PICACHU: Plug-In CGRA Handling Upcoming Nonlinear Operations in LLMs

Jiajun Qin[*]
New York University
New York, NY, USA
hobbitqia@gmail.com

Tianhua Xia
New York University
New York, NY, USA
tx856@nyu.edu

Cheng Tan[†]
Google
Mountain View, CA, USA
chengtan@google.com

Jeff Zhang
Arizona State University
Tempe, AZ, USA
jeffzhang@asu.edu

Sai Qian Zhang
New York University
New York, NY, USA
sai.zhang@nyu.edu

Abstract

Large language models (LLMs) have revolutionized natural language processing (NLP) domain by achieving state-of-the-art performance across a range of benchmarks. However, nonlinear operations in LLMs significantly contribute to inference latency and present unique challenges that have not been encountered previously. Addressing these challenges requires accelerators that combine efficiency, flexibility, and support for user-defined precision. Our analysis reveals that Coarse-Grained Reconfigurable Arrays (CGRAs) provide an effective solution, offering a balance of performance and flexibility tailored to domain-specific workloads.

This paper introduces PICACHU, a plug-in coarse-grained reconfigurable accelerator tailored to efficiently handle nonlinear operations by using custom algorithms and a dedicated compiler toolchain. PICACHU is the first to target all nonlinear operations within LLMs and to consider CGRA as a plug-in accelerator for LLM inference. Our evaluation shows that PICACHU achieves speedups of 1.86× and 1.55× over prior state-of-the-art accelerators in LLM inference.

CCS Concepts: • **Hardware → Application specific instruction set processors**; • **Computer systems organization → Single instruction, multiple data**; • **Computing methodologies → Neural networks**.

Keywords: Domain Specific Architecture (DSA), Coarse-Grained Reconfigurable Array (CGRA), Large Language Models (LLM)

[*]Also with Zhejiang University.
[†]Also with Arizona State University.

ACM Reference Format:
Jiajun Qin, Tianhua Xia, Cheng Tan, Jeff Zhang, and Sai Qian Zhang. 2025. PICACHU: Plug-In CGRA Handling Upcoming Nonlinear Operations in LLMs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3676641.3716013

1 Introduction

In the past decade, the landscape of artificial intelligence (AI) has undergone a profound transformation, with Transformer-based Large Language Models (LLMs) redefining the capabilities of AI. These models have emerged as powerful tools across various domains, including natural language processing (NLP) [14, 20, 50, 60, 84, 87, 88, 117, 123, 124] and computer vision (CV) [2, 8, 21, 66, 68, 121, 137], reshaping both industry and human life. Despite their transformative potential, transformer-based models pose substantial challenges due to their higher computational complexity relative to traditional neural networks like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). This underscores the growing demand for hardware acceleration. In response, the focus of AI accelerator development has increasingly shifted toward optimizing performance for transformer-based LLMs, which blend linear operations, such as matrix multiplication, with nonlinear operations like softmax and normalization, marking a significant trend in AI hardware innovation.

Although numerous hardware accelerators have been explored recently, the majority have focused primarily on enhancing the efficiency of General Matrix Multiplication (GEMM) operations [30, 31, 39, 59, 98, 138], often through algorithmic approaches such as pruning [32, 70, 72, 85, 86, 119, 149, 151], quantization [23, 30, 53, 65, 130, 139, 147, 152], and dataflow optimization [33, 36, 105, 133]. However, few of these studies have delved deeply into the implementation of nonlinear operations. As highlighted by early works [28, 49, 75, 136], during LLMs inferencing, nonlinear operations account for an even greater share of the computational and

Categories	Nonlinear Operations	Mathematical Operator	Representative LLMs
Activation Function	Softmax$(x_i) := \frac{\exp(x_i)}{\sum_{j=1}^{k}\exp(x_j)} = \frac{\exp(x_i-u)}{\sum_{j=1}^{k}\exp(x_j-u)}$; $u = \max_{j=1} x_j$	Division, Exponential	All
	$\mathrm{ReLU}(x) := \max(0, x)$	Maximum	OPT [145], T5 [90]
	$\mathrm{GeLU}(x) := 0.5x\left(1 + \mathrm{Tanh}(\sqrt{2/\pi}(x + 0.044715x^3)\right)$; $\mathrm{Tanh}(x) = (\exp(x) + \exp(-x))/(\exp(x) - \exp(-x))$	Division, Exponential	GPT [14, 84, 87, 88], BLOOM [57], Falcon [83], PanGu-α [144], Jurassic-1 [64], Gopher [89]
	$\mathrm{GeGLU}(x) := \mathrm{GeLU}(xW + b) \oplus (xV + c)$	Division, Exponential	LaMDA [110], GLM-130B [143]
	$\mathrm{SwiGLU}(x) := \mathrm{SiLU}(xW + b) \oplus (xV + c)$; $\mathrm{SiLU}(x) = x \cdot \mathrm{sigmoid}(x) = x \cdot \frac{1}{1+\exp(-x)}$	Division, Exponential	PaLM [17], LLaMA [113, 114], Qwen [7], DeepSeek [11], InternLM [15], Yi [135]
Normalization Function	LayerNorm$(x_i) := \frac{x_i-\mu}{\sigma}$; $\mu = \frac{1}{C}\sum_{i=1}^{C} x_i, \sigma = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(x_i-\mu)^2 + \epsilon}$	Inverted Square Root	GPT [14, 84, 87, 88], BLOOM [57], BERT [20], OPT [145], PanGu-α [144], Jurassic-1 [64]
	RMSNorm$(x_i) := \frac{x_i}{\sigma}$; $\sigma = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(x_i)^2 + \epsilon}$	Inverted Square Root	LLaMA [113, 114], T5 [90], Mistral [43], Qwen [7], DeepSeek [11], Gopher [89]
Positional Embedding	$\mathrm{RoPE}\begin{pmatrix} x_{2i-1} \\ x_{2i} \end{pmatrix} = \begin{pmatrix} x_{2i-1}\cos(m\theta_i) - x_{2i}\sin(m\theta_i) \\ x_{2i-1}\sin(m\theta_i) + x_{2i}\cos(m\theta_i) \end{pmatrix}$; $\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \ldots, d/2]$	Sine, Cosine	GPTNeo-20B [13], LLaMA [113, 114], PaLM [17], GLM-130B [143], Qwen [7], DeepSeek [11]

Table 1. Overview of nonlinear operations supported by PICACHU. Element-wise nonlinear operations are shown in black, while operations involving a reduction step followed by element-wise operations are highlighted in blue.

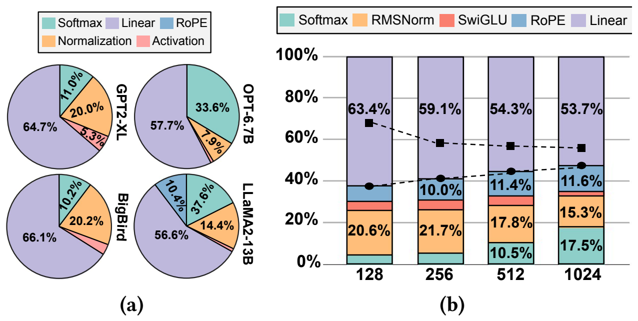


Figure 1. (a) Runtime breakdown for GPT2-XL, OPT-6.7B, BigBird and LLaMA2-13B execution with a sequence length of 1024. (b) Runtime breakdown for LLaMA2-7B execution across different sequence lengths.

implementation cost. This contrasts with traditional neural networks (e.g., CNN), where matrix multiplication typically dominates end-to-end runtime [28, 82, 92].

To illustrate the cost of nonlinear operations, we profile the GPT2-XL [88], OPT-6.7B [145], BigBird [141], LLaMA2-7B [114] and LLaMA2-13B models from Hugging Face, using half-precision (FP16) on an A100 GPU across samples of various sequence lengths. Figure 1 shows that nonlinear operations, such as softmax, layernorm, GeLU, and ReLU, have become a major bottleneck in processing latency. This issue intensifies with longer sequence lengths of 1024, with these operations accounting for up to 46.3% of the total inference latency.

However, supporting the nonlinear operations within LLMs poses unique challenges that previous research has yet to fully address. Firstly, unlike traditional neural networks, modern LLMs employ a broader and more complex range of nonlinear operations. As highlighted in Table 1, the variety of these operations has grown considerably. Traditional reliance on dedicated hardware units for nonlinear computations is no longer practical given this diversification. Secondly, LLMs are highly sensitive to the accuracy of these nonlinear operations, demanding high precision in the computations [10, 38, 140]. Prior works mainly focus on solely quantizing the linear layers while keeping the nonlinear operations in floating-point format to maintain the accuracy [23, 65, 96, 130, 132]. Thirdly, computing nonlinear operations within LLMs in integer arithmetic is challenging. While some works in deep neural networks (DNNs) [41, 49] address this, our evaluation in Table 2 shows that these methods cannot be directly applied to LLMs due to large accuracy loss. Therefore, nonlinear operations require careful consideration at the hardware design stage for LLM acceleration.

Consequently, the nonlinear acceleration solutions proposed in prior works [49, 75, 136] face significant limitations when utilized for practical LLM deployment. These solutions often lack the flexibility to accommodate the diverse range of nonlinear operations used in modern LLMs, as well as potential future expansions in the nonlinear operation landscape. Alternatively, some prior works may not have been validated on actual LLM workloads, lacking comprehensive accuracy evaluations [28]. These distinct challenges call for a holistic approach to accelerating nonlinear operations, one that can adapt to the evolving requirements of advanced LLMs.

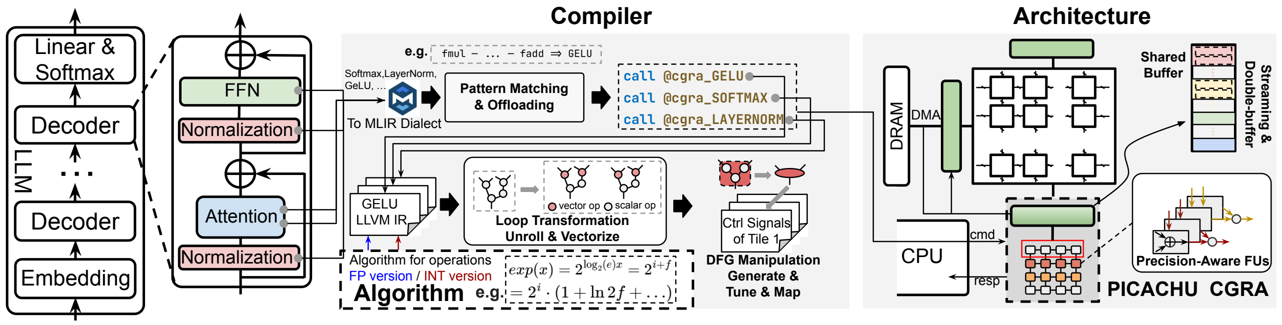


Figure 2. An overview of PICACHU.

Despite these challenges, we observe the opportunities to efficiently execute nonlinear operations. First, the nonlinearity in LLMs is introduced through a small set of basic nonlinear mathematical operators. Second, nonlinear operations share common loop-based computational patterns. Third, these operations exhibit high computational intensity.

These observations fit naturally with Coarse-Grained Reconfigurable Array (CGRA), a promising architectural solution for nonlinear operation acceleration. CGRAs are particularly well-suited for loop-based nonlinear operations due to their time-multiplexing and spatial-multiplexing capabilities [47, 62, 69]. Furthermore, CGRAs offer ultra flexible data flow, enabling support for a wide variety of nonlinear operations and data formats. This adaptability ensures that the accelerator can keep up with the rapidly evolving nonlinear operation landscape in LLMs, as new operations can be quickly implemented using basic arithmetic and control primitives on CGRAs.

Additionally, systolic arrays, widely adopted in AI accelerators like TPUs [44–46], are highly effective for linear computations such as matrix multiplications. We show in this paper that our CGRAs can be seamlessly integrated with systolic array architectures, enabling end-to-end LLM acceleration. In this paper, we introduce *PICACHU*: Plug-In Coarse-grAined-reConfigurable-accelerator Handling Upcoming nonlinear operations in LLMs (Figure 2). To our knowledge, this is the first accelerator designed for the extensive nonlinear operations in LLMs and the first to utilize CGRA as a plug-in accelerator for LLMs. Our contributions are listed as follows:

- We propose a high-precision approach to approximate current and emerging nonlinear functions in LLMs in both floating-point (FP) and integer (INT) arithmetic. Our accuracy evaluation demonstrates that our algorithm has the minimal impact on LLM accuracy.
- We propose a novel heterogeneous CGRA design tailored to accelerate nonlinear operations in LLMs. This CGRA architecture offers significant flexibility, supporting various data formats and precisions. Furthermore, the CGRA can be seamlessly integrated with systolic arrays for end-to-end acceleration.
- We implement an end-to-end compilation toolchain that translates an ML model at high level (e.g., Pytorch) to LLVM IR and maps it onto the CGRA architecture.
- We evaluate the end-to-end execution of PICACHU on various LLMs, and show that PICACHU achieves speedups of 1.86×, 1.55×, and 3.08× over state-of-the-art baselines including Gemmini [27], Tandem [28], and Nvidia A100 GPU, respectively.

2 Background and Related Work

2.1 Nonlinear Operations in LLMs

In LLMs, the integration of diverse nonlinear functions across multiple components—such as activation layers (e.g., Softmax, GeLU [35]), normalization layers (e.g., LayerNorm [5]), and positional embedding mechanisms (e.g., RoPE [104])—is vital for modeling the complex dynamics of natural language. Nonlinear functions in activation layers introduce essential nonlinearity, enabling models to capture intricate relationships that exceed the limitations of linear mappings. In normalization layers, functions like layer normalization not only stabilize and rescale internal representations but also introduce subtle nonlinear effects that enhance the learning capabilities of the model. Positional embedding layers, responsible for encoding the relative positions of tokens within sequences, add nonlinearity by introducing spatial dependencies that help the model comprehend word order and contextual relationships. A comprehensive summary of nonlinear operations and the representative LLMs that employ them is provided in Table 1.

Unlike in CNNs, where nonlinear functions contribute minimally to execution cost, they contribute to significant latency in LLMs due to the complexity and hardware inefficiency of mathematical operators like exp(.) and log(.). To mitigate this, previous research has aimed to optimize these operations for improved performance through two groups of methods: algorithmic modifications and software-hardware co-design approaches for nonlinear operations.

From an algorithmic perspective, prior research has primarily concentrated on operator-level approximations using

MethodModel	LLaMA-7B	LLaMA-13B	LLaMA2-7B	LLaMA2-13B
FP16	6.75	6.24	5.69	5.09
I-BERT [49]	7E+4	4E+4	6E+4	3E+4
Gemmlowp [41]	9.67	7.99	6.56	6.48

Table 2. PPL of LLaMA models over Wikitext2, a lower PPL indicates a better result. We follow the methods in [41, 49] to approximate nonlinear operations in integer arithmetic while keep the linear layers in FP format.

polynomials, which are computationally efficient and well-supported by GPUs without necessitating substantial hardware changes. For example, I-BERT [49] and I-LLM [38] use polynomials to approximate nonlinear operations specific for their target models. These approaches sacrifice generality, limiting their applicability to other functions, and may result in inconsistent accuracy across different models.

An alternative approach, used in NN-LUT [136] and Auto-LUT [71], involves using neural networks to approximate nonlinear functions. While this method can generalize to other functions, it requires extra training and lacks reliability evidence in LLM execution.

From the algorithm-hardware co-design perspective, most efforts focus on creating specialized hardware units with custom algorithms. Transformer accelerators [36, 61, 73, 76, 119] incorporate dedicated units for Softmax and normalization operations. Additionally, several accelerators specifically target Softmax [26, 40, 101, 103, 118, 120, 129, 150] or normalization functions [42, 122]. However, these accelerators are generally optimized for the operations specific to their models, making it difficult to generalize for other operations.

To address this limitation and achieve broader applicability, Tandem [28] focuses on designing a general-purpose processor for handling nonlinear operations in DNNs. However, the algorithms used in Tandem have not been evaluated for accuracy, which limits their applicability for LLMs, as these models are very sensitive to the precision of nonlinear operations. In contrast, PICACHU supports future operations and has been evaluated on various LLMs, with results showing that it can be implemented without accuracy loss.

2.2 CGRA

CGRA can be either loosely or tightly coupled with a CPU. In this paper, we focus on the loosely coupled approach, where the CGRA operates as an independent module, with the CPU handling only data and instruction transmission. The CGRA is composed of multiple tiles arranged in a grid, with each tile containing functional units (FUs) for performing basic operations, register files, configuration memory, and on-chip interconnects for communication with other tiles or memory. CGRA compiler produces the configuration signals so that, in each cycle, the tiles read their configurations and execute the corresponding operations. The routing between tiles and memory is also controlled by these configuration signals.

Given an application kernel, the compiler generates its data flow graph (DFG) and maps the DFG nodes to the CGRA tiles for computation and data routing. The initial interval (II), calculated by the compiler, is a crucial metric that represents the number of cycles between the initiation of sequential loop iterations. For loops with a large number of iterations, the II heavily influences the overall execution latency.

In comparison to ASICs, FPGAs, and GPUs, CGRA is among the most computationally efficient and highly customizable architectures, offering a promising approach to accelerate workloads involving extensive computations. Thanks to its configurability and domain-specific flexibility, CGRA can achieve superior performance and energy efficiency in the machine learning (ML) domain. Prior works have demonstrated that leveraging CGRA to accelerate DNNs [3, 6, 16, 22, 58, 97, 116, 134], graph neural networks (GNNs) [131, 153], and other ML workloads [1, 81] can lead to significant improvements in performance and cost. Additionally, some studies have explored the flexibility of CGRA in the ML domain by designing algorithms or frameworks for design space exploration (DSE) [78, 108, 109, 154]. Mozart [93] demonstrates the potential of using CGRA for arbitrary specialized computations like Softmax, offering a flexible and efficient solution. Building on this, our work specifically targets the domain of nonlinear operations within LLMs.

2.3 Systolic Array based DNN Accelerators

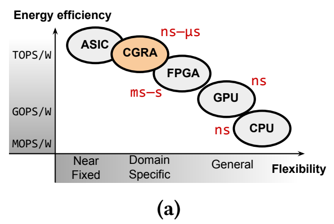
DNN accelerators based on systolic array architecture have revolutionized the efficiency of deep learning computations by optimizing the execution of matrix operations, which are central to neural networks. In this architecture, processing elements (PEs) are organized in a grid, enabling them to perform computations in a highly parallel and pipelined manner. Data flows rhythmically through the array, allowing each PE to operate on different pieces of data simultaneously, which significantly reduces latency and enhances throughput. Systolic array architecture is especially well-suited for tasks like convolution and matrix multiplication, which are frequently used in CNNs and transformer-based hardware accelerators [30, 31, 36, 52, 59, 72, 73, 98, 146–148].

3 Motivation

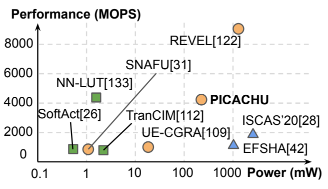
We begin by analyzing the characteristics of nonlinear operations (Section 3.1) and then explain why CGRA is well-suited for handling these operations (Section 3.2).

3.1 Characteristics of Nonlinear operations in LLMs

Nonlinear functions in LLMs consist of a limited set of basic functions. Although there are numerous operations, only a few basic operations are involved, as shown in Table 1. This allows us to focus on these essential operators, since



(a)



(b)

Figure 3. (a) Comparison of CGRA and other architectures, with reconfiguration times highlighted in red. (b) Representative works: ASIC in green, CGRA in orange, FPGA in blue.

computing them with low cost simplifies the overall nonlinear operations. Additionally, due to the broad applicability of these basic functions, we expect that supporting these core nonlinear mathematical operators will be adequate for addressing most current and future nonlinear operations.

Nonlinear operations share common computational patterns. Our analysis shows that all identified nonlinear operations can be represented as loop-based, with some additional computations outside the main loops. Based on the dataflow patterns, we categorize these operations into two classes: (1) *element-wise operations* (EO) and (2) *reduction followed by element-wise operations* (RE). As illustrated in Table 1, most nonlinear operations are EO, which can be implemented with a single loop with a 1D input tensor. For higher dimensions, tensors can be flattened into a 1D shape, avoiding nested loops, as the spatial arrangement of the tensor does not affect computations. In contrast, Softmax and all normalization operations belong to RE, requiring multiple single-layer loops for execution. For Softmax, there are three loops, the first two of which perform reduction, while normalization operations have two loops, with the first also being a reduction; the final loop of both Softmax and normalization operations consists of element-wise operations.

Nonlinear operations exhibit high computational intensity at the data flow graph level. Computational intensity is a widely used metric in CGRA [9, 80, 100]. At the DFG level, the computational intensity is calculated by the ratio of the nodes that perform computations and the nodes that access the memory where each node represents an operation of the kernel. In contrast to the classic Roofline model [126], which evaluates operations at the tensor level and classifies nonlinear operations as memory-bound, our analysis takes a more detailed approach by counting individual computations and memory accesses. The intensities of all operations, except for ReLU, exceed 5.3, with a maximum of 14.5. This high computational intensity indicates that nonlinear operations require minimal data movement, as each data item is read from memory, processed through multiple computations, and then written back.

3.2 Why CGRA?

Taking into account the challenges and characteristics outlined in Section 1 and Section 3.1, we conclude that to effectively accelerate nonlinear operations in LLMs, a *universal* accelerator with the following attributes is preferable.

3.2.1 Efficiency.

As shown in Figure 1, nonlinear operations constitute a large portion of runtime consumption in modern LLMs, with trends suggesting that this share continues to grow. Consequently, our accelerator must be designed to address the common patterns of these nonlinear operations, enabling significant efficiency gains to lower the costs associated with LLM inference. Figure 3b lists representative works focused on accelerations for nonlinear operations [24, 26, 40, 115, 136] and other workloads [29, 107, 112, 125], showing that CGRA can achieve good performance while maintaining relatively low energy consumption. We believe that CGRA is well-suited to achieve this because of its (1) integration of spatial and temporal computation [47, 69, 106] and (2) data-driven execution [48, 69, 95], making it an efficient accelerator for loop-based nonlinear operations.

3.2.2 Flexibility.

Flexibility is essential when handling diverse nonlinear operations, as a lack of it would hinder support for existing operations or future advancements. As illustrated in Figure 3a, CGRA achieves domain-specific flexibility with high energy efficiency, allowing adaptation to a wide range of current and future operations, as each tile can be configured to perform specific tasks. While CGRAs provide less flexibility than FPGAs or general-purpose processors, they suffice for nonlinear operations, which involve limited basic mathematical operators, as discussed in Section 4.1. CGRAs allow the integration of specialized hardware units for each tile, leveraging the characteristics of nonlinear operations to optimize energy use and resource allocation.

3.2.3 User-defined precision.

Since different LLMs use various nonlinear operations with varying sensitivity to their accuracy, this allows for hardware performance improvements with minimal accuracy loss by dynamically adjusting approximation levels for these operations. To support this, PICACHU enables adaptive approximation algorithms in Section 4.1 to ensure model accuracy. Furthermore, it incorporates a precision-aware design, as discussed in Section 5.3.3, enabling a trade-off between accuracy and performance.

4 Methodology

In this section, we offer a comprehensive overview of the PICACHU design. First, Section 4.1 details how nonlinear operations are transformed into multiple polynomial terms for flexible execution. Next, the PICACHU architecture is thoroughly discussed in Section 4.2, with an illustration provided in Figure 4 to support our algorithms. Finally, Section 4.3 presents the PICACHU compiler toolchain.

Operator	Calculation Method
$\exp(x)$	Step 1: Calculate $t = \log_2(e)x$ ($\exp(x) \Rightarrow \mathrm{pow}(2, t)$). Step 2: Split t into integer i and fraction f. Step 3: Calculate $\mathrm{pow}(2, i)$ directly. Step 4: Obtain $\mathrm{pow}(2, f) = 1 + \ln 2 \cdot f + \ln^2 2/2 \cdot f^2 + \ldots$ Step 5: Multiply the results in Step 3 and Step 4.
$\log(x)$	Step 1: Extract exponent e and mantissa m. Step 2: Obtain $\log_2(1+m) = 1/\ln 2 \cdot (m - m^2/2 + \ldots)$ Step 3: Sum the results in Step 3 and Step 4.
$\sin(x)$	Step 1: Obtain t with $\sin(t) = \sin(x), t \in [-\pi/2, \pi/2]$ Step 2: Obtain $\sin(t) = t - t^3/6 + \ldots$
$\cos(x)$	Step 1: Obtain t with $\cos(t) = \cos(x), t \in [-\pi/2, \pi/2]$ Step 2: Obtain $\cos(t) = 1 - t^2/2 + \ldots$

Table 3. Calculation methods for nonlinear mathematical operators. Suppose x is in FP format and for integer inputs we can convert them to FP first then apply methods above.

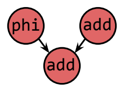
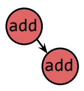
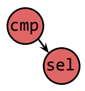
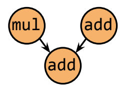
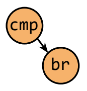
				
phi+add+add phi+add, phi, ...	add+add	cmp+select	mul+add+add mul+add, mul	cmp+br
100%	100%	32.5%	87.5%	100%

Table 4. Common patterns observed in DFGs across all nonlinear operations in LLMs. The three rows show their DFG patterns, corresponding LLVM IR operation chains and occurrence frequency across kernels, respectively.

4.1 PICACHU Algorithm

To facilitate the implementation of the nonlinear operations listed in Table 1, we employ Taylor expansion to decompose nonlinear mathematical operators into a sum of polynomial terms allowing these operations to be split into basic operations. PICACHU allows the users to adjust the level of approximation by selecting the number of polynomial terms, offering an ideal balance between computational cost and accuracy. Specifically, for the exponential and logarithmic operations (i.e., exp(.), log(.)), we adapt the methods from [40, 129] by dividing the input into two components through our special FUs proposed in Section 4.2.1. The first component enables us to directly compute values, while the second component, constrained within the range $[0, 1]$, is ideal for applying Taylor expansion. For the sine and cosine functions (i.e., sin(.) and cos(.)), we use a similar approach to that used for the exponential function: the input is first transformed to the range $[-\pi/2, \pi/2]$, then Taylor expansion is applied.

The division operation is directly implemented in FUs in a pipelined manner. Moreover, we do not account for the inverse square root, as it only appears outside the normalization loop and incurs minimal cost relative to the extensive computations within the loops. This operation can be handled by utilizing the CGRA to execute the standard method from GNU Libc [102] with negligible computational cost.

Some nonlinear operations cannot be accurately computed using basic arithmetic operations. For example, the GeLU activation function can be efficiently computed using the values of the Gaussian cumulative distribution function, denoted as $\Phi(\cdot)$. For these operations, we can leverage special function support proposed in Section 4.2.1 to handle them.

LLM inference typically uses FP32 or FP16 arithmetic, but current research is investigating quantization techniques that utilize INT arithmetic to reduce memory and computational costs. However, managing INT addition/subtraction with different scale factors can be challenging in INT arithmetic, complicating polynomial calculations. I-BERT [49] introduces methods for computing polynomials of quantized inputs through the technique of completing the square, which proves especially effective for calculating Taylor polynomials. For example, the polynomial $a + bx + cx^2$ can be rewritten as $c\left(x + \frac{b}{2c}\right)^2 + \left(a - \frac{b^2}{4c}\right)$, where x is quantized, and the other coefficients remain constant and can be quantized dynamically. Therefore, our algorithm supports both FP and INT calculations of nonlinear operations, motivating our architecture design of various data formats, as detailed in Section 4.2.1. In addition, since our approximation for nonlinear operations can achieve superior accuracy, we gain better performance at the expense of negligible accuracy loss through precision-aware design proposed in Section 5.3.3.

4.2 PICACHU Architecture

As shown in Figure 4, the PICACHU architecture primarily consists of a systolic array and a CGRA, both loosely coupled with the CPU. The CPU handles the control commands sent to these components. The PICACHU CGRA is pluggable, allowing it to function as a nonlinear unit connected to the systolic array via a Shared Buffer. It consists of 16 tiles interconnected through a mesh on-chip network. These tiles are classified into three types: *Branch-optimized Tile* (BrT), *Basic Tile* (BaT) and *Compute Tile* (CoT), each with distinct FUs, creating a heterogeneous CGRA. The PICACHU CGRA is different from conventional CGRAs because of its domain-specialized heterogeneous FUs, precision-awareness, and plugability via the shared memory.

4.2.1 Functional Units.

The FUs in the PICACHU CGRA can efficiently process both existing and emerging nonlinear operations while retaining general-purpose flexibility.

Operation fusion – Observing the DFGs of nonlinear operations, we identify several common patterns that occur frequently, mainly from Taylor polynomials discussed in Section 4.1. The prevalence of these patterns is illustrated in Table 4. We combine each recurring pattern into a single complex node that can run on the specialized FU in a single cycle, which reduces the size of the DFGs and the critical

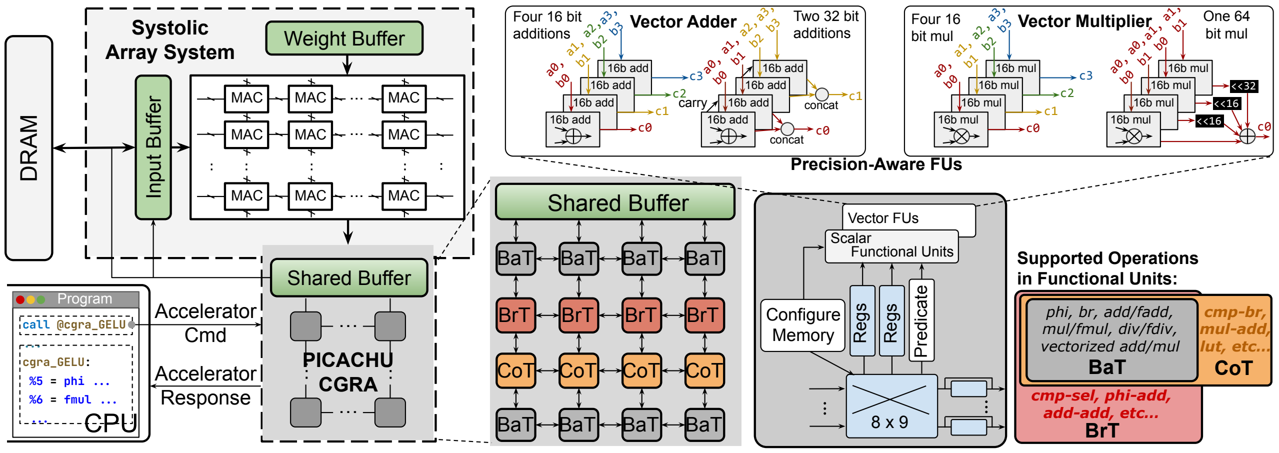


Figure 4. Overall architecture of PICACHU.

path (i.e., II), leading to higher overall speedup as shown in Figure 7a. The fused operations are classified in three categories, each of which is supported by different tiles (i.e., *BaT*, *BrT*, and *CoT*). This enables better workload distribution among more tiles and facilitates the DFG mapping, rather than concentrating the operations on a small number of tiles.

Special function support – To enhance support for nonlinear operations on the CGRA, we introduce two specialized hardware modules for distinct functions:

- **Floating-point to Fixed-point (FP2FX) Conversion Module:** This module extracts the integer and fractional components from a FP value, providing support for the algorithm outlined in Section 4.1 to perform exponential computations efficiently.
- **Look-up Table (LUT):** *CoTs* are equipped with LUTs to store pre-computed values of functions which are hard to compute (e.g., $\Phi(\cdot)$ in GeLU). Incorporating LUTs enhances PICACHU's versatility, enabling it to support both current and future nonlinear operations.

Data Format – Our CGRA is designed to support both FP and INT inputs through flexible tiles equipped with dedicated FP and INT units enabling dynamic reconfiguration of intermediate results as needed. The current architecture supports FP32/FP16 and INT32/INT16 for both inputs and outputs. FP inputs are converted to FP32 for intermediate computations, while INT inputs are processed directly using the precision-awareness techniques described in Section 4.2.2.

4.2.2 Precision-Awareness.

Quantization is essential for LLM inference, and researchers are actively exploring low-precision quantization with reduced bitwidth to minimize memory usage and computational costs [23, 30, 65, 130, 139, 152]. However, implementing a full set of arithmetic units for every integer bit width can be resource-inefficient. To mitigate this, prior study [99] shows that INT arithmetic can support effective hardware resource sharing, allowing units designed for higher bit widths to be multiplexed into several units for lower bit widths. Building on this insight, each PICACHU CGRA tile contains four lanes, each capable of performing 16-bit integer arithmetic. Two lanes can be combined for 32-bit addition by transferring carry bits between 16-bit adders. Similarly, all four lanes can be combined for 32-bit integer multiplication through additional shift and accumulation. Although a tile can perform two 32-bit additions using its four lanes, only half are enabled when operating in the INT32 format to ensure alignment between addition and multiplication operations. This approach offers flexibility in balancing vectorization factors and precision. Higher vectorization factor with INT16 allows greater speedup if slight accuracy loss is acceptable. Otherwise, INT32 will be used, albeit with reduced vectorization.

4.2.3 Shared Buffer.

As nonlinear operations are mostly memory-bound and tensor sizes are exceeding available memory capacity, optimizing the memory system is critical. To enhance memory efficiency, we utilize streaming and double-buffering techniques.

Streaming – Prior to CGRA execution, some data may be unavailable, causing an overlap between data generation and CGRA execution. In each pass, a tile of data is generated and sent to the CGRA for processing, with the next tile following the same process in subsequent passes.

Double-buffering – As discussed in Section 4.2.4, off-chip memory access via DMA is time-consuming and may create a bottleneck in our streaming execution. We implement a double-buffering technique with two input and two output buffers. While reading DRAM, one buffer is used for processing while the other stores data being transferred via DMA. This approach creates an overlap between execution and data movement, mitigating the impact of DMA latency.

4.2.4 Integration with Systolic Arrays.

Architectures based on systolic arrays are widely utilized for executing

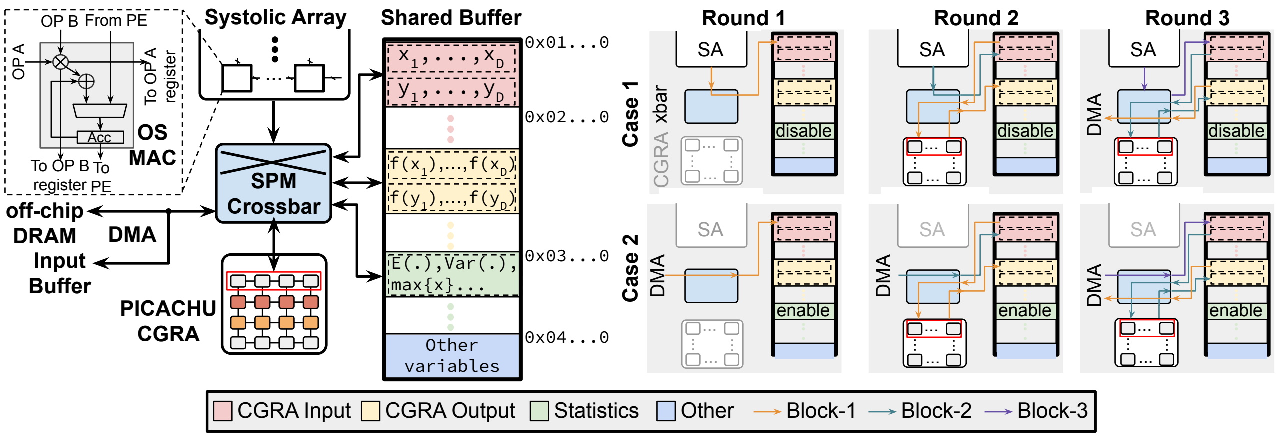


Figure 5. PICACHU CGRA integrate with the systolic array.

GEMM operations in neural networks (e.g., TPU [44–46]). A systolic array features its own on-chip memory subsystem, comprising input SRAM, weight SRAM, and output SRAM. On the other hand, conventional CGRA prefers to having its own dedicated memory space. We integrate the CGRA with the systolic array by multiplexing the output SRAM of the systolic array. This SRAM functions as the input, intermediate, and output memory (referred to as the *Shared Buffer*) for the PICACHU CGRA. This allows CGRA to easily access the results of matrix multiplications as inputs of nonlinear operations. Additionally, since some operations require inputs from off-chip DRAM instead of on-chip SRAM, the Shared Buffer is configured to read from and write to DRAM through DMA. The corresponding data flow during the execution is depicted in Figure 5.

As sequence lengths grow, there may be situations where SRAM cannot hold an entire tensor of shape $N \times D$, where N denotes the number of tokens and D represents the embedding dimension. This is where our streaming and double-buffering techniques, discussed in Section 4.2.3, come into play to minimize data movement and enhance data reuse. We propose three data flow strategies for nonlinear operations in LLMs and detail their execution in various scenarios.

Case 1 – For EOs, all operations are element-wise and single-looped (i.e., without reduction). Therefore, they can seamlessly overlap with the systolic array without concerns about tensor size. Each time the systolic array produces output, execution in the CGRA can commence immediately, as illustrated in Figure 5. The inputs for the Shared Buffer are sourced from the systolic array, the outputs are produced without buffering the intermediate statistics.

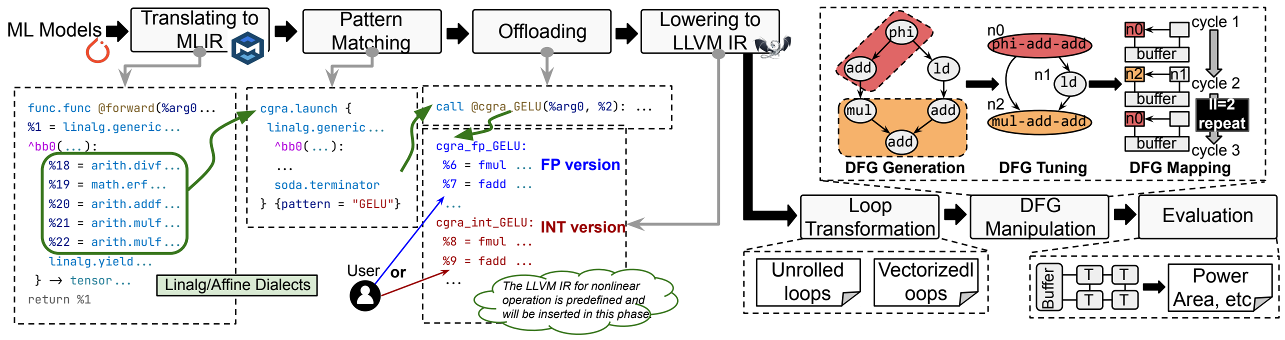
Case 2 – For REs and buffer cannot hold the entire tensor, we access DRAM to retrieve inputs channel by channel. This approach allows us to focus on processing a single vector for each data transfer. As described in Section 5.3.5, a Shared Buffer of 40KB is enough to hold the values within one channel for all the LLMs. Once processing is complete, the results are written back to DRAM via DMA. As shown in Figure 5, the inputs for the Shared Buffer are retrieved from off-chip DRAM, while intermediate statistics, such as mean and variance, are buffered for further computation. Note that both the inputs and outputs of normalization operations must reside in DRAM during execution while for softmax with three loops, as mentioned in Section 3.1, the second and third loops are the same as the normalization operations, as the first loop can overlap with the systolic array operations.

Case 3 – Despite the need for reduction in REs, algorithmic optimizations for nonlinear operations, such as FlashAttention [19], can enable them to fit within the buffer. In such cases, we can keep inputs in the Shared Buffer until statistics are obtained. Then, we can follow the data flows described in Case 1 to perform element-wise operations in the last loop of REs while keeping the Statistics module active.

4.3 PICACHU Compiler Toolchain

To support the mapping of ML models from high-level frameworks (e.g., PyTorch, ONNX) onto our CGRA, we have developed an end-to-end compiler framework built on top of MLIR [56]. As shown in Figure 6, this framework takes the models as inputs, lowering them to MLIR in Linalg/Affine dialects. Nonlinear operations are identified through predefined patterns and converted into specific tasks for offloading to the CGRA for execution. Subsequently, these operations are lowered to LLVM IR, where they undergo loop optimizations and DFG manipulations. Finally, the DFG is mapped onto the CGRA, allowing us to use our RTL framework tailored to the architecture proposed in Section 4.2 to evaluate performance. In the following sections, we introduce the implementation details of the proposed compiler framework.

Pattern Matching – Nonlinear operations will be split into multiple instructions in MLIR. For example, a GeLU operation is translated into five separate instructions, as illustrated in Figure 6. We implement a pattern matching that can locate



```
func.func @forward(%arg0...
%1 = linalg.generic...
^bb0(...):
  %18 = arith.divf...
  %19 = math.erf...
  %20 = arith.addf...
  %21 = arith.mulf...
  %22 = arith.mulf...
  linalg.yield...
 } → tensor...
return %1
```

```
cgra.launch {
 linalg.generic...
  ^bb0(...):
  ...
  soda.terminator
} {pattern = "GELU"}
```

```
call @cgra_GELU(%arg0, %2): ...
```

```
cgra_fp_GELU:
 %6 = fmul ...
 %7 = fadd ...
 ...
cgra_int_GELU:
 %8 = fmul ...
 %9 = fadd ...
 ...
```

Figure 6. Overall compiler toolchain of PICACHU.

such nonlinear operations and combine them into a specialized instruction. It supports future operations without the need to modify the MLIR dialect (as long as the front-end provides the lowering from PyTorch/ONNX to linalg).

Offloading – Once all nonlinear operations have been identified, we utilize the offload pass to lower them into the CGRA function calls, which can be initiated by the accelerator command, as shown in Figure 4. For computations other than nonlinear operations (e.g., matmul), they can also be tiled (output-stationary with same tiling factor as the nonlinear operations if tiling is necessary) and offloaded to the systolic array for end-to-end execution as shown in Figure 5. At this stage, we allow users to determine the data format of each offloaded kernel to trade-off between accuracy and speedup.

Lowering to LLVM IR – For nonlinear operations, we have predefined kernel codes that utilize the custom algorithms proposed in Section 4.1 across different data formats. These kernel codes are written in C++, parameterizable in terms of different tensor shapes, and translated into LLVM IR. After offloading, we insert these predefined kernels into the overall code, based on the desired data format.

Loop Transformations – This phase consists of loop unrolling and vectorization. (1) Loop Unrolling: We can unroll the kernel to increase the size of the DFGs, thereby improving the utilization of the CGRA resources and enhancing overall kernel performance. (2) Loop Vectorization: We leverage the LLVM Auto-Vectorization pass [55] to vectorize loops by replacing scalar operations with vector operations.

DFG Manipulation – This phase involves DFG generation, tuning, and mapping. (1) DFG Generation: Each DFG node represents one instruction in LLVM IR, with control-flow instructions converted to data flow through partial prediction [34]. (2) DFG Tuning: We fuse common patterns, as listed in Table 4, into a single node. For non-vectorized operations in our architecture (e.g., division), we split them into multiple nodes when vectorization is enabled. (3) DFG Mapping: Our framework employs a heuristic optimization algorithm to map the DFG onto the CGRA's Modulo Routing Resource Graph (MRRG) [77], minimizing the initiation interval (II). Mapping constraints include support for heterogeneous operations across tiles, memory access permissions, and the ability to perform special functions, all determined by the PICACHU CGRA specifications. Upon completing the mapping, we obtain the II and control signals for each tile to guide the execution of specific operations.

5 Experimental Evaluation

5.1 Experimental Setup

To evaluate the PICACHU algorithm, we test it across various LLMs, including OPT [145], GPT-2 [88], and LLaMA [114]. Each of these models incorporates multiple nonlinear operations as outlined in Table 3. For evaluating the hardware performance, we comapre PICACHU against Gemmini [27] and Tandem [28], both of which support a range of nonlinear operations in DNNs. In addition, we also compare PICACHU to the i7-11370H CPU and the A100 GPU.

We implement the PICACHU CGRA in RTL using synthesizable Verilog generated from VecPAC [107, 108] and synthesize it with Synopsys Design Compiler [54] using a 45 nm TSMC library for area and power estimation. Additionally, we utilize CACTI 6.5 [111] to evaluate the area and power of the SRAM. The PICACHU compiler framework is built on top of the MLIR [56, 74] and LLVM [55] infrastructures. For end-to-end evaluation, we employ Timeloop [4, 37, 82, 127, 128] to estimate latency, area, and power. In the following sections, we first assess accuracy in Section 5.2 using the algorithm from Section 4.1. Then, we evaluate performance and hardware efficiency in Section 5.3, followed by a comparison of end-to-end performance with other accelerators and GPUs.

5.2 Accuracy Performance

We evaluate the PICACHU algorithm, described in Section 4.1, on GPT2-XL, OPT-6.7B, OPT-13B, LLaMA2-7B, and LLaMA2-13B. Our focus is exclusively on the nonlinear operations of LLMs, while the linear components remain in FP16. The algorithm supports both FP and INT formats for nonlinear operations. Perplexity (PPL), a key metric for assessing language understanding and text generation quality,

MethodPPL (↓)Model	GPT2-XL	OPT-6.7B	OPT-13B	LLaMA2-7B	LLaMA2-13B
FP16	17.39	10.86	10.13	5.69	5.09
Ours (FP16)	0.00	↑ 0.02	0.00	↓ 0.21	↓ 0.21
Ours (INT16)	↓ 0.05	↑ 0.04	↑ 0.03	↓ 0.18	↓ 0.10

Table 5. Performance evaluation of PICACHU algorithm on Wikitext2 dataset, a lower value indicates a better result.

Model	Method	ARC-c (↑)	ARC-e (↑)	HS (↑)	PQ (↑)	WG (↑)	Avg. (↑)
GPT2-XL	FP16	28.49%	50.96%	50.79%	70.51%	58.32%	51.82%
	Ours (FP16)	0.00%	↑ 0.13%	↓ 0.08%	↓ 0.11%	↑ 0.16%	↑ 0.02%
	Ours (INT16)	↓ 0.08%	↑ 0.09%	↓ 0.06%	↓ 0.11%	↑ 0.16%	↑ 0.01%
OPT-6.7B	FP16	34.56%	60.06%	67.20%	76.55%	65.27%	60.83%
	Ours (FP16)	↑ 0.25%	0.00%	↑ 0.21%	↑ 0.06%	↓ 0.16%	↑ 0.07%
	Ours (INT16)	↑ 0.33%	↑ 0.33%	0.00%	↑ 0.11%	↓ 0.08%	↑ 0.05%
OPT-13B	FP16	35.67%	61.87%	69.87%	76.77%	64.96%	61.83%
	Ours (FP16)	↓ 0.26%	↑ 0.08%	↓ 0.87%	0.00%	↓ 0.16%	↓ 0.06%
	Ours (INT16)	↑ 0.42%	↓ 0.09%	↑ 0.05%	↑ 0.05%	↑ 0.08%	↑ 0.10%
LLaMA2-7B	FP16	46.25%	74.58%	75.99%	79.10%	68.90%	68.97%
	Ours (FP16)	↑ 0.08%	↓ 0.13%	↑ 0.03%	↓ 0.05%	↑ 0.24%	↑ 0.03%
	Ours (INT16)	↑ 0.25%	↑ 0.04%	↑ 0.02%	↓ 0.10%	↑ 0.32%	↑ 0.10%
LLaMA2-13B	FP16	49.15%	77.44%	79.38%	80.52%	72.14%	71.73%
	Ours (FP16)	↓ 0.09%	↑ 0.13%	0.00%	0.00%	↑ 0.24%	↑ 0.05%
	Ours (INT16)	↓ 0.17%	0.00%	↓ 0.08%	↓ 0.22%	0.00%	↓ 0.10%

Table 6. PICACHU performance on zero-shot tasks. The upward arrow indicates better accuracy.

indicates better performance with lower values. As shown in Table 5, the algorithm performs effectively on real-time LLMs for both FP and INT computations, with results evaluated on Wikitext2 [79]. We present performance of PICACHU on multiple NLP tasks, including PIQA [12], WinoGrande [91], HellaSwag [142], and ARC (Easy and Challenge) [18] using lm_eval==0.4.4 [25] for evaluation. The results in Table 6 show that PICACHU achieves accuracy comparable to the FP16 model, with average degradation remaining below 0.10%. Notably, for GPT2-XL, OPT-6.7B, and LLaMA2-7B, PICACHU even outperforms the FP16 model.

5.3 Hardware Evaluation

In this section, we evaluate the PICACHU CGRA with LLMs involving the nonlinear operations outlined in Table 1. First, we evaluate the power and area of the PICACHU CGRA in Section 5.3.1. Next, we assess the impact of FUs using a 4×4 homogeneous scalar CGRA without support for fused operations, special function units as baseline in Section 5.3.2. Then, we analyze the performance-accuracy trade-off in the precision-aware design in Section 5.3.3. Section 5.3.4 evaluates PICACHU's scalability (i.e., 3×3, 4×4, 5×5, and 4×8), and lastly, we study the effect of different Shared Buffer sizes on performance in Section 5.3.5.

5.3.1 Area and Power. We evaluate the area and power consumption of our PICACHU CGRA integrated with a 32×32 systolic array and a 40KB Shared Buffer, and the operational frequency is set to 1GHz. As shown in Table 7, PICACHU CGRA tiles occupy 14.9% of total area and 34.2% of total power. We quantify the overhead in the design of

	32×32 Systolic array		4×4 CGRA	Others
	SRAM	MAC		
Area (mm2)	5.3	0.4	1.0	0.1
Area distribution	77.6%	6.2%	14.9%	1.3%
Power (mW)	106.9	16.1	64.2	0.7
Power distribution	56.9%	8.6%	34.2%	0.3%

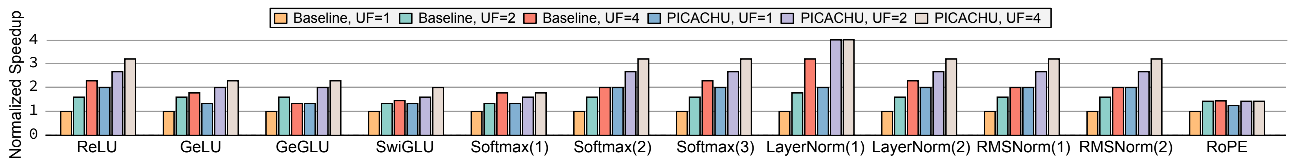
Table 7. Power and area breakdown of PICACHU.

FUs, expressed as a percentage of the extra overhead relative to the cost of a basic tile. The components - FP2FX unit, vectorized FUs, floating-point FUs and LUTs - contribute 1.7%, 59.8%, 11.6% and 0.5% to area overhead and 0.8%, 18.4%, 26.3% and 3.8% to power overhead, respectively.

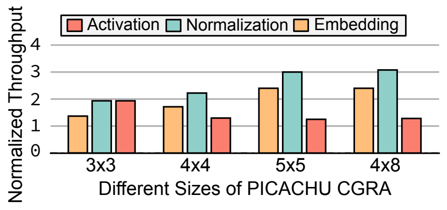
5.3.2 Comparison with baseline CGRA. Figure 7a shows the speedup achieved for each kernel (i.e., nonlinear operations) using our FUs and loop unrolling techniques. The results demonstrate that our PICACHU CGRA delivers an average speedup of 2.95×, with a maximum of 6.4×. Notably, RE operations consist of multiple loops, so we measure the individual II for each kernel. This highlights the effectiveness of the heterogeneous FUs discussed in Section 4.2.1 for accelerating nonlinear operations. The fusion of common patterns, as summarized in Table 4, significantly reduces the total number of operations per kernel, and the inclusion of specialized functions (e.g., FP2FX) simplifies the computation of nonlinear mathematical operators. Compared to the baseline CGRA in terms of overhead, PICACHU CGRA achieves 3.35× energy-efficiency and 2.11× area-efficiency.

5.3.3 Trade-off between performance and accuracy. While FP16/INT32 preserve LLM accuracy, INT16 achieves a vectorization factor of 4, delivering an aggressive average speedup of 2.77×, with a maximum of 3.5× with minimal accuracy loss as shown in Fig. 7d. Note that we only list vectorizable nonlinear operations, with accuracy measured based on complete operations (i.e., treating each RE as a single kernel rather than splitting it into multiple kernels). Furthermore, speedup may deviate from theoretical 4× due to non-vectorizable LLVM IR instructions (e.g., phi), which limit full vectorization benefits.

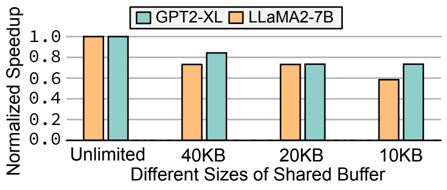
5.3.4 Scalability. Next, we evaluate scalability of PICACHU (Figure 7b) and observe that speedup does not consistently increase with CGRA size, particularly when the architecture is sufficiently large to handle nonlinear operations when plugged in with a 32×32 systolic array. Specifically, Figure 7b shows that the 4×8 CGRA offers less than 1.4× (< 2×) speedup over the 4×4 design, which is mainly due to the compiler's mapping capability [51, 63, 67, 94], which is out of the scope in this work. Notwithstanding, we can split the 4×8 CGRA into two 4×4 partitions accelerating two



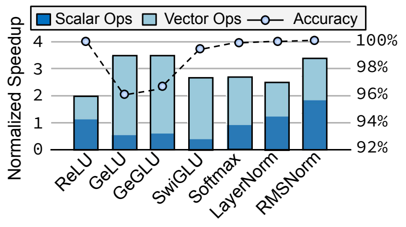
(a)



(b)



(c)



(d)

Figure 7. (a) Normalized speedup of different kernels over conventional 4×4 CGRA. For RE operations, the number in the parentheses of the operation indicates the index of the current loop. UF denotes unrolling factor. (b) Normalized throughput of different kernels running on PICACHU with various numbers of tiles over the baselines. (c) Normalized speedup of end-to-end LLMs running on PICACHU with various sizes of Shared Buffer over the one with unlimited size as the baselines. (d) Normalized speedup of different kernels under a vectorization factor of 4. We don't include non-vectorized kernels in this diagram.

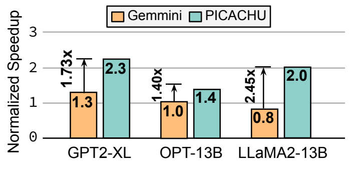
instances of one nonlinear operations, facilitated by double-buffering via the Shared Buffer, achieving a speedup of 2× while simplifying the mapping complexity. This approach maintains the favorable scalability seen in smaller CGRAs, even with larger configurations (e.g., a 64×64 systolic array).

5.3.5 Size of the Shared Buffer.

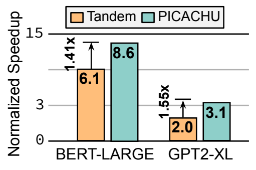
Figure 7c illustrates how Shared Buffer size affects LLM execution speedup. We evaluate GPT2-XL with an embedding dimension of 1600 and LLaMA2-7B with an embedding dimension of 4096. The 20KB/40KB buffers, respectively, match their token sizes, allowing us to apply the three cases of Section 4.2.4. The results show that the optimal Shared Buffer size is the threshold at which a token fits into memory, as larger sizes offer no additional benefits. At a size of 40KB, we observe improved speedup, but larger sizes provide no further performance gains, since data movement costs are hidden by streaming and double-buffering techniques in Section 4.2.3. This suggests that we can choose model-specific Shared Buffer sizes for optimal balance between performance and cost.

5.4 End-to-End Performance and Energy

We compare end-to-end latency and energy consumption of PICACHU to previous state-of-the-art accelerators for neural network, Gemmini [27] and Tandem [28], along with a CPU and the A100 GPU. For CPU evaluation, we use the CPU for all nonlinear operations in the LLM, while the systolic array handles GEMM. Gemmini is an end-to-end DNN accelerator with dedicated hardware units for nonlinear operations including ReLU, GeLU, Softmax, and LayerNorm. It offloads unseen ones to the on-chip RISC-V core. For a fair comparison, we use the same systolic array configuration and exclude all OS-related overhead. The DMA latency is measured using a Xilinx Alveo U280 FPGA card, where data movement occurs between off-chip DRAM and on-chip memory through DMA. As shown in Figure 8a, PICACHU outperforms CPU and Gemmini by 1.90× and 1.86× on average, respectively. For GPT2-XL and OPT-13B, Gemmini maintains comparable performance with PICACHU, as the nonlinear operations in these models can be efficiently handled by its dedicated hardware units. However, for LLaMA2-13B, which contains SwiGLU and RMSNorm, Gemmini must offload these operations to the on-chip RISC-V core, resulting in longer processing latency. Conversely, PICACHU outperforms both the CPU and Gemmini due to its wider nonlinear operator coverage and advanced memory optimizations, while the CPU and Gemmini incur additional data movement costs without the benefits of streaming and double-buffering.

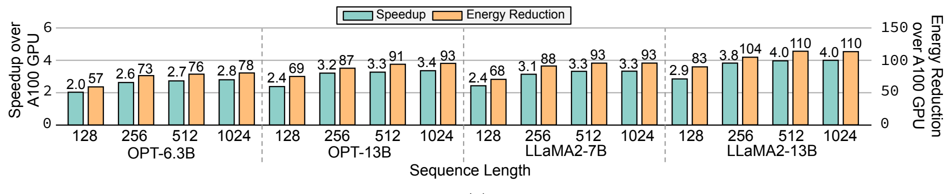


(a)

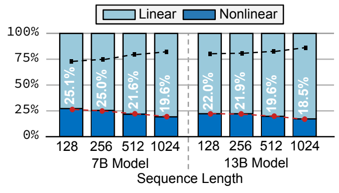


(b)

Figure 8. (a) Normalized speedup of Gemmini and PICACHU relative to CPU latency. (b) Normalized speedup of Tandem and PICACHU relative to A100 GPU execution latency.



(a)



(b)

Figure 9. (a) Normalized speedup and energy reduction of PICACHU relative to A100 GPU. (b) Latency breakdown of PICACHU on LLaMA-7B models and 13B models.

Tandem is another relevant work focused on non-GEMM acceleration in DNNs, using I-BERT and gemmlowp [41] to approximate nonlinear operations, which results in accuracy degradation as shown in Table 2. It reports hardware performance only for BERT and GPT2. As shown in Figure 8b, PICACHU outperforms Tandem on both LLMs with a maximum speedup of 1.55×. Figure 9a shows that PICACHU outperforms the A100 with average speedups of 2.80× on OPT and 3.36× on LLaMA. For a fair comparison, we follow Tandem [28] to scale up PICACHU's systolic array and CGRA to match the throughput of the A100. Figure 9b provides a latency breakdown of PICACHU. PICACHU achieves higher acceleration on LLaMA2 due to its more and more complex nonlinear operations. The end-to-end speedup results from systolic arrays accelerating linear operations and CGRA optimizing nonlinear ones. In LLaMA models, GEMM achieves 2.43× speedup while nonlinear operations reach 6.74×, yielding 3.36× overall acceleration. Notably, nonlinear operation latency in LLaMA2-7B/13B decreases from 42.4%/44.4% to 22.8%/20.5%, demonstrating the effectiveness of our techniques in accelerating these operations.

6 Conclusions

This work presents PICACHU, a plug-in coarse-grained reconfigurable accelerator specifically designed to efficiently handle nonlinear operations using custom algorithms and dedicated compiler toolchain. PICACHU targets all nonlinear operations within LLMs, leveraging CGRA as a plug-in accelerator for LLM inference. The evaluation results indicate that PICACHU outperforms previous state-of-the-art accelerators in LLM inference, presenting a promising avenue for future research on implementing nonlinear operations in LLMs.

References

[1] G Abarajithan, Zhenghua Ma, Zepeng Li, Shrideep Koparkar, Ravidu Munasinghe, Francesco Restuccia, and Ryan Kastner. 2024. CGRA4ML: A Framework to Implement Modern Neural Networks for Scientific Edge Computing. arXiv:2408.15561 [cs.AR] https://arxiv.org/abs/2408.15561

[2] Dosovitskiy Alexey. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv: 2010.11929* (2020).

[3] Kota Ando, Shinya Takamaeda-Yamazaki, Masayuki Ikebe, Tetsuya Asai, and Masato Motomura. 2017. A multithreaded CGRA for convolutional neural network processing. *Circuits and Systems* 8, 6 (2017), 149–170.

[4] Tanner Andrulis, Joel S. Emer, and Vivienne Sze. 2024. CiMLoop: A Flexible, Accurate, and Fast Compute-In-Memory Modeling Tool. In *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*.

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. arXiv:1607.06450 [stat.ML] https://arxiv.org/abs/1607.06450

[6] Inpyo Bae, Barend Harris, Hyemi Min, and Bernhard Egger. 2018. Auto-tuning CNNs for coarse-grained reconfigurable array-based accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2301–2310.

[7] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609* (2023).

[8] Yutong Bai, Xinyang Geng, Karttikeya Mangalam, Amir Bar, Alan L Yuille, Trevor Darrell, Jitendra Malik, and Alexei A Efros. 2024. Sequential modeling enables scalable learning for large vision models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 22861–22872.

[9] Mahesh Balasubramanian, Shail Dave, Aviral Shrivastava, and Reiley Jeyapaul. 2018. LASER: A hardware/software approach to accelerate complicated loops on CGRAs. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1069–1074.

[10] Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. 2019. Efficient 8-bit quantization of transformer neural machine language translation model. *arXiv preprint arXiv:1906.00532* (2019).

[11] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. 2024. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954* (2024).

[12] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 7432–7439.

[13] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. 2022. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745* (2022).

[14] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya

Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL] https://arxiv.org/abs/2005.14165

[15] Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, et al. 2024. Internlm2 technical report. *arXiv preprint arXiv:2403.17297* (2024).

[16] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138. doi:10.1109/JSSC.2016.2616357

[17] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling Language Modeling with Pathways. arXiv:2204.02311 [cs.CL] https://arxiv.org/abs/2204.02311

[18] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457* (2018).

[19] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. arXiv:2205.14135 [cs.LG] https://arxiv.org/abs/2205.14135

[20] Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[21] Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Bin Wang, Linke Ouyang, Xilin Wei, Songyang Zhang, Haodong Duan, Maosong Cao, et al. 2024. Internlm-xcomposer2: Mastering free-form text-image composition and comprehension in vision-language large model. *arXiv preprint arXiv:2401.16420* (2024).

[22] Clément Farabet, Berin Martini, Benoit Corda, Polina Akselrod, Eugenio Culurciello, and Yann LeCun. 2011. Neuflow: A runtime reconfigurable dataflow processor for vision. In *CVPR 2011 workshops*. IEEE, 109–116.

[23] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pretrained transformers. *arXiv preprint arXiv:2210.17323* (2022).

[24] Yuzhe Fu, Changchun Zhou, Tianling Huang, Eryi Han, Yifan He, and Hailong Jiao. 2024. SoftAct: A High-Precision Softmax Architecture for Transformers Supporting Nonlinear Functions. *IEEE Transactions on Circuits and Systems for Video Technology* (2024).

[25] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation. doi:10.5281/zenodo.10256836

[26] Yue Gao, Weiqiang Liu, and Fabrizio Lombardi. 2020. Design and implementation of an approximate softmax layer for deep neural networks. In *2020 IEEE international symposium on circuits and systems (ISCAS)*. IEEE, 1–5.

[27] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, et al. 2021. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 769–774.

[28] Soroush Ghodrati, Sean Kinzer, Hanyang Xu, Rohan Mahapatra, Yoonsung Kim, Byung Hoon Ahn, Dong Kai Wang, Lavanya Karthikeyan, Amir Yazdanbakhsh, Jongse Park, Nam Sung Kim, and Hadi Esmaeilzadeh. 2024. Tandem Processor: Grappling with Emerging Operators in Neural Networks. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 1165–1182. doi:10.1145/3620665.3640365

[29] Graham Gobieski, Ahmet Oguz Atli, Kenneth Mai, Brandon Lucia, and Nathan Beckmann. 2021. Snafu: an ultra-low-power, energy-minimal cgra-generation framework and architecture. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1027–1040.

[30] Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. 2023. Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–15.

[31] Cong Guo, Chen Zhang, Jingwen Leng, Zihan Liu, Fan Yang, Yun-Bo Liu, Minyi Guo, and Yuhao Zhu. 2022. ANT: Exploiting Adaptive Numerical Data Type for Low-bit Deep Neural Network Quantization. *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2022), 1414–1433. https://api.semanticscholar.org/CorpusID:251928917

[32] Tae Jun Ham, Sung Jun Jung, Seonghak Kim, Young H Oh, Yeonhong Park, Yoonho Song, Jung-Hun Park, Sanghee Lee, Kyoung Park, Jae W Lee, et al. 2020. Aˆ 3: Accelerating attention mechanisms in neural networks with approximation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 328–341.

[33] Tae Jun Ham, Yejin Lee, Seong Hoon Seo, Soosung Kim, Hyunji Choi, Sung Jun Jung, and Jae W Lee. 2021. ELSA: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 692–705.

[34] Mahdi Hamzeh, Aviral Shrivastava, and Sarma Vrudhula. 2014. Branch-aware loop mapping on CGRAs. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.

[35] Dan Hendrycks and Kevin Gimpel. 2023. Gaussian Error Linear Units (GELUs). arXiv:1606.08415 [cs.LG] https://arxiv.org/abs/1606.08415

[36] Seongmin Hong, Seungjae Moon, Junsoo Kim, Sungjae Lee, Minsub Kim, Dongsoo Lee, and Joo-Young Kim. 2022. DFX: A Low-latency Multi-FPGA Appliance for Accelerating Transformer-based Text Generation. arXiv:2209.10797 [eess.SY] https://arxiv.org/abs/2209.10797

[37] M. Horeni, P. Taheri, P. Tsai, A. Parashar, J. Emer, and S. Joshi. 2022. Ruby: Improving Hardware Efficiency for Tensor Algebra Accelerators Through Imperfect Factorization. In *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*.

[38] Xing Hu, Yuan Cheng, Dawei Yang, Zhihang Yuan, Jiangyong Yu, Chen Xu, and Sifan Zhou. 2024. I-LLM: Efficient Integer-Only Inference for Fully-Quantized Low-Bit Large Language Models. arXiv:2405.17849 [cs.LG] https://arxiv.org/abs/2405.17849

[39] Mingqiang Huang, Ao Shen, Kai Li, Haoxiang Peng, Boyu Li, and Hao Yu. 2024. Edgellm: A highly efficient cpu-fpga heterogeneous edge accelerator for large language models. *arXiv preprint arXiv:2407.21325* (2024).

[40] Muhammad Awais Hussain and Tsung-Han Tsai. 2021. An efficient and fast softmax hardware architecture (EFSHA) for deep neural

networks. In *2021 IEEE 3rd International conference on artificial intelligence circuits and systems (AICAS)*. IEEE, 1–4.

[41] Benoit Jacob and Pete Warden. 2017. gemmlowp: A small self-contained low-precision gemm library. *Retrieved June* 14 (2017), 2018.

[42] Seongho Jeong, Minseok Seo, Xuan Truong Nguyen, and Hyuk-Jae Lee. 2023. A Low-Latency and Lightweight FPGA-Based Engine for Softmax and Layer Normalization Acceleration. In *2023 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*. IEEE, 1–3.

[43] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).

[44] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, et al. 2023. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–14.

[45] Norman P Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David Patterson. 2020. A domain-specific supercomputer for training deep neural networks. *Commun. ACM* 63, 7 (2020), 67–78.

[46] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[47] Manupa Karunaratne, Dhananjaya Wijerathne, Tulika Mitra, and Li-Shiuan Peh. 2019. 4D-CGRA: Introducing branch dimension to spatio-temporal application mapping on CGRAs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[48] Himan Khanzadi, Yvon Savaria, and Jean Pierre David. 2017. A data driven CGRA Overlay Architecture with embedded processors. In *2017 15th IEEE International New Circuits and Systems Conference (NEWCAS)*. IEEE, 269–272.

[49] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. I-BERT: Integer-only BERT Quantization. *CoRR* abs/2101.01321 (2021). arXiv:2101.01321 https://arxiv.org/abs/2101.01321

[50] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.

[51] Xiangyu Kong, Yi Huang, Jianfeng Zhu, Xingchen Man, Yang Liu, Chunyang Feng, Pengfei Gou, Minggui Tang, Shaojun Wei, and Leibo Liu. 2023. Mapzero: Mapping for coarse-grained reconfigurable architectures with reinforcement learning and monte-carlo tree search. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–14.

[52] HT Kung, Bradley McDanel, and Sai Qian Zhang. 2019. Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 821–834.

[53] Hsiang-Tsung Kung, Bradley McDanel, and Sai Qian Zhang. 2020. Term quantization: Furthering quantization at run time. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–16.

[54] Pran Kurup and Taher Abbasi. 1997. *Logic synthesis using Synopsys®*. Springer Science & Business Media.

[55] C. Lattner and V. Adve. 2004. LLVM: a compilation framework for lifelong program analysis transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.* 75–86. doi:10.1109/CGO.2004.1281665

[56] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2020. MLIR: A Compiler Infrastructure for the End of Moore's Law. arXiv:2002.11054 [cs.PL] https://arxiv.org/abs/2002.11054

[57] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2023. Bloom: A 176b-parameter open-access multilingual language model. (2023).

[58] Jungi Lee and Jongeun Lee. 2021. NP-CGRA: Extending CGRAs for efficient processing of light-weight deep neural networks. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1408–1413.

[59] Jungi Lee, Wonbeom Lee, and Jaewoong Sim. 2024. Tender: Accelerating Large Language Models via Tensor Decomposition and Runtime Requantization. *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)* (2024), 1048–1062. https://api.semanticscholar.org/CorpusID:270620037

[60] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[61] Bingbing Li, Santosh Pandey, Haowen Fang, Yanjun Lyv, Ji Li, Jieyang Chen, Mimi Xie, Lipeng Wan, Hang Liu, and Caiwen Ding. 2020. Ftrans: energy-efficient acceleration of transformers using fpga. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 175–180.

[62] Zhaoying Li, Dhananjaya Wijerathne, Xianzhang Chen, Anuj Pathania, and Tulika Mitra. 2021. Chordmap: Automated mapping of streaming applications onto cgra. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 2 (2021), 306–319.

[63] Zhaoying Li, Dan Wu, Dhananjaya Wijerathne, and Tulika Mitra. 2022. Lisa: Graph neural network based portable mapping on spatial accelerators. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 444–459.

[64] Opher Lieber, Or Sharir, Barak Lenz, and Yoav Shoham. 2021. Jurassic-1: Technical details and evaluation. *White Paper. AI21 Labs* 1, 9 (2021).

[65] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. *Proceedings of Machine Learning and Systems* 6 (2024), 87–100.

[66] Ji Lin, Hongxu Yin, Wei Ping, Pavlo Molchanov, Mohammad Shoeybi, and Song Han. 2024. Vila: On pre-training for visual language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 26689–26699.

[67] Dajiang Liu, Yuxin Xia, Jiaxing Shang, Jiang Zhong, Peng Ouyang, and Shouyi Yin. 2024. E2EMap: End-to-End Reinforcement Learning for CGRA Compilation via Reverse Mapping. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 46–60.

[68] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2024. Visual instruction tuning. *Advances in neural information processing systems* 36 (2024).

[69] Leibo Liu, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin, and Shaojun Wei. 2019. A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications. *ACM Computing Surveys (CSUR)* 52, 6 (2019), 1–39.

[70] Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. 2023. Deja vu: Contextual sparsity for efficient llms

at inference time. In *International Conference on Machine Learning*. PMLR, 22137–22176.

[71] Haodong Lu, Qichang Mei, and Kun Wang. 2023. Auto-LUT: Auto Approximation of Non-Linear Operations for Neural Networks on FPGA. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[72] Liqiang Lu, Yicheng Jin, Hangrui Bi, Zizhang Luo, Peng Li, Tao Wang, and Yun Liang. 2021. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 977–991.

[73] Siyuan Lu, Meiqi Wang, Shuang Liang, Jun Lin, and Zhongfeng Wang. 2020. Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer. In *2020 IEEE 33rd International System-on-Chip Conference (SOCC)*. IEEE, 84–89.

[74] Yixuan Luo, Cheng Tan, Nicolas Bohm Agostini, Ang Li, Antonino Tumeo, Nirav Dave, and Tong Geng. 2023. ML-CGRA: an integrated compilation framework to enable efficient machine learning acceleration on CGRAs. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[75] Alberto Marchisio, Davide Dura, Maurizio Capra, Maurizio Martina, Guido Masera, and Muhammad Shafique. 2023. SwiftTron: An Efficient Hardware Accelerator for Quantized Transformers. In *2023 International Joint Conference on Neural Networks (IJCNN)*. 1–9. doi:10.1109/IJCNN54540.2023.10191521

[76] Alberto Marchisio, Davide Dura, Maurizio Capra, Maurizio Martina, Guido Masera, and Muhammad Shafique. 2023. SwiftTron: An Efficient Hardware Accelerator for Quantized Transformers. arXiv:2304.03986 [cs.LG] https://arxiv.org/abs/2304.03986

[77] Bingfeng Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins. 2003. Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling. In *2003 Design, Automation and Test in Europe Conference and Exhibition*. 296–301. doi:10.1109/DATE.2003.1253623

[78] Jackson Melchert, Kathleen Feng, Caleb Donovick, Ross Daly, Ritvik Sharma, Clark Barrett, Mark A Horowitz, Pat Hanrahan, and Priyanka Raina. 2023. Apex: A framework for automated processing element design space exploration using frequent subgraph analysis. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 33–45.

[79] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843* (2016).

[80] Tony Nowatzki, Vinay Gangadhar, Newsha Ardalani, and Karthikeyan Sankaralingam. 2017. Stream-dataflow acceleration. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 416–429.

[81] Westerley C Oliveira, Michael Canesche, Lucas Reis, José Augusto M Nacif, and Ricardo S Ferreira. 2023. Heterogeneous reconfigurable architectures for machine learning dataflows. *Concurrency and Computation: Practice and Experience* 35, 17 (2023), e6939.

[82] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer. 2019. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. 304–315.

[83] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116* (2023).

[84] Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277* (2023).

[85] Yubin Qin, Yang Wang, Dazheng Deng, Zhiren Zhao, Xiaolong Yang, Leibo Liu, Shaojun Wei, Yang Hu, and Shouyi Yin. 2023. Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–14.

[86] Zheng Qu, Liu Liu, Fengbin Tu, Zhaodong Chen, Yufei Ding, and Yuan Xie. 2022. Dota: detect and omit weak attentions for scalable transformer acceleration. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 14–26.

[87] A Radford. 2018. Improving language understanding by generative pre-training. (2018).

[88] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[89] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. 2021. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446* (2021).

[90] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[91] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Commun. ACM* 64, 9 (2021), 99–106.

[92] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883* (2018).

[93] Karthikeyan Sankaralingam, Tony Nowatzki, Vinay Gangadhar, Preyas Shah, Michael Davies, William Galliher, Ziliang Guo, Jitu Khare, Deepak Vijay, Poly Palamuttam, et al. 2022. The Mozart reuse exposed dataflow processor for AI and beyond: Industrial product. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 978–992.

[94] Nathan Serafin, Souradip Ghosh, Harsh Desai, Nathan Beckmann, and Brandon Lucia. 2023. Pipestitch: An energy-minimal dataflow architecture with lightweight threads. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 1409–1422.

[95] Jiang Sha, Wenbo Song, Yu Gong, and Yingying Zhao. 2020. Accelerating nested conditionals on CGRA with tag-based full predication method. *IEEE Access* 8 (2020), 109401–109410.

[96] Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. 2023. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv preprint arXiv:2308.13137* (2023).

[97] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. 2019. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 14–27.

[98] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Joon Kyung Kim, Vikas Chandra, and Hadi Esmaeilzadeh. 2017. Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network. *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)* (2017), 764–775. https://api.semanticscholar.org/CorpusID:21681898

[99] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Joon Kyung Kim, Vikas Chandra, and Hadi Esmaeilzadeh. 2018.

Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 764–775.

[100] Ivan Saraiva Silva and Francisco Carlos Silva Junior. 2023. X4-RARE: Revisiting the X4CP32 Coarse-Grained Reconfigurable Architecture Model. In *2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 1–6.

[101] Fanny Spagnolo, Stefania Perri, and Pasquale Corsonello. 2021. Aggressive approximation of the softmax function for power-efficient hardware implementations. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 3 (2021), 1652–1656.

[102] Richard Stallman et al. 1998. The GNU project. http://www.gnu.org/software/libc/.

[103] Jacob R Stevens, Rangharajan Venkatesan, Steve Dai, Brucek Khailany, and Anand Raghunathan. 2021. Softermax: Hardware/software co-design of an efficient softmax for transformers. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 469–474.

[104] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing* 568 (2024), 127063.

[105] Thierry Tambe, Coleman Hooper, Lillian Pentecost, Tianyu Jia, En-Yu Yang, Marco Donato, Victor Sanh, Paul Whatmough, Alexander M Rush, David Brooks, et al. 2021. Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 830–844.

[106] Cheng Tan, Miaomiao Jiang, Deepak Patil, Yanghui Ou, Zhaoying Li, Lei Ju, Tulika Mitra, Hyunchul Park, Antonino Tumeo, and Jeff Zhang. 2024. ICED: An Integrated CGRA Framework Enabling DVFS-Aware Acceleration. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1338–1352.

[107] Cheng Tan, Deepak Patil, Antonino Tumeo, Gabriel Weisz, Steve Reinhardt, and Jeff Zhang. 2023. VecPAC: A Vectorizable and Precision-Aware CGRA. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. doi:10.1109/ICCAD57390.2023.10323910

[108] Cheng Tan, Chenhao Xie, Ang Li, Kevin J Barker, and Antonino Tumeo. 2020. OpenCGRA: An open-source unified framework for modeling, testing, and evaluating CGRAs. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 381–388.

[109] Cheng Tan, Chenhao Xie, Ang Li, Kevin J Barker, and Antonino Tumeo. 2021. Aurora: Automated refinement of coarse-grained reconfigurable accelerators. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1388–1393.

[110] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239* (2022).

[111] Shyamkumar Thoziyoor, N Muralimanohar, J Ahn, and N Jouppi. 2009. Cacti 6.5. hpl.hp.com.

[112] Christopher Torng, Peitian Pan, Yanghui Ou, Cheng Tan, and Christopher Batten. 2021. Ultra-elastic cgras for irregular loop specialization. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 412–425.

[113] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL] https://arxiv.org/abs/2302.13971

[114] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL] https://arxiv.org/abs/2307.09288

[115] Fengbin Tu, Zihan Wu, Yiqi Wang, Ling Liang, Liu Liu, Yufei Ding, Leibo Liu, Shaojun Wei, Yuan Xie, and Shouyi Yin. 2022. TranCIM: Full-digital bitline-transpose CIM-based sparse transformer accelerator with pipeline/parallel reconfigurable modes. *IEEE Journal of Solid-State Circuits* 58, 6 (2022), 1798–1809.

[116] Fengbin Tu, Shouyi Yin, Peng Ouyang, Shibin Tang, Leibo Liu, and Shaojun Wei. 2017. Deep convolutional neural network architecture with reconfigurable computation patterns. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 8 (2017), 2220–2233.

[117] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

[118] Ihor Vasyltsov and Wooseok Chang. 2021. Efficient softmax approximation for deep neural networks with attention mechanism. *arXiv preprint arXiv:2111.10770* (2021).

[119] Hanrui Wang, Zhekai Zhang, and Song Han. 2021. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 97–110.

[120] Meiqi Wang, Siyuan Lu, Danyang Zhu, Jun Lin, and Zhongfeng Wang. 2018. A high-speed and low-complexity architecture for softmax function in deep learning. In *2018 IEEE asia pacific conference on circuits and systems (APCCAS)*. IEEE, 223–226.

[121] Wenhai Wang, Zhe Chen, Xiaokang Chen, Jiannan Wu, Xizhou Zhu, Gang Zeng, Ping Luo, Tong Lu, Jie Zhou, Yu Qiao, et al. 2024. Visionllm: Large language model is also an open-ended decoder for vision-centric tasks. *Advances in Neural Information Processing Systems* 36 (2024).

[122] Wenxun Wang, Shuchang Zhou, Wenyu Sun, Peiqin Sun, and Yongpan Liu. 2023. SOLE: Hardware-Software Co-design of Softmax and LayerNorm for Efficient Transformer Inference. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[123] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).

[124] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[125] Jian Weng, Sihao Liu, Zhengrong Wang, Vidushi Dadu, and Tony Nowatzki. 2020. A hybrid systolic-dataflow architecture for inductive matrix algorithms. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 703–716.

[126] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (apr 2009), 65–76. doi:10.1145/1498765.1498785

[127] Yannan Nellie Wu, Joel S Emer, and Vivienne Sze. 2019. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.

[128] Yannan N. Wu, Po-An Tsai, Angshuman Parashar, Vivienne Sze, and Joel S. Emer. 2022. Sparseloop: An Analytical Approach To Sparse Tensor Accelerator Modeling . In *ACM/IEEE International Symposium on Microarchitecture (MICRO)*.

[129] Tianhua Xia and Sai Qian Zhang. 2024. Hyft: A Reconfigurable Softmax Accelerator with Hybrid Numeric Format for both Training and Inference. arXiv:2311.13290 [cs.AR] https://arxiv.org/abs/2311.13290

[130] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*. PMLR, 38087–38099.

[131] Jiaqi Yang, Hao Zheng, and Ahmed Louri. 2024. Aurora: A Versatile and Flexible Accelerator for Graph Neural Networks. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 890–902.

[132] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems* 35 (2022), 27168–27183.

[133] Wenhua Ye, Xu Zhou, Joey Zhou, Cen Chen, and Kenli Li. 2023. Accelerating attention mechanism on fpgas based on efficient reconfigurable systolic array. *ACM Transactions on Embedded Computing Systems* 22, 6 (2023), 1–22.

[134] Shouyi Yin, Peng Ouyang, Shibin Tang, Fengbin Tu, Xiudong Li, Leibo Liu, and Shaojun Wei. 2017. A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications. In *2017 Symposium on VLSI Circuits*. IEEE, C26–C27.

[135] Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. 2024. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652* (2024).

[136] Joonsang Yu, Junki Park, Seongmin Park, Minsoo Kim, Sihwa Lee, Dong Hyun Lee, and Jungwook Choi. 2021. NN-LUT: Neural Approximation of Non-Linear Operations for Efficient Transformer Inference. *CoRR* abs/2112.02191 (2021). arXiv:2112.02191 https://arxiv.org/abs/2112.02191

[137] Lijun Yu, Yong Cheng, Zhiruo Wang, Vivek Kumar, Wolfgang Macherey, Yanping Huang, David Ross, Irfan Essa, Yonatan Bisk, Ming-Hsuan Yang, et al. 2024. Spae: Semantic pyramid autoencoder for multimodal generation with frozen llms. *Advances in Neural Information Processing Systems* 36 (2024).

[138] Zhihang Yuan, Lin Niu, Jia-Wen Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. 2023. RPTQ: Reorder-based Post-training Quantization for Large Language Models. *ArXiv* abs/2304.01089 (2023). https://api.semanticscholar.org/CorpusID:257913374

[139] Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. 2020. Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 811–824.

[140] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, 36–39.

[141] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems* 33 (2020), 17283–17297.

[142] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830* (2019).

[143] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414* (2022).

[144] Wei Zeng, Xiaozhe Ren, Teng Su, Hui Wang, Yi Liao, Zhiwei Wang, Xin Jiang, ZhenZhang Yang, Kaisheng Wang, Xiaoda Zhang, et al. 2021. Pangu-α: Large-scale autoregressive pretrained Chinese language models with auto-parallel computation. *arXiv preprint arXiv:2104.12369* (2021).

[145] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).

[146] Sai Qian Zhang, Bradley McDanel, and HT Kung. 2022. Fast: Dnn training under variable precision block floating point with stochastic rounding. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 846–860.

[147] Sai Qian Zhang, Bradley McDanel, HT Kung, and Xin Dong. 2021. Training for multi-resolution inference using reusable quantization terms. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 845–860.

[148] Sai Qian Zhang, Thierry Tambe, Nestor Cuevas, Gu-Yeon Wei, and David Brooks. 2024. CAMEL: Co-Designing AI Models and eDRAMs for Efficient On-Device Learning. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 861–875.

[149] Sai Qian Zhang, Thierry Tambe, Gu-Yeon Wei, and David Brooks. 2024. JointNF: Enhancing DNN Performance through Adaptive N: M Pruning across both Weight and Activation. In *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*. 1–6.

[150] Yuan Zhang, Yonggang Zhang, Lele Peng, Lianghua Quan, Shubin Zheng, Zhonghai Lu, and Hui Chen. 2022. Base-2 softmax function: Suitability for training and efficient hardware implementation. *IEEE Transactions on Circuits and Systems I: Regular Papers* 69, 9 (2022), 3605–3618.

[151] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. 2024. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems* 36 (2024).

[152] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems* 6 (2024), 196–209.

[153] Luca Zulberti, Matteo Monopoli, Pietro Nannipieri, and Luca Fanucci. 2022. Architectural implications for inference of graph neural networks on cgra-based accelerators. In *2022 17th Conference on Ph. D Research in Microelectronics and Electronics (PRIME)*. IEEE, 373–376.

[154] Luca Zulberti, Matteo Monopoli, Pietro Nannipieri, Luca Fanucci, and Silvia Moranti. 2023. Highly parameterised CGRA architecture for design space exploration of machine learning applications onboard satellites. In *2023 European Data Handling & Data Processing Conference (EDHPC)*. IEEE, 1–6.