# An Efficient Training Accelerator for Transformers With Hardware-Algorithm Co-Optimization

Haikuo Shao, Jinming Lu, Meiqi Wang, and Zhongfeng Wang, *Fellow, IEEE*

***Abstract*— Transformers have achieved significant success in deep learning, and training Transformers efficiently on resource-constrained platforms has been attracting continuous attention for domain adaptions and privacy concerns. However, deploying Transformers training on these platforms is still challenging due to its dynamic workloads, intensive computations, and massive memory accesses. To address these issues, we propose an Efficient Training Accelerator for TRansformers (TRETA) through a hardware-algorithm co-optimization strategy. First, a hardware-friendly mixed-precision training algorithm is presented based on a compact and efficient data format, which significantly reduces the computation and memory requirements. Second, a flexible and scalable architecture is proposed to achieve high utilization of computing resources when processing arbitrary irregular general matrix multiplication (GEMM) operations during training. These irregular GEMMs lead to severe under-utilization when simply mapped on traditional systolic architectures. Third, we develop training-oriented architectures for the crucial Softmax and layer normalization functions in Transformers, respectively. These area-efficient modules have unified and flexible microarchitectures to meet various computation requirements of different training phases. Finally, TRETA is implemented under Taiwan Semiconductor Manufacturing Company (TSMC) 28-nm technology and evaluated on multiple benchmarks. The experimental results show that our training framework achieves the same accuracy as the full precision baseline. Moreover, TRETA can achieve 14.71 tera operations per second (TOPS) and 3.31 TOPS/W in terms of throughput and energy efficiency, respectively. Compared with prior arts, the proposed design shows 1.4–24.5× speedup and 1.5–25.4× energy efficiency improvement.**

***Index Terms*— Algorithm-hardware codesign, general matrix multiplication (GEMM), nonlinear function, training accelerator, Transformer.**

## I. Introduction

In RECENT years, Transformer-based [1] deep neural networks (DNNs) have achieved impressive performance in various fields, such as natural language processing (NLP) [2], [3], [4], [5], computer vision (CV) [6], [7], [8], and speech processing [9], [10], [11]. Prevailing Transformer-based models are usually pretrained from large-scale unlabeled data, and the models can be fine-tuned on downstream tasks after pretraining, instead of training from scratch. These pretraining and fine-tuning processes of Transformers are typically deployed on high-end graphics processing units (GPUs) platform, which has powerful computing capacity and high energy supply. However, with the increasing application scope of Transformers, it is necessary to retrain models on the edge platform [12]. Because the data distribution in real application scenarios may be different from the training dataset, the model needs to be adjusted according to personal data. Uploading private data to servers for retraining will cause significant latency and privacy risks. Therefore, the efficient training of Transformers on edge devices is attracting continuous attention.

However, the model sizes of Transformers (e.g., BERT [2], ViT [6], RoBERTa [3], and GPTs [5], [13], [14]) are becoming increasingly massive and typically orders of magnitude larger than prior convolutional neural network (CNN) models. For example, the BERT-Large [2] contains 340M parameters, which is significantly larger than the classical CNN model ResNet-50 [15] with 25M parameters, and much larger Transformers [5], [14] with enormous parameters have been introduced in the past few years. The outstanding performance of Transformers comes with intensive computations and memory accesses. The tremendous resource requirements of Transformers training hinder its efficient deployment on edge platforms with limited computing and memory resources.

Previous studies mainly focused on the inference process and have tried various approaches to optimize the memory footprint, latency, and energy consumption of DNNs. Some model compression algorithms are used to reduce the computational burden, including pruning [16], quantization [17], knowledge distillation [18], and low-rank decomposition [19]. Meanwhile, many custom hardware accelerators [20], [21], [22] are designed for DNNs' deployment on resource-constrained edge platforms. However, these algorithm and hardware optimizations cannot be directly applied to solve the problems faced when training models on edge devices. Compared with inference, training tasks involve different computation patterns and memory requirements. For inference-only tasks, samples are propagated through the forward path to obtain predicted results. However, during the training process, all samples in the dataset are processed by the model in an iterative manner. Furthermore, each iteration

Manuscript received 13 April 2023; revised 5 July 2023; accepted 25 July 2023. Date of publication 14 September 2023; date of current version 24 October 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62174084 and in part by the National Key Research and Development Program of China under Grant 2022YFB4400604. *(Corresponding author: Zhongfeng Wang.)*

Haikuo Shao, Jinming Lu, and Zhongfeng Wang are with the School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China (e-mail: hkshao@smail.nju.edu.cn; jmlu@smail.nju.edu.cn; zfwang@nju.edu.cn).

Meiqi Wang is with the School of Integrated Circuits, Sun Yat-sen University, Shenzhen 518107, China (e-mail: wangmq53@mail.sysu.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2023.3305569.

Digital Object Identifier 10.1109/TVLSI.2023.3305569

involves more computation phases including forward propagation (FP), backward propagation (BP), weight gradient generation (WG), and weight update (WU). Therefore, training requires significantly more computing and memory resources than inference. Although there have been several works in recent years aimed at software and hardware optimizations of CNN training [23], [24], [25], [26], when these methods are directly transferred to Transformers, the model accuracy and deployment efficiency are unsatisfactory. The computation characteristics of sentence samples with changing lengths in NLP tasks are quite different from the regular image samples with fixed sizes processed by CNNs, which makes the previous hardware architectures designed for CNNs inefficient when dealing with Transformers. In conclusion, it is challenging to efficiently train Transformers under the condition of limited hardware resources. Some of the critical issues that reduce training accuracy and efficiency are listed below.

1) Low-bit (LB) data formats, such as 16-bit floating point (FP16) [23] and 8-bit integer (INT8) [27], have been adopted in CNN training. However, existing LB data formats and algorithms suffer from serious accuracy degradation when applied to Transformers. Therefore, a novel data format with high representation ability and an efficient training framework is needed to improve model accuracy.
2) General matrix multiplication (GEMM) accounts for the majority of workloads in Transformers training and is the primary target for acceleration. However, the sequence lengths of NLP samples are uncertain, and correspondingly, the dimensions of feature matrices change continuously. In addition, the matrices need to be transposed during training, so the shapes of operand matrices are extremely irregular, which makes GEMM operations quite inefficient when mapped on GPU, tensor processing unit (TPU) [28], and other traditional accelerators.
3) Softmax and layer normalization [1] are the fundamental nonlinear functions used in Transformer-based models, which play important roles in the convergence of training. The appropriate data precisions and efficient training-oriented hardware implementations of these nonlinear functions for ensuring sufficient training accuracy have not been much investigated.

In this work, we present an efficient training accelerator for Transformers, namely TRETA, to address the above issues. The main contributions are summarized as follows.

1) We present an efficient and hardware-friendly mixed precision training algorithm based on a novel and compact LB data format, named piecewise integer (PINT) [30]. PINT is applied in the representation and computation of the primary GEMM operations and FP16 is adopted for nonlinear functions. LB PINT can significantly reduce computing and memory requirements. Experimental results demonstrate that our training framework could achieve the same performance for various downstream tasks as the FP32 baseline.
2) We propose a training-oriented GEMM Core in TRETA with high flexibility and scalability to deal with arbitrary irregular GEMM operations of training. GEMM Core integrates well-organized systolic array (SA) units and flexible adder networks (FANs) for matrix multiplications (MMs) and partial sum reductions. SA units can support multiple dataflows and reduce data streaming latency based on a dedicated PINT processing element (PE) design. Compared with TPU-like SAs, our design can improve the mapping efficiency and hardware utilization of GEMMs by almost an order of magnitude.
3) A Softmax Core and a Layer Norm Core are specially designed to support the mixed-precision training framework. These modules are developed based on the FP16 arithmetic units in a deeply pipelined manner, which enables them to efficiently perform these nonlinear computations for all phases of training in unified architectures with minimal resources and low latency.
4) The proposed training accelerator is implemented under the Taiwan Semiconductor Manufacturing Company (TSMC) 28-nm process and evaluated on multiple benchmarks with various Transformer models. Based on the above optimization techniques, our design achieves up to 14.71 tera operations per second (TOPS) and 3.31 TOPS/W in terms of throughput and energy efficiency, respectively, which outperforms prior works and shows the feasibility of on-device training for Transformers.

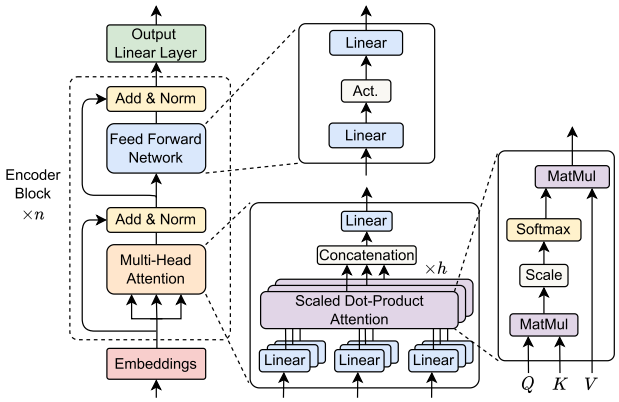



Fig. 1. BERT (left). MHA and FNN (middle). Self-attention (right).

The rest of this article is organized as follows. Section II gives an overview of Transformers training and related works. Section III describes the hardware-friendly training algorithm. The design of our hardware training accelerator is presented in Section IV, and the experimental results are reported in Section V. Finally, Section VI concludes this article.

## II. Background

### A. Model Architecture of Transformers

The overall architecture of a representative Transformer-based BERT model is shown in Fig. 1. BERT is a multilayer bidirectional encoder architecture based on the original Transformer implementation, including an input embedding layer, $n$ identical encoder blocks, and a task-specific output layer. Each encoder block is mainly composed of a multihead self-attention (MHA) module and a positionwise feed-forward

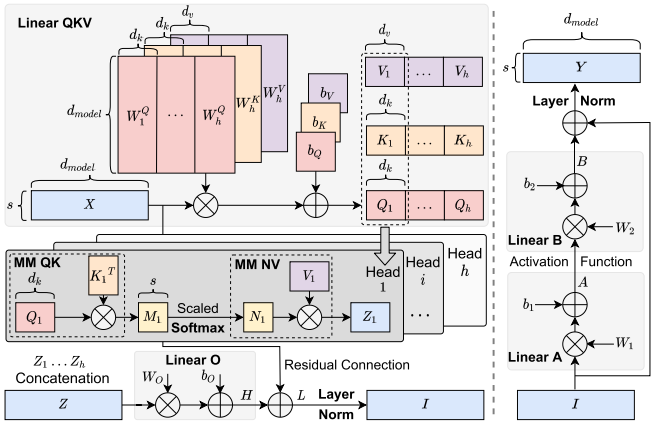



Fig. 2. Computations in the MHA ResBlock (left) and FFN ResBlock (right). The input matrix $X \in \mathbb{R}^{s\times d_{\text{model}}}$, output matrix $Y \in \mathbb{R}^{s\times d_{\text{model}}}$, and intermediate matrices $Z, H, L, I, B \in \mathbb{R}^{s\times d_{\text{model}}}$, $A \in \mathbb{R}^{s\times d_{\text{ff}}}$. The projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}}\times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}}\times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}}\times d_v}$, $W_O \in \mathbb{R}^{h\cdot d_v\times d_{\text{model}}}$, $W_1 \in \mathbb{R}^{d_{\text{model}}\times d_{\text{ff}}}$, and $W_2 \in \mathbb{R}^{d_{\text{ff}}\times d_{\text{model}}}$. MM represents matrix multiplication in self-attention.

TABLE I
SYMBOLS INVOLVED IN TRANSFORMER TRAINING

| Symbols | BERT-Base | Definitions |
|---|---|---|
| $n$ | 12 | number of encoder blocks |
| $h$ | 12 | number of heads in MHA |
| $s$ | - | sequence length |
| $d_k$ | 64 | dimension of query and key matrices |
| $d_v$ | 64 | dimension of value matrix |
| $d_{model}$ | 768 | number of input features |
| $d_{ff}$ | 3072 | dimension of the feedforward network |

network (FFN). The residual connection [15] is adopted in these two modules to build a deeper mode, followed by layer normalization (layer norm). As a kind of language understanding model, BERT uses learned embeddings to convert the input tokens to vectors of dimension $d_{\text{model}}$ (listed in Table I). The fine-tuning process of BERT on different tasks is implemented by incorporating the model with one additional task-specific output layer. Other Transformer decoder-based models (e.g., GPTs) share a similar architecture composed of these basic modules and have the same computational features as BERT.

The detailed computation processes inside MHA and FFN are displayed in Fig. 2. Transformers adopt a particular attention mechanism named scaled dot-product attention. Given the packed matrix representations of queries ($Q \in \mathbb{R}^{s\times d_k}$), keys ($K \in \mathbb{R}^{s\times d_k}$), and values ($V \in \mathbb{R}^{s\times d_v}$), the scaled dot-product attention results can be computed as follows:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{1}$$

The Softmax function is applied in a rowwise manner, and the dot-products of queries and keys are divided by $\sqrt{d_k}$ to alleviate the gradient vanishing problem. Transformers use multihead attention instead of simply performing a single attention function with keys, values, and queries of dimension $d_{\text{model}}$. The approach uses $h$ sets of different, learned linear projections to linearly project original $d_{\text{model}}$-dimensional queries, keys, and values to $d_k$, $d_k$, and $d_v$ dimensions, respectively. The output of a single head is computed with an attention function according to (1) for each set of projected $Q_i$, $K_i$, and $V_i$. Then the outputs of all heads are concatenated and projected back to a $d_{\text{model}}$-dimensional representation as follows:

$$\text{MHA}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O \tag{2}$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ and $Q = K = V = X$ in the Transformer encoder.

In addition to MHA sublayers, each block of an encoder contains a fully connected FFN, which consists of two linear transformations with an activation function in between. The computation of FFN can be summarized as follows:

$$\text{FFN}(X) = \text{Act}(I \times W_1 + b_1) \times W_2 + b_2 \tag{3}$$

where $W_1 \in \mathbb{R}^{d_{\text{model}}\times d_{\text{ff}}}$ and $W_2 \in \mathbb{R}^{d_{\text{ff}}\times d_{\text{model}}}$.

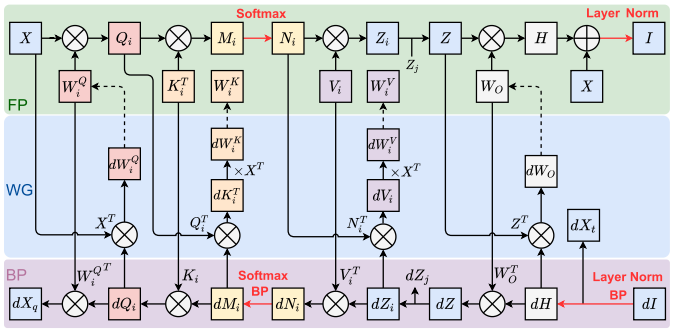



Fig. 3. Detailed training dataflow in MHA. The areas with green, purple, and blue colors in this figure represent the main computations in FP, BP, and WG phases, respectively, and the processes of using gradients to perform WU are represented by dotted lines.

### B. Training Process of Transformers

The training dataflow of the MHA module is shown in Fig. 3, where the gradients of parameter $X$ are expressed as $dX$. Generally, the training process can be divided into four phases, including FP, BP, WG, and WU. In the FP phase, input tokens are fed into the model to generate input matrices through the embedding layers, and then the encoder and output layer process the input matrix continuously. The loss function calculates the processing results to obtain the final loss. The BP phase and the WG phase begin with the loss and compute the gradients of feature values and weights with respect to loss according to the chain rule. In the WU phase, the weights are updated by the gradients and the optimizer. The process of linear projections of input and output matrices to different dimensions is completed by the linear layer. The calculation of the linear layer includes matrix multiplication between the input and weight, as well as optional bias. The computation of scaled dot-product attention includes the matrix multiplication of queries, keys, and values and the Softmax function. Therefore, we can divide the computations into three categories: matrix multiplication, Softmax, and layer normalization.

Given the input matrix $X$ and weight matrix $W$, the matrix multiplication at the FP phase can be simply expressed in the following equation. According to the matrix derivation principle and chain rule, the gradients $dX$ and $dW$ of $X$ and

$W$ with respect to loss can be obtained through the following equations, respectively,

$$Y = X \times W + b \tag{4}$$

$$dX = dY \times W^T \tag{5}$$

$$dW = X^T \times dY. \tag{6}$$

The computation $N = \text{Softmax}(M)$ in scaled dot-product attention is realized in a rowwise manner. Assuming a row vector $\boldsymbol{m} = (m_1, m_2, \ldots, m_s)$ in $M$, its corresponding result vector is $\boldsymbol{n} = (n_1, n_2, \ldots, n_s)$, where the element $n_i$ is obtained by the following equation:

$$n_i = \frac{e^{m_i}}{\sum_{j=1}^{s} e^{m_j}}, \quad \sum_{i=1}^{s} n_i = 1. \tag{7}$$

Through derivation of the Softmax function, we can get

$$\frac{\partial n_i}{\partial m_j} = \begin{cases} n_i - n_i n_i, & \text{if } i = j \\ 0 - n_i n_i, & \text{if } i \neq j. \end{cases} \tag{8}$$

Therefore, $(\partial \boldsymbol{n}/\partial \boldsymbol{m}) = \text{diag}(\boldsymbol{n}) - \boldsymbol{n}^T \cdot \boldsymbol{n}$, where $\text{diag}(\boldsymbol{n})$ represents the diagonal matrix constructed by $\boldsymbol{n}$. Finally, during the BP phase, $d\boldsymbol{m}$ can be obtained by the following equation:

$$d\boldsymbol{m} = d\boldsymbol{n} \cdot \frac{\partial \boldsymbol{n}}{\partial \boldsymbol{m}} = d\boldsymbol{n} \cdot (\text{diag}(\boldsymbol{n}) - \boldsymbol{n}^T \cdot \boldsymbol{n}). \tag{9}$$

Transformers use layer normalization to shift and scale the data. During the FP phase, suppose that the encoding vector corresponding to an input token is $\boldsymbol{x} = (x_1, x_2, \ldots, x_D)$, where $D$ represents the feature dimension, and the result vector obtained after layer norm is $\boldsymbol{y} = (y_1, y_2, \ldots, y_D)$, then the computation process of $\boldsymbol{y}$ is shown in the following equations, where $\epsilon$ is a small constant introduced to prevent the denominator from being zero, $\gamma$ and $\boldsymbol{\beta}$ are two learnable parameters:

$$\mu = \frac{1}{D}\sum_{i=1}^{D} x_i, \quad \nu = \frac{1}{D}\sum_{i=1}^{D} {x_i}^2, \ \sigma^2 = \nu - \mu^2 \tag{10}$$

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad y_i = \gamma_i \cdot \hat{x}_i + \beta_i. \tag{11}$$

The layer norm receives the gradient $d\boldsymbol{y} = (dy_1, dy_2, \ldots, dy_D)$ of $\boldsymbol{y}$ in the BP phase and computes $d\boldsymbol{x} = (dx_1, dx_2, \ldots, dx_D)$, $d\gamma$, and $d\boldsymbol{\beta}$ through the following equations:

$$d\beta_i = dy_i, \quad d\gamma_i = dy_i . \cdot \hat{x}_i. \tag{12}$$

$$dx_i = \frac{\gamma_i}{\sqrt{\sigma^2 + \epsilon}}\left(dy_i - \frac{\sum dy_j}{D} - \hat{x}_i \cdot \frac{\sum dy_j \cdot \hat{x}_j}{D}\right). \tag{13}$$

### C. Related Works

Many prior works have been proposed to train DNNs efficiently on resource-constrained platforms. Training with low-precision arithmetic has become a promising research field. Micikevicius et al. [23] introduced a methodology for training DNNs using half-precision floating-point (FP16) numbers. Other specific data formats, such as block floating point (BFP) [31] and brain floating point (BFLOAT) [32], have also been applied for efficient training. However, these formats require complicated floating-point computations and are not cost-effective for edge devices. Recently, some researchers [24], [27], [33] have proposed CNN training frameworks with INT8 arithmetic for efficient hardware implementation. However, these methods have the problem of accuracy degradation when transferred to large-scale Transformer-based models.

In addition, several hardware accelerators have also been developed for DNN training. Two generations of processors were presented in [25] and [34] for robust FP16 training. Lee et al. [26] proposed a learning accelerator that uses fine-grained mixed floating-point precision. In [35], a multicore chip was designed with aggressive architecture optimizations to achieve high training performance. SIGMA [36] also developed an architecture to handle sparse and irregular GEMMs. However, these works adopt floating-point arithmetic in the whole training process to maintain accuracy, which is inefficient for hardware implementation. Recently, Cambricon-Q [37] proposed a hybrid architecture to perform quantized training. A flexible training accelerator, named FlexBlock [38], was also designed to support multiple BFP precision.

## III. Hardware-Friendly Training Algorithm

This section presents an efficient and optimized training algorithm for Transformers with a novel and compact data format named PINT. Multiple Transformer-based models can be trained successfully using 8-bit PINT and half-precision floating-point (FP16) representations with negligible accuracy loss. By this means, the mixed-precision training algorithm's memory requirement and computation cost are significantly reduced. Combined with the frozen embedding layers, the training procedure is more hardware-friendly.

### A. Piecewise Integer

Quantization is a promising model compression method to reduce memory footprint and computation cost by representing parameters and activations with LB precision, for example, INT8 instead of 32-bit floating point (FP32). The uniform symmetric quantization scheme converts a continuous real number set $\boldsymbol{x}$ to $k$-bit integer value representations by

$$\boldsymbol{q} = \text{Clamp}\left(\text{Round}\left(\frac{\boldsymbol{x}}{f}\right), -2^{k-1}, 2^{k-1} - 1\right) \tag{14}$$

where $f = \max(|\boldsymbol{x}|)/2^{k-1}$ indicates the scaling factor used to normalize the data, and $\boldsymbol{q}$ is the integer representations.

Applying quantization techniques to Transformers training faces more challenges than inference-only tasks. Not only weights and activations, but also gradients need to be quantized to achieve full quantization training. The quantization of gradients is more challenging than that of weights and activations because it requires a higher precision representation to ensure training convergence and a more extensive dynamic range to cover data distribution. The gradients close to zero during training form an asymmetric distribution around zero and require a small quantization interval to reduce the quantization noise. The gradients away from zero need larger quantization intervals to cover their distribution with the limited

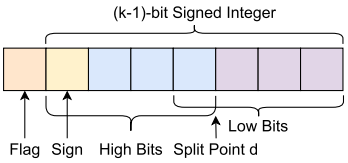



Fig. 4. Basic structure of a PINT(8, 3) number.

width of the LB data format. Traditional integer data formats (e.g., INT8) suffer from significant accuracy degradation in Transformers training due to their limited representation range and static quantization interval.

We introduce a compact but powerful data format, PINT [30], in the training process of Transformers, instead of directly applying INT8 quantization. Compared with INT8, PINT has a broader representation range and dynamic decimal accuracy. The PINT ($k$, $d$) data format is defined by two parameters, including the bit width $k$ and the split point $d$. The data structure of an (8, 3) PINT number is illustrated in Fig. 4, including one flag bit and one $(k-1)$-bit signed integer. The split point $d$ indicates how to divide a $k$-bit PINT into two overlapped sets, high-bits (HBs), and LBs. Regardless of the tensor-shared scaling factor, the actual integer value $x_p$ of the PINT is obtained by

$$x_p = \begin{cases} \text{Low bits signed integer}, & \text{flag} = 0 \text{ and } \odot(\text{HB}) = 1 \\ \text{Signed integer} \cdot 2^d, & \text{flag} = 1 \\ \text{Signed integer} \cdot 2^{k-2}, & \text{otherwise} \end{cases} \tag{15}$$

where $\odot$ represents XNOR operation, and $\odot(\text{HB}) = 1$ means that HBs are all "0" or all "1." Only LBs are significant in this case. The flag bit and $d$ divide the coding space into three parts, each of which corresponds to a separate scaling factor and has a different quantization interval. From (15), it can be further deduced that $k$-bit PINT could cover a dynamic range of $[-2^{2(k-2)}, 2^{2(k-2)} - 1]$, which is the same as that of the $(2k-3)$-bit INT.

### B. Mixed-Precision Training Algorithm

Based on the compact and efficient PINT data format introduced above, we designed a hardware-oriented optimized mixed-precision training algorithm for Transformers. As introduced in Section II-B, the computation patterns involved in training mainly include three types: matrix multiplication, Softmax, and layer normalization. Matrix multiplication is mainly used in the linear layer and attention mechanism. It is the main computation pattern during training, and the linear layer accounts for most of the computation cost. In our mixed-precision training algorithm, the computation of each training stage related to matrix multiplication is completed using the PINT data format. As for the nonlinear functions, including Softmax and layer normalization, considering that their accuracy is critical to the convergence of training, we use the FP16 format with higher precision for nonlinear computations in the mixed-precision training algorithm. Since the computation cost of nonlinear functions only accounts for a small part of training, the hardware overhead is not expensive while ensuring training accuracy.

Taking the FFN block as an example, in the mixed-precision training algorithm, the detailed dataflow and data format used in different training phases are shown in Fig. 5(a). In the FP, BP, and WG phases, the matrix multiplications involved in the two linear layers of the FFN block are all completed with the PINT data format, while the layer normalization function uses FP16 format data for computing. The inputs of matrix multiplication are in the PINT format, and the output results are in the INT32 format. Before transferring to the subsequent module, they need to be converted to the PINT or FP16 format. The input and output of the layer normalization module are FP16 data, and its FP16 results also need to be converted to the INT32 or PINT format to complete the subsequent computations.

To make the training algorithm more hardware-friendly, we have further optimized the training process. Some studies [39] found that during Transformers training, the majority of model information is learned by MHA and FFN blocks, while embeddings provide complementary but nonessential information. We conducted comparative experiments and found that freezing the parameters of embedding layers has negligible influence on the final model accuracy. For example, when BERT is fine-tuned on the SST-2 task of general language understanding evaluation (GLUE) [2] dataset, the loss variations of the model with frozen and unfrozen embedding layers are shown in Fig. 5(b). The frozen embedding layer has a negligible influence on the training convergence. Moreover, the parameter amounts of embedding layers are massive and the parameter updates are sparse, which is not suitable for hardware implementation. Therefore, we freeze the embedding layers in the training algorithm, so that the calculations of embedding layers can be completed in the host and reused in different epochs during the training process.

## IV. Efficient Hardware Architecture for Training

### A. Overall Architecture

In this section, we propose a TRETA, based on the presented hardware-friendly training algorithm. The overall architecture of our training accelerator is shown in Fig. 6. All involved data including weight parameters, input features, intermediate values, and gradients can be stored in off-chip DRAM. The direct memory access (DMA) engine is responsible for efficient data transfer between DRAM and on-chip buffers. Input, weight, output, and auxiliary memories are used to cache data such as input, weight, gradient, error, partial sum, and output. During different training phases, all modules in the accelerator cooperate orderly under the coordination of the global controller.

The **GEMM Core** is developed to perform arbitrary irregular matrix multiplications during training. GEMM Core consists of many SA units, which are connected to form a high-level 2-D architecture. Each SA unit is designed based on dedicated PINT PEs with scalable shape ($R \times C$) and flexible dataflow. The microarchitecture of PINT PE is illustrated in Section IV-B. To exploit the parallelism and data

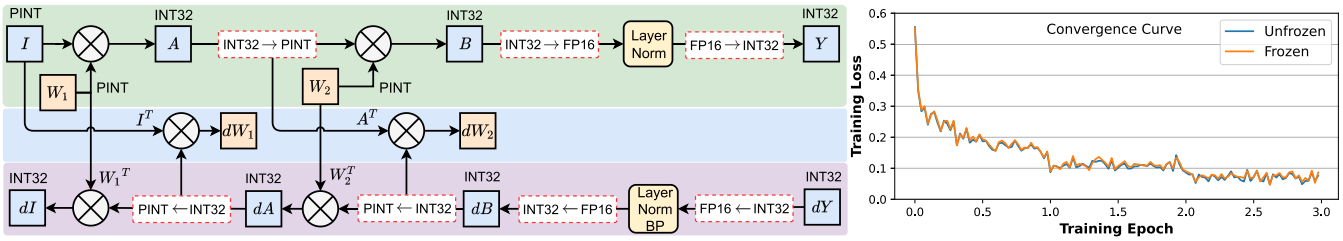



Fig. 5. (a) Detailed dataflow and data format configurations of the mixed-precision training algorithm in the FFN block. (b) Fine-tuning loss variations of BERT on SST-2 task with frozen and unfrozen embedding layers.

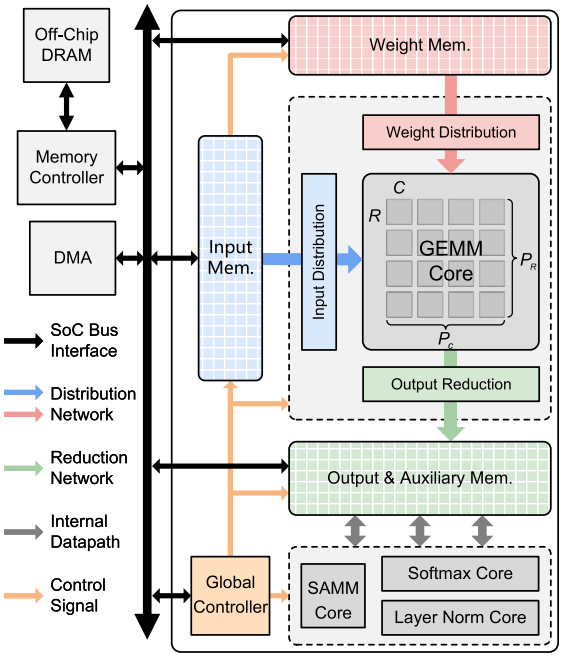



Fig. 6. Overall architecture of our training accelerator.

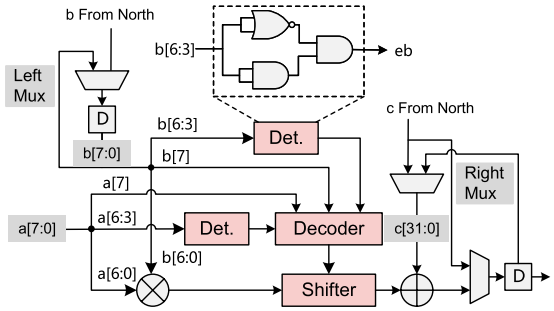



Fig. 7. Microarchitecture of a PINT(8, 3) PE.

reuse of matrix multiplication, GEMM Core is arranged in a hierarchical structure with a scalable size $P_R \times P_C \times R \times C$. In collaboration with data distribution and reduction networks, GEMM Core efficiently transfers data between on-chip buffers. Another self-attention matrix multiplication (SAMM) Core with 16 × 16 PE units is also integrated for the small amount of matrix multiplications in self-attention.

The **Softmax Core** and **Layer Norm Core** are composed of many parallel basic nonlinear function units. In our mixed-precision training algorithm, the FP16 data format is adopted for Softmax and layer normalization functions. Accordingly, we propose efficient and unified hardware modules based on elementary FP16 arithmetic units, which can effectively support the computations of these functions in all training stages.

### B. Dedicated PINT Processing Element

The dedicated PE is proposed to support the accurate computations of PINT data. The microarchitecture of a PE that supports PINT(8, 3) multiplication is shown in Fig. 7. A PINT multiplier is composed of a 7-bit signed integer multiplier, a data shifter, two detectors (Det. in Fig. 7), and a decoder. The detector determines whether HBs are all "1" or all "0," which includes an AND gate, a NOR gate, and an OR gate. The decoder takes the outputs of two detectors and flag bits as input and determines how to shift the product of the 7-bit multiplier. Then, the shifted product is accumulated by a 32-bit integer adder. Moreover, the dedicated PINT PE is designed as a reconfigurable structure to support multiple dataflows and applied in the architecture design introduced in Section IV-C. This flexible PE could reconfigure its datapath for different computing patterns through internal multiplexers (Mux). The left multiplexer in Fig. 7 can decide whether the multiplier's input data $b$ is from outside or remains unchanged. PE can work under weight stationary (WS) dataflow when $b$ remains unchanged. The multiplexer on the right can determine whether the input data $c$ of the accumulator is from another PE or its previous output, and PE changes to the output stationary (OS) dataflow when choosing the latter.

### C. Efficient GEMM Core Design

*1) Preliminary Experiments and Design Motivations:* GEMM operations are the primary workloads in Transformers training, and the computations of linear layers and attention mechanisms in different phases can be mapped to GEMMs. Therefore, the main target of our hardware optimizations is focused on GEMM computations. Prior training accelerators [34], [35], [37] usually choose SA as the basic architecture for accelerating GEMMs. A typical SA is built in a 2-D architecture with many interconnecting PEs, and there are three common dataflows for mapping GEMMs onto the array: WS, OS, and input stationary (IS). The systolic structure can significantly save SRAM bandwidth requirements and exploit data reuse opportunities provided by GEMMs.

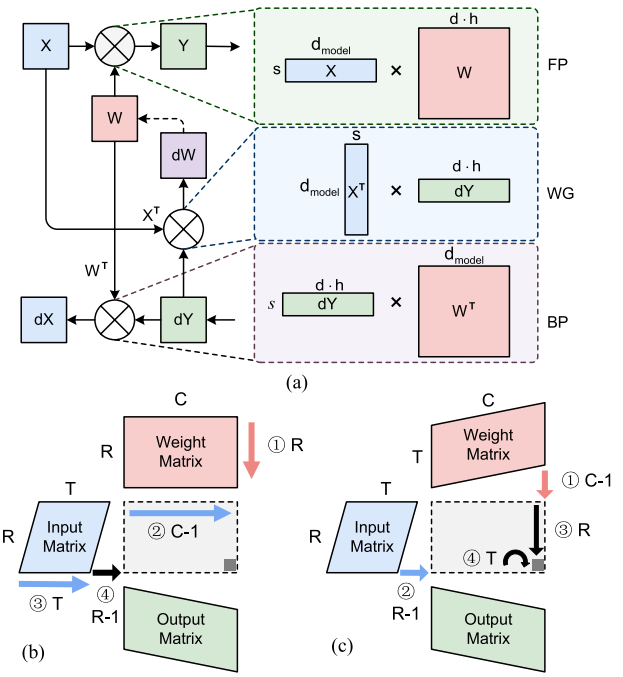



Fig. 8. (a) Various GEMM workloads of the linear layer during Transformer training, and runtime model of GEMMs under (b) WS and (c) OS dataflows. $R$ and $C$ are spatial dimensions and $T$ is the temporal dimension, and the runtime of GEMM consists of different computing periods ($t = ①+②+③+④$).

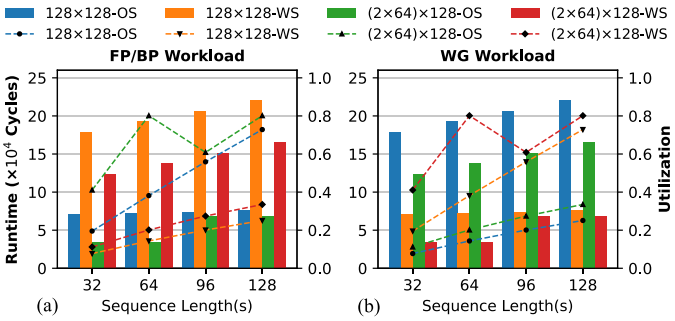



Fig. 9. Runtime (histogram) and efficiency comparisons of (a) FP/BP and (b) WG workloads on SAs with different configurations. (2 × 64) × 128-WS represents two parallel WS arrays with 64 rows and 128 columns.

However, the dynamic computation characteristics, especially the variation of sequence length dimensions, make the GEMM operations in Transformers training extremely inefficient when mapping on traditional SAs. Specifically, GEMMs of the representative linear $QKV$ layers in FP, BP, and WG phases can be expressed as matrix multiplications in Fig. 8(a). The dimensions of weight matrices are $d_{\text{model}} \times d \cdot h$ ($d = d_k$ or $d_v$), these parameters are fixed in a specific Transformer model, while the dimensions of input matrix ($X$) and error matrix ($dY$) are $s \times d_{\text{model}}$ and $s \times d \cdot h$, where $s$ refers to the sequence length, which is an uncertain variable (between 1 and 128 in the GLUE dataset, for example). The variation of $s$ makes GEMMs have irregular dimensions during training, which is quite different from previous CNN computations. The dimensions of feature maps and parameter matrices are identical for all image samples in the corresponding CV dataset during the CNN training process. Therefore, the computations of CNN training are invariant and preknown, which makes it achieve good efficiency on well-designed and fixed hardware architectures. However, the GEMM operations in Transformers training are irregular and variable, and therefore inefficient when mapped on traditional CNNs' accelerating architectures represented by SAs, which significantly decreases the training performance. It is necessary to compare and analyze the runtime and efficiency of various GEMM workloads on SAs with different features and further optimize the architectures for excellent performance.

We adopted the analytical model presented in SCALE-SIM [40] to simulate the runtime of GEMMs when mapping on SA. As shown in Fig. 8(b) and (c), in the WS dataflow, the dimensions of the input matrix and weight matrix are $R \times T$ and $R \times C$, respectively, while in the OS dataflow, the dimensions change to $R \times T$ and $T \times C$. The IS dataflow is similar to WS, with the difference of the stationary input matrix. Assuming that GEMM operations are exactly mapped to an SA with a size of $R \times C$ (corresponding to the number of rows and columns, respectively), the runtimes of both dataflows can be summarized in the following equation, taking into account the time of input distribution, partial-sum accumulation, and result transmit:

$$t = 2R + C + T - 2. \tag{16}$$

When the dimensions of the operand matrices do not match the size of the array, for example, the dimensions of the input matrix and weight matrix are $D_M \times D_K$ and $D_K \times D_N$, respectively, it is necessary to tile the matrices into blocks, and then the total runtime can be estimated by the following equation:

$$t_{\text{total}} = (2R + C + T - 2) \times \lceil D_M/R \rceil \times \lceil D_N/C \rceil. \tag{17}$$

The amount of required multiply-accumulate operations for GEMM is $D_M \times D_N \times D_K$, and the overall utilization of hardware resources can be obtained through the following equation:

$$\eta = \frac{D_M D_N D_K}{t_{\text{total}} \times R \times C}. \tag{18}$$

From the above analysis, it can be summarized that for a given workload, the number of PEs, the values of $R$ and $C$, and the choice of dataflow determine the GEMM runtime together and that less runtime means higher PE utilization.

The GEMM's runtime and efficiency comparisons of workloads with different sequence lengths on SAs with various sizes and dataflows are evaluated and listed in Fig. 9. The workloads include GEMMs of all linear layers in an MHA-FFN stack for different phases. Experimental results provide us insights into architecture design for achieving better training performance. GEMMs in the FP and BP phases are more suitable for the SA with the OS dataflow. In contrast, the WS dataflow is better for the WG phase. The appropriate choice of dataflow can drastically reduce the runtime of GEMMs. Moreover, allocating multiple arrays (known as scale-out) is a feasible method to achieve higher computing efficiency than creating a monolithic array (scale-up). Traditional SAs lack flexibility and face the problem of under-utilization in Transformer training due to their inherent features: fixed architectures and stationary interconnects.

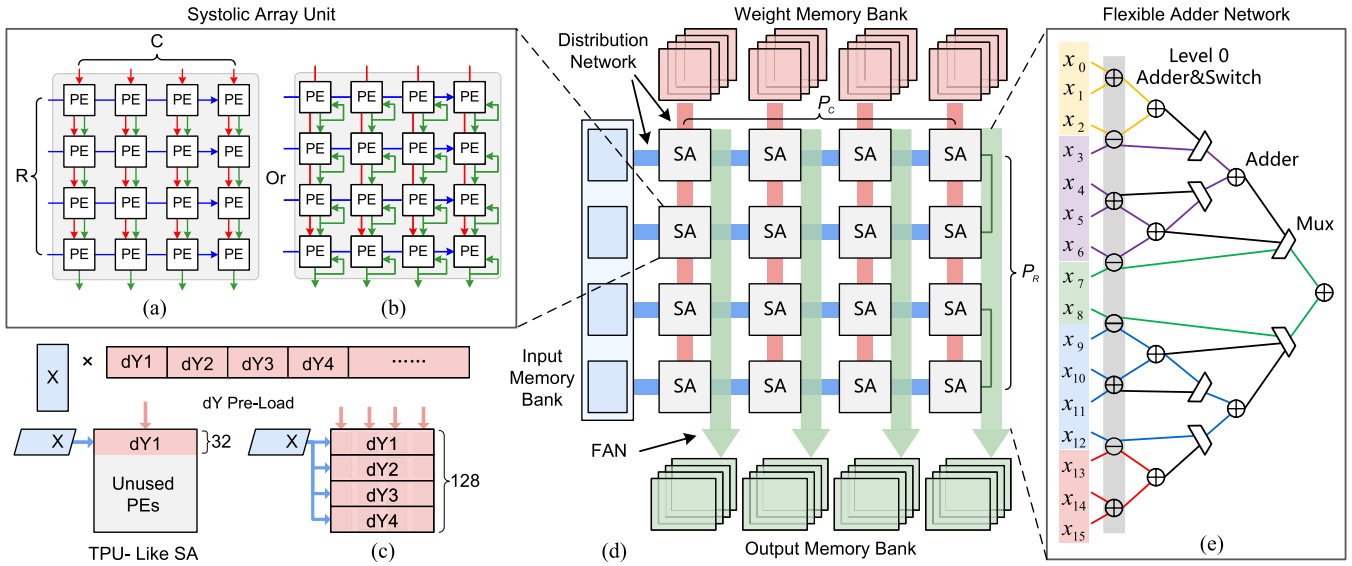



Fig. 10. SA unit under (a) WS and (b) OS dataflow. (c) GEMM workload mapping comparisons in the WG stage when $s = 32$. (d) Structure of GEMM core. (e) FAN architecture.

*2) Design of Efficient GEMM Core:* To improve the utilization of computing resources, reduce runtime, and promote overall training performance, we propose a flexible and scalable hardware architecture by allocating substantial SA units in a higher-level 2-D structure. The basic SA unit is built on the flexible PINT PE introduced in Section IV-B, which is designed to support both WS and OS dataflows through its reconfigurable structure.

The detailed dataflows of the SA unit under WS and OS patterns after optimizations are illustrated in Fig. 10(a) and (b), respectively. In WS dataflow, the data from weight memory first flow through registers in the PE among each column and remains unchanged after being preloaded to the corresponding PE. Then, the data from input memory are passed to PEs in the same row. Since the rows and columns of the SA unit are set fairly small (no more than 16), the input data of each cycle can be simultaneously broadcast to all PEs in the same row without streaming between registers. The broadcasting structure has the advantage of reducing the data input latency [$R$ in (16)] and eliminating the consumption of the input registers. Partial sums are still passed between output registers in the same column and out from the bottom side of the array. Under the OS dataflow, input data and weight data are concurrently broadcast to the array. The data between different rows and columns is not delayed, and a partial sum is accumulated within PEs. The results are transmitted from the bottom side of the array by column through output registers.

The SA units are arranged into a higher-level 2-D gird ($P_R \times P_C$) to build the GEMM Core, as shown in Fig. 10(d). The input and weight distribution networks are responsible for transferring the data in the memory banks to corresponding SA units. Inside the GEMM Core, SA units in the same row share data in the same input memory bank, while the weight data of different rows in the same column comes from different weight memory banks. The data distribution network can dynamically maximize the use of PE resources while taking advantage of GEMMs' parallelism. In the WS dataflow, output data of different SA rows need to be further reduced, and which rows of output data should be summed is uncertain and needs adjustment according to the data distribution strategy. Therefore, the traditional binary adder trees are no longer applicable. To address this issue, we implement a FAN that can perform arbitrary combinations of input data based on a predefined control signal, as shown in Fig. 10(e). The adders at the first level in the network are with the switch function, which can decide whether to export the sum of two input data or pass them to the next level unchanged according to the control signal. Different levels of adders are pipelined to improve throughput.

This fine-grained design not only supports flexible dataflows, but also greatly improves the mapping efficiency of GEMMs. Taking the workload in the WG stage during the training process as an example, it is assumed that the $s$ dimensions of the operation matrices equal 32 and the WS dataflow is adopted by SA, as shown in Fig. 10(c). In the traditional $128 \times 128$ TPU-like SA, gradient $dY$ can only be loaded into PEs in the first 32 rows of the array. The following data cannot be loaded into the remaining PEs as the subsequent matrix computations are irrelevant. Our GEMM core using the same computing resources ($R = C = 8$ and $P_R = P_C = 16$) can map the data to all PEs because the SA units in different rows can compute separately and then exploit FAN for accurate reduction. In addition, the smaller size of the SA unit can reduce the cycles for data preloading meanwhile the broadcast structure can decrease the data streaming time.

### D. Design of Softmax Core

We propose an efficient module named Softmax Core that contains many parallel Softmax Units, which can be used for Softmax computations in both FP and BP phases through flexible and reconfigurable structures. The basic architecture of the Softmax Unit is shown in Fig. 11(a). The upper and lower parts correspond to the FP and BP paths, respectively, and the arithmetic units in both paths support the FP16 data format, which is consistent with our training algorithm. The

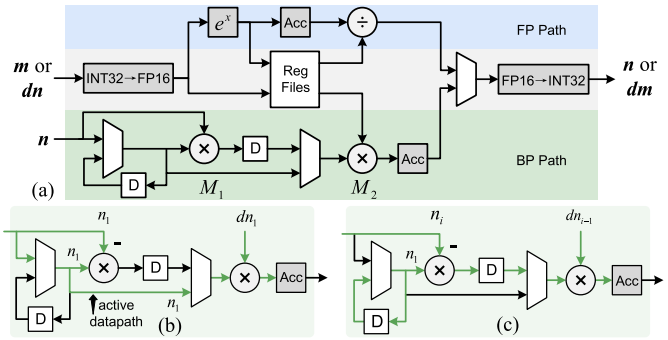



Fig. 11. (a) Microarchitecture of a single Softmax unit and its BP dataflow at the (b) first clock cycle and (c) $i$th clock cycle during training.

middle part includes two data format conversion units on input and output ports for conversions between INT32 and FP16 numbers, as well as a small piece of local Register Files for caching intermediate results. These relatively expensive modules in the middle are shared by FP and BP paths and work in a time-multiplexed manner for different training phases to achieve better efficiency. All modules in the Softmax Unit have been deeply pipelined to boost throughput.

In the FP phase, Softmax turns a vector $\boldsymbol{m}$ of $s$ real values into a vector $\boldsymbol{n}$ of $s$ real values that sum to 1 by the following equation:

$$n_i = \frac{e^{m_i}}{\sum_{j=1}^{s} e^{m_j}}, \quad \sum_{i=1}^{s} n_i = 1. \tag{19}$$

The FP path contains an exponential function unit, an accumulator (Acc), and a divider, which can efficiently perform corresponding computations. The result $\boldsymbol{n}$ of the FP phase is saved for the BP computation.

During the BP phase, Softmax computes gradient $d\boldsymbol{m}$ of vector $\boldsymbol{m}$ according to gradient vector $d\boldsymbol{n}$ through the following equation:

$$d\boldsymbol{m} = d\boldsymbol{n} \cdot \frac{\partial \boldsymbol{n}}{\partial \boldsymbol{m}} = d\boldsymbol{n} \cdot (\mathrm{diag}(\boldsymbol{n}) - \boldsymbol{n}^T \cdot \boldsymbol{n}). \tag{20}$$

Taking the first element $dm_1$ in $d\boldsymbol{m}$ as an example, the computation of its value can be expanded as follows:

$$dm_1 = dn_1(n_1 - {n_1}^2) + dn_2(-n_2 n_1) + \cdots + dn_s(-n_s n_1). \tag{21}$$

The BP path is flexible and efficient to support the $d\boldsymbol{m}$ computation at different periods accurately through its reconfigurable datapath defined by multiplexers. The elements of $dn$ are first converted to FP16 format by conversion unit and then stored in local Register Files. The detailed computation process of $dm_1$ is illustrated in Fig. 11(b) and (c). At the first clock cycle, the BP datapath is shown in Fig. 11(b). The inputs of $M_1$ are $n_1$ and $-n_1$, while the inputs of $M_2$ are $dn_1$ and $n_1$. The accumulator takes the output of $M_2$, $n_1 \cdot dn_1$, as input. At cycle $i (i > 1)$, the dataflow is changed to Fig. 11(c). Data $n_1$ and $-n_i$ from the auxiliary memory are sent to $M_1$, and the inputs of $M_2$ are $dn_{i-1}$ and $-n_1 \cdot n_{i-1}$, which is the previous clock output of $M_1$. The accumulator receives $dn_{i-1}(-n_1 n_{i-1})$ as input. The output of the accumulator is $dn_1 \cdot n_1 - \sum_{i=2}^{s+1} dn_{i-1} \cdot n_1 n_{i-1} = dn_1 \cdot n_1 + dn_1(-{n_1}^2) + dn_2(-n_1 n_2) + \cdots + dn_s(-n_1 n_s) = dm_1$ when the calculation reaches the $s+1$ clock cycles. In other words, $dm_1$ can be obtained after $s+1$ cycles. The computation process of the remaining elements in $d\boldsymbol{m}$ is similar to that of $dm_1$.

**Algorithm 1** Layer Normalization Training Process

**Forward Propagation:**

**Input:** $\boldsymbol{x} = (x_1, x_2, \ldots, x_D)$, $\boldsymbol{\gamma} \in \mathbb{R}^D$, $\boldsymbol{\beta} \in \mathbb{R}^D$;

**Output:** $\boldsymbol{y} = (y_1, y_2, \ldots, y_D)$;

$\mu \leftarrow \frac{1}{D}\sum_{i=1}^{D} x_i$

$\nu \leftarrow \frac{1}{D}\sum_{i=1}^{D} {x_i}^2$

$\sigma^2 \leftarrow \nu - \mu^2$

$\lambda \leftarrow 1/\sqrt{\sigma^2 + \epsilon}$

$\hat{x}_i \leftarrow (x_i - \mu) \times \lambda, \quad 1 \leqslant i \leqslant D$

$y_i \leftarrow \hat{x}_i \times \gamma_i + \beta_i, \quad 1 \leqslant i \leqslant D$

**Backward Propagation:**

**Input:** $\mathrm{d}\boldsymbol{y} = (\mathrm{d}y_1, \mathrm{d}y_2, \ldots, \mathrm{d}y_D)$, $\boldsymbol{\gamma}$, $\hat{\boldsymbol{x}}$;

**Output:** $\mathrm{d}\boldsymbol{x} = (\mathrm{d}x_1, \mathrm{d}x_2, \ldots, \mathrm{d}x_D)$;

$sum_1 \leftarrow \sum_{i=1}^{D} \mathrm{d}y_i$

$sum_2 \leftarrow \sum_{i=1}^{D} \mathrm{d}y_i \times \hat{x}_i$

$mean_1 \leftarrow \frac{1}{D} \times sum_1$

$mean_2 \leftarrow \frac{1}{D} \times sum_1$

$\gamma_i \lambda \leftarrow \gamma_i \times \lambda$

$\mathrm{d}x_i \leftarrow \gamma_i \lambda \times (\mathrm{d}y_i - mean_1 - \hat{x}_i \times mean_2), \quad 1 \leqslant i \leqslant D$

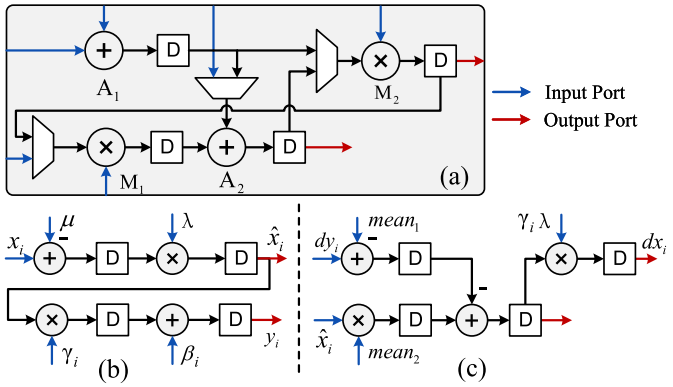



Fig. 12. (a) Microarchitecture of the layer norm unit and its detailed dataflow in the (b) FP and (c) BP phases during training.

## E. Design of the Layer Norm Core

We present a high-throughput module, namely Layer Norm Core, which contains plenty of efficient and parallel layer norm units. The microarchitecture of the layer norm unit is shown in Fig. 12(a), which consists of two adders, two multipliers, and three multiplexers, all of which support the FP16 data format. The adders and multipliers are responsible for linear transformation. During different training phases, the layer norm unit can reconfigure its internal structure by the multiplexers, using a unified architecture to flexibly support various computations of normalization.

Layer normalization in Transformers operates on the embedding vector of a single token in a feature matrix. In the FP phase, assuming the input vector is $\boldsymbol{x} = (x_1, x_2, \ldots, x_D)$, the normalized result is expressed as $\boldsymbol{y} = (y_1, y_2, \ldots, y_D)$, where $D$ represents the feature numbers ($d_{\text{model}}$). While in the BP phase, the input and output of the layer norm are gradients $d\boldsymbol{y}$ and $d\boldsymbol{x}$, respectively. We have introduced the basic principle

of the layer norm in Section II, and its detailed computation process is explained in Algorithm 1.

Before the linear transformations (lines 5, 6, and 12 in Algorithm 1) of the layer norm, statistics ($\mu$, $\nu$, $\text{mean}_1$, and $\text{mean}_2$) computation (lines 1, 2, and 7–10) and local variables ($\sigma^2$, $\lambda$, and $\gamma_i\lambda$) computation (lines 3, 4, and 11) are required for the input data. The input data of layer normalization comes from its previous linear layer, which is completed by GEMM Core. When the results of the prior linear layer are streamed out from GEMM Core, their statistics computation can be accomplished synchronously. The mean value $\mu$ of the FP phase can be obtained by an accumulator and a constant divider, and the computation of the mean square value $\nu$ requires an additional multiplier. The computation process of statistical values $\text{mean}_1$ and $\text{mean}_2$ in the BP phase are similar to $\mu$ and $\nu$, except that $\text{mean}_2$ needs to calculate the product of $dy_i$ and $\hat{x}_i$. Note that $\hat{x}_i$ is the intermediate result saved in the FP phase. In both the FP and BP phases, these local variables need to be computed only once for one normalization process. The square root, reciprocal, and multiplication of $\lambda$ and $\gamma_i\lambda$ can all be accomplished by basic FP16 arithmetic units.

Based on precomputed statistics and local variables, the linear transformation processes (lines 5, 6, and 12) can be performed continuously in the layer norm unit. During the FP phase, the dataflow of the layer norm unit is shown in Fig. 12(b). In this pattern, the input data are mean value $\mu$, local variable $\lambda$, and model parameters $\gamma_i$ and $\beta_i$. The element $x_i$ of the input vector is streamed in, and the normalized result $y_i$ and intermediate value $\hat{x}_i$ are streamed output in turn. The value $\hat{x}_i$ needs to be saved for BP computation. In the BP phase, the internal connection of the layer norm unit is transformed into Fig. 12(c). The input values are changed to the statistics $\text{mean}_1$, $\text{mean}_2$, and the local variable $\gamma_i\lambda$. The gradient $dy_i$ and intermediate value $\hat{x}_i$ are sent into the module, and the corresponding gradient $dx_i$ is streamed out from the upper port.

## V. Evaluation

### A. Experimental Setup

To demonstrate the effectiveness of our hardware-friendly mixed-precision training algorithm, we fine-tune the BERT and ALBERT [4] (a lightweight BERT variant) on the GLUE [2] benchmark with various downstream tasks. The GLUE benchmark is a collection of diverse natural language understanding tasks. We fine-tune GPT-2 [5] (117M parameters) on multiple language modeling tasks and report the perplexities, including Penn Treebank (PTB) [41], WikiText [42], and Children's Book Test (CBT) [43]. We also fine-tune ViT [6] on several image classification tasks to demonstrate the applicability of our training algorithm on Vision Transformer, including CIFAR-10 [44], CIFAR-100 [44], and ImageNet [45]. We evaluate the results of the FP32 baseline and PINT(8, 3) + FP16 quantization scheme and apply the traditional INT8 + FP16 quantization simultaneously for comparison. The frozen embedding method is adopted in both quantization schemes. We use the PyTorch framework to simulate the training processes of different data formats on NVIDIA RTX 3090 GPUs. The fine-tuning batch size is 8 for GPT-2 and 32 for BERT/ALBERT, and ViT is fine-tuned with a batch size of 256. We select the same optimal learning rate and optimizer for different training schemes and fine-tune the pretrained models for 3 epochs over the downstream tasks.

Our training accelerator is described in RTL, synthesized with Synopsys Design Compiler (DC), and placed and routed by Synopsys IC Compiler under the TSMC 28-nm technology. The accelerator operates at 500-MHz frequency, and the GEMM Core is configured as $8 \times 8 \times 16 \times 16$. Softmax Core and Layer Norm Core integrate 32 and 128 corresponding nonlinear function units, respectively. The sizes of input, weight, and output memories are 128, 256, and 512 KB, respectively. The input and output memory are simple dual-port SRAM, and the weight memory is single-port SRAM with a ping-pong buffer manner. We evaluate the training accelerator with BERT on each task of the GLUE benchmark. The detailed configurations of training are consistent with the algorithm. The switching activity interchange format (SAIF) is generated after function simulation and then used to estimate power consumption by Synopsys PrimeTime-PX. We also developed a TPU-like SA architecture as the baseline to demonstrate the advantages of our design.

### B. Results and Analysis

*1) Model Accuracy:* The experimental results and comparisons between different training schemes are summarized in Tables II–IV. Compared with the FP32 baseline, PINT(8, 3) + FP16 training can achieve equivalent performance for various downstream tasks. The accuracy degradations of fine-tuned BERT and ALBERT using PINT are less than 1% for most tasks, and the results are even better on several tasks, such as QNLI and SST-2. The perplexity of GPT-2 fine-tuned with PINT on various language modeling tasks increases by less than 1, which implies that the accuracy is almost unaffected. The accuracy degradations of fine-tuned ViT using PINT are less than 0.6% on different tasks compared to the FP32 baseline. Under the same experimental configurations, the performance of the INT8 + FP16 training in each task drops dramatically. The results demonstrate that the PINT data format, which has a broader numerical range and dynamic resolution than the common INT format with the same bit width, can significantly improve the accuracy of the fine-tuned models.

*2) Performance:* The irregular GEMM operations during the Transformer training process make the traditional TPU-like SA inefficient. Fig. 13(a) analyzes the influence of sequence length ($s$) on the PE utilization of $128 \times 128$ TPUs and $8 \times 8 \times 16 \times 16$ GEMM Core. We summarized the GEMM computations involved in all training phases of a single sample and deployed them on our GEMM Core and TPUs with different dataflows. The runtime and hardware utilization are evaluated using SCALE-SIM [40] analytical model. When the sequence length ($s$) is short (<40), the utilization of TPUs is extremely low, less than 20%. The efficiency of TPUs increases along with sequence length, but the best utilization is still less than 50%. The utilization of TPU with the OS dataflow is better, due to the GEMM computations in both FP

TABLE II
BERT AND ALBERT FINE-TUNING RESULTS AND COMPARISONS BETWEEN DIFFERENT TRAINING SCHEMES

| Model | Precision | MNLI-m | MNLI-mm | QQP | QNLI | SST-2 | CoLA | STS-B | MRPC | RTE | Avg.† |
|---|---|---|---|---|---|---|---|---|---|---|---|
| BERT-Base | FP32 Baseline | 84.12 | 84.28 | 87.93 | 91.45 | 92.20 | 57.54 | 88.34 | 84.80 | 64.98 | 81.74 |
| | PINT(8,3)+FP16 | 84.61 | 84.57 | 87.73 | 91.52 | 92.66 | 57.26 | 88.16 | 85.53 | 63.53 | 81.73 |
| | Difference | +0.49 | +0.29 | -0.20 | +0.07 | +0.44 | -0.28 | -0.18 | +0.73 | -1.45 | -0.01 |
| | INT8+FP16 | 48.81 | 49.85 | 65.82 | 74.06 | 82.11 | 35.56 | 77.86 | 71.81 | 55.23 | 62.35 |
| ALBERT | FP32 Baseline | 84.91 | 85.62 | 87.20 | 91.30 | 91.74 | 56.57 | 89.91 | 90.87 | 70.39 | 83.17 |
| | PINT(8,3)+FP16 | 84.74 | 85.34 | 87.03 | 91.70 | 92.31 | 55.59 | 90.73 | 90.60 | 70.04 | 83.12 |
| | Difference | -0.17 | -0.28 | -0.17 | +0.40 | +0.57 | -0.98 | +0.82 | -0.27 | -0.35 | -0.05 |
| | INT8+FP16 | 32.59 | 33.15 | 63.18 | 49.26 | 50.92 | 0* | 0* | 68.38 | 52.43 | 49.90 |

* Training ALBERT with INT8 precision on CoLA and STS-B can not get convergence, and the evaluation score was zero.
† WNLI task in GLUE is excluded from experiments as it has relatively small dataset and shows an unstable behaviour.

TABLE III
GPT-2 (SMALL) FINE-TUNING PERPLEXITIES ON DIFFERENT LANGUAGE MODELING TASKS

| Dataset | FP32 Baseline | PINT(8,3) +FP16 | INT8 +FP16 |
|---|---|---|---|
| PTB | 20.07 | 20.95 | 713.85 |
| WikiText-2 | 21.15 | 21.74 | 821.58 |
| WikiText-103 | 16.57 | 16.92 | 622.13 |
| CBT | 18.07 | 18.58 | 530.88 |

TABLE IV
VIT-BASE FINE-TUNING RESULTS ON DIFFERENT IMAGE CLASSIFICATION TASKS

| Top-1 Accuracy | FP32 Baseline | PINT(8,3) +FP16 | INT8 +FP16 |
|---|---|---|---|
| CIFAR-10 | 98.66 | 98.57 | 95.36 |
| CIFAR-100 | 91.03 | 91.45 | 86.68 |
| ImageNet | 83.77 | 83.51 | 81.44 |

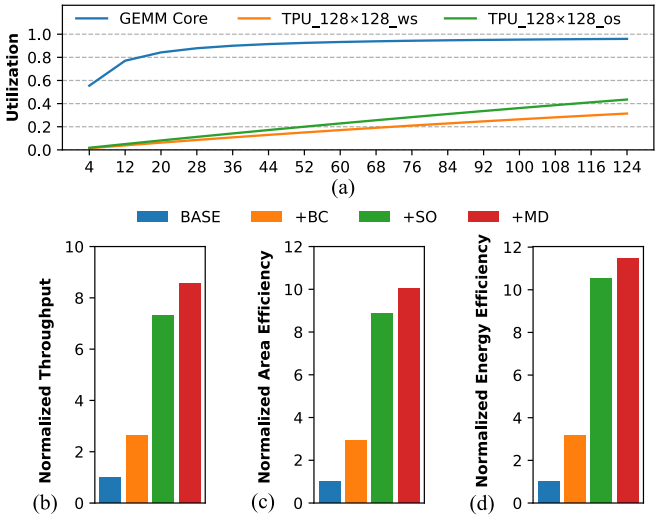



Fig. 13. (a) Comparison of utilization with different sequence lengths. (b)–(d) Performance improvement with proposed hardware techniques.

and BP stages being more suitable for the OS dataflow. In contrast, the hardware efficiency of our GEMM Core has been maintained at a high level. For the range (>12) where most of the sample length in the dataset is located, the PE utilization of GEMM Core exceeds 80%. We further conducted ablation studies to analyze the performance improvements of different design methods. First, we developed a TPU-like SA with the

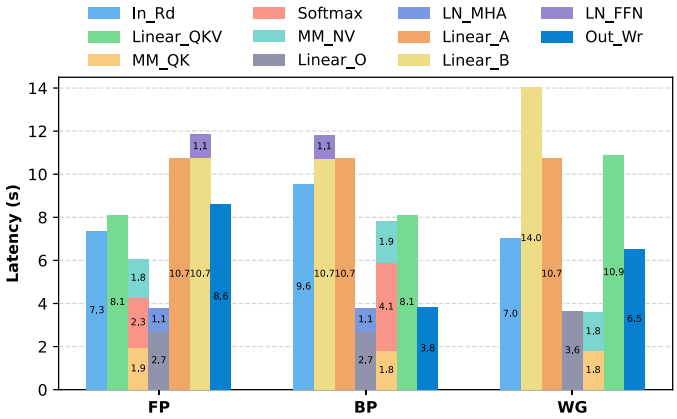



Fig. 14. Latency breakdown per epoch across different phases.

WS dataflow as the baseline. All samples in the GLUE dataset accomplish the GEMM computation of a training epoch on each hardware. Then, we append different design methods one by one: broadcast ("+BC"), scaling out ("+SO"), and multiple dataflows ("+MD"). The improvements in throughput (2.64–8.56×), area efficiency (2.91–10.05×), and energy efficiency (3.17–11.45×) of each scheme are listed in Fig. 13(b)–(d). The results prove the orthogonality of our design methods.

The overall performance of hardware is affected by computing resources and memory bandwidth. Benefiting from our hardware and algorithm co-optimization, the LB data format and high data reuse opportunity significantly reduce the bandwidth requirements. Fig. 14 shows the latency breakdown of each component in the Transformer encoder at different training phases in TRETA. The GEMM operations from the linear layers and self-attentions (MM) in MHA account for most of the computing time, while the nonlinear functions including Softmax and Layer Norm (LN) only take up a small part. The time spent on input (activations, weights, errors, etc.) reading (In_Rd) and output (activations, errors, gradients, etc.) writing (Out_Wr) is less than the total computing time. Therefore, the transaction time between DRAM and on-chip buffers can be overlapped by the ping-ping buffer manner.

*3) Performance and Efficiency Versus TPUs:* Fig. 15 evaluates the performance and efficiency of TPU-like SAs and GEMM Core under different intensive GEMM workloads with the same computing resources. In this experiment, we evaluated TPUs with different aspect ratios or dataflows and

TABLE V
COMPARISONS WITH PREVIOUS WORKS

| | [25] | LNPU [26] | [34] | SIGMA [36] | [35] | [37] | FlexBlock [38] | TRETA |
|---|---|---|---|---|---|---|---|---|
| Irregular GEMM Optimization | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |
| Process (nm) | 14 | 65 | 14 | 28 | 7 | 45 | 65 | 28 |
| Area ($mm^2$) | 9 | 16 | 9.8 | 65.10 | 19.6 | 8.96 | 33.86 | 17.26 |
| Frequency (MHz) | 750 | 200 | 1000 | 500 | 1000 1600 | 1000 | 333 | 500 |
| Precision | FP16 | FP8 | FP16 | FP16 | FP8 | INT8 | FB12 FB16 FB24† | PINT(8,3) |
| Throughput (TOPS) | 0.75* | 0.6* | 2.0* | 10.8 | 16.38* 26.2* | 2.0* | 8.78* 3.35* 0.95* | 14.71 |
| Speedup (times)‡ | 0.44 | 0.35 | 1.16 | 6.29 | 9.54 15.3 | 1.16 | 5.11 1.95 0.55 | 8.56 |
| Power (W) | - | 0.367 | 1.43 | 22.33 | 4.58 13.4 | 0.89 | 8.27 7.80 7.36 | 4.45 |
| Energy Efficiency (TOPS/W) | - | 1.63* | 1.4* | 0.48 | 3.58* 1.95* | 2.24* | 1.06* 0.43* 0.13* | 3.31 |
| Energy Efficiency Improvement (times)‡ | - | 4.19 | 3.59 | 1.23 | 9.19 5.01 | 5.75 | 2.72 1.10 0.33 | 8.51 |

* These throughput and energy efficiency results are reported without taking into account irregular GEMMs utilization degradation.
† The FlexBlock formats are defined as FB12 for 4-bit, FB16 for 8-bit, and FB24 for 16-bit mantissas with 8-bit shared exponents.
‡ Speedup and energy efficiency improvement are compared with TPU 128×128-ws of effective 1.72 TOPS throughput and 4.41 W power consumption. The TPU-like systolic array is synthesized and implemented under the same TSMC 28 nm technology as TRETA.

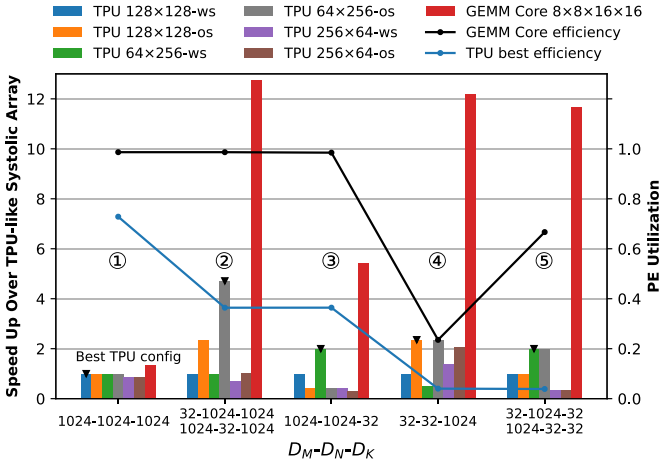



Fig. 15. Speedup and efficiency of TPU and GEMM Core with different configs for various workloads.

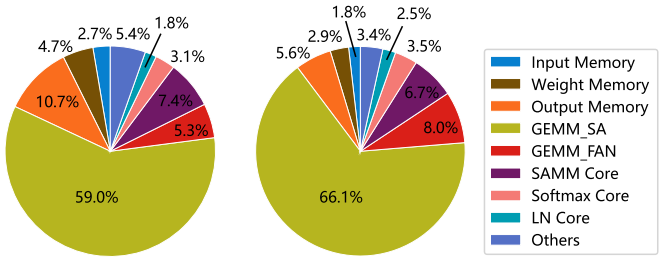



Fig. 16. Area (left) and power (right) breakdown of TRETA.

GEMM Core with $8 \times 8 \times 16 \times 16$ ($R \times C \times P_R \times P_C$) configuration. For example, the TPU of $64 \times 256$-ws has 64 rows, 256 columns of PE units, and the WS dataflow. Each row and column can read one data element per cycle. The GEMM workloads are represented by $D_M$-$D_N$-$D_K$ (meaning that the dimensions of two operand matrices are $D_M \times D_K$ and $D_K \times D_N$, respectively). Considering the transposition rule of matrix multiplication, the 32-1024-1024 and 1024-32-1024 workloads are regarded as one case for analysis. For the workload with a regular shape (① in Fig. 15), the performances of all hardware are close. GEMM Core benefits from its broadcast structure for better performance. For workloads with relatively small spatial $D_M$ or $D_N$ dimensions (②), TPUs with fewer rows of PE and OS dataflows perform better than those with square shapes and WS dataflows. However, the best efficiency of TPUs is still reduced by half compared with regular workload. Besides, TPU with the WS dataflow performs better when the temporal dimension $D_K$ is relatively small (③). As for the irregular spatial and temporal dimensions (④ and ⑤), the efficiency of TPUs has decreased greatly. Unlike TPUs, GEMM Core can dynamically adjust dataflow and data distribution strategy according to workload requirements to maintain high efficiency, as we described in Section IV-C. Only when $D_M$ and $D_N$ are both small (④), the efficiency of GEMM Core decreases significantly, as it cannot handle both small spatial dimensions. Nevertheless, this workload (④) is insignificant in Transformers training and can be efficiently processed with the supplementary SAMM Core. In conclusion, our training architecture performs much better than TPUs for all workloads and is more suitable for complicated Transformer training tasks.

*4) Area and Power:* The area and power breakdown of our training accelerator are shown in Fig. 16. Most area and power consumptions come from modules related to GEMMs. SA in the GEMM core occupies 59.0% area and 66.1% power. The FAN introduced for result reductions brings 5.3% area and 8.0% power consumption. The SAMM Core responsible for matrix multiplication in self-attention comprises 7.4% area and 6.7% power. Softmax and Layer Norm modules only take less than 5% resources, since these nonlinear functions only account for a small part of the computation in the training process.

## C. Comparison and Discussion

Table V shows comparisons between TRETA and prior DNN training accelerators. There exist apparent differences among these works in terms of training precision, process, and evaluation methodology. Nevertheless, our design shows significant superiority in training throughput and efficiency.

FP16 training precision is adopted in [25], [34], and [36], to provide high model accuracy. While Lee et al. [26], [35], Zhao et al. [37], and Noh et al. [38] proposed more compact and aggressive LB integer or floating-point DNN training. However, they are primarily verified on the traditional CNNs rather than Transformers. Fleischer et al. [25] presented a primitive scalable 2-D compute array, which can promote throughput by arranging more PEs. However, as we analyzed in Section III, it is inefficient to build a monolithic array through scaling up. LNPU [26] is composed of 16 sparse deep learning cores, and each core has four PE lines with 48 Pes. When using the FP8 precision training dense model at 200 MHz, LNPU can achieve 1.63-TOPS/W energy efficiency. A multicore processor is proposed in [34], and each core is organized as dual corelets, which has a 2-D SA of equivalent 8 × 64 size. Allocating fewer rows reduces the opportunity for data reuse in GEMM operations and increases the consumption of buffer access. Compared with them [25], [26], [34], TRETA shows 7.4-24.5× higher throughput and 2.0-2.4× higher energy efficiency.

SIGMA [36] provides a flexible and scalable microarchitecture to handle the sparsity and irregularity in GEMMs. When performing dense irregular GEMM operations, SIGMA can achieve a PE utilization close to our work. However, SIGMA focuses on sparsity and lacks the support of nonlinear functions. Cambricon-Q [37] presented a quantized training framework and is evaluated on the basic Transformer model with a 2.3% accuracy drop. The hardware of Cambricon-Q integrates a 64 × 64 PE array, and this square array is not optimal for complicated GEMM workloads. The experimental results of FlexBlock [38] show that its MAC array is less efficient than TPU-like SA when training Transformers. In comparison with these training accelerators [36], [37], [38], our design provides 1.4–15.5× speed up and 1.5–25.4× energy efficiency improvement. Lee et al. [35] have reached a throughput of 25.6 TOPS at 1.6 GHz for FP8 training, benefiting from advanced 7-nm technology and well-organized architecture. However, FP8 training cannot fully utilize the resources of its mixed-precision engines, and extra inference units reduce the overall efficiency. Moreover, the requirement for hybrid FP8 formats at different training stages increases the hardware complexity. Therefore, TRETA has achieved 1.7× higher energy efficiency than [35]. When working at 1 GHz, Lee et al. [35] can achieve a peak performance of 16.38 TOPS and 3.58 TOPS/W. However, considering the PE utilization (<80%) of irregular GEMMs on its architecture with basic computing parallelism of 16, the effective performance of [35] is lower than TRETA.

## VI. Conclusion

In this article, we propose an energy-efficient training accelerator for Transformers, namely TRETA, with hardware-algorithm co-optimization. First, a mixed-precision training framework is presented based on a novel and compact data format. This framework significantly reduces the computation and memory requirements, while maintaining the same accuracy as the full-precision baseline. TRETA integrates efficient GEMM and nonlinear modules to perform the computations of various Transformer components during the training process. The flexible and scalable GEMM Core can maintain high utilization when processing arbitrary irregular GEMM operations in Transformer training, achieving 10.05× area efficiency and 11.45× energy efficiency than traditional TPU-like SAs. The nonlinear function units designed for Softmax and layer normalization are flexible to support various computations during training in unified architectures, which work in a time-multiplexed manner and consume minimal resources. The experimental results demonstrate that TRETA achieves 14.71 TOPS in throughput and 3.31 TOPS/W in energy efficiency and realizes 1.4–24.5× speedup and 1.5–25.4× energy efficiency over prior DNN training accelerators. This article, as the efficient work on hardware and algorithm co-optimized training accelerator for Transformers, also brings some inspiration to Transformer-based large model training.

## References


[1] A. Vaswani et al., "Attention is all you need," in *Proc. NIPS*, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv*, vol. abs/1810.04805, 2019.

[3] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*.

[4] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," 2019, *arXiv:1909.11942*.

[5] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[6] A. Dosovitskiy et al., "An image is worth 16 × 16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.

[7] Z. Liu et al., "Swin Transformer: Hierarchical vision transformer using shifted windows," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 9992–10002.

[8] N. Parmar et al., "Image transformer," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 4055–4064.

[9] L. Dong, S. Xu, and B. Xu, "Speech-Transformer: A no-recurrence sequence-to-sequence model for speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2018, pp. 5884–5888.

[10] X. Chen, Y. Wu, Z. Wang, S. Liu, and J. Li, "Developing real-time streaming transformer transducer for speech recognition on large-scale dataset," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2021, pp. 5904–5908.

[11] C.-Z. A. Huang et al., "Music transformer," 2018, *arXiv:1809.04281*.

[12] J. Lee and H.-J. Yoo, "An overview of energy-efficient hardware accelerators for on-device deep-neural-network training," *IEEE Open J. Solid-State Circuits Soc.*, vol. 1, pp. 115–128, 2021.

[13] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," OpenAI Blog, 2018. [Online]. Available: https://openai.com/blog/language-unsupervised/

[14] T. B. Brown et al., "Language models are few-shot learners," 2020, *arXiv:2005.14165*.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)* 2015, pp. 770–778.

[16] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–14.

[17] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021, *arXiv:2106.08295*.

[18] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, pp. 1789–1819, Mar. 2020.

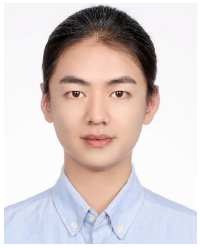
[19] Y. Idelbayev and M. Á. Carreira-Perpiñán, "Low-rank compression of neural nets: Learning the rank of each layer," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 8046–8056.

[20] J. Lee, S. Kang, J. Lee, D. Shin, D. Han, and H. Yoo, "The hardware and algorithm co-design for energy-efficient DNN processor on edge/mobile devices," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 10, pp. 3458–3470, Oct. 2020.

[21] B. Keller et al., "A 95.6-TOPS/W deep learning inference accelerator with per-vector scaled 4-bit quantization in 5 nm," *IEEE J. Solid-State Circuits*, vol. 58, no. 4, pp. 1129–1141, Apr. 2023.
[22] Y. Wang et al., "A 28 nm 27.5TOPS/W approximate-computing-based transformer processor with asymptotic sparsity speculating and out-of-order computing," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 1–3.
[23] P. Micikevicius et al., "Mixed precision training," 2017, *arXiv:1710.03740*.
[24] F. Zhu et al., "Towards unified INT8 training for convolutional neural network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1966–1976.
[25] B. Fleischer et al., "A scalable multi-TeraOPS deep learning processor core for AI training and inference," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 35–36.
[26] 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.
[27] M. Wang, S. Rasoulinezhad, P. H. W. Leong, and H. K.-H. So, "NITI: Training integer neural networks using integer-only arithmetic," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 3249–3261, Nov. 2022.
[28] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.
[29] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[30] J. Lu, C. Ni, and Z. Wang, "ETA: An efficient training accelerator for DNNs based on hardware-algorithm co-optimization," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Feb. 8, 2022, doi: 10.1109/TNNLS.2022.3145850.
[31] M. P. Drumond, T. Lin, M. Jaggi, and B. Falsafi, "Training DNNs with hybrid block floating point," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 1–11.
[32] D. Kalamkar et al., "A study of BFLOAT16 for deep learning training," 2019, *arXiv:1905.12322*.
[33] K. Zhao et al., "Distribution adaptive INT8 quantization for training CNNs," 2021, *arXiv:2102.04782*.
[34] 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.
[35] S. K. Lee et al., "A 7-nm four-core mixed-precision AI chip with 26.2-TFLOPS hybrid-FP8 training, 104.9-TOPS INT4 inference, and workload-aware throttling," *IEEE J. Solid-State Circuits*, vol. 57, no. 1, pp. 182–197, Jan. 2022.
[36] E. Qin et al., "SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 58–70.
[37] Y. Zhao et al., "Cambricon-Q: A hybrid architecture for efficient training," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 706–719.
[38] S.-H. Noh, J. Koo, S. Lee, J. Park, and J. Kung, "FlexBlock: A flexible DNN training accelerator with multi-mode block floating point support," 2022, *arXiv:2203.06673*.
[39] B. Thompson et al., "Freezing subnetworks to analyze domain adaptation in neural machine translation," 2018, *arXiv:1809.05218*.
[40] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of DNN accelerators using SCALE-SIM," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Aug. 2020, pp. 58–68.
[41] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn Treebank," *Comput. Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.
[42] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016, *arXiv:1609.07843*.
[43] F. Hill, A. Bordes, S. Chopra, and J. Weston, "The Goldilocks principle: Reading children's books with explicit memory representations," 2015, *arXiv:1511.02301*.
[44] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
[45] 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.*, Jun. 2009, pp. 248–255.

**Haikuo Shao** received the B.S. degree in electronic information science and technology and the M.S. degree in integrated circuit engineering from Nanjing University, Nanjing, China, in 2019 and 2022, respectively, where he is currently working toward the Ph.D. degree in information and communication engineering.

His current research interests include VLSI design and compression algorithms for machine learning.



**Jinming Lu** received the B.S. degree in microelectronics from Nankai University, Tianjin, China, in 2018, and the Ph.D. degree in information and communication engineering from Nanjing University, Nanjing, China, in 2023.

His current research interests include automatic speech recognition and deep learning, especially its hardware acceleration and compression algorithms.

**Meiqi Wang** received the B.S. degree in electronic information science and technology and the Ph.D. degree in information and communication engineering from Nanjing University, Nanjing, China, in 2018 and 2023, respectively.

She was a Visiting Student with the Wangxuan Institute of Computer Technology, Peking University, Beijing, China, in 2019. She is currently an Assistant Professor with the College of Integrated Circuits, Sun Yat-sen University, Shenzhen, China. Her current research interests include efficient algorithm and hardware codesign for deep neural networks, VLSI design, and in-memory computing.

**Zhongfeng Wang** (Fellow, IEEE) received the B.E. and M.S. degrees from the Department of Automation, Tsinghua University, Beijing, China, in 1988 and 1990, respectively, and the Ph.D. degree from the University of Minnesota, Minneapolis, MN, USA, in 2000.

He was with Oregon State University, Corvallis, OR, USA, and National Semiconductor Corporation, Santa Clara, CA, USA. He was with Broadcom Inc., San Jose, CA, USA, from 2007 to 2016, as a Leading VLSI Architect. Since 2016, he has been with Nanjing University, Nanjing, China, as a Distinguished Professor. He is a World-Recognized Expert on Low-Power High-Speed VLSI Design for Signal Processing Systems. He has authored over 200 technical articles with multiple best paper awards received from the IEEE technical societies, among which is the VLSI Transactions Best Paper Award of 2007. He has edited one book *VLSI* and held more than 20 U.S. and Chinese patents. His current research interests include optimized VLSI design for digital communications and deep learning.

Dr. Wang was elevated to the fellow of IEEE for contributions to VLSI design and implementation of FEC coding in 2015. He has also served as a TPC member and various chairs for tens of international conferences. Moreover, he has contributed significantly to industrial standards. So far, his technical proposals have been adopted by more than 15 international networking standards. In the current record, he has had many articles ranking among the top 25 most (annually) downloaded manuscripts in the IEEE TRANSACTIONS ON VLSI SYSTEMS (TVLSI). In the past, he has served as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I, TRANSACTIONS ON CIRCUITS AND SYSTEMS II, and TVLSI for many terms.