Y**
Supply Voltage (V)	NA	0.78 - 1.1	0.54 - 0.62	0.7 - 1.1	0.55 - 0.75	**0.6 - 1.1**
Frequency (MHz)	1410	50 - 200	1000 - 1500	50 - 200	1000 - 1600	**60 - 1040**
Precision	FP64/32/16, TF32 BF16, INT8/4	FP8, FP16	FP16, FP32	FP8, FP16	HFP8, FP16, FP32	**Posit8**
Power (mW)	40000	43.1 - 367	NA	58 - 647	NA	**11 - 343**
Peak Performance (TFLOPS)	19.5@FP32 78@FP16	0.3@FP16 0.6@FP8	3@FP16	0.54@FP16 1.08@FP8	12.8@FP16 25.6@HFP8	**0.532@Posit8**
Peak Energy Efficiency (TFLOPS/W)	0.0488@FP32 0.195@FP16	1.74@FP16 3.48@FP8	1.41@FP16	1.81@FP16 3.62@FP8	1.8@FP16 3.5@HFP8	**3.87@Posit8[5)] 4.51@Posit8[6)]**
Peak Area Efficiency ($TFLOPS/mm^2$)	0.0236@FP32 0.0944@FP16	0.019@FP16 0.038@FP8	0.3@FP16	0.017@FP16 0.033@FP8	0.65@FP16 1.31@HFP8	**0.101@Posit8**
Max Frequency Energy Efficiency (TFLOPS/W)	0.195@1.41GHz	1.63@0.2GHz	1.1@1.5GHz	1.66@0.2GHz	1.9@1.6GHz	**1.55@1.04GHz**
Real Model Training Efficiency (TFLOPS/W)	NA	0.66[2)]	NA	0.57[3)]	NA	**1.21@Posit8[4)]**

1) They are evaluated with 0% sparsity
2) It uses the benchmark model of ResNet18 on ImageNet with mixed precision of FP8-FP16, achieving 0.66TFLOP/W at 0% sparsity.
3) It uses the benchmark model of CycleGAN, achieving 0.57TFLOP/W at 0% sparsity.
4) We evaluate PL-NPU with the benchmark model of ResNet18 on ImageNet with Posit8 at 0% sparsity with 1040MHz.
5) We evaluate PL-NPU at 0.6V, 28nm.
6) We further explore the voltage-frequency curve and normalize the energy efficiency of PL-NPU at 0.75V, 65nm with library characterization.

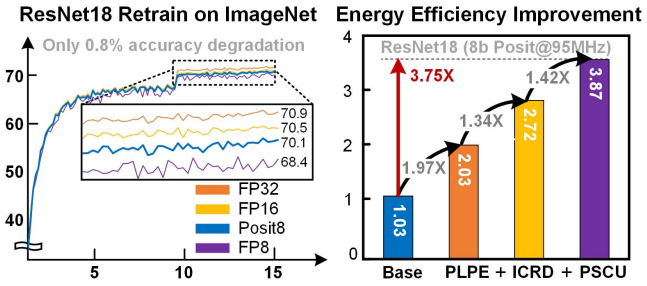


Fig. 17. Retraining curve and the energy efficiency improvement with three proposed innovations.

with the posit-only codec unit. In conclusion, it reduces DRAM access by 65.7% and improves energy efficiency by 1.42×, boosting the energy efficiency of PL-NPU from 2.72TFLOPS/W@Posit8 to 3.87TFLOPS/W@Posit8.

C. Comparison With State-of-Art Processor

Table VI compares the PL-NPU against the prior state-of-art training processors [5]–[8], [47] at 0% sparsity, and left subfigure in Fig. 17 shows the retraining curve comparsion among different precisions. PL-NPU achieves the highest peak energy efficiency of 3.87TFLOPS/W@Posit8. When compared with FP16 training processors, PL-NPU outperforms the [47] and [7] by 19.8× and 2.74×, since PL-NPU is dedicated for the edge device low-precision local training. It avoids the power-consuming FP16 MAC unit and excessive memory access for 16b data format. Apart from peak energy efficiency, even though PL-NPU adopts 8b posit, it achieves considerable accuracy thanks to posit's dynamic exponent size, which fully utilizes the DNN data distribution characteristics. Besides, the peak energy efficiency of PL-NPU is 1.07×, 1.11×, and 1.11× higher than 8b training processors [5], [8] and [6], respectively. It comes from the reason that PLPE facilitates PL-NPU to convert the power-consuming linear domain multiplication into simple logarithm-domain addition and perform most of the accumulation with the fixed-point accumulator. The simplicity of PLPE further enables PL-NPU to operate in a high frequency and a corresponding high energy efficiency, which is 1.55TFLOPS/W with 1.04GHz under 1.1V. To further fairly demonstrate the energy efficiency in a real training scenario, we adopt the metric real model training efficiency proposed by [25], which requires us to additionally take hardware utilization and data movement in all three training phases into consideration. Benefited from the utilization optimization in WG by ICRD and the significant reduction in memory access gained from both ICRD and PSCU, PL-NPU is able to implement ResNet18 with a fairly high real model training energy efficiency of 1.21TLFOPS/W without much degradation compared with the max frequency energy efficiency. On the contrary, even though [8] and [5] have slightly higher max frequency energy efficiencies, they suffer from severe energy efficiency degradation due to two reasons. First, their energy efficiencies are commonly evaluated in FF and BP, but they lack WG optimization, incurring hardware utilization degradation in WG that harms the real model training energy efficiency. Second, [8] fails to adopt compression, and the strict data repetition frequency

requirement of run-length compression used in [5] also limits its memory access reduction. The above two reasons results in relative real model training energy efficiency reduction of 65.7% and 59.5% for [8] and [5] compared with max frequency one, while it is 21.9% for PL-NPU. Thus, PL-NPU's real model training energy efficiency is 1.83× and 2.12× higher than [8] and [5] at 0% sparsity. With proposed three techniques, PL-NPU reaches 3.75× higher energy efficiency and offers 1.68× speedup when training a ResNet18 model.

VIII. Conclusion

In this paper, we design a PL-NPU to support the low-bit training, which is practical for edge-device training. Three innovations proposed for PL-NPU allow it to effectively achieve energy-efficient local DNN training. By transforming the complicated linear-domain computation into logarithm domain and performing high precision accumulation in HFPA with PLPE, PL-NPU can reduce the massive computation power and shorten the crucial critical path. With ICRD, the PL-NPU enhances the hardware utilization and reduces the considerable memory access during training, which is essential for higher training efficiency. With PSCU, PL-NPU encodes the offset zero-centric small data into low-bit EG format and the others into fixed-length posit format, drastically reducing both on-chip communication and off-chip data movement while maintaining the high training accuracy. Benefiting from the proposed methods, PL-NPU achieves a peak energy efficiency of 3.87TFLOPS/W@Posit8, illuminating a prospect of life-long learning on edge devices.

References

[1] Y. LeCun, "1.1 Deep learning hardware: Past, present, and future," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 12–19.

[2] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[3] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 26, 2013, pp. 2553–2561.

[4] M. Song *et al.*, "*In-situ* AI: Towards autonomous and incremental deep learning for IoT systems," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 92–103.

[5] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "7.7 LNPU: A 25.3TFLOPS/W sparse deep-neural-network learning processor with fine-grained mixed precision of FP8-FP16," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 142–144.

[6] A. Agrawal *et al.*, "9.1 A 7nm 4-core AI chip with 25.6TFLOPS hybrid FP8 training, 102.4TOPS INT4 inference and workload-aware throttling," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 144–146.

[7] J. Oh *et al.*, "A 3.0 TFLOPS 0.62 V scalable processor core for high compute utilization AI training and inference," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2020, pp. 1–2.

[8] S. Kang *et al.*, "7.4 GANPU: A 135TFLOPS/W multi-DNN training processor for GANs with speculative dual-sparsity exploitation," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 140–142.

[9] S. Choi, J. Sim, M. Kang, Y. Choi, H. Kim, and L.-S. Kim, "An energy-efficient deep convolutional neural network training accelerator for *in situ* personalization on smart devices," *IEEE J. Solid-State Circuits*, vol. 55, no. 10, pp. 2691–2702, Oct. 2020.

[10] P. Micikevicius *et al.*, "Mixed precision training," in *Proc. Int. Conf. Learn. Represent.*, 2018. [Online]. Available: https://openreview.net/forum?id=r1gs9JgRZ

[11] N. Wang, J. Choi, D. Brand, C. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst., (NeurIPS)*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Montreal, QC, Canada, Dec. 2018, pp. 7686–7695.

[12] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2018. [Online]. Available: https://openreview.net/forum?id=HJGXzmspb

[13] L. Cambier, A. Bhiwandiwalla, T. Gong, O. H. Elibol, M. Nekuii, and H. Tang, "Shifted and squeezed 8-bit floating point format for low-precision training of deep neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2020. [Online]. Available: https://openreview.net/forum?id=Bkxe2AVtPS

[14] X. Sun *et al.*, "Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst., Annu. Conf. Neural Inf. Process. Syst., (NeurIPS)*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds. Vancouver, BC, Canada, Dec. 2019, pp. 4901–4910.

[15] S. Lee, J. Park, and D. Jeon, "Toward efficient low-precision training: Data format optimization and hysteresis quantization," in *Proc. Int. Conf. Learn. Represent.*, 2022. [Online]. Available: https://openreview.net/forum?id=3HJOA-1hb0e

[16] G. Yang, T. Zhang, P. Kirichenko, J. Bai, A. G. Wilson, and C. D. Sa, "SWALP: Stochastic weight averaging in low precision training," in *Proc. 36th Int. Conf. Mach. Learn., (ICML)*, vol. 97, K. Chaudhuri and R. Salakhutdinov, Eds. Long Beach, CA, USA: PMLR, Jun. 2019, pp. 7015–7024.

[17] NVIDIA. (2017). *NVDLA Deep Learning Accelerator*. [Online]. Available: http://nvdla.org

[18] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Jun. 2015, pp. 92–104.

[19] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[20] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. Int. Symp. Comput. Archit.*, vol. 44, no. 3, 2016, pp. 243–254.

[21] N. Samimi, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Res-DNN: A residue number system-based DNN accelerator unit," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 2, pp. 658–671, Feb. 2020.

[22] C. Kim, S. Kang, D. Shin, S. Choi, Y. Kim, and H.-J. Yoo, "A 2.1TFLOPS/W mobile deep RL accelerator with transposable PE array and experience compression," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 136–138.

[23] Y. Wang *et al.*, "A 28nm 276.55TFLOPS/W sparse deep-neural-network training processor with implicit redundancy speculation and batch normalization reformulation," in *Proc. Symp. VLSI Circuits*, Jun. 2021, pp. 1–2.

[24] S. Choi, J. Sim, M. Kang, and L.-S. Kim, "TrainWare: A memory optimized weight update architecture for on-device convolutional neural network training," in *Proc. Int. Symp. Low Power Electron. Design*, Jul. 2018, pp. 1–6.

[25] J. Park, S. Lee, and D. Jeon, "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*, vol. 57, no. 3, pp. 965–977, Mar. 2022.

[26] C. Yang, Z. Wu, J. Chee, C. D. Sa, and M. Udell, "How low can we go: Trading memory for error in low-precision training," 2021, *arXiv:2106.09686*.

[27] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[28] J. S. Vitter, "Design and analysis of dynamic Huffman codes," *J. ACM*, vol. 34, no. 4, pp. 825–845, Oct. 1987.

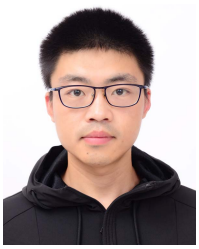
[29] J. L. Gustafson and I. T. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomput. Frontiers Innov.*, vol. 4, no. 2, pp. 71–86, 2017.

[30] P. W. Group. (2018). *Posit Standard Documentation Release 3.2-Draft*. [Online]. Available: https://posithub.org/docs/posit_standard.pdf

[31] J. Lu, C. Fang, M. Xu, J. Lin, and Z. Wang, "Evaluations on deep neural networks training using posit number system," *IEEE Trans. Comput.*, vol. 70, no. 2, pp. 174–187, Feb. 2021.

[32] B. Liu *et al.*, "A 22nm, 10.8 μW/15.1 μw dual computing modes high power-performance-area efficiency domained background noise aware keyword-spotting processor," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 12, pp. 4733–4746, Dec. 2020.

[33] N. J. Higham, "The accuracy of floating point summation," *SIAM J. Sci. Comput.*, vol. 14, no. 4, pp. 783–799, 1993.
[34] J. Müller *et al.*, *Handbook Floating-Point Arithmetic*, 2nd Ed. Cham, Switzerland: Springer, 2018.
[35] H. F. Langroudi, Z. Carmichael, D. Pastuch, and D. Kudithipudi, "Cheetah: Mixed low-precision hardware & software co-design framework for DNNs on the edge," 2019, *arXiv:1908.02386*.
[36] J. Johnson, "Rethinking floating point for deep learning," 2018, *arXiv:1811.01721*.
[37] J. N. Coleman, E. I. Chester, C. I. Softley, and J. Kadlec, "Arithmetic on the European logarithmic microprocessor," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 702–715, Jul. 2000.
[38] F. Taylor, "An extended precision logarithmic number system," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-31, no. 1, pp. 232–234, Feb. 1983.
[39] Y. Uguen, L. Forget, and F. de Dinechin, "Evaluating the hardware cost of the posit number system," in *Proc. 29th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2019, pp. 106–113.
[40] S. Choi, J. Shin, and L.-S. Kim, "A deep neural network training architecture with inference-aware heterogeneous data-type," *IEEE Trans. Comput.*, vol. 71, no. 5, pp. 1216–1229, May 2022.
[41] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.*, vol. C-24, no. 12, pp. 1145–1155, Dec. 1975.
[42] S. Golomb, "Run-length encodings (corresp.)," *IEEE Trans. Inf. Theory*, vol. IT-12, no. 3, pp. 399–401, Jul. 1966.
[43] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," in *Handbook of Systemic Autoimmune Diseases*, vol. 1, no. 4. 2009.
[44] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2009, pp. 248–255.
[45] R. F. Rice, "Some practical universal noiseless coding techniques," Jet Propuls. Lab., Pasadena, CA, USA, Tech. Rep. 79-22, Mar. 1979.
[46] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *Proc. Int. Conf. Learn. Represent.*, 2019. [Online]. Available: https://openreview.net/forum?id=HJz6tiCqYm
[47] J. Choquette and W. Gandhi, "NVIDIA A100 GPU: Performance & innovation for GPU computing," in *Proc. IEEE Hot Chips 32 Symp. (HCS)*, Aug. 2020, pp. 1–43.



Yang Wang received the B.S. degree in electronic science and technology from Xidian University, Xi'an, China, in 2014, and the Ph.D. degree in microelectronics and solid state electronics from the Institute of Microelectronics, Chinese Academy of Sciences, in 2019. He is currently a Post-Doctoral Researcher with the School of Integrated Circuits, Tsinghua University. His research interests include VLSI DSP, deep learning, and neural network acceleration.

Dazheng Deng received the B.S. degree in microelectronic science and engineering from Tsinghua University, Beijing, China, in 2020, where he is currently pursuing the M.S. degree with the School of Integrated Circuits. His current research interests include deep learning, computer architecture, and VLSI design.

Leibo Liu (Senior Member, IEEE) received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 1999, and the Ph.D. degree from the Institute of Microelectronics, Tsinghua University, in 2004. He is currently a Professor with the School of Integrated Circuits, Tsinghua University. His research interests include reconfigurable computing, mobile computing, and VLSI DSP.

Shaojun Wei (Fellow, IEEE) was born in Beijing, China, in 1958. He received the Ph.D. degree from the Faculte Polytechnique de Mons, Belgium, in 1991. In 1995, he became a Professor with the Institute of Microelectronics, Tsinghua University. His main research interests include VLSI SoC design, EDA methodology, and communication ASIC design. He is a Senior Member of the Chinese Institute of Electronics (CIE).

Shouyi Yin (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2000, 2002, and 2005, respectively.

He has worked with Imperial College London, U.K., as a Research Associate. He is currently a Full Professor and the Vice Director of the School of Integrated Circuits, Tsinghua University. He has published more than 100 journal articles and more than 50 conference papers. His research interests include reconfigurable computing, AI processors, and high-level synthesis. He has served as a Technical Program Committee Member in the top VLSI and EDA conferences, such as A-SSCC, MICRO, DAC, ICCAD, and ASPDAC. He is an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, *ACM Transactions on Reconfigurable Technology and Systems* (TRETS), and *Integration, the VLSI Journal*.