# FIGNA: Integer Unit-based Accelerator Design for FP-INT GEMM Preserving Numerical Accuracy

Jaeyong Jang†[1], Yulhwa Kim‡[1], Juheun Lee†, and Jae-Joon Kim†[*]
Seoul National University†
{jaeyongjang, juheun.lee, kimjaejoon}@snu.ac.kr†
Sungkyunkwan University‡
yulhwa.k@gmail.com‡

2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 979-8-3503-9313-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/HPCA57654.2024.00064

***Abstract*—The weight-only quantization has emerged as a promising technique for alleviating the computational burden of large language models (LLMs) by employing low-precision integer (INT) weights, while retaining full-precision floating point (FP) activations to ensure inference quality. Despite the memory footprint reduction achieved through decreased bit-precision of weight parameters, the actual computing performance is often not improved significantly due to FP-INT multiply-accumulation (MAC) operations being performed on the floating point unit (FPU) after dequantizing the INT weight values to FP values, owing to the lack of dedicated FP-INT arithmetic units. In this study, we investigate the impact of introducing a dedicated FP-INT unit on overall performance and find that such specialization does not yield substantial improvements. As an alternative approach, we propose FIGNA, an accelerator based on INT units designed specifically for FP-INT MAC operations. A key feature of FIGNA is its ability to achieve the same *numerical* accuracy as the FPU while relying solely on the integer-unit, a departure from prior methods that relied on integer-units with numerical approximations for FP arithmetic results, albeit claiming similar *inference* accuracy through dedicated network training. Through comprehensive experiments on FP-INT quantized networks for LLMs, including OPT and BLOOM, we demonstrate the superior performance of FIGNA compared to conventional FPUs in terms of performance per area (TOPS/$mm^2$) and energy efficiency (TOPS/W) across various input and weight precision combinations. For instance, in the FP16-INT4 case, FIGNA shows 6.34x higher TOPS/$mm^2$ and 2.19x higher TOPS/W compared to the baseline.**

## I. Introduction

Large language models (LLMs) have demonstrated exceptional performance across wide range of natural language processing (NLP) tasks and have become the backbone of numerous applications ranging from machine translation and sentiment analysis to chatbots and question answering systems [6], [44], [48], [57]. However, the success of LLMs also presents significant challenges. The massive number of parameters leads to increased memory consumption and higher computational costs, limiting their accessibility [31], [49]. Hence, there is a growing need to address these challenges and develop techniques to optimize LLMs, making them more efficient, compact, and accessible without compromising their linguistic capabilities. One of the promising solutions for enhancing the efficiency of LLM is quantization. The concept of quantization involves representing high-precision FP values with lower-precision data types. Traditional quantization techniques in DNNs involve quantizing both the weights and activations to low-precision integer, significantly reducing both memory consumption and computational complexity of the model [11], [28], [50], [51]. However, quantizing activations in language models poses challenges due to their dynamic nature and the sensitivity of generative tasks to precision loss [17]. In this context, weight-only quantization provides a compelling alternative [2], [7], [14], [16], [17], [27], [30], [39], [52], [53].

Quantizing the weights substantially reduces memory consumption. For example, without quantization, representative LLMs with ∼175B parameters such as GPT-3, OPT, and Bloom require more than 350 GB of memory to load their model parameters [14], [48], [57]. As a single NVIDIA A100 GPU has 80 GB memory, at least 5 A100 GPUs are required for processing these models [38]. On the other hand, by reducing the weights to 8-bit or even 4-bit integers, the model sizes can be reduced to 175 GB or 88 GB. The inference of the quantized models can run on only 2 or 3 NVIDIA A100 GPUs, showcasing the substantial reduction in memory and resources. Hence, weight-only quantization approach strikes a balance between compression and accuracy, offering a practical solution for optimizing LLMs.

As the importance of the weight-only quantization grows, NVIDIA recently introduced a new FP-INT GEMM kernel in its popular CUDA template, CUTLASS [22], and other researchers also have proposed specialized GPU kernels [14], [17]. However, these kernels still rely on existing FP units for computations of FP-INT GEMM due to the lack of dedicated FP-INT arithmetic units on the hardware. Consequently, INT weight parameters need to dequantized to FP values before being fed to the FP units, incurring computational overhead. As a result, the primary achievement of these kernels lies in leveraging the reduced memory footprint in weight-quantized cases. In other words, these kernels can achieve speedup in memory-bound scenarios, such as a single batch text generation. However, real-world LLM applications often involve multi-batch processing, as batching of generative tasks can be easily accomplished [45], [54]. Since multi-batch processing is typically compute-bound, enhancing inference speed or throughput with weight-only quantization becomes challenging, even with specialized kernels.

[1]Co-first authors.
[*]Corresponding author.

2378-203X/24/$31.00 ©2024 IEEE
DOI 10.1109/HPCA57654.2024.00064

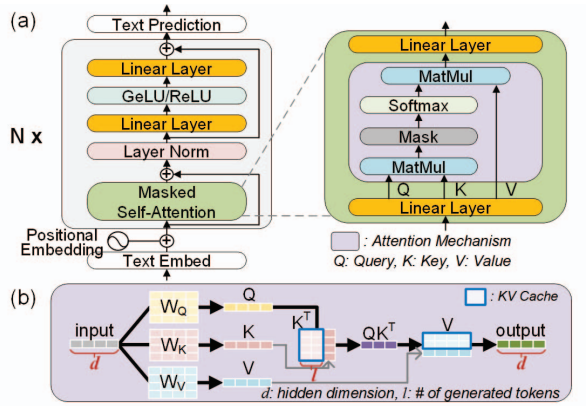


Fig. 1. (a) A general architecture of LLMs. (b) Matrix multiplications within the attention mechanism during a single batch processing with KV cache.

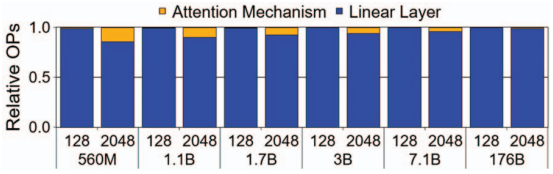


Fig. 2. The relative number of operations (OPs) for GEMM in attention mechanisms and linear layers of Bloom models across various model size for text generation tasks with token lengths of 128 and 2048. The relative OPs are independent of batch size since the OPs linearly scales with batch size both in attention mechanism and linear layers.

To overcome the limitation in weight-only quantization and improve the performance, we propose FIGNA, a specialized hardware architecture tailored to effectively accelerate FP-INT quantized networks. FIGNA focuses on the optimization of FP-INT GEMM operation, a critical operation in LLMs. The optimization involves pre-alignment, where FP inputs are converted to INT format before computations by sharing exponents, allowing all computations to be performed using integer operations. To ensure the advantages of integer operations at the system level, we minimize the overhead due to format conversion through application-transparent optimization techniques, such as truncation of the pre-aligned mantissa and chunk-based mantissa allocation, as well as the FIGNA architecture. While this architecture shares a design philosophy with previous block floating point (BFP) hardware in terms of sharing the exponent of the inputs, a crucial difference exists between the BFP hardware and FIGNA in terms of numerical accuracy. Previous works using BFP format suffered from higher numerical errors compared to conventional FP MAC, but attempted to maintain neural network accuracy by customizing neural network for the given hardware. In contrast, through theoretical analysis, we design FIGNA to maintain the same level of numerical accuracy as that of conventional FP MAC. Consequently, FIGNA allows users to directly deploy the given neural network to the hardware without the need for customization specific to the hardware constraints.

Our contributions can be summarized as follows:

- We propose an integer-based FP-INT GEMM computing scheme, along with application-transparent optimizations, including truncation of pre-aligned mantissas and chunk-based allocation.
- Through theoretical analysis, we carefully design the amount of the truncation and introduce 0-less INT format to ensure that the proposed approach preserves the numerical accuracy of conventional FP arithmetic.
- We introduce a novel FIGNA architecture, purpose-built to handle integer-based FP-INT GEMM. FIGNA capitalizes on the benefits of simple integer units, leading to substantial performance enhancements.

## II. Background

### A. LLM Inference and Quantization

LLMs typically consist of a series of decoder blocks, which include a masked self-attention and linear layers (Fig. 1(a)) [48], [57]. In the masked self-attention, the input values are first processed with a linear layer to generate Query (Q), Key (K), and Value (V) vectors. These vectors are then used to perform the attention mechanism. The attention results are subsequently post-processed with another linear layer. Following the attention block, the output is further processed with multiple linear layers. Within LLMs, both linear layers and attention mechanisms involve GEMM computations, but the difference lies in what gets multiplied. Linear layers execute GEMM operations between weights and activations, whereas attention mechanisms perform GEMM operations between activations. When LLM has a hidden dimension denoted as $d$ and a total of $l$ tokens generated up to a point, taking into account KV cache [37], [54], the number of operations associated with linear layers is proportional to $d^2$, whereas that of the attention mechanisms is proportional to $(l+1)d$ as shown in Fig. 1(b). It is important to note that the hidden dimension $d$ in LLMs is usually much larger than the number of generated tokens $l$. For example, $d$ is 14,336 in a BLOOM-176B model while $l$ typically falls within the range of [1, 2048] for general text-generation tasks. Consequently, the GEMM operations carried out by the linear layers tend to dominate the computational workload within LLMs (Fig. 2). Hence, it is crucial to optimize and accelerate the GEMM operations associated with linear layers to boost the performance and efficiency in LLM computations [20].

To enhance the efficiency of LLM inference, quantization techniques tailored for LLMs have been actively studied recently. However, activation quantization is more challenging for LLMs compared to other DNNs since their activation values tend to have wide distributions, and outliers play a crucial role [11]. Consequently, quantizing LLM activations often leads to a degradation in accuracy. To reduce the quantization error, more sophisticated quantization techniques, such as vector-wise quantization (e.g. VS-Quant [8]), block-wise quantization (e.g. MX [10], and RPTQ [55]) have been proposed. While these techniques show promise in mitigating accuracy degradation by finely defining quantization parameters such as quantization offset and scale, they are not universally effective. Depending on LLM models and tasks, accuracy issues can still arise [55]. As a result, for broader applications of LLMs,

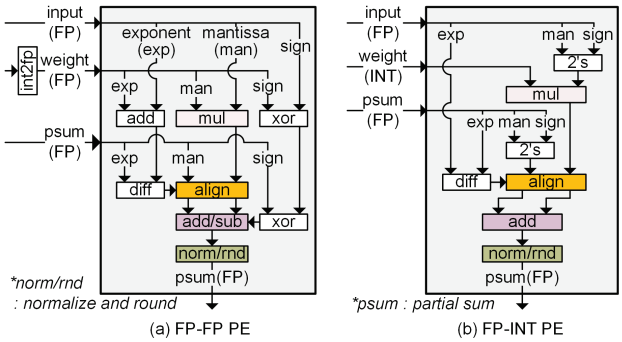


Fig. 3. Naive PE designs for FP-INT MAC operation. (a) FP-FP PE adopts a typical FP MAC unit along with dequantization of INT weights to FP values. (b) FP-INT PE tailors the FP MAC units specially for the FP-INT operation.

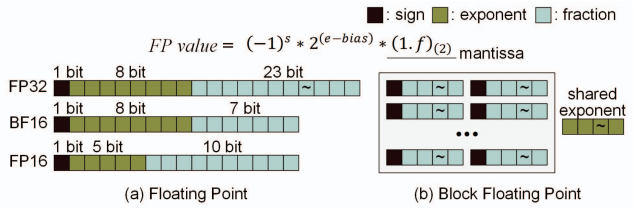


Fig. 4. Conventional (a) floating point and (b) block floating point formats.

weight-only quantization has been adopted as a pragmatic solution to enhance LLM inference efficiency [14], [20], [30]. In this scenario, all the linear layers of LLMs are required to perform FP-INT GEMM operations between activations and weights.

### B. Processing FP-INT GEMM in LLMs

FP-INT GEMM consists of FP-INT Multiply-Accumulate (MAC) operations, where a FP activation is multiplied by an INT weight, and the resulting FP multiplication is accumulated. In conventional processors, there are no specialized arithmetic units specifically designed to handle operations between FP values and INT values. Consequently, the FP-INT MAC operation is typically performed using a FP MAC unit [15], [41], referred to as a FP-FP PE in Fig. 3(a). Before the computation, the integer weights need to be converted into FP format (Fig. 4(a)), which is composed of sign, mantissa, and exponent, to ensure compatibility with the FP activation [36].

As a result, the FP-FP PE, while versatile for general FP MAC operations, leads to inefficient computation for FP-INT MAC operations. To overcome this inefficiency, a dedicated FP-INT PE can be considered as an alternative (Fig. 3(b)) The FP-INT PE eliminates the need for the weight data type conversion and directly utilizes low-precision integer weights for multiplication, resulting in a reduction in the size of the multiplier. For instance, when handling a FP32 activation and an INT8 weight, the FP-INT PE requires a 25-bit $\times$ 8-bit multiplier for multiplication between the signed-mantissa of the FP activation and the integer weight, while the FP-FP PE requires a 24-bit $\times$ 24-bit multiplier for the multiplication between the mantissas of two FP32 values.

Fig. 5 presents a summary of the power savings achieved by the FP-INT PE in comparison to the conventional FP-FP PE. Notably, when using FP32 activations, the FP-INT PE exhibits

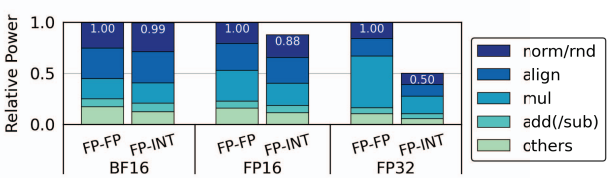


Fig. 5. The relative power consumption of FP-FP PE and FP-INT PE during FP-INT8 MAC operation. The power is normalized with the total power of FP-FP PE for each FP type.

substantial reductions in power consumption, with reductions of up to 50%. However, the performance gain of the FP-INT PE diminishes when dealing with FP16 or BF16 activations. This is primarily because the significant benefit of the FP-INT PE comes from reducing the size of the multiplier, but for FP16 and BF16 cases, the reduction in multiplier size is limited due to the smaller number of mantissa digits in these formats. For example, when processing FP16 activation and INT8 weight, the FP-INT PE replaces the 11-bit $\times$ 11-bit multiplier of the FP-FP PE with a 12-bit $\times$ 8-bit multiplier, resulting in a modest reduction in the multiplier size. For BF16 activation, the FP-INT PE replaces the 8-bit $\times$ 8-bit multiplier of the FP-FP PE with a 9-bit $\times$ 8-bit multiplier, leading to a slight increase in the multiplier size.

The breakdown of PE components in Fig. 5 reveals that alignment and normalization operations, essential for handling the scale of FP values in the accumulation stage, occupy a significant portion of the PE, regardless of the target data types. These operations involve shifting the mantissas of operands based on exponent differences, which demands sizable barrel shifters. Therefore, development of a new arithmetic unit structure that has smaller overhead associated with alignment and normalization is needed for enhancing the efficiency of FP-INT GEMM.

### C. Block Floating Point

Block Floating Point (BFP) has been proposed as a compromise between the wide dynamic range of traditional FP formats and the simplicity of INT formats [47]. BFP assigns a common exponent to a group of values, allowing them to be collectively scaled (Fig. 4(b)). This characteristic allows BFP to achieve a higher compression ratio than INT formats, while also facilitating straightforward computations using integers within each group [24], [56]. On the other hand, similar to other quantization techniques, utilizing BFP for DNN compression introduces numerical deviations from the original FP values, potentially resulting in neural network accuracy loss.

To address this accuracy limitation, previous works have adopted BFP-DNN co-design approaches with a focus on preserving DNN accuracy while minimizing bit precision of BFP formats. One of the approaches employs exclusive searching methods to identify appropriate BFP formats for specific DNN models [8], [9], [13], [29], [43]. However, this approach comes with challenges as the optimal BFP format and the required hardware support may vary across different DNN models. Another approach is BFP-aware training, where DNNs are specifically optimized for a chosen BFP format through training [9], [10], [35], [56]. While this approach

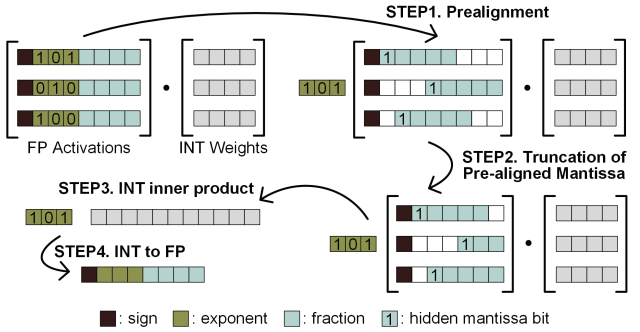


Fig. 6. Overview of the proposed integer-based FP-INT GEMM operation

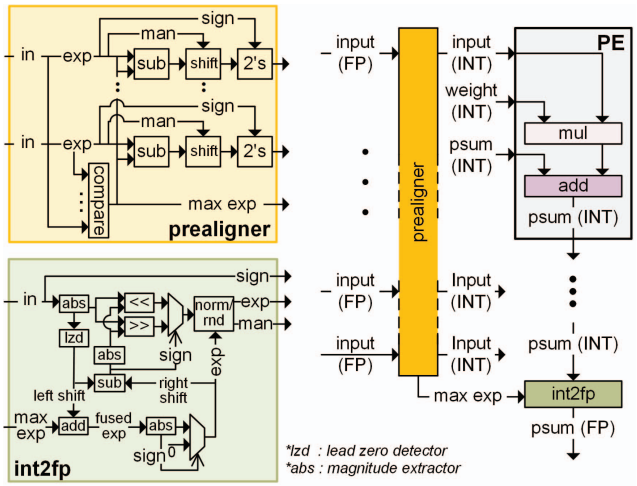


Fig. 7. The key hardware components of FIGNA for integer-based FP-INT GEMM: *prealigner*, PE, and *int2fp* unit. The *prealigner* is responsible for converting FP input values to INT values by aligning all the mantissas to the same scale. The FIGNA PE conducts the MAC operation using INT units, and after the entire computation of the inner product, the computation results are converted back into FP values using $int2fp$ unit.

can improve accuracy, it comes with the drawback of an expensive training process, making it less adaptable for large-scale models that require significant computational resources. Additionally, if the target BFP format changes, the DNNs need to be retrained for the new block size, posing practical challenges in deploying BFP-based solutions across different hardware configurations. Hence, in this paper, rather than developing intricate quantization methods that demand extensive effort, we develop a reliable approach to process GEMM operations with FP activations accurately and efficiently by adopting the shared exponent concept in the BFP format.

## III. Integer-based FP-INT GEMM Preserving Numerical Accuracy

In this section, we introduce a computational methodology and an accelerator design called FIGNA for FP-INT GEMM, with a primary focus on preserving numerical accuracy while adopting the concept of the shared exponent from BFP. Based on a comprehensive theoretical analysis of computational reliability, we can make the proposed scheme operate in an application-transparent manner. Therefore, unlike conventional BFP methods, the proposed design eliminates the need to consider hardware when deploying the neural network.

### A. Overview of Integer-based FP-INT GEMM

Fig. 6 illustrates the overview of the proposed integer-based FP-INT GEMM, and Fig. 7 presents the essential hardware components of FIGNA, specifically designed to handle integer-based FP-INT GEMM. Before the computation begins, the *prealigner* converts FP values into INT values. It decomposes exponent and mantissa of each FP value and handles the pre-alignment process, dynamically aligning mantissas of FP activations for the subsequent GEMM computation. When *prealigner* decomposes FP values, it derives mantissa by combining the hidden leading bit 1 with fraction, as illustrated in Fig. 4. On the other hand, *prealigner* also detects subnormal FP values by examining exponent values that are zero. In the case of subnormal values, since they lack a hidden leading bit, their fractions are directly used as mantissas. Next, the *prealigner* identifies the maximum exponent within a vector of the FP activation matrix. It then adjusts the mantissa of each FP activation based on the difference between its own exponent and the maximum exponent, ensuring that all pre-aligned mantissas lie in the same scale. The pre-alignment makes it possible to represent the FP activations using integer values, sharing a common maximum exponent to indicate the scale of the activation vector. It is important to note that the dynamic nature of pre-alignment differentiates the usage of shared exponents in our approach from that in BFP. In BFP, shared exponents are a component of the numeric format used to represent specific numbers. In contrast, in our proposed approach, shared exponents are introduced as a step of computation and are not visible to users. With the pre-alignment in place, we can replace the FP-INT MAC operation with an INT-INT MAC operation, eliminating the need for alignment and normalization in each MAC operation. After completing the entire inner product with the weight matrix, the *int2fp* unit converts the computation results back into the FP format by following the FP formatting and exception handling rules used in the normalization part of conventional FP units.

Despite the benefits of integer-based FP-INT GEMM in eliminating alignment/normalization overhead in each PE, it can lead to a significant increase in the multiplier size if not handled carefully. For instance, consider the case of BF16 activations, where the maximum amount of mantissa shifting in the pre-alignment process is 255 due to 8-bit exponents. The pre-aligned mantissa should be represented with 263-bit integer to precisely maintain their values, and after combining with the sign value, it requires 264-bit integer. As a result, for the MAC operation between BF16 and INT8, while a FP-INT PE necessitates a 9-bit $\times$ 8-bit multiplier, a FIGNA PE requires a 264-bit $\times$ 8-bit multiplier, increasing the multiplier size by 29.3 times. In this case, the overhead introduced by the increased multiplier size outweighs the advantage of eliminating alignment/normalization overhead. Therefore, to achieve meaningful performance gain with integer-based FP-INT GEMM, it is crucial to develop a scheme to reduce the bit-precision of the pre-aligned mantissas while preserving the numerical accuracy of the original FP-INT GEMM.

### B. Mathematical Proof of Numerical Accuracy Preserving Truncation of Pre-Aligned Mantissas for FP-INT MAC

The proposed integer-based FP-INT GEMM has a potential to have same level of numerical accuracy as the FP operations even with the truncation of pre-aligned mantissas since the results of FP operations cannot be fully represented with the given FP format due to the limited number of mantissa bits. Rounding is applied in such cases to approximate the results to the nearest representable FP values. This rounding process introduces errors in FP operations, bounded by the smallest increment between two representable FP numbers, which can be denoted as $\epsilon = 2^{-p}$ ($p$: bit precision of the mantissa). On the other hand, if we truncate pre-aligned mantissas, the truncation error arises before the actual computation takes place. We assume that the computational results pass through the accumulation path with integer values precisely by increasing the bit-width of the integer accumulator throughout the accumulation path. This approach is feasible because the increase in bit-width is minor, and it depends on the logarithm scale of the number of accumulations. Therefore, the truncation of pre-aligned mantissas can be seen as shifting the error source of the FP operations from the end of the operation to the start of the operation.

To ensure numerical accuracy, it is essential to carefully examine the truncation of the pre-aligned mantissa to guarantee that the error introduced in the integer-based FP-INT MAC operation does not exceed $\epsilon$. The impact of the mantissa truncation on FP summation has been investigated in a previous work [23], and in this paper, we extend the mathematical proof presented in [23] to describe the effect of the mantissa truncation on the FP-INT MAC operation. The level of truncation is controlled by the parameter $\delta$, which determines the number of extra bits allowed for the pre-aligned mantissas. In this context, the pre-aligned mantissas preserve only top $t(=p+\delta)$ bits, and any remaining bits are discarded. To preserve the overall accuracy of the FP-INT GEMM, it is essential to ensure that the error bound for individual FP-INT MAC operation is equal to that of the conventional operation. This ensures that the numerical accuracy is maintained at each step of the computation, leading to accurate results for the entire GEMM. Therefore, we begin by analyzing the impact of the truncation on the integer-based FP-INT MAC operation $wx+w'x'$, using two FP values $x$ and $x'$ ($x \geq x'$ without loss of generality) and two integer values $w$ and $w'$. We assume that all these values are non-negative, and we examine the impact of the sign during the accumulation stage, considering both addition and subtraction. Note that precise results are computed when: 1) $x = x'$, 2) $x' = 0$, or 3) $\delta \geq k$ where $k$ is the exponent difference between $x$ and $x'$. Therefore, for the following analysis, we focus on the scenario where $x > x' > 0$ and $k > \delta$. Additionally, we assume that $w, w' \in [1,\ 2^{p_w}-1]$, where $p_w$ is the bit precision of the maximum value. The impact of the zero integer value would be discussed in the next section.

After the pre-alignment and truncation of $x'$, we obtain $\bar{x'}$, and the difference between $x'$ and $\bar{x'}$ is bounded as follows:

$$|x'-\bar{x'}| = 0.0\cdots0b_t\cdots b_{(k+p-1)} \leq 2^{-(p+\delta-1)}(1-2^{-(k-\delta)}) \quad (1)$$

Here, $b_i$ denotes the binary state of $i$-th fraction bit. As the absolute scale of $x$ and $x'$ does not affect the amount of mantissa shifting and truncation in the pre-alignment stage, we assume that the exponent of $x$ is 0 (which means $x \geq 1$) and that of $x'$ is $-k$ without loss of generality.

In the case that the addition is followed by the multiplication, the relative error of the MAC operation is defined as follows:

$$e_{add} = \frac{|(wx+w'x')-(wx+w'\bar{x'})|}{|wx+w'x'|} = \frac{|w'|\times|x'-\bar{x'}|}{|wx+w'x'|} \quad (2)$$

Because $|wx+w'x'| \geq 1$, by applying Eq. 1, we can get

$$e_{add} \leq (2^{p_w}-1)\times 2^{-(p+\delta-1)}(1-2^{-(k-\delta)}) \leq 2^{-(p+\delta-1-p_w)} \quad (3)$$

The upper bound of $e_{add}$ is $\epsilon$ when $\delta = p_w + 1$.

In the case that the subtraction is followed by the multiplication, the relative error is defined as follows:

$$e_{sub} = \frac{|(wx-w'x')-(wx-w'\bar{x'})|}{|wx-w'x'|} = \frac{|w'|\times|x'-\bar{x'}|}{|wx-w'x'|}. \quad (4)$$

Here, $wx \geq 1$ and $w'x'$ is bounded as follows:

$$w'x' \leq (2^{p_w}-1)(1.b_1\cdots b_{p-1})2^{-k} < 1.1\cdots1\times2^{-k+p_w} \quad (5)$$

If $\delta = p_w + 1$, $wx$ is always greater than $w'x'$ when $k > \delta$, so $|wx-w'x'|$ is bounded as follows:

$$|wx-w'x'| \geq 2^{-1}+\cdots+2^{p_w-k+1} = 1-2^{p_w-k+1} \quad (6)$$

By applying Eq. 6 to Eq. 4, we can get $e_{sub} \leq \epsilon$. Please note that this proof is also valid for the exceptional scenario involving subnormal FP numbers as computation results. Subnormal FP numbers have an exponent of 0 and lack a hidden leading bit 1 for the mantissa. In this case, the rounding error, which depends on the smallest difference between two subnormal FP numbers, is larger than $\epsilon$. Meanwhile, as the proposed scheme is designed to guarantee the error level smaller than $\epsilon$, integer-based FP-INT GEMM can maintain the numerical accuracy when dealing with subnormal numbers too.

Therefore, the integer-based FP-INT MAC operation preserve the same level of numerical accuracy as that of conventional FP-INT operation when $\delta = p_w + 1$. The above analysis is independent of the specific FP format, making this finding applicable to any FP format. Additionally, increasing $\delta$ by 1, i.e., setting $\delta = p_w + 2$, reduces the error bound of the integer-based FP-INT MAC operation to half of $\epsilon$. This enhanced error tolerance is especially valuable in FP-INT GEMM, where an additional conversion of the integer-based computation results back into the original FP format is required at the end of the PE array. **As a result, the integer-based FP-INT GEMM can maintain the numerical accuracy with $\boldsymbol{\delta = p_w + 2}$.**

In the inner product computation of GEMM, MAC operations are executed sequentially, and errors accumulate with each MAC operation. Therefore, if we can match the error

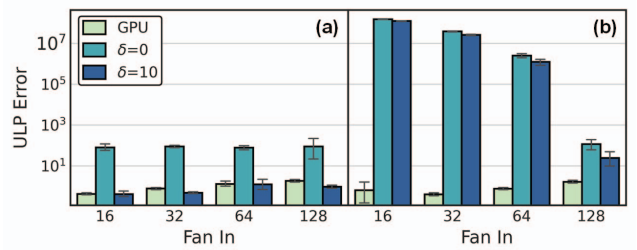


Fig. 8. The computation error of FP32-INT8 GEMM executed with the NVIDIA A100 GPU and the proposed integer-based FP-INT GEMM (with pre-aligned mantissa truncation parameterized by $\delta$) in the comparison to the accurate FP-INT GEMM. The (a)/(b) subplots represent the cases where 0 values are excluded/included in the INT8 weights. Each error bar represents a 95% confidence interval.

level of MAC computation between the pre-aligned method and conventional FP operation in the case of two inputs, we can anticipate the same level of numerical accuracy in GEMM for both methods, since the same level of error accumulates throughout the entire GEMM processing. Fig. 8 illustrates unit in the last place (ULP) error of FP-INT GEMM with the NVIDIA A100 GPU against the accurate operation and that of the proposed integer-based FP-INT GEMM against the accurate operation [42]. For the evaluation on the real FP-INT GEMM operations, we collect a dataset of 50,000 FP-INT inner product cases for each fan-in value. Here, fan-in refers to the number of input values considered for each inner product operation, and the reported error represents the average error observed in these sampled cases. When weights are within the range $1 \leq w \leq 2^{p_w} - 1$, the mantissa truncation with $\delta = 0$ substantially increases the computation error of the integer-based FP-INT GEMM. In contrast, we can observe that the proposed method shows the same numerical accuracy as the NVIDIA A100 GPU with $\delta = p_w + 2$ as predicted by our mathematical analysis (Fig. 8(a)).

One exception case is when the weight range includes 0 values ($0 \leq w \leq 2^{p_w} - 1$). In such cases, it can be observed that the error level of the proposed method increases significantly even with $\delta = p_w + 2$ so that the numerical accuracy cannot be guaranteed (Fig. 8(b)). Hence, we need to find the root cause of the higher level of numerical errors when 0 is included and develop the solution.

### C. 0-less Signed INT Format for Reliable Computation

A key insight from the analysis in Section III-B is that when performing the multiplication of non-zero integer values with FP values, the scale change of the FP values remains within a finite window. Moreover, the accumulation process does not change the scale of the operands. This particular property enables us to aggressively truncate the less significant bits of the pre-aligned mantissas before computation without incurring a significant impact on the numerical accuracy. In contrast, achieving reliable computation with the pre-aligned mantissa truncation becomes unfeasible when integer weights contain 0 values, as the scale change after the multiplication becomes unpredictable. For example, consider the case when two FP activations, $1.011 \times 2^{23}$ and $1.001 \times 2^{-1}$, multiplied with weight values 0 and 1, respectively. The correct FP-INT MAC operation result should be $1.001 \times 2^{-1}$. However, when we apply the concept of pre-alignment and mantissa truncation with an extra bit of 1, the pre-aligned mantissas are represented as $1.0110$ and $0.0000$, sharing the maximum exponent $2^{23}$. After multiplying the weight values and accumulating, the computational result becomes 0, which does not reflect the correct value of $1.001 \times 2^{-1}$.

To address the challenges posed by configuring significant bits with pre-aligned mantissas when encountering 0 weight values, we propose a 0-less signed INT format that maintains high compatibility with conventional signed INT formats for weight quantization. In the commonly used 2's complement format for signed integers, each bit position (indexed by $k$) represents either the value 0 or $2^{k-1}$ with binary states 0 or 1, except for the most significant bit (MSB), which represents either 0 or $-2^{MSB-1}$. When all binary digits in a 2's complement integer representation are set to 0, the resulting integer value is 0. To avoid complications with 0 weight values during computations with pre-aligned mantissas, the 0-less signed INT format is designed such that each bit position represents either $-2^{k-1}$ or $+2^{k-1}$ with binary states 0/1. The conversion from the conventional 2's complement integer value $W$ to the 0-less signed integer value $W'$ can be achieved with a simple linear equation: $W' = 2W + 1$. It also results in $YW' = 2YW + Y$ ($Y$: input activation vector) in the GEMM operation, requiring simple scaling and summation of activations for the conversion. Therefore, even when the weights are represented using 0-less signed INT format, we can utilize 2's complement multipliers and adders for the PE design by incorporating units for activation summation.

In general quantization algorithms, quantization offset and scale (typically FP values) are introduced to provide flexibility in asymmetric quantizer design, with these offset and scale factors used to adjust computation results after the GEMM operation. By carefully tuning the quantization offset and scale during the weight quantization process, it is possible to obtain weight values quantized into different INT formats (e.g. signed INT, unsigned INT, or even 0-less signed INT format) without altering the effectively represented values of each quantization stage. Therefore, asymmetric quantization demonstrates the same level of network accuracy regardless of INT formats [3].

The $p_w$-bit 0-less signed integer has $2^{p_w} - 1$ as its maximum absolute value, which means the maximum scale shifting of the pre-aligned mantissas after the weight multiplication is $2^{p_w} - 1$. Therefore, when the weights are represented in the 0-less signed INT format, the pre-aligned mantissas truncation with $\delta = p_w + 2$ extra bits can ensure computational reliability of the integer-based FP-INT GEMM.

### D. Chunk-based Mantissa Allocation

Fig. 9 shows the area/power reduction achieved by the FIGNA PE with the proposed pre-aligned mantissa truncation and 0-less integer values compared to the baseline FP-FP PE. The FIGNA PE shows remarkable reductions in both area and power consumption, reaching up to 89% and 88% respectively, for FP32 activations. However, for BF16 activations with INT8

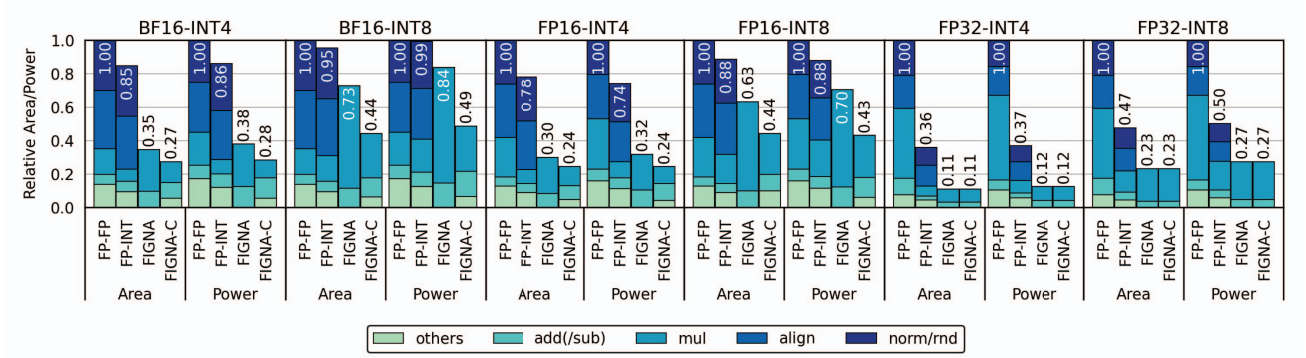


Fig. 9. The relative area and power consumption of FP-FP PE, FP-INT PE, FIGNA PE, and FIGNA-C PE for FP-INT MAC operation. The area and power is normalized with the total area and power of FP-FP PE for each data type, respectively. The evaluation includes three FP formats (BF16, FP16, and FP32) and two INT formats (INT4 and INT8).

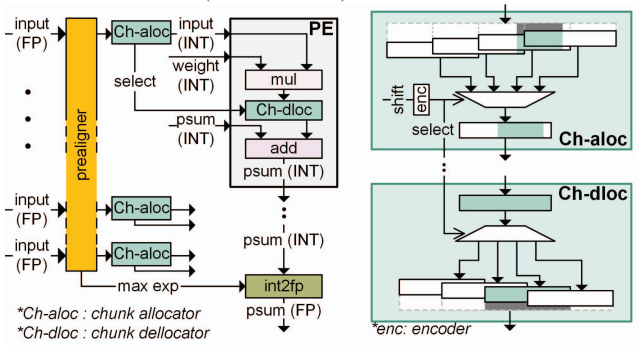


Fig. 10. The hardware components of FIGNA for chunk-based mantissa allocation. The pre-aligned mantissas are divided into chunks, and the chunk allocator selects and passes a set of consecutive chunks that contain all the valid mantissa values to the PE. The chunk deallocator ensures that the multiplication results are correctly located in the accumulator based on $select$, the chunk location. The chunk location is determined by the amount of mantissa shifting for the prealignment.

weights, the reduction is relatively marginal, with a 27% reduction in area and a 16% reduction in power. Similarly, the reduction is modest for FP16 activations. These findings suggest that BF16 and FP16 activations may not fully exploit the potential benefits of the FIGNA PE, prompting further investigation and optimization to better leverage the advantages of the FIGNA PE for these specific activation types.

In Section III-B, we discussed the calculation of extra bits for the pre-aligned mantissa, assuming that the data type of FP activation and the FP accumulator are the same. However, in practical scenarios, FP32 accumulators are often preferred for DNN processing, even when working with BF16 and FP16 activations [5], [34]. The reason for this preference is that FP32 accumulators possess higher-resolution mantissas, which effectively minimize rounding errors and lead to more accurate computation results. In such cases, the bit requirement of the pre-aligned mantissa should be calculated based on the accumulator data type, since the mantissa truncation should mimic the rounding of the conventional operation. Consequently, regardless of the data type of FP activations, the pre-aligned mantissa should consist of $p_{fp32} + n + 2$ bits, where $p_{fp32}$ is the bit precision of FP32 mantissa, which is 24. However, allocating $p_{fp32} + n + 2$ bits for the pre-aligned mantissa, irrespective of the data type of FP activations, can lead to inefficiency, particularly for lower-precision formats such as BF16 and FP16. For instance, when using 8-bit weights, the pre-aligned mantissas would require 34 bits. Yet, for BF16 and FP16 activations, only 8 and 11 bits of the pre-aligned mantissa are valid, respectively. Hence, a significant portion of the allocated bits may remain unused, leading to the under-utilization of computational resources.

To reduce the effective bit count of the pre-aligned mantissa, we propose a chunk-based mantissa allocation method (Fig. 10). In this method, we introduce the concept of a chunk, which is a set of binary digits, serving as the basic unit to represent pre-aligned values. For instance, when using a 7-bit chunks, it requires a total of 5 chunks to represent the original 34-bit pre-aligned mantissas. However, in the case of BF16 activation, only 2 consecutive chunks contain valid values, while the remaining 3 chunks are filled with zeros. To improve efficiency, the *chunk allocator* represents the pre-aligned mantissa by passing 14-bit values composed of the 2 consecutive chunks containing all the valid values, along with the position of these 2 chunks. Since there are 4 possible positions for the chunks containing valid values, this positional information requires additional 2 bits for representation. The chunk location (*select*) is determined based on the amount of mantissa shifting (*shift*) in the prealignment stage, as it defines the position of the valid values within the pre-aligned mantissa chunks. By adopting this chunk-based approach, we can effectively reduce the size of the pre-aligned mantissa from 34 bits to 16 bits, without any loss of relevant information. This reduction in the number of bits enables better utilization of computational resources, leading to improved efficiency in the FIGNA PE. Without the chunk-based approach, the FIGNA PE performs the multiplication between 35-bit signed-mantissa and the INT8 weights. In contrast, the chunk-based approach reduces the multiplier overhead by performing multiplication between 15-bit signed-mantissa and the INT8 weights. The *chunk deallocator* uses a 1-to-4 DEMUX to determine the required amount of shifting for the multiplication result, based on the chunk location. This alignment is performed before the accumulation process to ensure the proper adjustment of the scale of the multiplication result.

In this chunk-based mantissa allocation approach, finding the optimal chunk size is crucial due to the trade-off between reducing effective bits and minimizing multiplexing overhead.

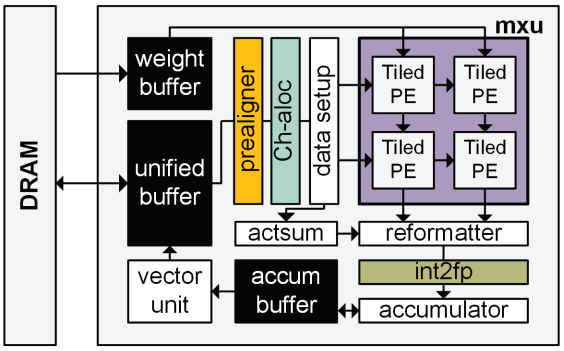


Fig. 11. High-level overview of FIGNA architecture

Higher-precision chunks reduce the number of chunks and multiplexing overhead but may introduce more redundant bits and larger multipliers. Conversely, lower-precision chunks reduce redundant bits and multiplier size but increase the number of chunks and potential multiplexing overhead. To find the optimal chunk size, we evaluated the overall efficiency of the FIGNA PE operation for different chunk sizes, and found that 7-bit chunks for BF16 activations and 5-bit chunks for FP16 activations show the best results. As a result, the FIGNA-C PE, a PE designed for integer-based FP-INT MAC and chunk-based mantissa allocation, significantly enhances area/power efficiency compared to other PE designs (Fig. 9). In the case of BF16 activations and INT8 weights, FIGNA-C PE improves area/power efficiency by 1.66/1.71 times compared to FIGNA PE without chunk-based allocation and by 2.27/2.04 times compared to the baseline FP-FP PE. For FP32 activations, 34-bit chunks achieve the best area/power efficiency, indicating that the pre-aligned mantissas should be fed to PEs without any multiplexing. Therefore, the design of FIGNA-C PE and FIGNA PE is identical for the FP32 cases.

Overall, the integer-based FP-INT MAC, combined with all the proposed optimization techniques, reduces PE area and power by 51% to 89% compared to the baseline FP-FP PE (Fig. 9). These results demonstrate the effectiveness of the proposed approach in enhancing the efficiency of FP-INT MAC operations.

### *E. Overall FIGNA Architecture*

Fig. 11 illustrates the overall architecture of FIGNA. First, the *prealigner* converts FP input activations to integer values and *chunk allocator* selects valid chunks among pre-aligned integer values. The *data setup* unit ensures the proper timing of input feeding to the matrix multiplication unit (*mxu*), which is designed based on a 2D systolic array architecture of Google TPU [19], [25]. The *mxu* processes FP-INT GEMM operation leveraging integer units of FIGNA-C PEs, following typical weight stationary dataflows. The PEs are equipped with registers for storing weight values, and the 2D weight matrix is directly mapped onto the 2D PE array. A vector of activations, used for inner product computations, is sequentially fed into the PE array. This vector is shared across the columns of the PE array to maximize input vector reuse, allowing multiple calculations with the same input. Each PE performs the MAC

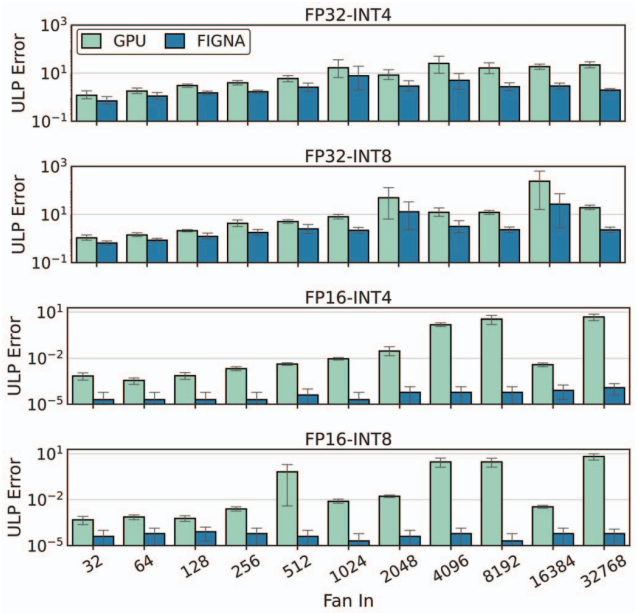


Fig. 12. The numerical error of FP-INT GEMM executed with the NVIDIA GPU and the proposed FIGNA in the comparison to the accurate operation. The fan-in values ranging from 32 to 32768 are evaluated to cover the large fan-in values of LLMs. Each error bar represents a 95% confidence interval.

operation between input activation and weight, and the MAC results are passed down to the next PE for accumulation. In this way, each column of PEs effectively computes the inner product of the GEMM. The PEs are organized in a tiled manner based on the weight-stationary dataflow [18]. At the end of the *mxu*, *int2fp* unit converts integer computation results back into FP values. When the fan-in exceeds the row count of the *mxu*, the input vector is divided into sub-vector to fit to the *mxu*. Each sub-vector is then sequentially fed into the *mxu* for processing. To complete the entire GEMM operation, the computation results from different sub-vectors are merged using *accumulator*, which is composed of FP32 adders. Additionally, *actsum* and *reformatter* are designed to support the conversion between the 2's complement INT format and the proposed 0-less signed INT format. In FIGNA, on-chip buffers are divided into three types. The *unified buffer* is responsible for buffering both input and output data. The *weight buffer* is dedicated to storing weight values, and the *accum buffer* is utilized to store the partial sums generated during computations.

## IV. Evaluation

### *A. Accuracy Evaluation*

*1) **Numerial Accuracy**:* In this section, we extensively analyze the numerical accuracy of the proposed FIGNA using various FP formats (FP32 and FP16) and INT formats (INT4 and INT8) commonly adopted in LLMs. We evaluate the fan-in values ranging from 32 to 32768 to cover the large fan-in value of LLMs. A dataset of 50,000 FP-INT inner product cases is sampled for each fan-in value. We use a Gaussian distribution to generate the INT weights, which mirrors the typical weight distribution in LLMs. The FP activations are sampled from a

TABLE I
ACCURACY RESULTS OF LANGUAGE MODELS INFERENCED USING NVIDIA GPU AND FIGNA. BERT, BLOOM, OPT MODELS ARE EVALUATED ON GLUE, PTB, AND WIKITEXT-2, RESPECTIVELY.

| | BERT-base | | | | | | | | BLOOM | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | QNLI | SST-2 | MRPC | STSB | QQP | MNLI-m/mm | CoLA | RTE | 560M | 1.1B | 1.7B | 3B | 7.1B | 176B | 176B' |
| GPU | 91.54 | 92.43 | 89.35 | 88.59 | 89.34 | 84.12/84.68 | 59.57 | 72.56 | 41.26 | 46.98 | 27.92 | 23.13 | 19.40 | 13.58 | 14.02 |
| FIGNA | 91.54 | 92.43 | 89.35 | 88.59 | 89.34 | 84.12/84.68 | 59.57 | 72.56 | 41.29 | 46.98 | 27.92 | 23.13 | 19.40 | 13.58 | 14.02 |
| | BERT-large | | | | | | | | OPT | | | | | | |
| | QNLI | SST-2 | MRPC | STSB | QQP | MNLI-m/mm | CoLA | RTE | 125M | 350M | 1.3B | 2.7B | 6.7B | 13B | 30B | 66B |
| GPU | 92.28 | 93.46 | 90.01 | 90.15 | 89.59 | 86.50/85.79 | 62.91 | 73.29 | 27.66 | 22.03 | 14.68 | 12.49 | 10.86 | 10.12 | 9.55 | 9.34 |
| FIGNA | 92.37 | 93.12 | 90.02 | 90.19 | 89.58 | 86.64/85.92 | 63.89 | 73.65 | 27.67 | 22.02 | 14.68 | 12.49 | 10.86 | 10.12 | 9.55 | 9.34 |

wide range of values, allowing us to demonstrate the operation with outliers.

Fig. 12 summarizes the ULP error of FP-INT GEMM with the NVIDIA GPU or the proposed FIGNA, compared to the accurate operation. When using FP32 activations, the proposed FIGNA exhibits slightly lower error compared to sthe GPU. The observed improvement is attributed to the conservative truncation of pre-aligned mantissas. Based on the error bound analysis in Section III-B, FIGNA is designed with sufficient extra bits $\delta$ to preserve numerical accuracy even in the worst-case scenario, where the maximum weight is multiplied by the minimum activation value, and the minimum weight is multiplied by the maximum activation value. However, in real computing scenarios, such conditions are infrequent, resulting in relatively small errors for FIGNA. For FP16, FIGNA demonstrates much lower error compared to the GPU. In conventional DNN computing systems, the accumulator uses FP32 format regardless of the data type of FP activations, which results in the pre-aligned mantissa in FIGNA consisting of $p_{fp32}+n+2$ bits, equivalent to 34 bits for cases with INT8 weight. This configuration leads to 23 extra mantissa bits for FP16 after the prealignment. Consequently, if the exponent difference between the maximum activation and the other activations falls within these extra bits, there is no truncation error in FIGNA, ensuring accurate operation. Since such large differences are also infrequent in real computing scenarios, FIGNA achieves numerical accuracy better than conventional FP units. As a result, the proposed FIGNA successfully preserves or even achieves better numerical accuracy compared to GPU in various computing scenarios for FP-INT GEMM operations.

*2) **Inference Accuracy**:* To assess the effectiveness of FIGNA in handling language model inference tasks, we evaluate the inference accuracy of language models when processed using NVIDIA GPU and FIGNA. We examine both encoder-based models (BERT [12]) and decoder-based models (BLOOM [48] and OPT [57]). For BERT, we evaluate its performance on the General Language Understanding Evaluation (GLUE) benchmark [46]. We consider two varients of BERT models, BERT-base and BERT-large, for the evaluation. BERT-base uses FP32 activations and INT8 weights, while BERT-large uses BF16 activations and INT8 weights. Regarding OPT and BLOOM, we measure the perplexity of text generation on the WikiText-2 [33] and Penn Treebank (PTB) [32] datasets, respectively. In this experiment, we cover all available size options for OPT and BLOOM. All these models use FP16 activations and INT8 weights. Additionally, we also report the perplexity of BLOOM-176B with INT4 weights (denoted as BLOOM-176B') for comprehensive analysis. Table I summarizes the evaluation results. Due to the preserved numerical accuracy in FIGNA, there is no noticeable difference between the inference results of the GPU and FIGNA.

TABLE II
HARDWARE CONFIGURATION OF FIGNA AND BASELINE DESIGNS

| | FP-FP | FP-INT | [23] | FIGNA |
|---|---|---|---|---|
| *mxu* | 128×128 | 128×128 | 128×(128×W*) | 128×128 |
| INT weight | ✗ | ✓ | ✓ | ✓ |
| INT unit | ✗ | ✗ | ✓ | ✓ |
| weight bitwidth scalability | ✓ | ✓ | ✗ | ✓ |
| chunk allocation | - | - | ✗ | ✓ |

** : weight bitwidth*

## B. Hardware Evaluation Setup

*1) **Hardware Configuration**:* We compare FIGNA with 3 baseline designs: FP-FP Engine, FP-INT Engine, and [23] (Table II). We include the comparison with [23] because their approach also addresses the challenges of weight-only quantized DNNs. Similar to FIGNA, [23] also employs the concept of prealignment to convert FP activations to INT values before performing computations. However, [23] is specifically designed to process weights with non-uniform quantization using the binary-coded format. In such cases, weights cannot be represented with traditional INT values, so the design from [23] tackles the GEMM operation in a bit-serial manner to leverage INT addition units. This is achieved by processing the summation of activation values where each PE handles the binary value of each weight bit position. For multi-bit weight cases, [23] merges the summation result of each weight bit position by multiplying the significance of the bit position (represented as an FP value for non-uniform quantization) and then performing add operations. For the comparison, the hardware design of [23] is properly modified to handle INT weight values. For FIGNA, chunk-based mantissa allocation

is adopted for FP16 and BF16 activations. However, it is not used in FP32 input cases.

To measure the area and power of the designs, we synthesize the designs using Synopsys Design Compiler with the 28nm CMOS technology, and measure power consumption using PrimePower [26]. Since LLM models are being quantized into various formats, all designs are evaluated for six cases: input based on FP32, FP16, and BF16 formats, and weight based on INT4 and INT8 formats. We compile SRAM memory array for input and weight buffers and implement a register file for accumulation buffer to support relatively frequent access. The access energy for SRAM and register file are measured as 0.1053 pJ/bit and 0.0625 pJ/bit, respectively. For off-chip memory characteristics, we use HBM2 memory which has the access energy of 3.9 pJ/bit and a bandwidth of 256 GB/s [18].

For a fair comparison, we adjust all the designs to have the same maximum performance (TOPS) target. FP-FP Engine, FP-INT Engine and FIGNA have a 128×128 systolic array. For [23], we use a 128×128×4 systolic array when the weight is 4-bit, and a 128×128×8 systolic array when the weight is 8-bit, in order to match the TOPS target. All designs are synthesized targeting a clock frequency of 100MHz. Note that the experiment is focused on assessing GEMM operations involved in linear layers, as these operations are the most significant components of LLM inference.

*2) **Simulator Setup**:* To measure the performance of each design, we implement a System-C based simulator [1], [4]. During the simulation, we obtain the DRAM access count, SRAM access count, and operation count for inference of each network, and also measure the latency. We calculate the energy caused by memory access by multiplying the memory access count with the energy per bit mentioned in Section IV-B1. Using the power measurement value from the logic synthesis, we calculate the energy per operation and multiply it by the operation count obtained from the simulation to determine the dynamic energy due to logic operation. We multiply the leakage power by the latency to obtain the leakage energy of the design. We evaluate all the linear layers that involve GEMM operation between input activations and weight matrix. Note that these GEMM operations account for the majority of total computing time primarily due to their large hidden dimension [21], [40].

*3) **Target Workload**:* We measure the TOPS/$mm^2$ and TOPS/W of FIGNA and baseline designs in processing all the language models introduced in Section IV-A, which include BERT, BLOOM and OPT. For the BERT models, which are encoder-based models, we assume that they encode sentences with a length of 128 tokens. Each linear layer in these models handles 128 input vectors simultaneously, effectively representing 128 batches of data. The BLOOM and OPT models are decoder-based models, and they perform text generation using an autoregressive approach. In our experiments, we utilize a batch size of 32 for theses models, taking into account recent research indicating that batching generative tasks in language models for real-world applications can be conducted efficiently without incurring additional computational costs with this batch size [54].

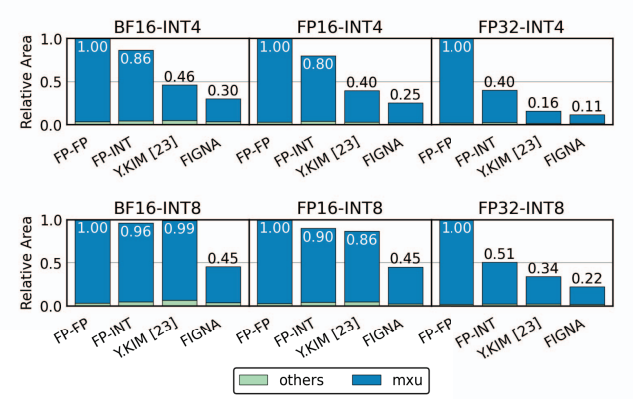


Fig. 13. Area breakdown of FIGNA and baseline designs

## C. Analysis on Area Efficiency

In this section, we compare the area efficiency (TOPS/$mm^2$) of FIGNA with the baseline designs.

Fig. 13 provides the area breakdown for each design with six different combinations of data types. Across the six data types, the *mxu* of FP-FP Engine has the biggest area due to the expensive FP unit used in the PE. On the other hand, FP-INT Engine reduces the complexity of the *mxu* by employing narrower bitwidths for the multipliers, leading to a reduction in the size of the *mxu*. Despite this reduction, the components required for alignment and normalization logic within *mxu* are relatively large, and hence *mxu* still retains a considerable area, with only up to 26.2% reduction on average. Both [23] and FIGNA further reduce the complexity of the *mxu* by employing pre-alignment, resulting in the area reduction by 46.5% and 70.3% on average, respectively.

A notable point is that when the input is BF16 or FP16 and the weight is INT8, FIGNA shows a significant reduction in area compared to other designs, while [23] shows either minimal reduction or even a slight increase in area. This is because each design exhibits different sensitivity to changes in weight bitwidth. FP-FP Engine internally converts the INT weight into FP format, ensuring that even with an increase in the weight bitwidth, the area of the PE remains unchanged. The FP-INT Engine experiences an increase in multiplier area directly proportional to the weight bitwidth. However, considering that the multiplier's contribution to the overall area is relatively small (Fig. 9), the increase in its area does not significantly affect the total area. On the other hand, [23] undergoes a notable increase in area, as the weight bitwidth increases. In [23], the weight is divided into 1-bit slices, which are then distributed to different columns of the *mxu*. Subsequently, an adder is employed to carry out the accumulation operation, and each bit plane is merged at the end of the *mxu*. This approach follows the design philosophy of the bit-serial computing, in which the number of adders and registers increases proportional to the weight's bitwidth. Consequently, the total area also experiences a substantial increase. In contrast, FIGNA uses a multiplier to process all

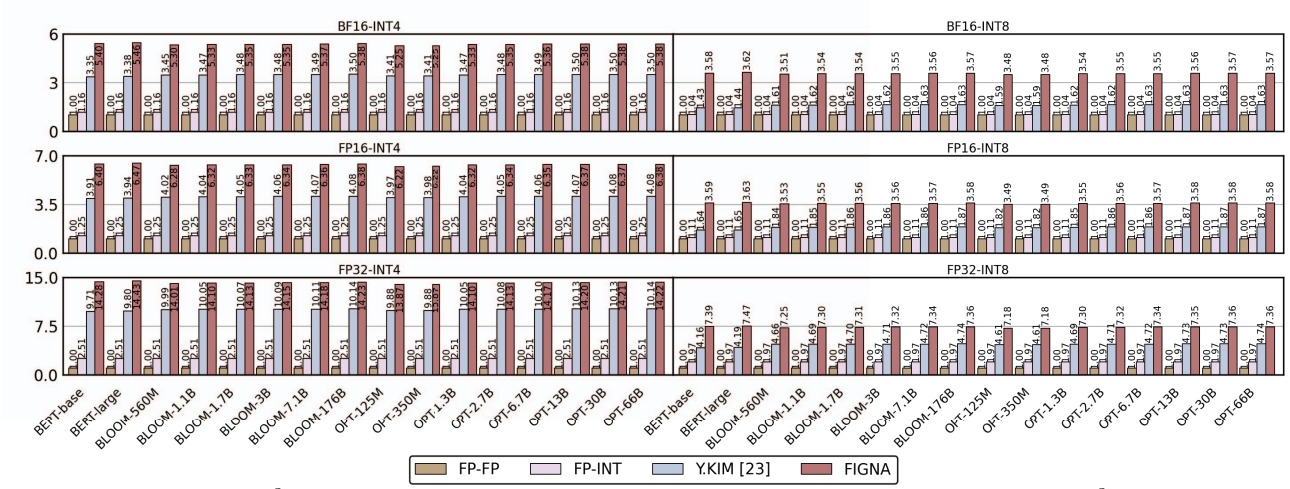


Fig. 14. Normalized TOPS/$mm^2$ of FIGNA and baseline designs for inferencing language models using six data types. TOPS/$mm^2$ is normalized with that of FP-FP Engine for each case.

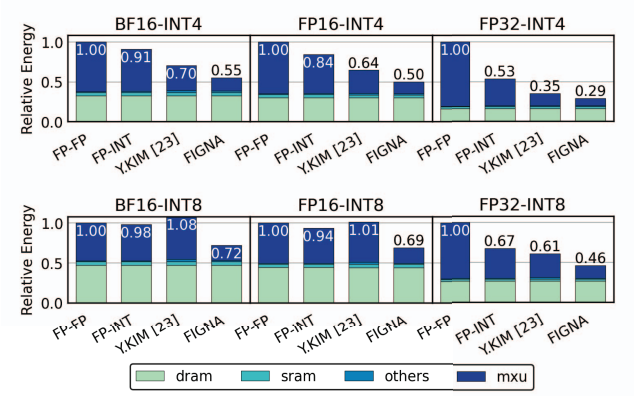


Fig. 15. Energy breakdown of FIGNA and baseline designs. Energy consumption during the OPT-66B inference is evaluated.

bits of the weight within a single column of the *mxu* in bit-parallel computing style. As a result, it requires much smaller additional registers when the bitwidth of weights increases compared to [23]. Therefore, when the weights use INT8 format, FIGNA demonstrates a smaller increase in area than [23], making it a more efficient choice for handling larger bitwidth of weights.

Fig. 14 shows the normalized TOPS/$mm^2$ for the six data type combinations. For each data type, all TOPS/$mm^2$ is normalized against that of FP-FP Engine. FIGNA shows higher TOPS/$mm^2$ than baseline designs across all models and data types due to its notable reduction in logic complexity facilitated by prealigning and chunk selection. Compared to FP-FP Engine and FP-INT Engine, FIGNA exhibits up to 14.3 times higher TOPS/$mm^2$. In addition, FIGNA outperforms [23] by up to 2.51 times thanks to the chunk selection feature and superior scalability with respect to weight bitwidth. The results indicate that optimizing PE complexity by utilizing integer-based unit and chunk selection techniques can lead to substantial improvements in TOPS/$mm^2$, offering promising directions for efficient processing in various LLM models and data types.

### D. Analysis on Energy Efficiency

In this section, we compare the energy efficiency (TOPS/W) of FIGNA with the baseline designs.

Fig. 15 shows the energy breakdown for each data type when running OPT-66B. Similar to the trend in the area analysis, the proposed FIGNA exhibits lower energy consumption than baseline designs across all data formats. In INT4 cases, the energy consumption follows the expected order: FP-FP Engine, FP-INT Engine, [23], and FIGNA. However, for BF16-INT8 and FP16-INT8 cases, the increase in energy consumption in [23] is particularly noticeable compared to INT4 cases. Interestingly, in such scenarios, the energy consumption in [23] becomes even higher than that of FP-FP Engine. The unexpected behavior is due to signficant increase in clock distribution power in [23], which is needed to accomodate the increased number of registers in proportion to the increased weight bitwidth, a characteristic of bit-serial computing style. In contrast, FIGNA requires fewer additional registers when weight bitwidth is increased, resulting in significantly lower energy consumption than baseline designs, even in INT8 cases.

Fig. 16 presents the comparison of TOPS/W for the six data types when performing inference for various LLM models. TOPS/W demonstrates similar trend to that of TOPS/$mm^2$ for reasons previously discussed in Section IV-C. For all models and data formats, FIGNA demonstrates highest TOPS/W. The advantage of FIGNA over [23] is most noticeable in BF16-INT8 and FP16-INT8 format cases.

### E. Dependency on On-Chip Buffer Size

In this section, we explore whether FIGNA consistently exhibits better TOPS/W and TOPS/$mm^2$ across various on-chip buffer sizes. In LLM inference, DRAM access energy accounts for a significant portion of the overall energy consumption. One effective approach to mitigate this is increasing the size of on-chip buffers, which helps reduce the amount of DRAM access, thereby potentially achieving higher TOPS/W. However, on-chip buffers have a larger area per bit compared to logic, which could lead to a trade-off: incorporating more on-chip

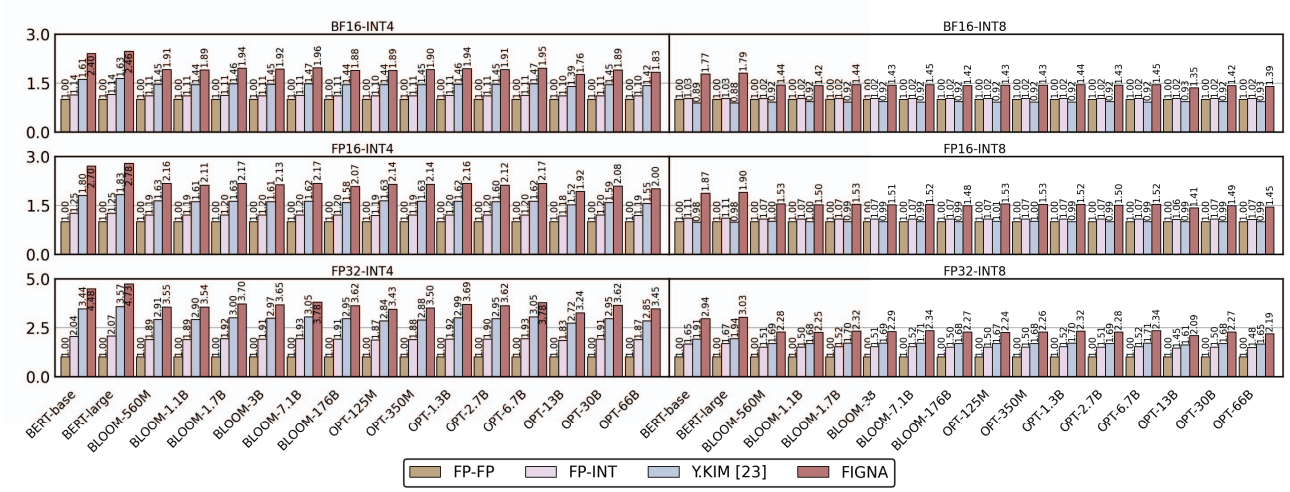


Fig. 16. Normalized TOPS/W of FIGNA and baseline designs for inferencing language models using six data types. TOPS/W is normalized with that of FP-FP Engine for each case.

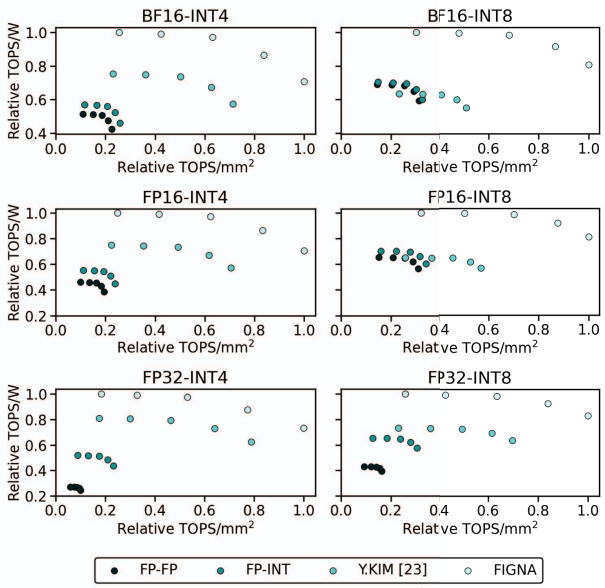


Fig. 17. Trade-off between TOPS/W and TOPS/$mm^2$ of FIGNA and baseline designs with different on-chip buffer sizes.

buffers may increase the overall chip area, potentially reducing TOPS/$mm^2$. Note that the TOPS/$mm^2$ values examined in this section include SRAM area, while the TOPS/$mm^2$ values examined in previous sections only considered the logic area.

Fig. 17 shows the normalized TOPS/W and TOPS/$mm^2$ for OPT-1.3B with a fixed *mxu* size as mentioned in Section IV-B1, while varying the on-chip buffer size. It can be observed that increasing the on-chip buffer size leads to an increase in TOPS/W and a decrease in TOPS/$mm^2$ for all designs. However, all data points for FIGNA are observed in the upper-right corner of the graph compared to baseline designs, indicating that FIGNA outperforms other designs in both TOPS/$mm^2$ and TOPS/W, irrespective of memory sizes.

## V. Conclusion

In this paper, we propose a novel computing scheme and hardware architecture called FIGNA to effectively accelerate quantized models in the FP-INT format, aiming to reduce the computational cost of Large Language Models (LLMs). FIGNA achieves this by converting input data from FP format to INT format using shared exponents before involving them in computations, enabling the construction of efficient integer-based PE. A key difference from previous works that also used shared exponents of FP values is that FIGNA preserves the numerical accuracy of FP-INT MAC computations while previous works try to preserve the neural network accuracy with higher level of numerical errors. For FIGNA, we mathematically derive the required number of extra bits for the pre-aligned mantissas to preserve the numerical accuracy. Ensuring numerical accuracy reduces the burden on users, as they do not need to invest efforts in retraining or fine-tuning models to compensate for any numerical errors caused by hardware constraints. Experimental results demonstrate that when using quantized networks in FP16-INT8, FIGNA achieves higher area efficiency (TOPS/$mm^2$) and energy efficiency (TOPS/W) of 3.56x and 1.55x, respectively, compared to the baseline FP-FP Engine.

## Acknowledgements

This work was supported in part by Institute of Information communications Technology Planning Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2020R1A2C2004329, NeuroHub: Web-based Open Simulation Platform for Collaborative In-Memory Neural Network Hardware Research, IITP-2023-RS-2023-00256081: artificial intelligence semiconductor support program to nurture the best talents, No. 2021-0-01343: Artificial Intelligence Graduate School Program (Seoul National University)), Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2022R1A6A3A01087416), BK21 FOUR program at Seoul National University, and Inter-university Semiconductor Research Center at Seoul National University. The EDA tool was supported by the IC Design Education Center (IDEC).

## REFERENCES


[1] I. S. Association *et al.*, "Ieee standard for standard systemc language reference manual," *IEEE Computer Society*, 2012.

[2] Y. Bai, Y.-X. Wang, and E. Liberty, "Proxquant: Quantized neural networks via proximal operators," *arXiv preprint arXiv:1810.00861*, 2018.

[3] Y. Bhalgat, J. Lee, M. Nagel, T. Blankevoort, and N. Kwak, "Lsq+: Improving low-bit quantization through learnable offsets and better initialization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 696–697.

[4] D. C. Black and J. Donovan, *SystemC: From the ground up*. Springer, 2004.

[5] N. Burgess, J. Milanovic, N. Stephens, K. Monachopoulos, and D. Mansell, "Bfloat16 processing for neural networks," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, 2019, pp. 88–91.

[6] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, "Palm: Scaling language modeling with pathways," 2022.

[7] I. Chung, B. Kim, Y. Choi, S. J. Kwon, Y. Jeon, B. Park, S. Kim, and D. Lee, "Extremely low bit transformer quantization for on-device neural machine translation," *arXiv preprint arXiv:2009.07453*, 2020.

[8] S. Dai, R. Venkatesan, M. Ren, B. Zimmer, W. Dally, and B. Khailany, "Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 873–884, 2021.

[9] B. Darvish Rouhani, D. Lo, R. Zhao, M. Liu, J. Fowers, K. Ovtcharov, A. Vinogradsky, S. Massengill, L. Yang, R. Bittner *et al.*, "Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point," *Advances in neural information processing systems*, vol. 33, pp. 10 271–10 281, 2020.

[10] B. Darvish Rouhani, R. Zhao, V. Elango, R. Shafipour, M. Hall, M. Mesmakhosroshahi, A. More, L. Melnick, M. Golub, G. Varatkar *et al.*, "With shared microexponents, a little shifting goes a long way," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[11] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Llm.int8 (): 8-bit matrix multiplication for transformers at scale," *arXiv preprint arXiv:2208.07339*, 2022.

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[13] H. Fan, G. Wang, M. Ferianc, X. Niu, and W. Luk, "Static block floating-point quantization for convolutional neural networks on fpga," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 28–35.

[14] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[15] E. Hokenek, R. Montoye, and P. Cook, "Second-generation risc floating point with multiply-add fused," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 5, pp. 1207–1213, 1990.

[16] Y. Jeon, C. Lee, E. Cho, and Y. Ro, "Mr. biq: Post-training non-uniform quantization based on minimizing the reconstruction error," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 329–12 338.

[17] Y. Jeon, B. Park, S. J. Kwon, B. Kim, J. Yun, and D. Lee, "Biqgemm: matrix multiplication with lookup table for binary-coding-based quantized dnns," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–14.

[18] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma *et al.*, "Ten lessons from three generations shaped google's tpuv4i: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1–14.

[19] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[20] S. Kim, C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. W. Mahoney, and K. Keutzer, "Squeezellm: Dense-and-sparse quantization," *arXiv preprint arXiv:2306.07629*, 2023.

[21] S. Kim, C. Hooper, T. Wattanawong, M. Kang, R. Yan, H. Genc, G. Dinh, Q. Huang, K. Keutzer, M. W. Mahoney *et al.*, "Full stack optimization of transformer inference: a survey," *arXiv preprint arXiv:2302.14017*, 2023.

[22] Y. J. Kim, R. Henry, R. Fahim, and H. H. Awadalla, "Who says elephants can't run: Bringing large scale moe models into cloud scale production," 2022.

[23] Y. Kim, J. Jang, J. Lee, J. Park, J. Kim, B. Kim, S. J. Kwon, D. Lee *et al.*, "Winning both the accuracy of floating point activation and the simplicity of integer arithmetic," in *The Eleventh International Conference on Learning Representations*, 2023.

[24] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, L. Hornof *et al.*, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[25] S. Kung, "Vlsi array processors," *IEEE ASSP Magazine*, vol. 2, no. 3, pp. 4–22, 1985.

[26] P. Kurup and T. Abbasi, *Logic synthesis using Synopsys®*. Springer Science & Business Media, 2012.

[27] S. J. Kwon, J. Kim, J. Bae, K. M. Yoo, J.-H. Kim, B. Park, B. Kim, J.-W. Ha, N. Sung, and D. Lee, "Alphatuning: Quantization-aware parameter-efficient adaptation of large-scale pre-trained language models," *arXiv preprint arXiv:2210.03858*, 2022.

[28] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu, "Brecq: Pushing the limit of post-training quantization by block reconstruction," *arXiv preprint arXiv:2102.05426*, 2021.

[29] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, "High-performance fpga-based cnn accelerator with block-floating-point arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1874–1885, 2019.

[30] J. Lin, J. Tang, H. Tang, S. Yang, X. Dang, and S. Han, "Awq: Activation-aware weight quantization for llm compression and acceleration," 2023.

[31] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, "Estimating the carbon footprint of bloom, a 176b parameter language model," *arXiv preprint arXiv:2211.02001*, 2022.

[32] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," 1993.

[33] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[34] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," 2018.

[35] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu *et al.*, "Fp8 formats for deep learning," *arXiv preprint arXiv:2209.05433*, 2022.

[36] J.-M. Muller, N. Brisebarre, F. De Dinechin, C.-P. Jeannerod, V. Lefevre, G. Melquiond, N. Revol, D. Stehlé, S. Torres *et al.*, *Handbook of floating-point arithmetic*. Springer, 2018.

[37] NVIDIA, "Fastertransformer," accessed: 2023-10-10. [Online]. Available: https://github.com/NVIDIA/FasterTransformer

[38] NVIDIA, "Nvidia a100 tensor core gpu," accessed: 2023-07-19. [Online]. Available: https://www.nvidia.com/ko-kr/data-center/a100/

[39] G. Park, B. Park, S. J. Kwon, B. Kim, Y. Lee, and D. Lee, "nuqmm: Quantized matmul for efficient inference of large-scale generative language models," *arXiv preprint arXiv:2206.09557*, 2022.

[40] Y. Qin, Y. Wang, D. Deng, Z. Zhao, X. Yang, L. Liu, S. Wei, Y. Hu, and S. Yin, "Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–14.

[41] E. Quinnell, E. E. Swartzlander, and C. Lemonds, "Floating-point fused multiply-add architectures," in *2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers*, 2007, pp. 331–337.

[42] J. Richard Shewchuk, "Adaptive precision floating-point arithmetic and fast robust geometric predicates," *Discrete & Computational Geometry*, vol. 18, pp. 305–363, 1997.

[43] Z. Song, Z. Liu, and D. Wang, "Computation error analysis of block floating point arithmetic oriented convolution neural network accelerator design," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[44] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023.

[45] A. K. Vijayakumar, M. Cogswell, R. R. Selvaraju, Q. Sun, S. Lee, D. Crandall, and D. Batra, "Diverse beam search: Decoding diverse solutions from neural sequence models," 2018.

[46] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[47] J. H. Wilkinson, *Rounding errors in algebraic processes*. Courier Corporation, 1994.

[48] B. Workshop, :, T. L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé, J. Tow, A. M. Rush, S. Biderman, A. Webson, P. S. Ammanamanchi, T. Wang, B. Sagot, N. Muennighoff, A. V. del Moral, O. Ruwase, R. Bawden, S. Bekman, A. McMillan-Major, I. Beltagy, H. Nguyen, L. Saulnier, S. Tan, P. O. Suarez, V. Sanh, H. Laurençon, Y. Jernite, J. Launay, M. Mitchell, C. Raffel, A. Gokaslan, A. Simhi, A. Soroa, A. F. Aji, A. Alfassy, A. Rogers, A. K. Nitzav, C. Xu, C. Mou, C. Emezue, C. Klamm, C. Leong, D. van Strien, D. I. Adelani, D. Radev, E. G. Ponferrada, E. Levkovizh, E. Kim, E. B. Natan, F. D. Toni, G. Dupont, G. Kruszewski, G. Pistilli, H. Elsahar, H. Benyamina, H. Tran, I. Yu, I. Abdulmumin, I. Johnson, I. Gonzalez-Dios, J. de la Rosa, J. Chim, J. Dodge, J. Zhu, J. Chang, J. Frohberg, J. Tobing, J. Bhattacharjee, K. Almubarak, K. Chen, K. Lo, L. V. Werra, L. Weber, L. Phan, L. B. allal, L. Tanguy, M. Dey, M. R. Muñoz, M. Masoud, M. Grandury, M. Šaško, M. Huang, M. Coavoux, M. Singh, M. T.-J. Jiang, M. C. Vu, M. A. Jauhar, M. Ghaleb, N. Subramani, N. Kassner, N. Khamis, O. Nguyen, O. Espejel, O. de Gibert, P. Villegas, P. Henderson, P. Colombo, P. Amuok, Q. Lhoest, R. Harliman, R. Bommasani, R. L. López, R. Ribeiro, S. Osei, S. Pyysalo, S. Nagel, S. Bose, S. H. Muhammad, S. Sharma, S. Longpre, S. Nikpoor, S. Silberberg, S. Pai, S. Zink, T. T. Torrent, T. Schick, T. Thrush, V. Danchev, V. Nikoulina, V. Laippala, V. Lepercq, V. Prabhu, Z. Alyafeai, Z. Talat, A. Raja, B. Heinzerling, C. Si, D. E. Taşar, E. Salesky, S. J. Mielke, W. Y. Lee, A. Sharma, A. Santilli, A. Chaffin, A. Stiegler, D. Datta, E. Szczechla, G. Chhablani, H. Wang, H. Pandey, H. Strobelt, J. A. Fries, J. Rozen, L. Gao, L. Sutawika, M. S. Bari, M. S. Al-shaibani, M. Manica, N. Nayak, R. Teehan, S. Albanie, S. Shen, S. Ben-David, S. H. Bach, T. Kim, T. Bers, T. Fevry, T. Neeraj, U. Thakker, V. Raunak, X. Tang, Z.-X. Yong, Z. Sun, S. Brody, Y. Uri, H. Tojarieh, A. Roberts, H. W. Chung, J. Tae, J. Phang, O. Press, C. Li, D. Narayanan, H. Bourfoune, J. Casper, J. Rasley, M. Ryabinin, M. Mishra, M. Zhang, M. Shoeybi, M. Peyrounette, N. Patry, N. Tazi, O. Sanseviero, P. von Platen, P. Cornette, P. F. Lavallée, R. Lacroix, S. Rajbhandari, S. Gandhi, S. Smith, S. Requena, S. Patil, T. Dettmers, A. Baruwa, A. Singh, A. Cheveleva, A.-L. Ligozat, A. Subramonian, A. Névéol, C. Lovering, D. Garrette, D. Tunuguntla, E. Reiter, E. Taktasheva, E. Voloshina, E. Bogdanov, G. I. Winata, H. Schoelkopf, J.-C. Kalo, J. Novikova, J. Z. Forde, J. Clive, J. Kasai, K. Kawamura, L. Hazan, M. Carpuat, M. Clinciu, N. Kim, N. Cheng, O. Serikov, O. Antverg, O. van der Wal, R. Zhang, R. Zhang, S. Gehrmann, S. Mirkin, S. Pais, T. Shavrina, T. Scialom, T. Yun, T. Limisiewicz, V. Rieser, V. Protasov, V. Mikhailov, Y. Pruksachatkun, Y. Belinkov, Z. Bamberger, Z. Kasner, A. Rueda, A. Pestana, A. Feizpour, A. Khan, A. Faranak, A. Santos, A. Hevia, A. Unldreaj, A. Aghagol, A. Abdollahi, A. Tammour, A. HajiHosseini, B. Behroozi, B. Ajibade, B. Saxena, C. M. Ferrandis, D. Contractor, D. Lansky, D. David, D. Kiela, D. A. Nguyen, E. Tan, E. Baylor, E. Ozoani, F. Mirza, F. Ononiwu, H. Rezanejad, H. Jones, I. Bhattacharya, I. Solaiman, I. Sedenko, I. Nejadgholi, J. Passmore, J. Seltzer, J. B. Sanz, L. Dutra, M. Samagaio, M. Elbadri, M. Mieskes, M. Gerchick, M. Akinlolu, M. McKenna, M. Qiu, M. Ghauri, M. Burynok, N. Abrar, N. Rajani, N. Elkott, N. Fahmy, O. Samuel, R. An, R. Kromann, R. Hao, S. Alizadeh, S. Shubber, S. Wang, S. Roy, S. Viguier, T. Le, T. Oyebade, T. Le, Y. Yang, Z. Nguyen, A. R. Kashyap, A. Palasciano, A. Callahan, A. Shukla, A. Miranda-Escalada, A. Singh, B. Beilharz, B. Wang, C. Brito, C. Zhou, C. Jain, C. Xu, C. Fourrier, D. L. Periñán, D. Molano, D. Yu, E. Manjavacas, F. Barth, F. Fuhrimann, G. Altay, G. Bayrak, G. Burns, H. U. Vrabec, I. Bello, I. Dash, J. Kang, J. Giorgi, J. Golde, J. D. Posada, K. R. Sivaraman, L. Bulchandani, L. Liu, L. Shinzato, M. H. de Bykhovetz, M. Takeuchi, M. Pàmies, M. A. Castillo, M. Nezhurina, M. Sänger, M. Samwald, M. Cullan, M. Weinberg, M. D. Wolf, M. Mihaljcic, M. Liu, M. Freidank, M. Kang, N. Seelam, N. Dahlberg, N. M. Broad, N. Muellner, P. Fung, P. Haller, R. Chandrasekhar, R. Eisenberg, R. Martin, R. Canalli, R. Su, R. Su, S. Cahyawijaya, S. Garda, S. S. Deshmukh, S. Mishra, S. Kiblawi, S. Ott, S. Sang-aroonsiri, S. Kumar, S. Schweter, S. Bharati, T. Laud, T. Gigant, T. Kainuma, W. Kusa, Y. Labrak, Y. S. Bajaj, Y. Venkatraman, Y. Xu, Y. Xu, Y. Xu, Z. Tan, Z. Xie, Z. Ye, M. Bras, Y. Belkada, and T. Wolf, "Bloom: A 176b-parameter open-access multilingual language model," 2023.

[49] C.-J. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. Aga, J. Huang, C. Bai *et al.*, "Sustainable ai: Environmental implications, challenges and opportunities," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 795–813, 2022.

[50] X. Wu, Z. Yao, M. Zhang, C. Li, and Y. He, "Xtc: Extreme compression for pre-trained transformers made simple and efficient," *Advances in Neural Information Processing Systems*, vol. 35, pp. 3217–3231, 2022.

[51] G. Xiao, J. Lin, M. Seznec, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," *arXiv preprint arXiv:2211.10438*, 2022.

[52] C. Xu, J. Yao, Z. Lin, W. Ou, Y. Cao, Z. Wang, and H. Zha, "Alternating multi-bit quantization for recurrent neural networks," *arXiv preprint arXiv:1802.00150*, 2018.

[53] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He, "Zeroquant: Efficient and affordable post-training quantization for large-scale transformers," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 168–27 183, 2022.

[54] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-Based} generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.

[55] Z. Yuan, L. Niu, J. Liu, W. Liu, X. Wang, Y. Shang, G. Sun, Q. Wu, J. Wu, and B. Wu, "Rptq: Reorder-based post-training quantization for large language models," *arXiv preprint arXiv:2304.01089*, 2023.

[56] S. Q. Zhang, B. McDanel, and H. Kung, "Fast: Dnn training under variable precision block floating point with stochastic rounding," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 846–860.

[57] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "Opt: Open pre-trained transformer language models," 2022.