2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA) | 979-8-3503-2658-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISCA59077.2024.00079

MECLA: Memory-Compute-Efficient LLM Accelerator with Scaling Sub-matrix Partition

Yubin Qin
School of Integrated Circuits, BNRist
Tsinghua University
Beijing, China
qyb20@mails.tsinghua.edu.cn

Yang Wang*
School of Integrated Circuits, BNRist
Tsinghua University
Beijing, China
wangyang_imec@mail.tsinghua.edu.cn

Zhiren Zhao
School of Integrated Circuits, BNRist
Tsinghua University
Beijing, China
zhaozr21@mails.tsinghua.edu.cn

Xiaolong Yang
School of Integrated Circuits, BNRist
Tsinghua University
Beijing, China
yangxl21@mails.tsinghua.edu.cn

Yang Zhou
School of Integrated Circuits, BNRist
Tsinghua University
Beijing, China
zhou-y22@mails.tsinghua.edu.cn

Shaojun Wei
School of Integrated Circuits, BNRist
Tsinghua University
Beijing, China
wsj@tsinghua.edu.cn

Yang Hu
School of Integrated Circuits, BNRist
Tsinghua University
Beijing, China
hu_yang@tsinghua.edu.cn

Shouyi Yin*
School of Integrated Circuits, BNRist
Tsinghua University
Beijing, China
yinsy@tsinghua.edu.cn

***Abstract*—Large language models (LLMs) have been showing surprising performance in processing language tasks, bringing a new prevalence to deploy LLM from cloud to edge. However, being a scaling auto-regressive Transformer with a huge parameter amount and generating output one by one, LLM introduces overwhelming memory footprints and computation during its inference, especially from its linear layers. For example, generating 32 output tokens with LLaMA-7B LLM requires 14GB of weight data and performs over 400 billion operations (98% from linear layers), which is far beyond the capability of consumer-level GPU and traditional accelerators. To solve these issues, we propose a memory-compute-efficient LLM accelerator, MECLA, with a parameter-efficient scaling sub-matrix partition method (SSMP). It decomposes large weight matrices into several tiny-scale source sub-matrices (SS) and derived sub-matrices (DS). Each DS can be obtained by scaling the corresponding SS with a scalar. For memory issues, SSMP avoids accessing the full weight matrix but only requires small SS and DS scaling scalars. For computation issues, the proposed MECLA processor fully exploits the intermediate data reuse of matrix multiplication via on-chip matrix regrouping, inner-product multiplication re-association, and outer-product partial sum reuse. Experiments on 20 benchmarks show that MECLA reduces memory access and computation by 83.6% and 72.2%. It achieves an energy efficiency of 7088GOPS/W. Compared to V100 GPU and state-of-the-art Transformer accelerator SpAtten and FACT, MECLA saves 113.14×, 12.99×, and 1.62× higher energy efficiency.**

***Index Terms*—large language model, transformer, accelerator, artificial intelligence**

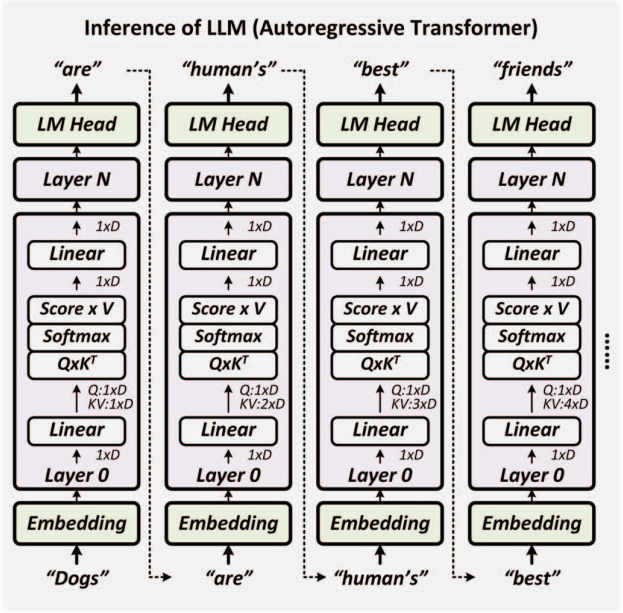
This work was supported in part by the NSFC under Grant 62125403 and Grant 92164301; in part by the National Science and Technology Major Project under Grant 2022ZD0115201; in part by the National Key Research and Development Program under Grant 2021ZD0114400; Beijing S&T Project Z221100007722023; in part by the Beijing National Research Center for Information Science and Technology; and in part by the Beijing Advanced Innovation Center for Integrated Circuits. (Corresponding author: Yang Wang and Shouyi Yin.)

I. Introduction

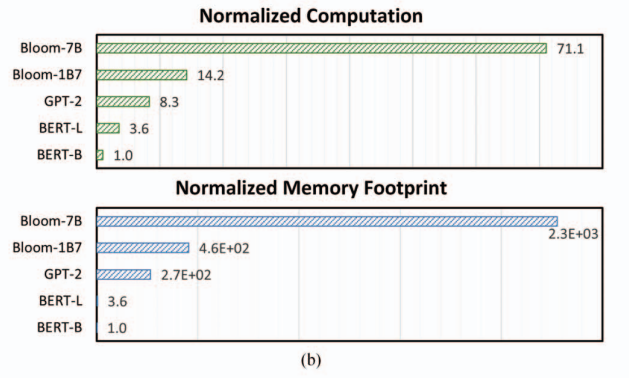
Large language models (LLMs), such as GPT-4 [50], PaLM [2], [14], and LLaMA [71], [72], mark a significant milestone of the Transformer model [73] in the field of artificial intelligence. LLMs have exhibited remarkable capabilities in natural language processing (NLP) and beyond, giving rise to innovative applications, including recommendation systems [4], [23], chatbots [50], [92], logic reasoning assistance [3], [37], etc. The immense power of LLMs has sparked a new trend in its adoption from cloud-based computing to end devices [92]. However, LLM inference is so compute and memory intensive that requires quite a lot of hardware resources such as GPUs. For instance, GLM-130B (INT4), being the only 100B-scale LLM that can run inference on consumer-grade GPUs, demands 4×RTX 3090 (24GB). Even the smallest LLaMA requires 14GB memory and 14 billion operations for a single-word response, which is 4.6× higher than the largest GPT-2. Given these substantial memory and computation burdens, a memory-compute-efficient LLM inference hardware system becomes of paramount importance.

LLM is based on the autoregressive Transformer structure [58], [59], commonly referred to as a *decoder*. It has two main features. First, it takes the user's input prompt and generates the output tokens one by one, as shown in Figure 1(a). Each newly generated token is computed based on the given input

979-8-3503-2658-1/24/$31.00 ©2024 IEEE
DOI 10.1109/ISCA59077.2024.00079



(a)



(b)

Fig. 1. Workflow of large language model (a). Normalized computation and memory effort comparison of LLM and traditional Transformers (b).

and all the previously generated output tokens. Whenever a new output token is computed, it is combined with the previous outputs for the next token generation. This process iterates for several tens to thousands of times until the whole output is completed. Such an autoregressive generation process results in a substantial amount of weight parameter access since the computation unit accesses the entire model's weight from beginning to end for each new output computation. While previous *encoder* Transformer like BERT [18] uses non-regressive inference and accesses the weight only once. Second, LLM has significantly larger dimensions after scaling compared to traditional Transformers, which further exacerbates the issue of weight access and introduces a substantial computational load. For example, a typical LLM Bloom-7B requires about 8.53× more memory access and operations for generating one output token compared to GPT-2.

We analyze the computation and memory efforts of LLM

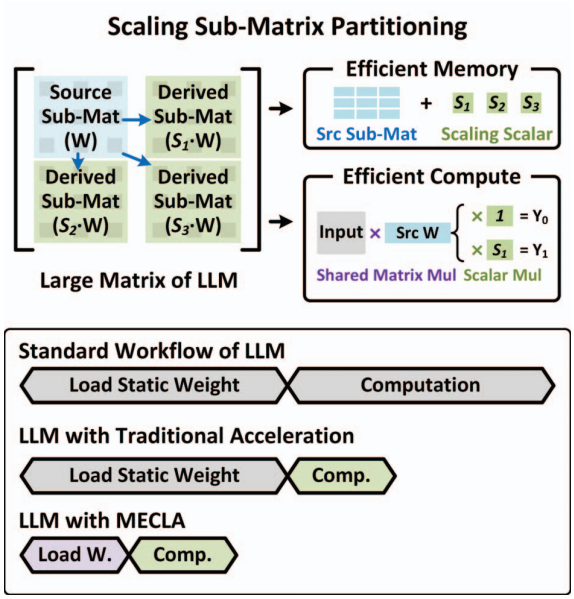


Fig. 2. Principle of scaling sub-matrix partitioning (SSMP) method and optimization coverage comparison with previous works.

and traditional Transformers, and Figure 1(b) shows the results. The computation effort increases several dozen times, and the memory footprint increases over four orders of magnitude. Additionally, [55] shows that the computation of linear layers is dominant in most Transformer models and tasks. Our experiment shows that this trend still exists in LLM tasks. The memory and computation effort of linear layers takes up over 98% of the total when generating sentences with Bloom-1B7 and Bloom-7B. Therefore, the key to efficient LLM lies in the memory and computation of linear layers.

Based on the aforementioned observations for LLM inference efficiency, we propose the scaling sub-matrix partition (SSMP) method, a hardware-friendly matrix partition method enabling high parameter efficiency. Figure 2 illustrates the approach. For the substantial static weight of LLM linear layers, SSMP employs a "generative sub-matrix" strategy. It decomposes the large-scale weight matrix into small-scale *source sub-matrices* (SS) and *derived sub-matrices* (DS). Each SS is linked to a number of DSs nearby, and each DS can be obtained by multiplying a scaling scalar with its corresponding SS. By this means, accessing a linear layer no longer requires reading the full matrix, but only need to read the SS parameters and scaling scalar of each DS, and the whole weight matrix can be re-generated on-chip. Since all parameters in a DS share the same scaling scalar, SSMP method reduces the memory footprint greatly.

In addition, SSMP method is of great potential to improve computation efficiency. Intuitively, SS and DS differ only by a scalar factor. In matrix multiplication, for SS and DS corresponding to the same input channels, their partial sums (PSum) also differ by the same scalar factor. This characteristic can avoid redundant block matrix multiplication calculation

according to the size of SS and the number of DS belonging to an SS. Utilizing this feature with specially designed circuits can gain computation efficiency.

We propose a hardware accelerator, MECLA, based on the SSMP matrix partition method and computation optimization to achieve memory-compute-efficient LLM inference. Experiment on 20 evaluation tasks shows MECLA reduces static memory access by 83.6% and improves computational efficiency by 5.27× on the geometric average. It achieves an energy efficiency of 7088GOPS/W, which is 113.14×, 12.99×, and 1.62× higher than running the same tasks on NVIDIA V100 and state-of-the-art accelerator SpAtten [77] and FACT [55]. Our key innovations are as follows:

- We propose SSMP, a parameter-efficient matrix partition method that splits a large weight matrix into small SS and DS sub-matrices while DS can be obtained by scaling SS. Fine-tuning method for training weight into SSMP is also provided.
- We propose matrix regrouping and matrix-multiplication re-association mechanism, which dynamically adjusts the matrix multiplication workload and matrix arrangement so that the hardware can fully utilize the PSum reusability brought by SSMP for efficiency.
- We design MECLA, a specialized hardware accelerator that exploits parameter and computation efficiency for LLM inference. Experiment on several typical LLM models and tasks demonstrates its capabilities and efficiency gain over conventional hardware.

II. Background and Motivation

In this section, we demonstrate the process of typical LLM inference and the challenges in terms of memory and computation during the inference process.

A. Language Modeling Task and Models

The core of LLM is language modeling. It models the probability of a token sequence $(x_1, x_2, ..., x_n)$, where a token can be a word, sub-word, etc. Because natural language has sequential ordering, researchers use the product of conditional probabilities to factories the joint probabilities of the token sequence [6], [31], as equation 1. This method is referred to as "autoregressive language modeling", which can be used to predict the probability of the next token based on the previous tokens.

$$P(x) = P(x_1, ..., x_n) = \prod_{n=1}^{N} P(x_n | x_1, ..., x_{n-1}) \quad (1)$$

Transformer models have shown outstanding capabilities in language modeling and become the *de facto* choices [50], [92] . It models the text sequence with the self-attention mechanism, as Figure 1 and Algorithm 1 depict. Given input sequence $(x_1, ..., x_n) \in \mathbb{R}^{n \times d}$, it first uses QKV linear transformation to project the input into *query*, *key*, and *value* space. Then it computes *attention score* with the obtained *query* and *key*, and the results are used to perform a weighted sum on the value matrix. Finally, it uses linear layers to transform the intermediate results to get the output. According to the decomposition as equation 1, the model generates one new token, such as the next work of the output sentence, during one inference iteration. Then it joins the newly generated token with the previously generated ones as a new input sequence for another inference iteration, and this process continues until the whole output sentence is generated or exceeds the maximum length limitation. This one-by-one generation process is referred to as "autoregressive generation".

Algorithm 1: Autoregressive Transformer

Input : Input prompt I, Transformer model M, max output length L_{max}
Output: Generated output sequence S_{out}

$X \leftarrow I$;
$S_{out} \leftarrow \{\}$;
while $len(S_{out}) \leq L_{max}$ *and* $S_{out}[-1] \neq [EOS]$ **do**
  for *Layer in* M **do**
    if *Layer is attention layer* **then**
      $q, k, v \leftarrow Layer.QKV\_Linear(X)$;
      $A \leftarrow softmax(q \times k^T / sqrt(M.dim))$;
      $X \leftarrow A \times v$;
      $X \leftarrow Layer.Linear(X)$;
    else
      $X \leftarrow Layer(X)$;
    end
  end
  $S_{out}.append(decode(X))$;
  $X \leftarrow S_{out}$;
end
return S_{out}

TABLE I
Optimization Performance Comparison of State-of-the-Art Transformer Accelerators

Accelerator	Computation		Memory	
	QKV	FFN	QKV	FFN
A3 [27]	No	No	No	No
ELSA [28]	No	No	No	No
JSSC'22 [80]	No	No	No	No
DOTA [57]	No	No	No	No
Sanger [46]	No	No	No	No
SpAtten [77]	Yes	Yes	No	No
FACT [55]	Yes	Yes	Low	No
MECLA	Yes	Yes	Yes	Yes

B. Motivation

Running LLMs poses distinct challenges compared to traditional Transformer models such as BERT [18] and GPT-2 [59].

Firstly, LLMs exhibit significantly larger weight matrices, magnitudes beyond those of traditional models. For instance, the GLM-130B model, with an embedding dimension of 12288 and a feed-forward network dimension of 32768, results in a weight size of 402MB for a single layer. This is 96 / 52 times larger than the linear layer of the largest BERT / GPT-2. Consequently, the computational load has increased proportionally. Furthermore, the substantial dimensions lead to a small proportion of attention computation ($Q \times K^T$ and P×V), amounting to less than 2% when the token length is under 1024. These characteristics collectively underscore the challenge of the high computational and storage efforts associated with the linear layer in LLM inference.

The autoregressive generation of LLM worsens the issue. In Algorithm 1, the QKV linear layer (line.6) and linear layer (line.9) both require large amounts of weight data. These weight data cannot be retained in the local cache of the PE array since it is required to run all the other layers before using the weight data again and the weight matrix is too large for local storage. Therefore, with the generation of each new output token, the PE array undergoes a complete traversal of the model data, and this process continues until the inference is complete. This characteristic further exacerbates the memory footprint issue in LLMs. Furthermore, due to the use of matrix-vector multiplication in autoregressive computations instead of matrix-matrix multiplication, the lack of input tensor reuse poses a potential underutilization problem. These features make optimization the memory and computation efficiency of LLM linear layer urgent.

Unfortunately, we find out that the state-of-the-art Transformer accelerators cannot solve the aforementioned computation and memory issues effectively. Table 1 provides an overview of the effectiveness of these approaches in optimizing the QKV linear layer and FFN linear layer, which are two main sources of computation/memory bottleneck in LLM. However, the majority of the work [27], [28], [46], [57], [80] concentrates on alleviating the computational bottleneck in attention layers for very long sequences. FACT [55] and SpAtten [77] realize challenges with the linear layer. However, their sparsity method cannot reduce the weight memory footprint. It motivates us to propose a memory-compute-efficient LLM accelerator design.

III. SSMP Partition and Fine-tuning

To address the memory bounding issue in LLMs, we propose a novel approach called Scaling Sub-matrix Partitioning (SSMP). This method leverages matrix partitioning and the reuse of intermediate computations to effectively reduce the weight access and computational effort in the linear layers of LLMs, including FFN linear and QKV linear. This section illustrates the matrix decomposition of SSMP and how it is employed in LLMs.

Figure 3 shows our observation for building a memory-compute-efficient LLM design. Since the weight matrix of LLM is huge, such as 4k×11k in LLaMA, it is necessary to compress the weight, and a typical method is to only keep the important weight which is less than 1% of the total and use quantization like per-channel quantization to compress the rest of the weight [34], [43], [85], which can reduce the 16-bit weight matrix into 8bit or even less. Also, some previous works discover the computation redundancy due to the pigeonhole's principle and exploit the similarity or duplication of weight tensors [64], [87]. These two aspects inspire us to explore the spatial-wise (input-channel or output-channel) similarity or duplication for weight compression. For one thing, the input and output channel of LLM weight is much more than traditional models, which involves more pigeonhole principle occurrence. For another, extending the similarity from previous channel-wise to 2D spatial-wise can further reduce the computation effort, which is a solution to the LLM memory and computation burden. Additionally, the small amount of important weight remains floating-point, which consumes little memory burden but keeps the accuracy of the model, making sufficient space for compressing the unimportant but massive weight values. Since the amount of important weight is too small compared to the remaining ones, their storage is not explicitly listed out in the rest of the paper for simplicity.

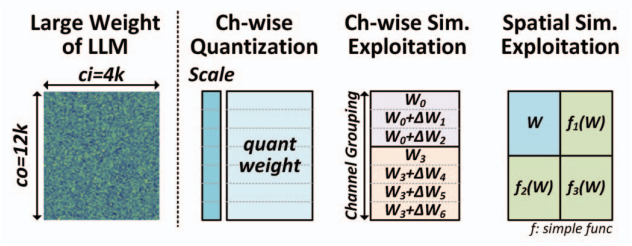


Fig. 3. Comparison of memory and compute efficient methods.

A. SSMP Partitioning

Figure 4 illustrates the principle of SSMP. It partitions the weight matrix into "source sub-matrix (SS)" and "derived sub-matrix (DS)." Each DS can be obtained by multiplying its corresponding SS by a scaling parameter. The configuration of SSMP is represented by a quadruple (x, y, n_x, n_y), where x/y indicates the dimensions of SS in the vertical/horizontal direction, and n_x, n_y indicate the extension numbers of DS in the vertical/horizontal direction. For example, in the case of Figure 4, the SS is a 2×2 block sub-matrix, covering the remaining 5 DS of the matrix. Therefore, its configuration is $(2, 2, 2, 3)$. Note that all SS and DS in a matrix have the same matrix dimension $[x, y]$, and each set of SS and DS forms a region with the same size $[x \cdot n_x, y \cdot n_y]$. The rest of the matrix is composed of the same pattern.

SSMP effectively reduces the number of parameters that need to be accessed during model computation. For a weight matrix of size $[D_x, D_y]$, when applied with SSMP with configuration (x, y, n_x, n_y), it is divided into $n = D_x D_y/(x \cdot n_x \cdot y \cdot n_y)$ regions. Each region contains one SS and $n_x n_y - 1$ DS. The storage requirements with SSMP include saving n

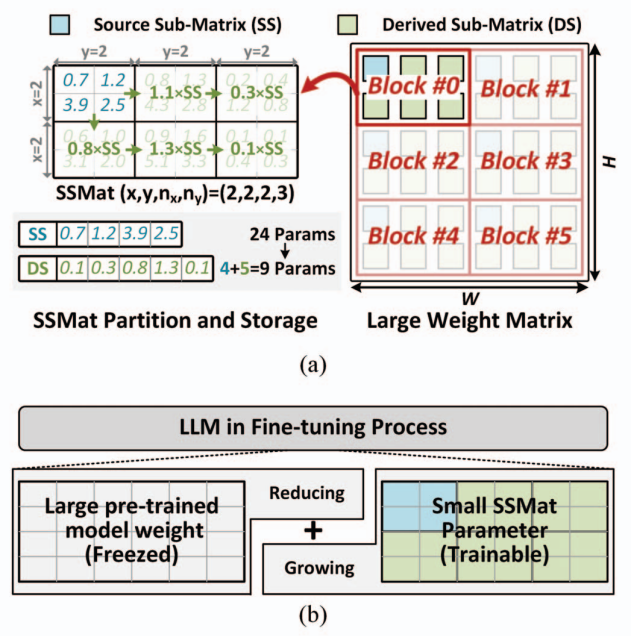


Fig. 4. SSMP partition and storage pattern.

SS of size $[x, y]$ and $n \cdot (n_x n_y - 1)$ scaling parameters. Since scaling is shared within a DS, the storage requirement for each DS is reduced to $1/(xy)$ of the original. Additionally, as all DS within a region share the parameters of one SS, the storage requirement is further reduced to $1/(n_x n_y)$ of the original.

Our experimental results indicate that large models can adopt SSMP configurations of (8, 8, 4, 4) or even more aggressive while maintaining acceptable end-to-end accuracy degradation (details in Section V). Under the above condition, SSMP reduces weight memory access by an average of 83.6%. For instance, it reduces the parameters in one linear layer of Bloom-7B from 67.1MB to 5.2MB or even smaller, which significantly optimizes the model's memory access bottleneck.

In addition to reducing memory footprint, SSMP can also reduce computation strength through data reuse. This is because the weights of DS and SS can be scalable with a shared parameter S. From the perspective of the output channel, the two weight kernels (scaled by S) perform matrix-vector multiplication with the same input vector slice, thus their partial sum is also scaled by S. From the input channel perspective, the weights corresponding to the two input tensor slices that need to be accumulated are scaled by S. This scaling can be transferred to the input tensor, and through optimized circuit design, computational power consumption can be further reduced. Theoretically, SSMP configured as (8, 8, 4, 4) can reduce over 72% of computational power consumption. Section IV will demonstrate how we utilize these characteristics to further optimize the hardware implementation for computation.

Algorithm 2: SSMP Fine-tuning

Data: Pre-trained LLM weight W
Result: SSMP-style LLM weight W'
1 **Initialization**
2 $W_{SS}, S, \sigma \leftarrow Init$; `// Trainable param`
3 FreezeParam(M);
4 **Training Loop**
5 **for** *trainSet in trainLoader* **do**
6 $W_{DS} \leftarrow W_{SS} \times S$;
7 $W_{new} \leftarrow$Concat(W_{SS}, W_{DS}); `// SSMP weight`
8 $W_{new} \leftarrow \sigma \cdot W + (1 - \sigma) \cdot W_{new}$;
9 $Output \leftarrow Forward(trainSet, W_{new})$;
10 Loss $\leftarrow L_{LM}(Output) + L(\sigma)$;
11 Update(W_{SS}, S, σ);
12 **end**
13 $W' \leftarrow \{W_{SS}, S, \sigma\}$;
14 **return** W';

B. Fine-tuning towards SSMP

Although SSMP shows efficient memory and computation characteristics, directly training an LLM with SSMP from scratch is challenging and expensive [92]. We propose an SSMP-oriented fine-tuning method, which fine-tunes a pre-trained LLM on an end-to-end task while adjusting its weight characteristics to align with SSMP.

The fine-tuning process is shown in Algorithm 2. It takes the pre-trained model as the input and freezes all the parameters for parameter efficiency. It creates two small learnable tensors: source sub-matrix weight tensor W_{SS} and derived sub-matrix scaling parameter tensor S_{DS}. During the training process, the model weight is composed of two parts: the pre-trained weight and the SS/DS weight. While the pre-trained weight is kept frozen, only updating the SS/DS weight, which includes the W_{SS} and the scaling factor S. In addition, the algorithm introduces a forget factor σ (Line 9) to help turn the weight matrix into SSMP style. The σ is initialized as 1, which means using the pre-trained weight at the very beginning of fine-tuning. We adopt regularization to punish the σ towards 0, and when its magnitude falls below 10^{-4} which is chosen by experiment, we can safely remove the pre-trained weight and only keep the SSMP weight.

Through fine-tuning, the accuracy of using SSMP can be fully recovered. Note that the trainable parameter, i.e., the W_{SS}, S, and σ is far smaller than the original weight. Thus the fine-tuning process is parameter-efficient and affordable. This method is similar to LoRA [30] since they both train a "biased" weight with small parameters but with a fundamental difference: it is the efficiency of LLM inference that the proposed method is designed to solve, rather than the training. Section V provides a detailed analysis of the training process and results.

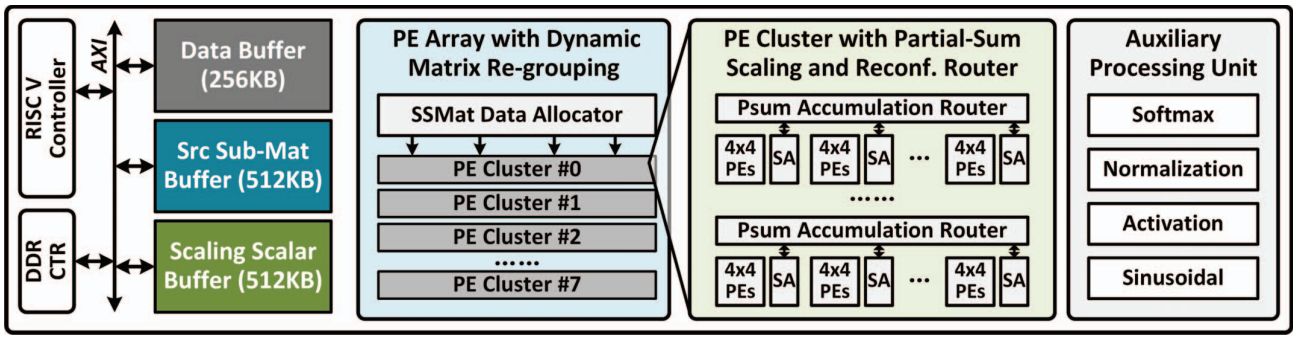


Fig. 5. Overall architecture of MECLA Processor.

IV. Architecture and Hardware Innovation of MECLA Processor

A. Motivation

Although SSMP provides the potential to improve the memory and compute efficiency of LLMs through the reuse of intermediate data and the reduction of computation strength, it is still a tough task to achieve effective acceleration on traditional general-purpose hardware platforms with rigid dataflow. The first reason is that SSMP requires fine-grained partial-sum level data reuse. The second reason is the reuse matrix varies, i.e., the SS and DS scaling factor matrix can both be the ones to be reused, which cannot be simply handled in previous works. To ensure the efficiency of computing with SSMP, we propose a specialized accelerator: MECLA. This section shows its hardware design details.

B. Overall Architecture

Figure 5 illustrates the overall architecture of MECLA. It is primarily composed of a RISC-V core, DDR controller, on-chip buffer totaling 1.25MB, 8 PE clusters, and an auxiliary unit. Communication between components is facilitated through the AXI bus. The RISC-V core serves as the central controller of MECLA, fetching instructions from the external host and controlling the processor's operation. The on-chip buffer contains a 256KB data buffer, a 512KB source sub-matrix buffer, and a 512KB scaling scalar buffer. The input tensor stored in the data buffer is broadcasted to the 8 PE clusters, while the source sub-matrix buffer and scaling scalar matrix buffer are distributed in each cluster. Each PE cluster consists of 16 sets of 4×4 PEs and 16 scaling accumulators (SA). The 4x4 PEs calculate partial sums (PSums) in matrix multiplication, which are multiplied and accumulated through the SA. The auxiliary processing unit is responsible for softmax, normalization, activation, and sinusoidal embedding calculations, ensuring that MECLA fully supports the inference process of LLMs.

As indicated in Section III, SSMP decomposes the weight matrix into several SS and DS, where SS represents weights of size $[x, y]$, and DS can be dynamically generated in real-time by SS and a scaling scalar matrix of size $[n_x, n_y]$. MECLA focuses on optimizing computations using this feature. We propose an on-the-fly matrix regrouping and dual-mode mapping strategy to achieve these objectives. The following two subsections will elaborate on these designs.

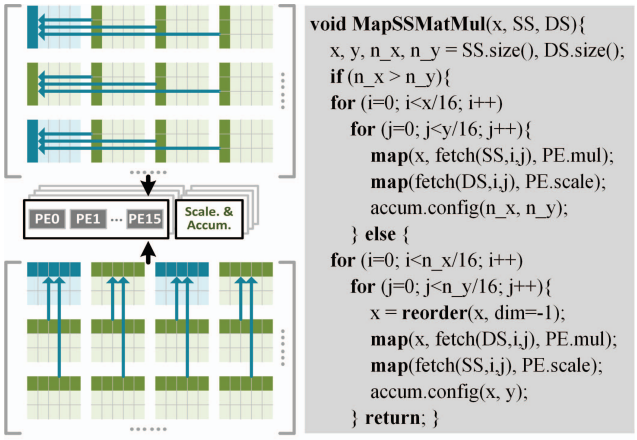


Fig. 6. PE workload mapping with source/derived sub-matrix re-grouping.

C. PE Array Design and On-the-fly Matrix Regrouping

MECLA applies reordered data mapping to exploit PSum reuse. After reading SS and DS, the matrix can be reconstructed online following the SSMP approach. By this means, each SS sub-matrix is scattered with intervals of $[x \times n_x, y \times n_y]$. To ensure hardware utilization, it is necessary to centralize data with relevance together to one or a few PE clusters, thereby avoiding redundant computation and data access.

SSMP involves inner product (input channel) and outer product (output channel) data reuse. One SS sub-matrix's x output channels and y input channels are reused by n_x / n_y times, respectively. Taking the SS in the upper left corner of Figure 6 as an example, its $(k{\cdot}x)^{th}$ row weight can be obtained by multiplying the 0^{th} row weight of the SS by the DS scaling factor, where k is an integer from 1 to $n_x - 1$. Similarly, using the i^{th} row of the SS sub-matrix can obtain the $(k{\cdot}x{+}i)^{th}$ row in the DS sub-matrix. Further, in terms of the inner product data reuse, using the j^{th} column of the SS sub-matrix can

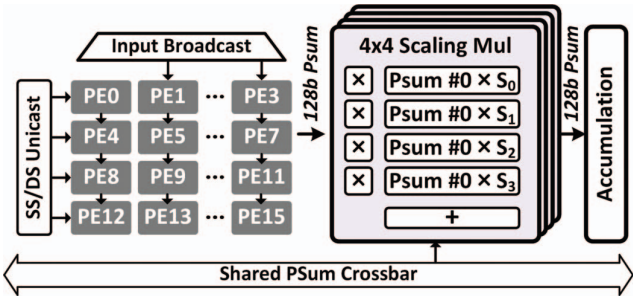


Fig. 7. PE array with scaling multiplier design in MECLA processor.

recover the $(k \times y + j)^{th}$ column of the DS. Thus, in MECLA's data mapping, it performs an on-the-fly matrix regrouping by putting the related sub-matrix rows and columns, with interval n_x or n_y together, as shown in 6.

Another feature of MELCA's data mapping is the unfixed weight matrix. This is because the SSMP partition method produces various n_x, n_y given different tasks, LLM models, and even different layers in one LLM, which introduces various requirements of data reuse. When n_x is large, the computation focuses more on how to reuse the PSums over different output channels. On the contrary, when n_y is large, the processor should be configured to exploit inner-product level redundancy and duplication.

MECLA applies a dual-mode source / derived sub-matrix gathering in its mapping, as shown in Figure 6. It selects the optimal matrix regrouping strategy for the data reuse according to SSMP configuration. If $n_x > n_y$, it maps the SS sub-matrix to the PE weight buffer, and maps the DS scaling scalar to the scale buffer to reuse the PSum of the PE array. Otherwise, it changes the matrix multiplication sequence (detailed in Section IV.D) and maps the DS scaling scalar to the weight buffer and the SS sub-matrix weight to the scale buffer. In this situation, the weight of SS are fully reused, which is in line with the inner-product reuse computation.

Based on the above observation and design, MECLA's PE cluster consists of 16 PE arrays, and the design of each array is illustrated in Figure 7. It comprises a set of 4x4 PEs forming a matrix-vector multiplier. In this set, the input data (activation vector) is broadcasted, and the weight data (SS or DS scaling scalar matrix) is unicast. Matrix multiplication generates 4 32-bit partial sums (PSum), which are then transmitted to a 4x4 scaling multiplier array for further computation. Each PSum corresponds to 4 multipliers, and these multipliers multiply the PSum by up to 4 scaling factors. The results are aggregated as the output of the PE cluster, which is summed with the results from other clusters to obtain the final output. Additionally, there is an accumulator involved in the scaling multiplier array, which directly accumulates the input (PSum) and serves as the output. This design addresses the data flow efficiency issue for long inner product dimension y.

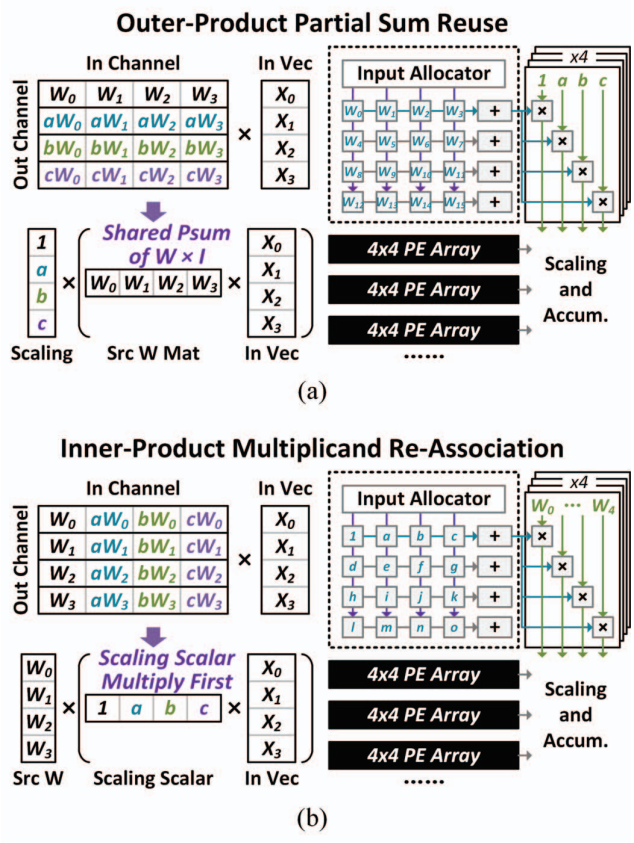


Fig. 8. Different PE array workflow of MECLA processor. (a) Outer-product optimization. (b) Inner-product optimization

D. Outer-product Reuse and Inner-product Re-association

Given a matrix partitioned by the SSMP method, it shows different data reuse features from inner-product and outer-product perspectives. This section shows how MECLA's PE array deals with the two situations.

If the weight matrix emphasizes outer-product reuse, its matrix-vector multiplication can be illustrated as shown in Figure 8(a) after grouping the related rows (indicated with the same color) together with the regrouping method in Section IV.C. In this example, the weight data is regrouped by re-ordering the output channel. The 4x4 weight matrix can be viewed as a 4×1 scaling vector multiplying a 1×4 weight vector. The former is a slice of DS, and the latter is a slice of SS. According to the associative law of matrix multiplication, the 1×4 weight vector can be multiplied with the 4×1 input vector first to obtain a 1×1 shared PSum, followed by the multiplication between PSum and 4 scaling scalars. In terms of the data mapping, the 4 weight data is stored in 4 PEs in one row a 4×4 PE array, and the scaling factor a, b, c is stored in the scaling multiplier array, and the unused scaling multipliers are gated for energy saving. Similarly, the other 12 PEs compute the PSum of another three 1×4 SS weight slices with the same input.

When the weight matrix has more inner-product reuse, the aforementioned PSum sharing strategy can no longer improve the efficiency. MECLA adopts computation re-association for this issue, as shown in 8(b). The weight data in SS is regrouped by re-ordering the input channel, and a 4×4 weight matrix can be viewed as a 4×1 weight vector multiplying a 1×4 scaling vector. Unlike the outer-product mode, MECLA computes the multiplication of the scaling vector and the input vector first with the PEs, and then multiplies the PSum with the weight data in the scaling multiplier array.

MECLA's data mapping and reuse mechanism does not treat the weight matrix in its original sequence. It dynamically regroups the matrix and swap the multiplication order to ensure data reuse. By reusing the PSum, the number of computation operations is reduced from 28 muls and 16 adds to 4 muls and 4 adds, and the power consumption is reduced by 85.6% in the given example.

E. Design Details

Operation and pipeline overview. MECLA uses fully pipeline dataflow under the control of the RISC V core to guarantee hardware utilization. The processor takes an input matrix and a weight matrix from DDR decomposed by SSMP as input to generate the output of a whole attention layer. As soon as the inputs are ready, the auxiliary unit performs the preprocessing, such as embedding and normalization. The result returns back to the data buffer and the accelerator starts computing the matrix multiplication with SSMP weight. During the computation, the controller finds the optimal mapping strategy and data reuse pattern and maps the weight data, composed by SS sub-matrix in SS buffer and DS scaling scalar in DS buffer into the PE array. The SSMP multiplication applies to the QKV linear and FFN linear of a transformer layer, which is the absolute dominant computation and memory access bottleneck according to the analysis in Section II. Note that there are a few matrix partitions that cannot be decomposed with SSMP, and the processor fetches these weight data and compensates them with standard matrix multiplication as soon as the corresponding input tokens are mapped onto the PE array and finishes the computation with SSMP partitions. Along with the QKV linear, the DDR fetches the KV cache to the SS and DS buffer for attention computation, which starts when the QKV linear finishes. This memory access and computation scheme is applied to all the computations until the processor calculates the output token embedding.

Memory modules. MECLA contains three SRAM implemented on chip, and is designed for LLM inference with 8b. The 256KB data buffer is used to store the token, intermediate and output matrix. A typical input vector of LLM has an embedding dimension of 16384 (in the linear layer of 7B scale model), which requires 16KB for one token vector. Due to parallelism strategies, input prompt and batch concerns, we support at most 16 tokens to store on-chip. The Src sub-mat buffer and scaling scalar buffer are used to store the weight with SSMP. A typical static weight in LLM can be reduced to 64KB-1MB, and the two buffer is capable for the majority of situations. Larger weights are split into several output channels for sequential computation. Additionally, these two buffers stores the weight matrix which cannot apply SSMP and the KV cache. It stores a maximum of 4k token length of a single head in a typical 7B scale LLM.

PE array and scaling multiplier. The PE array of MECLA is designed for PSum reuse. However, SSMP has various configuration and reuse patterns, which require a reconfigurable PE array to suit all the cases. We use a 4×4 PE array along with 4×4 scaling multipliers as the basic unit. According to algorithm evaluation results (detailed in Section V.II), the max reust times is set at 32, and 8 basic units forms a group. The 8 scaling multipliers in a group communicate with each other using the internal crossbar, which supports sharing the PSum results for more reuse times.

V. Evaluation

A. Experiment Setup

Workloads and setup. We use several language models and tasks to evaluate the SSMP and MECLA processor. The models include open-source language model RoBERTa base, RoBERTa large [44], Bloom 1B7, Bloom 7B [84], and LLaMA-2 7B [72]. We use 10 tasks in total, including 8 tasks from general language understanding evaluation (GLUE) datasets [75] (linguistic acceptability CoLA [81], sentiment analysis SST-2 [67], sentence similarity and paraphrasing MRPC [19], QQP, and STS-B [9], and natural language inference MNLI [82], question answering QNLI [60], and textual entailment RTE [17]), databricks-dolly-15k general natural language processing, and wikitext-2 language modeling task [49]. We use the PyTorch [53] and Hugging Face library [83] to implement the model with SSMP, and use RoBERTa large, Bloom 7B, and LLaMA 13B as the teacher models to fine-tune the SSMP model with knowledge distillation. In each fine-tuning, we use a learning rate of {5e-4, 1e-4, 5e-5} and batch size of {16, 32} and train the model for 20 epochs. After the fine-tuning, we capture the model weight and run-time activation data to perform post-layout hardware simulation to obtain the hardware metrics.

Hardware implementations. We perform RTL design for MECLA processor and complete the synthesis using Synopsys Design Compiler under 28nm CMOS technology. We use Synopsys IC Compiler II to complete the placement and routing for the chip and generate the netlist and post layout, as shown in Figure 13. To obtain the power and latency, we use Synopsys VCS and PrimeTime tools for post-layout power analysis with the data and waveform captured from PyTorch as mentioned above.

B. Accuracy and Model Compression Evaluation

We use the fine-tuned pre-trained language models to evaluate the model accuracy of SSMP method with a total of 20 models and tasks. SSMP has 4 configurable hyperparameters:$[x, y, n_x, n_y]$, and the model's compression ratio increases as each value gets larger. All the linear layers of a model share the same set of $[x, y, n_x, n_y]$ under a

TABLE II
ACCURACY OF DIFFERENT LANGUAGE MODELS WITH MECLA OPTIMIZATION

Model	RoBERTa Large								RoBERTa Base								LLaMA-2 7B		Bloom 7B	Bloom 1B7
Task	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Dolly	Wiki2	Wiki2	Wiki2
Baseline	90.6	96.2	90.2	68.2	94.8	91.6	85.2	92.3	87.5	95.1	89.7	63.4	93.3	90.8	86.6	91.5	29.7	5.9	12.3	29.9
MECLA (standard)	89.8	96.0	89.5	67.3	94.1	90.9	84.5	92.2	87.1	95.0	89.6	62.8	93.2	90.7	86.0	91.2	29.6	6.1	12.8	30.0
MECLA (aggressive)	88.9	94.6	88.6	66.4	93.7	90.0	83.5	90.9	86.1	93.7	88.6	61.9	91.7	89.7	85.6	90.3	28.1	6.4	13.4	30.6

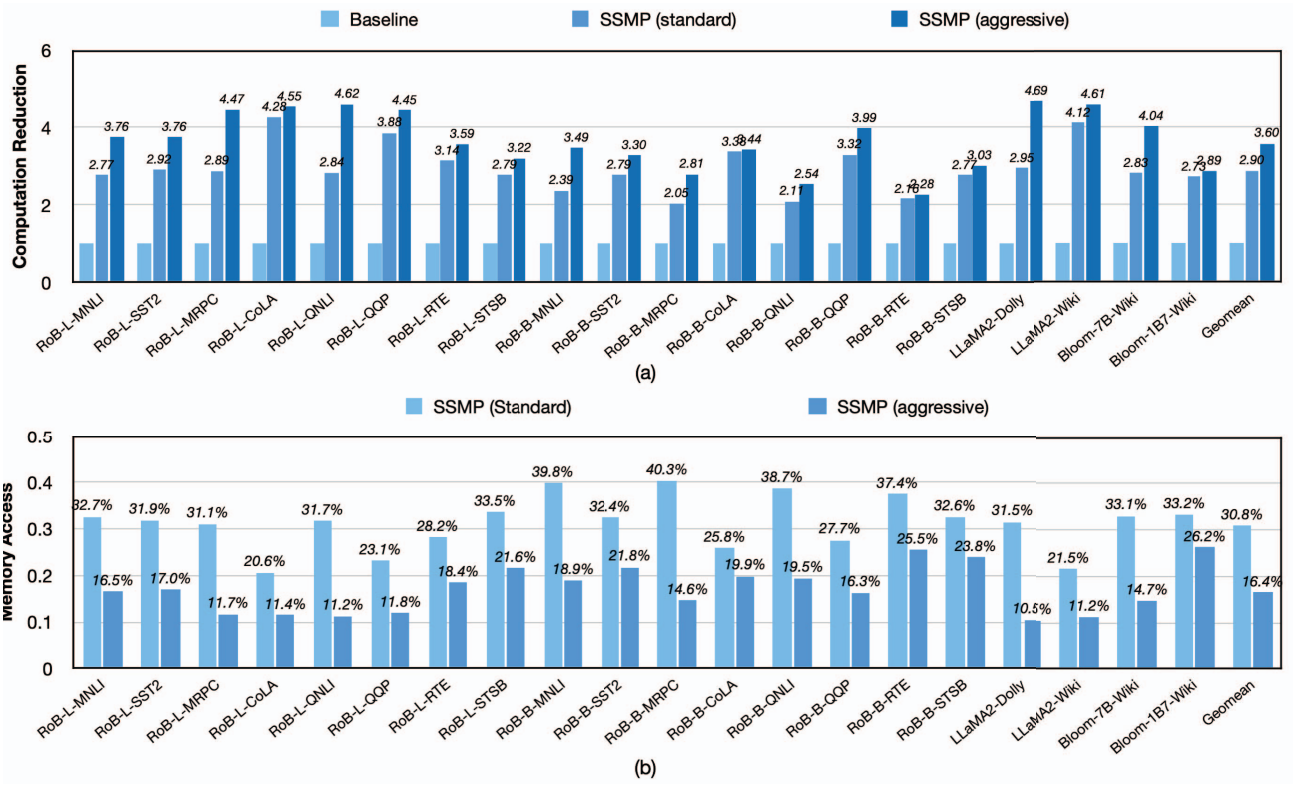


Fig. 9. Model inference computation reduction (a) and memory footprint reduction (b) on MECLA processor with SSMP method.

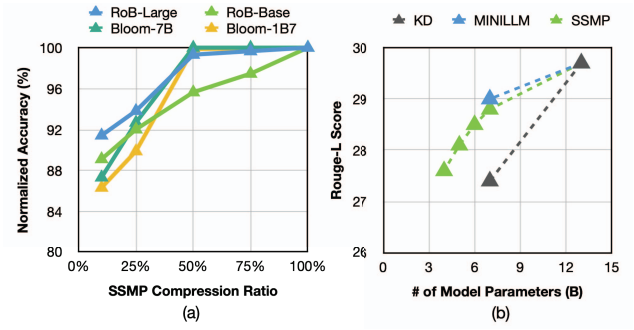


Fig. 10. SSMP compression for different scale language model (a) and comparison with state-of-the-art distillation methods (b).

given configuration of fine-tuning. We set the search space of each hyperparameter in $\{2, 4, 8, 16\}$ and use grid search and successive halving method to get the configuration within the finetuning of each task. Since MECLA and SSMP are designed for efficient inference, and the fine-tuning parameter is also reduced greatly by SSMP, we assume this search effort is acceptable.

For the accuracy baseline, we use the accuracy of models fine-tuned with LoRA (dim=16) [30] with the same training parameters. Based on the baseline accuracy, we obtain two settings of MECLA: standard and aggressive. MECLA standard refers to the setting with the smallest computation and memory access effort while the accuracy degradation is within 2% for GLUE tasks and 5% for wikitext language modeling perplexity and dolly Rouge-L score. MECLA aggressive refers to the smallest model setting within 5% accuracy degradation for GLUE and 10% for wikitext or dolly. Table II shows the end-to-end accuracy (F1 for MRPC, QQP, accuracy for SST-2, QNLI, MNLI, RTE, Matthews correlation for CoLA, Pearson correlation for STS-B, perplexity for wikitext-2 language modeling, and Rouge-L score for dolly dataset).

As shown in the results, using SSMP for inference is

feasible and introduces decent degradation. For example, when processing the GLUE benchmark with RoBERTa large model, the SSMP introduces an average of 0.6% / 1.1% with standard / aggressive SSMP, respectively, let alone its capability for reducing the computation and memory access efforts. This feature guarantees outstanding acceleration of MECLA since a high compression ratio with SSMP is guaranteed from the perspective of accuracy.

Figure 10(a) shows the SSMP performance with different scales of RoBERTa (base and large) and Bloom (1B7 and 7B) models. Since SSMP detects and removes the duplicated computation of the big weight matrix according to a given ratio, it works for models with different scales. E.g., when reducing approximately 50% of weight parameters with SSMP, the Bloom 1B7 and Bloom 7B show 1.2 and 0.7 increase of wikitext-2 perplexity. The result also shows that for larger scale LLMs, the SSMP can achieve higher compression ratio given the same accuracy loss limitation. This is because the occurrence probability of sub-matrix similarity increases due to larger weight matrix dimension, which leads to better SSMP compression performance.

Figure 10(b) compares the LLaMA model compression results of SSMP method with state-of-the-art LLM compression methods: knowledge distillation (KD) and MiniLLM [26]. They all use the LLaMA 13B as the teacher for distillation. Compared to naive KD, SSMP can better preserve the capability to process general language tasks of LLM. When the compressed model has approximately 7B parameter number, the Rough-L score of SSMP is 1.4 higher than KD, and is only 0.2-point less than the LLM-oriented fine-tuning method MiniLLM. The number of parameters with SSMP is less than 57.1% of KD when achieving the almost the same Rouge-L score of 27.4. Further, unlike KD and MiniLLM, which involve distilling a large-scale LLM (13B) into a fixed-scale smaller LLM (7B), the SSMP method aims to find out and avoid computational redundancies within the LLM. Therefore, SSMP offers more choices in terms of parameter quantity and precision. Within 1-point score degradation compared to the MiniLLM 7B distillation, the SSMP provides 3 compression options from 5B to 7B. Thus, SSMP yields LLM compression results comparable to state-of-the-art methods while offering a broader range of model sizes suitable for various devices, facilitating a nuanced balance between model performance and storage constraints. Notably, SSMP introduces the potential for computational reuse, further enhancing its versatility and applicability in practical scenarios.

C. Performance Evaluation

To evaluate the performance with MECLA, we obtain the runtime input and weight data from PyTorch. We split the weight data into SS and DS matrices of SSMP and generate the RISC-V instructs based on model information, such as matrix dimensions. We use these data to perform post-layout simulation with VCS to obtain the hardware performance, and we use DDR4 simulator to simulate the data load movement for MECLA processor. Figure 11 shows the computation and

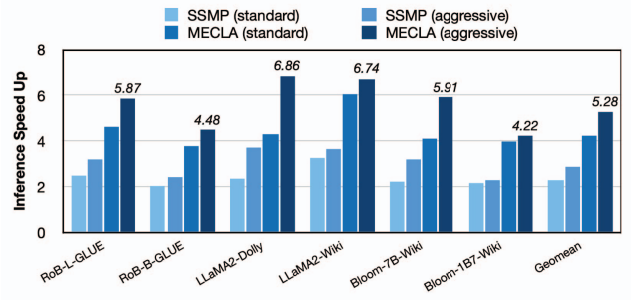


Fig. 11. Inference speed up with SSMP on GPU and MECLA processor.

memory reduction of MECLA with no optimization and the proposed features. Since the SSMP is able to compress the 7B or 13B-scale LLM down to half or even one-tenth, current consumer-level devices have enough DRAM to store the complete model, thus we do not consider model parallelism in figure 11's evaluation. MECLA standard / aggressive achieves an average reduction of 65.5% / 72.2% in computation and 69.2% / 83.6% in memory footprint across 20 benchmarks.

We also compare MECLA's inference performance on single 32GB NVIDIA V100 GPU. Since the peak performance of V100 is 125TOPS (INT8, 1GHz), we use 32 MECLA processors (total with 131.2TOPS@INT8) with data and model parallelism for performance comparison. We set each MECLA processor to have a maximum of 8GB of external memory for storing data. For models with storage requirements of less than 8GB, such as RoBERTa, Bloom 1B7, and highly compressed LLaMA 7B, 32 MECLA processors utilize data parallelism to maximize throughput. However, for other scenarios, such as LLaMA and Bloom 7B with compression rates lower than 50%, we employ 2-model parallelism and 16-data parallelism to meet computational demands. The data bandwidth between parallel MECLA processors is set 900GB/s as NVlink in our simulation. Additionally, since both the V100 and the MECLA cluster can store the whole model and all the cached files with its memory in the evaluation case, thus we ignore all the external memory access costs such as weight data load for the two hardware platforms. According to the experiment results, using SSMP standard / aggressive on V100 achieves 2.32× / 2.88× of inference speed improvement on an average of 20 benchmarks, with a peak improvement of 3.29× / 3.75× at the wikitext and dolly general language task with LLaMA. The improvement is limited because of two reasons. 1) Although the SSMP reduces the parameter amount, the GPU still needs to recover the matrix, which introduces extra computation requirement. 2) Small proportion of SSMP weight matrix cannot be partitioned into SS and DS and needs sparse matrix computation, which suffers from low utilization on GPU compared to ASIC processors. With the aid of specifically designed circuits in MECLA, the throughput improvement reaches an average of 4.25× / 5.28×, and the peak gain is up to 6.26× / 6.86× compared to naive inference on V100.

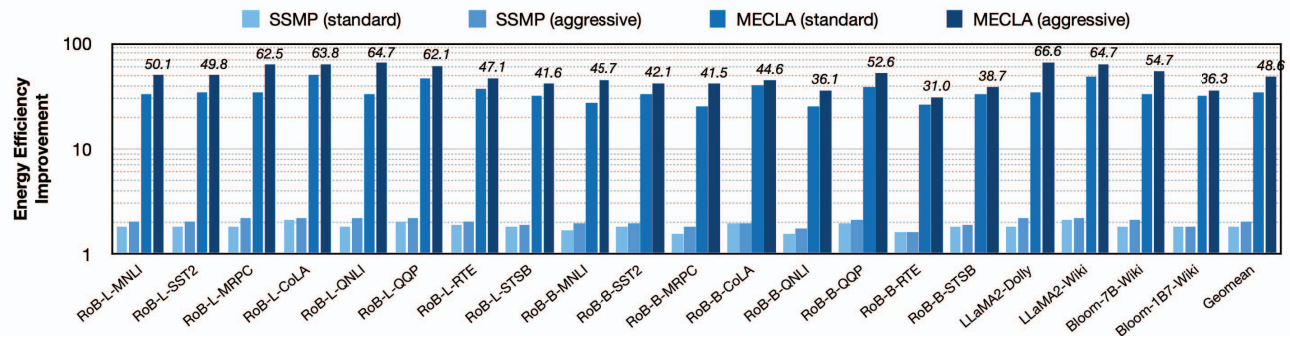


Fig. 12. Energy efficiency improvement and breakdown analysis. +SSMP: V100 with SSMP method. +MECLA: MECLA processor with SSMP method.

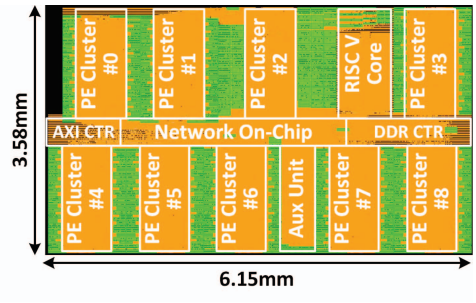


Fig. 13. Post layout of MECLA processor.

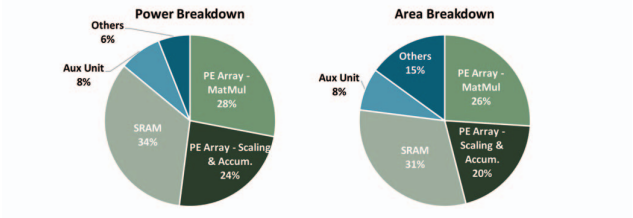


Fig. 14. Power and area breakdown of MECLA processor.

D. Area and Energy Efficiency Evaluation

We complete the RTL synthesis, placement, and routing for MECLA processor with 28nm 1P8M CMOS for the chip area. Then we use Synopsys PrimeTime and VCS to perform post simulation and power analysis with actual running data for precise energy evaluation. Figure 13 shows its post layout, and Figure 14 shows its power and area breakdown. MECLA has a die area of 22.02 mm^2. The PE array, on-chip buffer, and scaling accumulator take up 26%, 31%, and 20% of the area, respectively. During operation at 1GHz, the processor's average power is 2.87W, with the PE array and memory consuming 52.3% and 34.4% of the energy. In comparison, the latest A17 Pro mobile SoC with 3nm technology consumes 11-14W of power when running heavy tasks. Only 2.56% of energy overhead (normalized to 3nm technology) would be required if MECLA processor is embedded in the mobile SoC to enable efficient LLM inference.

TABLE III
COMPARISON OF ACCELERATION EFFECT AND SPECIFICATIONS WITH STATE-OF-THE-ART ACCELERATORS

	SpAtten$_{1/8}$ [77]	Sanger [46]	FACT [55]	MECLA
Accelerate	Decoder (attention only)	Encoder (Attention only)	Encoder (whole model)	**LLM**[1] **(whole model)**
Optimization[2]	C Only	C Only	C Only	**C & M**
Technology (nm)	40	55	28	**28**
Area (mm^2)	1.55	16.9	6.03	**22.02**
Throughput (GOPS)	360	529	928	**14008**
Energy Efficiency (GOPS/W)	382	192	4388	**7088**

1) MECLA's acceleration works for both encoder and decoder, and performs better on the latter.
2) C: computation. M: memory access.

Figure 12 shows the energy efficiency improvement of MECLA processor and V100 GPU compared to naive implementation baseline. On geometric average, using standard and aggressive SSMP optimization for inference with GPU improves the energy consumption by an average of 1.83× / 2.01×. Combined with the specially designed circuits in MECLA, the efficiency improvement is improved by 34.3×/ 48.6×, achieving 5.01 / 7.09 TOPS/W on average. The peak energy efficiency reaches 7.43 and 9.71TOPS/W at 1.0V, 1GHz, using MECLA standard and aggressive. Such high improvements come from two main factors. For memory concerns, MECLA saves 80.4%-89.5% weight parameter access, which greatly reduces the memory footprint and alleviates the bandwidth bounding. For computation concerns, the PSum reuse reduces 76.6%-78.7% computation with the special PE array design.

E. Comparison with Other Accelerators

We compare MECLA with the state-of-the-art Transformer accelerator SpAtten [77], Sanger [46], and FACT [55]. These works are typical accelerators that exploit the attention mechanism to improve the throughput and energy efficiency of Transformer inference. Table III summarizes their specifications. Sanger and FACT are both optimized for the "encoder" Transformer, such as BERT and ViT. SpAtten focuses on the auto-regressive "decoder" like GPT-2 via token pruning. These works are all designed for small-scale Transformers by finding redundancy of attention [46] and linear computation [55], [77].

However, LLM has extremely expanded parameter sizes and computation, emphasizing the optimization for both memory and computation. Also, the linear layer in LLM consumes over 95% of the computation and memory, which is the dominant bottleneck for LLM and makes attention-oriented optimization such as [27], [28], [46] less effective. To the best of our knowledge, MECLA is so far the first work that uses parameter efficiency strategies for LLM inference, reducing both memory and computation effort for the linear layer of LLM. It achieves a peak energy efficiency of 9715GOPS/W and an average of 7088GOPS/W, which is 12.99×, 18.80×, and 1.62×, and 113.4× higher than the three counterparts and V100, with different technology normalized to 28nm for fair comparison. Note that when we compare the improvement of average energy efficiency, we compare the metric proposed in each paper. However, when [55] handles autoregressive LLM like LLaMA, GPT, and Bloom, its efficiency is degraded by over 5.0× since it is designed for encoder-based models like BERT and ViT and cannot handle the memory access bottleneck for LLM. Therefore, for LLM efficiency, MECLA has over 8.1× efficiency improvement when processing LLM compared with [55].

VI. Related Work

Computation Efficiency of Transformer. The outstanding performance and large computation of the Transformer model have attracted lots of interest in improving the computation efficiency. Many works discover the existence of unimportant and neglectable computation introduced by attention mechanism [73] and softmax function, and make efforts in sparse attention algorithm [5], [13], [15], [16], [22], [36], [56], [61], [70], [74], [78], [91], sparse attention processor [20], [21], [25], [28], [41], [42], [46], [55], [57], [76], [77], [88], [94], low precision or approximation attention computation unit [27], [80], [89] to improve the computation efficiency. Some works notice the heavy burden of linear layers and propose pruning [55], [77] or low precision computation [55] to alleviate the burden. Our MECLA focuses on the linear layer in LLM after computation bottleneck analysis, but it uses fine-grained level data reuse instead of the token or weight pruning method as previous work does. This is because the former suffers from a low optimization effect, and the latter is hard to realize since full parameter retraining of LLM towards sparsity is of huge expense w.r.t. time and resources.

Parameter Efficiency for Transformer. Research has discovered that Transformer can have higher accuracy and performance with scaling (increasing the model size), and more precise, versatile large Transformer model have been showcasing their value [2], [7], [8], [14], [18], [50], [71], [72], [84], [92]. The issue of large parameters has attracted lots of concern. Some works use distilling [63], [68] or quantization [89], [90] to obtain a small-scale model from pre-trained large model. Some works propose sharing parameters, by making different layers [39], [69] or heads [65] the same weight to reduce parameter amount. These works either have a low compression ratio or can be applied to limited models or tasks. For LLMs with even larger weight sizes, previous works focus on model quantization [84], [92], distillation [26], or low parameter fine-tuning [30], [48]. Unlike these works, our MECLA uses fine-grained data sharing inside each matrix multiplication to exploit the data reuse, which greatly improves the parameter efficiency while keeping the versatility of LLM.

Neural Network Accelerator. Neural networks (NNs) have demonstrated their ability in a variety of scenarios which even outperform humans. Deploying NN accelerator for efficient inference is an effective solution to NN's high computation and memory costs. Previous accelerators improve the computation efficiency from sparsity [1], [21], [24], [27], [28], [33], [35], [40], [45], [47], [54], [55], [62], [66], [79], [88], [93], hardware utilization [10], [11], [32], data mapping and reuse [29], [38], [69], [87], and energy-efficient PE designs [27], [52], [80]. This work is so far the first NN accelerator for LLM, which employs a novel matrix partition strategy and data reuse pattern to optimize the huge computational and memory access bottlenecks of LLM.

Moreover, MECLA is an NN accelerator designed for applying LLMs from the cloud to the edge, distinguished from prior works that focused on optimizing for specific scenarios. On cloud servers, the low compute-to-memory ratio in LLM's matrix-vector multiplication results in low computation efficiency, and previous works propose optimization methods such as introducing input batchsize [12], [51], [86]. This paper, however, offers an alternative perspective from weight parameter efficiency. For edge devices, even small-scale LLMs like LLaMA-7B (14GB) introduce significant storage burdens, alongside challenges related to computation and power consumption. MECLA's improvements in computational efficiency and model compression effectively mitigate these issues.

VII. Conclusion

We propose MECLA, which is, to our knowledge, the first memory-compute-efficient accelerator for cloud-to-edge LLM inference. We first propose the SSMP matrix partition method, which splits a large parameter matrix into two tiny-scale source sub-matrix and derived sub-matrix, and the latter can be obtained by scaling the former. Also, we propose a fine-tuning method to turn a pre-trained LLM into SSMP format. Then, we propose an efficient hardware processor, MECLA, to exploit the PSum reuse to save computation effort. MECLA effectively reduces the memory footprint and computation by 69.2-83.6% and 65.5-72.2% on an average of 20 benchmarks. Compared to SOTA accelerators, it is 113.4×, 12.99×, and 1.62× more energy saving compared to V100 GPU, SpAtten, and FACT.

References

[1] J. Albericio, P. Judd, T. H. Hetherington, T. M. Aamodt, N. D. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 2016, pp. 1–13. [Online]. Available: https://doi.org/10.1109/ISCA.2016.11

[2] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen, E. Chu, J. H. Clark, L. E. Shafey, Y. Huang, K. Meier-Hellstern, G. Mishra, E. Moreira, M. Omernick, K. Robinson, S. Ruder, Y. Tay, K. Xiao, Y. Xu, Y. Zhang, G. H. Abrego, J. Ahn, J. Austin, P. Barham, J. Botha, J. Bradbury, S. Brahma, K. Brooks, M. Catasta, Y. Cheng, C. Cherry, C. A. Choquette-Choo, A. Chowdhery, C. Crepy, S. Dave, M. Dehghani, S. Dev, J. Devlin, M. Díaz, N. Du, E. Dyer, V. Feinberg, F. Feng, V. Fienber, M. Freitag, X. Garcia, S. Gehrmann, L. Gonzalez, G. Gur-Ari, S. Hand, H. Hashemi, L. Hou, J. Howland, A. Hu, J. Hui, J. Hurwitz, M. Isard, A. Ittycheriah, M. Jagielski, W. Jia, K. Kenealy, M. Krikun, S. Kudugunta, C. Lan, K. Lee, B. Lee, E. Li, M. Li, W. Li, Y. Li, J. Li, H. Lim, H. Lin, Z. Liu, F. Liu, M. Maggioni, A. Mahendru, J. Maynez, V. Misra, M. Moussalem, Z. Nado, J. Nham, E. Ni, A. Nystrom, A. Parrish, M. Pellat, M. Polacek, A. Polozov, R. Pope, S. Qiao, E. Reif, B. Richter, P. Riley, A. C. Ros, A. Roy, B. Saeta, R. Samuel, R. Shelby, A. Slone, D. Smilkov, D. R. So, D. Sohn, S. Tokumine, D. Valter, V. Vasudevan, K. Vodrahalli, X. Wang, P. Wang, Z. Wang, T. Wang, J. Wieting, Y. Wu, K. Xu, Y. Xu, L. Xue, P. Yin, J. Yu, Q. Zhang, S. Zheng, C. Zheng, W. Zhou, D. Zhou, S. Petrov, and Y. Wu, "Palm 2 technical report," *CoRR*, vol. abs/2305.10403, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2305.10403

[3] S. Arora, A. Narayan, M. F. Chen, L. J. Orr, N. Guha, K. Bhatia, I. Chami, and C. Ré, "Ask me anything: A simple strategy for prompting language models," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/pdf?id=bhUPJnS2g0X

[4] K. Bao, J. Zhang, Y. Zhang, W. Wang, F. Feng, and X. He, "Tallrec: An effective and efficient tuning framework to align large language model with recommendation," in *Proceedings of the 17th ACM Conference on Recommender Systems*, ser. RecSys '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1007–1014. [Online]. Available: https://doi.org/10.1145/3604915.3608857

[5] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *CoRR*, vol. abs/2004.05150, 2020.

[6] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, 2003. [Online]. Available: http://jmlr.org/papers/v3/bengio03a.html

[7] J. Betker, G. Goh, L. Jing, T. Brooks, J. Wang, L. Ouyang, J. Zhuang, J. Lee, Y. Guo, W. Manassra, P. Dhariwal, C. Chu, Y. Jiao, and A. Ramesh, "Improving image generation with better captions," 2023, https://cdn.openai.com/papers/dall-e-3.pdf.

[8] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[9] D. M. Cer, M. T. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia, "Semeval-2017 task 1: Semantic textual similarity - multilingual and cross-lingual focused evaluation," *CoRR*, vol. abs/1708.00055, 2017. [Online]. Available: http://arxiv.org/abs/1708.00055

[10] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Architectural Support for Programming Languages and Operating Systems, ASPLOS 2014, Salt Lake City, UT, USA, March 1-5, 2014*, R. Balasubramonian, A. Davis, and S. V. Adve, Eds. ACM, 2014, pp. 269–284. [Online]. Available: https://doi.org/10.1145/2541940.2541967

[11] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[12] Z. Cheng, J. Kasai, and T. Yu, "Batch prompting: Efficient inference with large language model apis," *CoRR*, vol. abs/2301.08721, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2301.08721

[13] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *CoRR*, vol. abs/1904.10509, 2019.

[14] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, "Palm: Scaling language modeling with pathways," *J. Mach. Learn. Res.*, vol. 24, pp. 240:1–240:113, 2023. [Online]. Available: http://jmlr.org/papers/v24/22-1144.html

[15] G. M. Correia, V. Niculae, and A. F. T. Martins, "Adaptively sparse transformers," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds. Association for Computational Linguistics, 2019, pp. 2174–2184. [Online]. Available: https://doi.org/10.18653/v1/D19-1223

[16] B. Cui, Y. Li, M. Chen, and Z. Zhang, "Fine-tune BERT with sparse self-attention mechanism," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds. Association for Computational Linguistics, 2019, pp. 3546–3551. [Online]. Available: https://doi.org/10.18653/v1/D19-1361

[17] I. Dagan, O. Glickman, and B. Magnini, "The PASCAL recognising textual entailment challenge," in *Machine Learning Challenges, Evaluating Predictive Uncertainty, Visual Object Classification and Recognizing Textual Entailment, First PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, April 11-13, 2005, Revised Selected Papers*, ser. Lecture Notes in Computer Science, J. Q. Candela, I. Dagan, B. Magnini, and F. d'Alché-Buc, Eds., vol. 3944. Springer, 2005, pp. 177–190. [Online]. Available: https://doi.org/10.1007/11736790_9

[18] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186.

[19] W. B. Dolan and C. Brockett, "Automatically constructing a corpus of sentential paraphrases," in *Proceedings of the Third International Workshop on Paraphrasing, IWP@IJCNLP 2005, Jeju Island, Korea, October 2005, 2005*. Asian Federation of Natural Language Processing, 2005. [Online]. Available: https://aclanthology.org/I05-5002/

[20] H. Fan, T. Chau, S. I. Venieris, R. Lee, A. Kouris, W. Luk, N. D. Lane, and M. S. Abdelfattah, "Adaptable butterfly accelerator for attention-based nns via hardware and algorithm co-design," in *55th IEEE/ACM International Symposium on Microarchitecture, MICRO 2022, Chicago, IL, USA, October 1-5, 2022*. IEEE, 2022, pp. 599–615.

[21] C. Fang, A. Zhou, and Z. Wang, "An algorithm–hardware co-optimized framework for accelerating n:m sparse transformers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 11, pp. 1573–1586, 2022.

[22] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *J. Mach. Learn. Res.*, vol. 23, pp. 120:1–120:39, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-0998.html

[23] Y. Gao, T. Sheng, Y. Xiang, Y. Xiong, H. Wang, and J. Zhang, "Chat-rec: Towards interactive and explainable llms-augmented recommender system," *CoRR*, vol. abs/2303.14524, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2303.14524

[24] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. N. Vijaykumar, "Sparten: A sparse tensor accelerator for convolutional neural networks," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019*. ACM, 2019, pp. 151–165.

[25] S. Goyal, A. R. Choudhury, S. Raje, V. T. Chakaravarthy, Y. Sabharwal, and A. Verma, "Power-bert: Accelerating BERT inference via progressive word-vector elimination," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 3690–3699. [Online]. Available: http://proceedings.mlr.press/v119/goyal20a.html

[26] Y. Gu, L. Dong, F. Wei, and M. Huang, "MiniLLM: Knowledge distillation of large language models," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=5h0qf7IBZZ

[27] T. J. Ham, S. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J. Park, S. Lee, K. Park, J. W. Lee, and D. Jeong, "A^3: Accelerating attention mechanisms in neural networks with approximation," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22-26, 2020*. IEEE, 2020, pp. 328–341.

[28] T. J. Ham, Y. Lee, S. H. Seo, S. Kim, H. Choi, S. J. Jung, and J. W. Lee, "ELSA: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *Proceedings of the 48th Annual International Symposium on Computer Architecture*, ser. ISCA '21. IEEE Press, 2021, p. 692–705.

[29] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. W. Fletcher, "Ucnn: Exploiting computational reuse in deep neural networks via weight repetition," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. IEEE Press, 2018, p. 674–687. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00062

[30] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9

[31] F. Jelinek, "Interpolated estimation of markov source parameters from sparse data," in *Proc. Workshop on Pattern Recognition in Practice, 1980*, 1980.

[32] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, p. 1–12, jun 2017. [Online]. Available: https://doi.org/10.1145/3140659.3080246

[33] S. Kang, D. Han, J. Lee, D. Im, S. Kim, S. Kim, J. Ryu, and H.-J. Yoo, "Ganpu: An energy-efficient multi-dnn training processor for gans with speculative dual-sparsity exploitation," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 9, pp. 2845–2857, 2021.

[34] B. Keller, R. Venkatesan, S. Dai, S. G. Tell, B. Zimmer, C. Sakr, W. J. Dally, C. T. Gray, and B. Khailany, "A 95.6-tops/w deep learning inference accelerator with per-vector scaled 4-bit quantization in 5 nm," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 4, pp. 1129–1141, 2023.

[35] S. Kim, J. Lee, S. Kang, J. Lee, and H.-J. Yoo, "A 146.52 tops/w deep-neural-network learning processor with stochastic coarse-fine pruning and adaptive input/output/weight skipping," in *2020 IEEE Symposium on VLSI Circuits*, 2020, pp. 1–2.

[36] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=rkgNKkHtvB

[37] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," in *NeurIPS*, 2022.

[38] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," *SIGPLAN Not.*, vol. 53, no. 2, p. 461–475, mar 2018. [Online]. Available: https://doi.org/10.1145/3296957.3173176

[39] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[40] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "7.7 lnpu: A 25.3tflops/w sparse deep-neural-network learning processor with fine-grained mixed precision of fp8-fp16," in *2019 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2019, pp. 142–144.

[41] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, M. Xie, L. Wan, H. Liu, and C. Ding, "Ftrans: Energy-efficient acceleration of transformers using fpga," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 175–180. [Online]. Available: https://doi.org/10.1145/3370748.3406567

[42] Z. Li, S. Ghodrati, A. Yazdanbakhsh, H. Esmaeilzadeh, and M. Kang, "Accelerating attention through gradient-based learned runtime pruning," in *ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022*, V. Salapura, M. Zahran, F. Chong, and L. Tang, Eds. ACM, 2022, pp. 902–915.

[43] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for llm compression and acceleration," in *MLSys*, 2024.

[44] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019.

[45] Z. G. Liu, P. N. Whatmough, Y. Zhu, and M. Mattina, "S2TA: exploiting structured sparsity for energy-efficient mobile CNN acceleration," in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2022, Seoul, South Korea, April 2-6, 2022*. IEEE, 2022, pp. 573–586.

[46] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 977–991.

[47] M. Mahmoud, I. Edo, A. H. Zadeh, O. M. Awad, G. Pekhimenko, J. Albericio, and A. Moshovos, "Tensordash: Exploiting sparsity to accelerate deep neural network training," in *53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2020, Athens, Greece, October 17-21, 2020*. IEEE, 2020, pp. 781–795.

[48] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, and B. Bossan, "Peft: State-of-the-art parameter-efficient fine-tuning methods," https://github.com/huggingface/peft, 2022.

[49] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[50] OpenAI, "GPT-4 technical report," *CoRR*, vol. abs/2303.08774, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2303.08774

[51] L. Orr, "Manifest," https://github.com/HazyResearch/manifest, 2022.

[52] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. S. Emer, S. W. Keckler, and W. J. Dally, "SCNN: an accelerator for compressed-sparse convolutional neural networks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*. ACM, 2017, pp. 27–40. [Online]. Available: https://doi.org/10.1145/3079856.3080254

[53] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[54] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22-26, 2020*. IEEE, 2020, pp. 58–70.

[55] Y. Qin, Y. Wang, D. Deng, Z. Zhao, X. Yang, L. Liu, S. Wei, Y. Hu, and S. Yin, "Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589057

[56] J. Qiu, H. Ma, O. Levy, W. Yih, S. Wang, and J. Tang, "Blockwise self-attention for long document understanding," in *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, ser. Findings of ACL, T. Cohn, Y. He, and Y. Liu, Eds., vol. EMNLP 2020. Association for Computational Linguistics, 2020, pp. 2555–2565. [Online]. Available: https://doi.org/10.18653/v1/2020.findings-emnlp.232

[57] Z. Qu, L. Liu, F. Tu, Z. Chen, Y. Ding, and Y. Xie, "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*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 14–26.

[58] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2019.

[59] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[60] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, J. Su, X. Carreras, and K. Duh, Eds. The Association for Computational Linguistics, 2016, pp. 2383–2392. [Online]. Available: https://doi.org/10.18653/v1/d16-1264

[61] A. Roy, M. Saffar, A. Vaswani, and D. Grangier, "Efficient content-based sparse attention with routing transformers," *Trans. Assoc. Comput. Linguistics*, vol. 9, pp. 53–68, 2021.

[62] F. Sadi, J. Sweeney, T. M. Low, J. C. Hoe, L. T. Pileggi, and F. Franchetti, "Efficient spmv operation for large and highly sparse matrices using scalable multi-way merge parallelization," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019*. ACM, 2019, pp. 347–358.

[63] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter," *CoRR*, vol. abs/1910.01108, 2019.

[64] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: bit-level dynamically composable architecture for accelerating deep neural networks," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. IEEE Press, 2018, p. 764–775. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00069

[65] N. Shazeer, "Fast transformer decoding: One write-head is all you need," *CoRR*, vol. abs/1911.02150, 2019. [Online]. Available: http://arxiv.org/abs/1911.02150

[66] J. H. Shin, A. Shafiee, A. Pedram, H. Abdel-Aziz, L. Li, and J. Hassoun, "Griffin: Rethinking sparse optimization for deep learning architectures," in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2022, Seoul, South Korea, April 2-6, 2022*. IEEE, 2022, pp. 861–875.

[67] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2013, pp. 1631–1642. [Online]. Available: https://aclanthology.org/D13-1170/

[68] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: a compact task-agnostic BERT for resource-limited devices," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault, Eds. Association for Computational Linguistics, 2020, pp. 2158–2170. [Online]. Available: https://doi.org/10.18653/v1/2020.acl-main.195

[69] T. Tambe, C. Hooper, L. Pentecost, T. Jia, E. Yang, M. Donato, V. Sanh, P. N. Whatmough, A. M. Rush, D. Brooks, and G. Wei, "Edgebert: Sentence-level energy optimizations for latency-aware multi-task NLP inference," in *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021*. ACM, 2021, pp. 830–844.

[70] Y. Tay, D. Bahri, L. Yang, D. Metzler, and D. Juan, "Sparse sinkhorn attention," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 9438–9447.

[71] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," *CoRR*, vol. abs/2302.13971, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2302.13971

[72] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. Canton-Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," *CoRR*, vol. abs/2307.09288, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.09288

[73] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 5998–6008.

[74] A. Vyas, A. Katharopoulos, and F. Fleuret, "Fast transformers with clustered attention," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[75] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[76] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, "Hat: Hardware-aware transformers for efficient natural language processing," in *Annual Conference of the Association for Computational Linguistics*, 2020.

[77] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021, Seoul, South Korea, February 27 - March 3, 2021*. IEEE, 2021, pp. 97–110.

[78] S. Wang, L. Zhou, Z. Gan, Y. Chen, Y. Fang, S. Sun, Y. Cheng, and J. Liu, "Cluster-former: Clustering-based sparse transformer for long-range dependency encoding," *CoRR*, vol. abs/2009.06097, 2020.

[79] Y. Wang, Y. Qin, D. Deng, J. Wei, T. Chen, X. Lin, L. Liu, S. Wei, and S. Yin, "A 28nm 276.55tflops/w sparse deep-neural-network training processor with implicit redundancy speculation and batch normalization reformulation," in *2021 Symposium on VLSI Circuits*, 2021, pp. 1–2.

[80] Y. Wang, Y. Qin, D. Deng, J. Wei, Y. Zhou, Y. Fan, T. Chen, H. Sun, L. Liu, S. Wei, and S. Yin, "An energy-efficient transformer processor exploiting dynamic weak relevances in global attention," *IEEE Journal of Solid-State Circuits*, pp. 1–16, 2022.

[81] A. Warstadt, A. Singh, and S. R. Bowman, "Neural network acceptability judgments," *Trans. Assoc. Comput. Linguistics*, vol. 7, pp. 625–641, 2019. [Online]. Available: https://doi.org/10.1162/tacl_a_00290

[82] A. Williams, N. Nangia, and S. R. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, M. A. Walker, H. Ji, and A. Stent, Eds. Association for Computational Linguistics, 2018, pp. 1112–1122. [Online]. Available: https://doi.org/10.18653/v1/n18-1101

[83] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on*

Empirical Methods in Natural Language Processing: System Demonstrations. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45.

[84] B. Workshop, :, T. L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé, J. Tow, A. M. Rush, S. Biderman, A. Webson, P. S. Ammanamanchi, T. Wang, B. Sagot, N. Muennighoff, A. V. del Moral, O. Ruwase, R. Bawden, S. Bekman, A. McMillan-Major, I. Beltagy, H. Nguyen, L. Saulnier, S. Tan, P. O. Suarez, V. Sanh, H. Laurençon, Y. Jernite, J. Launay, M. Mitchell, C. Raffel, A. Gokaslan, A. Simhi, A. Soroa, A. F. Aji, A. Alfassy, A. Rogers, A. K. Nitzav, C. Xu, C. Mou, C. Emezue, C. Klamm, C. Leong, D. van Strien, D. I. Adelani, D. Radev, E. G. Ponferrada, E. Levkovizh, E. Kim, E. B. Natan, F. D. Toni, G. Dupont, G. Kruszewski, G. Pistilli, H. Elsahar, H. Benyamina, H. Tran, I. Yu, I. Abdulmumin, I. Johnson, I. Gonzalez-Dios, J. de la Rosa, J. Chim, J. Dodge, J. Zhu, J. Chang, J. Frohberg, J. Tobing, J. Bhattacharjee, K. Almubarak, K. Chen, K. Lo, L. V. Werra, L. Weber, L. Phan, L. B. allal, L. Tanguy, M. Dey, M. R. Muñoz, M. Masoud, M. Grandury, M. Šaško, M. Huang, M. Coavoux, M. Singh, M. T.-J. Jiang, M. C. Vu, M. A. Jauhar, M. Ghaleb, N. Subramani, N. Kassner, N. Khamis, O. Nguyen, O. Espejel, O. de Gibert, P. Villegas, P. Henderson, P. Colombo, P. Amuok, Q. Lhoest, R. Harliman, R. Bommasani, R. L. López, R. Ribeiro, S. Osei, S. Pyysalo, S. Nagel, S. Bose, S. H. Muhammad, S. Sharma, S. Longpre, S. Nikpoor, S. Silberberg, S. Pai, S. Zink, T. T. Torrent, T. Schick, T. Thrush, V. Danchev, V. Nikoulina, V. Laippala, V. Lepercq, V. Prabhu, Z. Alyafeai, Z. Talat, A. Raja, B. Heinzerling, C. Si, D. E. Taşar, E. Salesky, S. J. Mielke, W. Y. Lee, A. Sharma, A. Santilli, A. Chaffin, A. Stiegler, D. Datta, E. Szczechla, G. Chhablani, H. Wang, H. Pandey, H. Strobelt, J. A. Fries, J. Rozen, L. Gao, L. Sutawika, M. S. Bari, M. S. Al-shaibani, M. Manica, N. Nayak, R. Teehan, S. Albanie, S. Shen, S. Ben-David, S. H. Bach, T. Kim, T. Bers, T. Fevry, T. Neeraj, U. Thakker, V. Raunak, X. Tang, Z.-X. Yong, Z. Sun, S. Brody, Y. Uri, H. Tojarieh, A. Roberts, H. W. Chung, J. Tae, J. Phang, O. Press, C. Li, D. Narayanan, H. Bourfoune, J. Casper, J. Rasley, M. Ryabinin, M. Mishra, M. Zhang, M. Shoeybi, M. Peyrounette, N. Patry, N. Tazi, O. Sanseviero, P. von Platen, P. Cornette, P. F. Lavallée, R. Lacroix, S. Rajbhandari, S. Gandhi, S. Smith, S. Requena, S. Patil, T. Dettmers, A. Baruwa, A. Singh, A. Cheveleva, A.-L. Ligozat, A. Subramonian, A. Névéol, C. Lovering, D. Garrette, D. Tunuguntla, E. Reiter, E. Taktasheva, E. Voloshina, E. Bogdanov, G. I. Winata, H. Schoelkopf, J.-C. Kalo, J. Novikova, J. Z. Forde, J. Clive, J. Kasai, K. Kawamura, L. Hazan, M. Carpuat, M. Clinciu, N. Kim, N. Cheng, O. Serikov, O. Antverg, O. van der Wal, R. Zhang, R. Zhang, S. Gehrmann, S. Mirkin, S. Pais, T. Shavrina, T. Scialom, T. Yun, T. Limisiewicz, V. Rieser, V. Protasov, V. Mikhailov, Y. Pruksachatkun, Y. Belinkov, Z. Bamberger, Z. Kasner, A. Rueda, A. Pestana, A. Feizpour, A. Khan, A. Faranak, A. Santos, A. Hevia, A. Unldreaj, A. Aghagol, A. Abdollahi, A. Tammour, A. HajiHosseini, B. Behroozi, B. Ajibade, B. Saxena, C. M. Ferrandis, D. McDuff, D. Contractor, D. Lansky, D. David, D. Kiela, D. A. Nguyen, E. Tan, E. Baylor, E. Ozoani, F. Mirza, F. Ononiwu, H. Rezanejad, H. Jones, I. Bhattacharya, I. Solaiman, I. Sedenko, I. Nejadgholi, J. Passmore, J. Seltzer, J. B. Sanz, L. Dutra, M. Samagaio, M. Elbadri, M. Mieskes, M. Gerchick, M. Akinlolu, M. McKenna, M. Qiu, M. Ghauri, M. Burynok, N. Abrar, N. Rajani, N. Elkott, N. Fahmy, O. Samuel, R. An, R. Kromann, R. Hao, S. Alizadeh, S. Shubber, S. Wang, S. Roy, S. Viguier, T. Le, T. Oyebade, T. Le, Y. Yang, Z. Nguyen, A. R. Kashyap, A. Palasciano, A. Callahan, A. Shukla, A. Miranda-Escalada, A. Singh, B. Beilharz, B. Wang, C. Brito, C. Zhou, C. Jain, C. Xu, C. Fourrier, D. L. Periñán, D. Molano, D. Yu, E. Manjavacas, F. Barth, F. Fuhrimann, G. Altay, G. Bayrak, G. Burns, H. U. Vrabec, I. Bello, I. Dash, J. Kang, J. Giorgi, J. Golde, J. D. Posada, K. R. Sivaraman, L. Bulchandani, L. Liu, L. Shinzato, M. H. de Bykhovetz, M. Takeuchi, M. Pàmies, M. A. Castillo, M. Nezhurina, M. Sänger, M. Samwald, M. Cullan, M. Weinberg, M. D. Wolf, M. Mihaljcic, M. Liu, M. Freidank, M. Kang, N. Seelam, N. Dahlberg, N. M. Broad, N. Muellner, P. Fung, P. Haller, R. Chandrasekhar, R. Eisenberg, R. Martin, R. Canalli, R. Su, R. Su, S. Cahyawijaya, S. Garda, S. S. Deshmukh, S. Mishra, S. Kiblawi, S. Ott, S. Sang-aroonsiri, S. Kumar, S. Schweter, S. Bharati, T. Laud, T. Gigant, T. Kainuma, W. Kusa, Y. Labrak, Y. S. Bajaj, Y. Venkatraman, Y. Xu, Y. Xu, Y. Xu, Z. Tan, Z. Xie, Z. Ye, M. Bras, Y. Belkada, and T. Wolf, "BLOOM: A 176b-parameter open-access multilingual language model," *CoRR*, vol. abs/2211.05100, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2211.05100

[85] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: Accurate and efficient post-training quantization for large language models," in *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[86] T. Xie, Z. Cheng, Y. Xu, P. Shi, and T. Yu, "A framework for human-readable prompt-based method with large language models," https://github.com/hkunlp/humanprompt, 2022.

[87] J. Yang, Z. Zhang, Z. Liu, J. Zhou, L. Liu, S. Wei, and S. Yin, "Fusekna: Fused kernel convolution based accelerator for deep neural networks," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 894–907.

[88] A. Yazdanbakhsh, A. Moradifirouzabadi, Z. Li, and M. Kang, "Sparse attention acceleration with synergistic in-memory pruning and on-chip recomputation," in *55th IEEE/ACM International Symposium on Microarchitecture, MICRO 2022, Chicago, IL, USA, October 1-5, 2022*. IEEE, 2022, pp. 744–762.

[89] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "GOBO: quantizing attention-based NLP models for low latency and energy efficient inference," in *53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2020, Athens, Greece, October 17-21, 2020*. IEEE, 2020, pp. 811–824.

[90] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8BERT: quantized 8bit BERT," in *Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition, EMC2@NeurIPS 2019, Vancouver, Canada, December 13, 2019*. IEEE, 2019, pp. 36–39. [Online]. Available: https://doi.org/10.1109/EMC2-NIPS53020.2019.00016

[91] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontañón, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big bird: Transformers for longer sequences," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[92] A. Zeng, X. Liu, Z. Du, Z. Wang, H. Lai, M. Ding, Z. Yang, Y. Xu, W. Zheng, X. Xia, W. L. Tam, Z. Ma, Y. Xue, J. Zhai, W. Chen, Z. Liu, P. Zhang, Y. Dong, and J. Tang, "GLM-130B: an open bilingual pre-trained model," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/pdf?id=-Aw0rrrPUF

[93] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "Sparch: Efficient architecture for sparse matrix multiplication," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22-26, 2020*. IEEE, 2020, pp. 261–274.

[94] G. Zhao, J. Lin, Z. Zhang, X. Ren, Q. Su, and X. Sun, "Explicit sparse transformer: Concentrated attention through explicit selection," *CoRR*, vol. abs/1912.11637, 2019. [Online]. Available: http://arxiv.org/abs/1912.11637