Efficient Transformer Inference Through Hybrid Dynamic Pruning

Ghadeer A. Jaradat, Mohammed F. Tolba, Ghada Alsuhli, Hani Saleh, *Member, IEEE*, Mahmoud Al-Qutayri, *Member, IEEE*, Thanos Stouraitis, *Life Fellow, IEEE*

***Abstract*—In the world of deep learning, transformer models have become very significant, leading to improvements in many areas from understanding language to recognizing images, covering a wide range of applications. Despite their success, the deployment of these models in real-time applications, particularly on edge devices, poses significant challenges due to their computational intensity and memory demands. To overcome these challenges, we introduce a novel Hybrid Dynamic Pruning (HDP) technique, an efficient algorithm-architecture co-design approach that accelerates transformers using head sparsity, block sparsity and approximation to reduce computations in attention and reduce memory access. With the observation of the huge redundancy in attention scores and attention heads, we propose a novel integer-based block pruning to prune unimportant blocks in the attention matrix at run time. We also propose integer-based head pruning to detect and prune unimportant heads at an early stage at run time. Also, we propose an approximation method that reduces attention computations. To efficiently support these methods with lower latency, we propose the HDP Accelerator (HDPA) as a co-processor architecture, synthesized in two configurations— HDPA-edge and HDPA-server— to meet the needs of mobile and server platforms. Extensive experiments with different transformer models and benchmarks demonstrate that HDPA-server achieves $481\times$ and $381\times$ speedup in attention layer computation over Intel i7-1185G7 CPU and NVIDIA T4 GPU, respectively. Compared to other state-of-the-art accelerators, HDPA achieves $1.26\times$ to $2.08\times$ higher throughput, $1.3\times$ to $18\times$ greater MAC efficiency, and $1.1\times$ to $5.1\times$ improved energy efficiency, when normalized to the same computational load.**

***Impact Statement*—Transformer models have revolutionized natural language processing and computer vision, yet their high computational and memory demands restrict deployment on resource-limited edge devices. This work introduces a co-designed algorithm and hardware approach for attention layer, that addresses these challenges by significantly reducing computations and memory requirements. The proposed ASIC-based accelerator achieves up to 481× speedup over CPUs and 381× over GPUs, and delivers achieves $1.26\times$ to $2.08\times$ higher throughput, $1.3\times$ to $18\times$ greater MAC efficiency, and $1.1\times$ to $5.1\times$ improved energy efficiency over state-of-the-art accelerators with the same computational budget. This advancement enables efficient, real-time transformer applications in latency-sensitive areas such as autonomous systems, mobile devices, and healthcare, bringing high-performance deep learning capabilities to edge and low-power environments.**

***Index Terms*—Hardware acceleration, dynamic pruning, approximation, self attention, transformer.**

This work was supported by SoC Lab under Khalifa University Center for Cyber-Physical Systems (Fund code: 8474000137).

Ghadeer A. Jaradat, Mohammed F. Tolba, Ghada Alsuhli, Hani Saleh, Mahmoud Al-Qutayri, and Thanos Stouraitis are with SoC Center, Khalifa University, Abu Dhabi 127788, UAE (e-mail: 100062811@ku.ac.ae; mf1173@fayoum.edu.eg; ghada.alsuhli@ku.ac.ae; hani.saleh@ku.ac.ae; mahmoud.alqutayri@ku.ac.ae; thanos.stouraitis@ku.ac.ae).

I. Introduction

WITH their attention mechanism, transformer models, such as BERT [1], GPT [2], T5 [3], and others [4] [5], have transformed Natural Language Processing (NLP) cutting-edge performance in tasks like question-answering [6], text classification [3], and machine translation [7]. The transformer architecture uses the self-attention mechanism [8] and it is highly parallelizable on modern Graphical Processing Units (GPUs), providing major benefits over older models, like Long Short Term Memories (LSTMs) and Recurrent Neural Networks (RNNs). This advancement has accelerated progress in computer vision tasks such as object recognition and detection [9], image classification [10], and segmentation [11].

Deploying large transformer models on devices with limited resources is challenging, due to their high computation and memory requirements. For example, a BERT-Base transformer needs 440 MB of memory and over 176 Giga FLoating-point OPerations (GFLOPs) [12]. The computations are particularly intensive because of the attention operations whose computational complexity is quadratically related to the length of input sequences [13]. Attention operations in transformer models become increasingly dominant as the input sequence length grows. For a BERT-Base transformer deployed on ARM Cortex edge platforms, with a sequence length of 512 token, the attention operations account for about half of the total execution time, and this figure rises to 70% when the sequence length extends to 768 [14]. Therefore, finding efficient ways to handle attention operations is crucial for speeding up transformers.

Sparsity refers to reducing the number of non-zero elements in data or computations, enabling more efficient processing by focusing only on essential information. In the context of Deep Neural Networks (DNNs), including transformers, sparsity can significantly improve computational efficiency, memory usage, and power consumption, which is particularly beneficial when deploying models on specialized hardware accelerators. Many studies utilize sparsity to mitigate the quadratic time and complexity issue. Some techniques save computations by using fixed or static sparse attention patterns [15] [13] [16], but their performance is limited [17], since the sparsity pattern in attention is naturally dynamic, and depends only on the input. Other techniques focus on dynamic sparsity. For example, A^3 [18] uses various approximation methods to skip calculations of

near-zero values, aiming to decrease computational demands, but it requires loading all data onto the chip, which doesn't decrease off-chip DRAM access. SpAtten [19] introduces a cascaded token-pruning mechanism that gradually eliminates less important tokens to simplify the workload using a Top-K strategy. Despite being tailored for dynamic decision making in hardware, Top-K requires significant computational cost. Energon [14] uses a mixed-precision, multi-stage filtering technique to mimic the Top-K pruning approach, but it relies on a special unit to handle sparsity. AccelTran [20] prunes values in all transformer matrix multiplications, if they fall below a certain predetermined threshold. DOTA [21], identifies and removes weak connections between tokens using low-rank linear transformations to approximate attention scores. While this method effectively detects weak attention links, it leads to unstructured pruning, which presents significant challenges for hardware efficiency. Additionally, the coarse-grained block pruning method proposed in [22] for accelerating transformers employs an excessively coarse sparsity pattern, leading to a suboptimal balance between sparsity and accuracy. Even at moderate sparsity levels, this method struggles to preserve acceptable accuracy, often necessitating retraining to recover performance, which in turn increases computational overhead. A^3, Energon, AccelTran, and DOTA leverage unstructured sparsity to enable efficient deployment of transformers. However, unstructured sparsity incurs considerable overhead due to its dynamic nature, requiring specialized hardware controllers to manage it efficiently. These controllers often consist of a large number of clocking elements [23] and impose significant hardware overhead, necessitating complex interconnects, non-zero matching, and load-balancing mechanisms [24]. This makes them less practical for high-performance hardware execution. Furthermore, unstructured sparsity leads to non-uniform data access patterns, reducing computational efficiency. The unpredictable nature of sparsity patterns further complicates execution and introduces performance bottlenecks, ultimately slowing down processing speed.

Other research efforts have been directed towards the removal of attention heads in transformer models, based on the understanding that not all heads contribute significantly to the transformer's performance. Trainable gating parameters have been attached to each head and regularized in [25]. In other work, the importance of each attention head is gauged based on its sensitivity to the overall loss [26]. This sensitivity is utilized as an indirect measure to determine the significance of each head. Another work termed "single-shot meta-pruner" involves training a compact convolutional neural network with the specific purpose of identifying and selecting the attention heads that are crucial for preserving the distribution of attention within the model [27]. All of these studies perform pruning at compile time, not run time, and require retraining to recover the accuracy drop. SpAtten [19] performs a cascaded head pruning at run time using the Top-K approach, where the importance of each attention head is derived by aggregating the absolute values of its attention outputs of the head throughout the layers, resulting in an aggregate score of importance for each head. SpAtten uses a separate unit to perform head Top-K strategy, which also requires significant computational cost. A^3, Energon, and AccelTran do not support head pruning.

We propose integer-based Hybrid Dynamic Pruning (HDP), an algorithm architecture co-design to enable efficient attention inference. This method operates on multiple levels within the attention matrix, reducing complexity by concentrating on blocks and heads. This method incorporates head pruning to selectively eliminate less important heads, and employs fine-grained block pruning for removing smaller, less critical blocks, all based on the integer parts of inputs. The pruning is done dynamically; applied during inference without relying on fixed patterns for pruning, and without fine tuning or retraining. The main features of our method:

1) We propose integer-based fine-grained block pruning, which prunes unimportant, small-sized blocks, with the observation that self attention primarily relies on a few critical query-key pairs, highlighting significant redundancy in the self-attention mechanism. HDP utilizes the integer part of the input to identify and prune less important blocks, focusing subsequent operations only on the unpruned blocks. This technique reduces computations and memory access.
2) We introduce an early head-pruning strategy that identifies and eliminates less important heads, based on the integer parts of the input at the initial stages of computation, unlike methods that perform pruning after all computations. This method decreases computational demands and minimizes memory access.
3) We approximate attention calculation by breaking down the multiplication of Query (Q) and Key (K) matrices into three smaller ones operating on integer and fractional parts. By summing them together, we approximate the attention outcome and achieve near-zero pruning, by omitting multiplication of fractional parts. This method effectively reduces computational complexity, while maintaining model accuracy during inference.
4) We design and implement an ASIC-based architecture, HDP Accelerator (HDPA), to efficiently execute HDP algorithm. Our architecture functions as a co-processor, compatible with existing neural network accelerators. HDPA achieves higher throughput compared to other accelerators when normalized to the same computational budget.

The article is structured as follows: Section II provides background information on transformer acceleration. Section III details the methodology of the HDP framework. Section IV describes the hardware architecture of HDPA co-processor. Section V outlines the experimental setup, the baselines used for comparison, and discusses the results. Finally, in Section VI we offer conclusion.

II. Background and Motivation

A. *Transformer Algorithm*

Since their initial introduction in 2017 [8], transformers have shown state-of-the-art (SOTA) performance in NLP. They are often regarded as alternatives to classic CNNs and RNNs in various real-world situations due to their exceptional efficiency and generality. Transformers consist

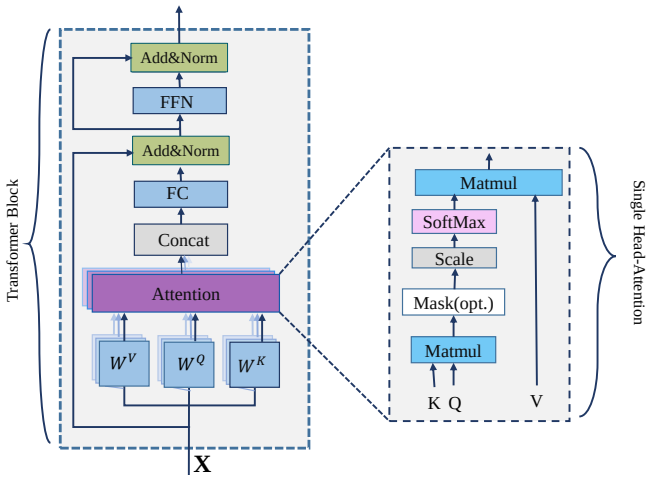


Fig. 1: Transformer block. The left side shows the transformer block structure, including multi-head attention, feed-forward network (FFN), and Add &Norm layers. The input feature X is projected using the projection weights (W^Q, W^K, W^V) into query (Q), key (K), and value (V) features. The right side zooms into the single-head attention, detailing the matrix multiplications, scaling, and Softmax operations.

Algorithm 1: Attention Mechanism [8]

Input: $\{K, Q, V\} \in \mathbb{R}^{l \times d}$
l: sequence length, d: input feature dimension
Number of heads: H;
$Q_h, K_h, V_h \leftarrow$ Split Q, K, V into H chunks;
$d_h = d/H$;
for $h_{id} \leftarrow 0$ **to** H **do**
  $S_a = Q_h {K_h}^T$;
  /* $S_a \in \mathbb{R}^{l \times l}$ */
  $S_a = \frac{S_a}{\sqrt{d_h}}$;
  for $i \leftarrow 0$ **to** l **do**
    $P_a = \text{softmax}(S_a[i])$;
  end
  $result[h_{id}] = P_a \cdot V_h$;
end
$O_a = \text{concat}(result[h_{id}])$;
Output: $O_a \in \mathbb{R}^{l \times d}$;

of two essential components: the encoder and the decoder, which are formed by layering many transformer blocks. As depicted in Fig. 1, a transformer block comprises three primary elements: linear layers, multi-head self-attention, and the normalization layer. The linear layers include the Fully-Connected layer (FC), Feed Forward Network (FFN), and the projection layer. For a block in a transformer model, the input X is a sequence of l tokens. The input is projected through the projection layer to $Query$ (Q), Key (K), and $Value$ (V) features using the weights W^Q, W^K, W^V respectively. Following this, attention mechanisms are utilized on these features to comprehend the long-term relational dependencies present in the input sequence.

The attention mechanism, as shown in Algorithm 1, splits the Q, K , V matrices into H smaller sets Q_h, K_h, V_h, where H is defined as the number of heads. Each of these sets forms an "attention head", see Fig. 1 right side. Inside each head, the output is computed as

$$\text{Attention}(Q_h, K_h, V_h) = \text{softmax}\left(\frac{Q_h {K_h}^T}{\sqrt{d_k}}\right) V_h.$$

The process starts with the computation of the dot product between Q_h and ${K_h}^T$, followed by scaling. This step computes the alignment or similarity between each pair of tokens, the fundamental components of a sequence. The resulting matrix represents each token's relationship with every other token. Then a row-wise softmax is applied to obtain the attention probability. When processing a particular token, the attention probability weights tell the model how much attention to give to each token in the sequence. After that, the attention probability is multiplied with the Value vector V_h, resulting in a weighted sum that forms the output of the single head attention layer. The resulting output is a combination of information from all tokens, with each token's importance and relevance determined and weighted by the attention mechanism. Each head independently calculates the attention result. The results are concatenated to get the end result. The concept entails that each head possesses the ability to concentrate on distinct segments of the input sequence or capture different types of relations within the data.

The attention mechanism fulfills various functions: it enables the model to focus on the most significant tokens of the input sequence, captures long-term relationships within the sequence, and greatly enhances the model's awareness of the context. This selective attention and context-aware processing are critical for complicated tasks, such as language translation [28], text summarization [29], and other NLP tasks.

In addition to the multi-head attention component, transformer models also employ other processes, such as normalization and FFN. The FFN in a transformer is composed of two FC layers. FFN's function can be expressed as

$$\text{FFN}(X) = \text{GELU}(XW_1 + b_1)W_2 + b_2,$$

where $X, W_1, W_2, b_1,$ and b_2 represent the input to the FFN, the weight matrices, and the bias vectors, respectively. The Gaussian Error Linear Unit (GELU), serves as a special activation function [30], widely used in transformers. Our focus in this work is the attention layer, as it is the bottleneck for transformer models.

B. *Dynamic sparsity in Multi-Head Self Attention*

Sparsity in attention is driven by the recognition that not all attention probabilities are significant. After applying a row-wise softmax function to get the attention probabilities, P_a in Algorithm 1, most likely a small subset of the attention scores, S_a, in each row will impact the output of the attention, O_a. Similarly, sparse multi-head attention

strategies are typically motivated by the recognition that not all heads in the attention mechanism have significant impact on the final output. Some heads in the attention model might have a minimal impact on the overall outcome. These observations point to the existence of redundant heads and insignificant P_a, indicating that the model's performance could be enhanced by ignoring the non crucial heads and removing unimportant S_a.

By examining the attention probability matrix (P_a) generated from different heads across various layers and inputs, we notice that the significance of individual heads for the multi-head attention mechanism depends not only on the input data but also on their position within the model's layered architecture. Fig. 2 presents the attention probabilities across different heads, layers, and inputs in a BERT-Base model fine-tuned on the SST-2 dataset [31]. This visualization provides insights into how attention distributions vary across layers and how they respond differently to different inputs. As shown in Fig. 2a, attention probabilities exhibit significant variation across layers for the same head. For instance, Head 11, highlighted by the red box, demonstrates noticeable fluctuations in attention values across layers 9, 10, and 11, suggesting that different layers process and refine contextual information differently, even within the same attention head. The attention distribution of the same head within the same layer also changes significantly depending on the input. Fig. 2 highlights this, where the blue boxes emphasize the contrast in attention probabilities for Heads 0 and 1 in Layer 11. Specifically, for Input 1, these heads exhibit low attention weights, indicating a weaker association with tokens in the sequence. However, for Input 2, the same heads display higher attention values, suggesting that the attention mechanism dynamically adjusts its focus based on input characteristics. This variation highlights how attention computations in transformers adapt dynamically based on input data. Additionally, most attention heads do not distribute focus uniformly across tokens. Instead, only a subset of attention probabilities exhibit high magnitudes, while others remain relatively low. This irregular distribution suggests that there is no fixed pattern or shape in how important attention values emerge, reinforcing the adaptiveness of transformer models in assigning relevance dynamically.

This variability highlights the complex, dynamic nature of how transformers process information. For a given input, certain heads in specific layers may become more active, focusing on particular aspects of the data. This activation can differ from one input to another, reflecting the adaptive response of each head to the unique characteristics of the data it encounters. Similarly, the significance of a head may differ across layers. Furthermore, the subset of high-magnitude attention probabilities is dynamic depending on the input sequences, and different heads have different subsets. This data-dependent behavior of attention heads is a critical consideration in model analysis and optimization

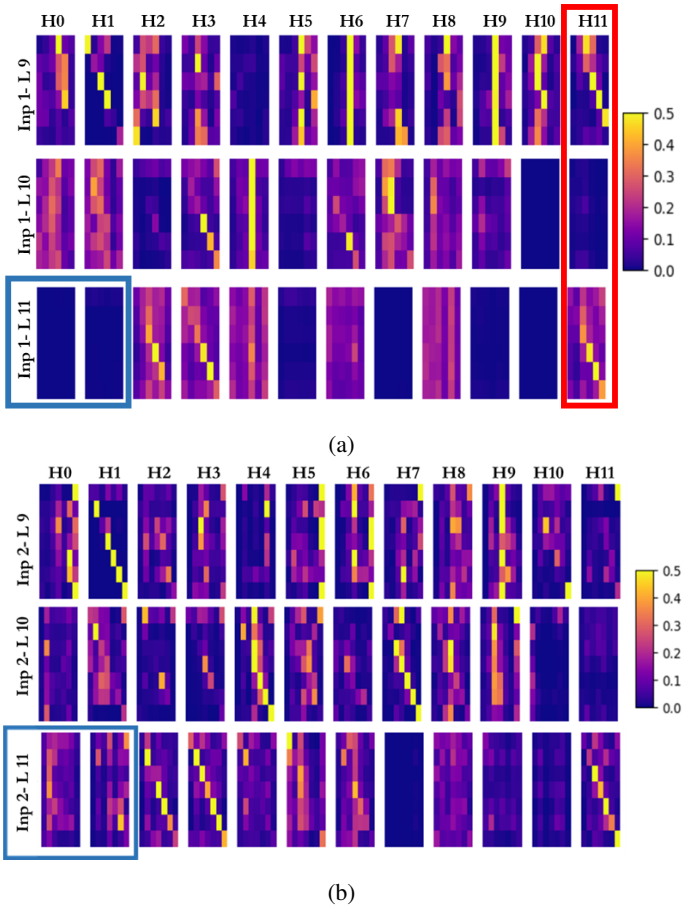


Fig. 2: Attention probability analysis in BERT-Base model: (a) Input 1, (b) Input 2. The red box highlights attention variability across layers for Head 11, while the blue boxes show differences in attention probabilities for the same heads and layers across different inputs.

and provides motivation for exploring effective methods to eliminate unimportant heads and Query-Key relations and thus save computations. Many methods have been used to prune such relations. The Top-K pruning method [19] is used to prune the attention weights, where a whole row can be pruned, but this requires retraining to recover the lost accuracy. Also, it requires specialized hardware to get the K most significant attention weights. Energon [14] avoided the Top-K selection and used mean filtering as a practical approximation instead, but it still has a separate unit to perform this operation, and faces data duplication overhead due to the multi-round filtering method employed. In Energon and AccelTran [20], pruning is done in an element-wise pattern, which results in an irregular sparse matrix, where zeros are randomly spread over the matrix, and this results in irregular memory accesses.

III. Algorithmic Optimization

In this section, we discuss algorithmic optimizations that enhance the transformer model's efficiency and performance,

Algorithm 2: Block, Head Pruning and Approximation (One Head)

0 **Input:** Quantized $\{K, Q, V\} \in \mathbb{R}^{l \times d}$
1 Block size: $c \times c$
2 Block Pruning Ratio: ρ_B
3 Head Pruning Threshold: τ_H
4 Integer part of Q: Q_I, Fractional part of Q: Q_F
5 Integer part of K: K_I, Fractional part of K: K_F
6 $S_I \leftarrow Q_I \cdot {K_I}^T$;
/* For the S_I matrix with $c \times c$ block size, iterate over the l/c rows of blocks */
7 **for** $i \leftarrow 0$ **to** l/c **do**
/* Process each $c \times c$ block within the row of blocks */
8 **for** $j \leftarrow 0$ **to** l/c **do**
9 $\theta_j \leftarrow \sum_{x \in block_j} |x|$;
10 $\theta_H \leftarrow \theta_H + \theta_j$;
11 **end**
12 $min_i \leftarrow \min(\theta_j) \quad \forall \quad \theta_j : \quad j = 0, ..., l/c$;
13 $max_i \leftarrow \max(\theta_j) \quad \forall \quad \theta_j : \quad j = 0, ..., l/c$;
14 $mean_i \leftarrow \sum_{0}^{l/c} \theta_j \Big/ (l/c) : \quad j = 0, ..., l/c$;
15

$$\Theta_i = \begin{cases} \text{if } 0 \le \rho_B < 1 : \\ \quad \rho_B \times max_i + (1 - \rho_B) \times mean_i \\ \text{if } -1 < \rho_B < 0 : \\ \quad -\rho_B \times min_i + (1 + \rho_B) \times mean_i \end{cases}$$

$Mask[i, j] \leftarrow (\theta_j < \Theta_i)?0 : 1; \quad j = 0, ..., l/c$;
/* S_I results for pruned blocks are not needed */
16 $S_I \leftarrow S_I \odot Mask$;
17 **end**
18 **if** $\theta_H > \tau_H$ **then**
19 **for** $i \leftarrow 0$ **to** l/c **do**
20 **for** $j \leftarrow 0$ **to** l/c **do**
21 **if** $Mask[i, j] == 1$ **then**
22 $S_{Fr1}[i, j] \leftarrow Q_{I_i} \cdot {K_F}_j^T$;
23 $S_{Fr2}[i, j] \leftarrow Q_{F_i} \cdot {K_I}_j^T$;
24 **end**
25 **end**
26 **end**
/* approximated attention score */
27 $approx \leftarrow S_I + S_{Fr1} + S_{Fr2}$;
28 $S_a \leftarrow approx/\sqrt{d}$;
29 $P_a \leftarrow softmax(S_a)$;
30 $result \leftarrow P_a \cdot V$;
31 **else**
/* Prune the whole head */
32 $result$ = 0;
33 **end**
34 **Output:** $result \in \mathbb{R}^{l \times d}$;

focusing on block pruning, head pruning, and approximation techniques. These optimizations are key to reducing computational complexity and memory access.

A. *Block Pruning*

Unlike the Top-K method [19], we propose integer-based block pruning, where the pruning decision is based on the integer parts of the inputs. We employ a small block size for pruning, to eliminate the necessity for retraining and to guarantee a more organized and hardware-amenable sparsity pattern.

Initially, multiplication is conducted only on the integer parts of Q and K to obtain an integer attention score (S_I) matrix. For each $c \times c$ block, we calculate its importance, θ, as the absolute sum of the values within the block, shown in Algorithm 2, line 9. For each row of blocks, we determine the pruning threshold, Θ, by applying mean filtering method used in Energon, which involves calculating the minimum, maximum, and mean of the importance values, along with a predefined pruning ratio, ρ_B, as shown in Algorithm 2, line 15. A mask is generated to record the status of each block. Blocks with importance θ falling below the row-specific threshold, Θ, are pruned and a mask value for the block is assigned to 0. When a block is pruned, subsequent computations for that block are omitted. Conversely, retained blocks (mask is 1), their final attention result is approximated, with the approximation technique detailed in section III-B.. The block pruning mechanism is detailed in Algorithm 2, specifically from lines 6 to 16, and is visually represented in Fig. 3. This figure illustrates that after computing S_I, the thresholds can be determined to identify the pruned blocks, generating a mask that will be utilized in subsequent processing stages.

B. *Approximation*

For blocks with mask value equal to 1 and θ exceeding Θ, we proceed to approximate the final attention outcome after obtaining the S_I. This involves calculating two products: the product of Q_I by ${K_F}^T$, and the product of Q_F by ${K_I}^T$, yielding S_{Fr1} and S_{Fr2}, respectively. The ultimate attention score is obtained by summing these two with the S_I. This method not only approximates the attention outcome, but also enables near-zero pruning, which has minimal impact on model accuracy during inference [32]. Specifically, when two numbers are close to zero, their integer parts are zero, forcing all three components to be zero. Consequently, the multiplications of fractional parts are omitted, resulting in effective near-zero pruning. The pruning pattern in the approximation stage remains consistent with the pattern used in block pruning, as pruned blocks do not proceed to the approximation phase. This process is outlined in Algorithm 2, specifically from lines 18 to 27, and is visually represented within the black box in Fig. 3. This figure illustrates that, based on the mask value, only unpruned blocks undergo approximation, where S_{Fr1} and S_{Fr2} are computed and

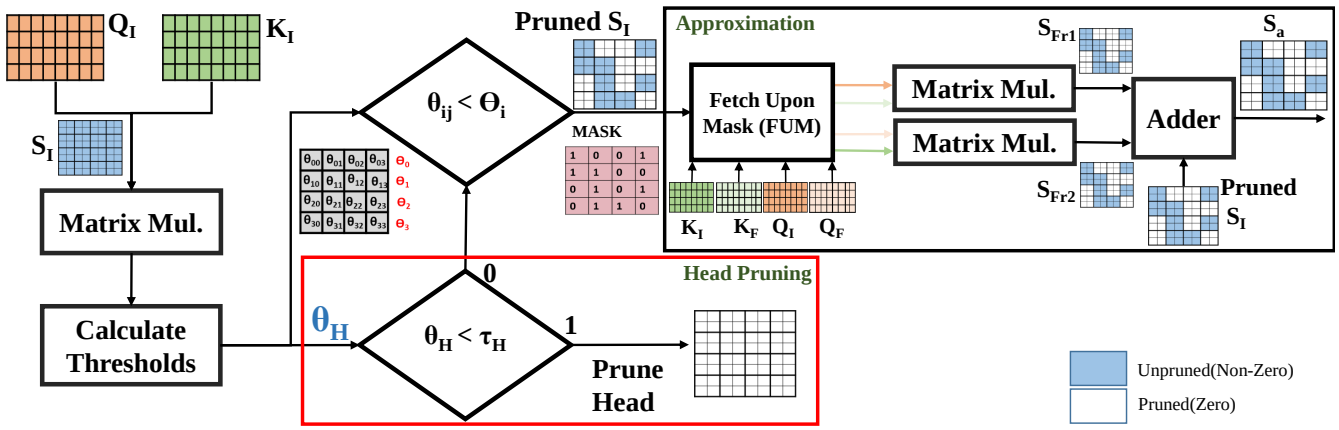


Fig. 3: Block pruning, head pruning, and approximation in HDP: Pruning is applied to $Q_I \times K_I$ based on block importance θ and row threshold Θ. Head pruning (red border) removes heads with θ_H below τ_H. The approximation stage (black border) refines pruned integer results by adding fractional components. The figure illustrates an example with Q of size 8×4, K of size 4×8, and a block size of 2×2.

subsequently summed with S_I to obtain the approximated attention score S_a.

C. *Head Pruning*

Not all heads are essential, and many can be pruned without affecting the overall performance. Unlike other methods [19], where head importance is assessed after completing all computations for the head, we introduce an early head pruning approach. To evaluate head importance, θ_H, we compute the absolute summation of all values in S_I. Heads with θ_H below a predefined threshold τ_H are completely pruned and the rest of computations of these head are skipped. The threshold τ_H is a parameter that will be profiled. The head pruning process is detailed in Algorithm 2 at line 32 and is visually highlighted by the red box in Fig. 3. The figure illustrates that if the value of θ_H falls below τ_H, the corresponding head is pruned. Otherwise, the head continues to the approximation stage for further processing.

IV. Hardware Architecture

This section introduces a customized HDP Accelerator (HDPA). This accelerator is developed to function as a co-processor and made to be compatible with a variety of neural network accelerators for easy integration.

A. *Architecture Overview*

The HDPA architecture depicted in Fig. 4, consists of multiple cores, with the architecture of each individual core shown in Fig. 4 (middle part). Each core is composed of an array of processing elements (PE Array), a Sparsity Engine (SE), an adder, and a softmax unit. The PE Array handles matrix multiplication tasks such as $Q \times K^T$ and $P_a \times V$. It also calculates importance values. The SE is tasked with identifying which blocks to prune and deciding whether a head should be pruned or not.

Workflow: Once Q, K, and V are generated and quantized by another processor in fixed-point, 16-bit format and stored in memory, HDPA processes each attention head sequentially. It employs tiled matrix multiplication for these operations. Initially, the integer components of Q and K are retrieved from off-chip memory into on-chip memory for the computation of $Q_I \times {K_I}^T$ using tiling. The SE then uses the computed importance values for each block to create a mask indicating which blocks are not pruned. This approach prevents unnecessary data fetching for pruned blocks, reducing memory access and computational overhead. Once $Q_I \times {K_I}^T$ computation is complete, and the head importance is assessed, the SE decides whether a head will be pruned. If so, the remaining computations for that head will be skipped and the next head will be assessed. For heads that remain unpruned, a Fetch Upon Mask (FUM) strategy is employed, where data fetching is determined by a mask deermined by the computed threshold values. If the mask value is 0 (indicating a pruned block), the corresponding K values will not be fetched, and the computation for that block is skipped. If the mask value is 1, the corresponding Q and K values are fetched, and the PE array calculates the two remaining partials ($Q_I \times {K_F}^T$ and $Q_F \times {K_I}^T$) simultaneously. These results, along with the integer results from the previous step, will be added together using the ADDER module to obtain the total S_a. After performing all $Q \times K$ calculations, a row-wise softmax will be applied to the S_a. Then, the PEs will be utilized to compute $S_a \times V$. Upon completion, the attention results for each tile will be immediately stored in DRAM memory. The host DNN accelerators can access these results to perform subsequent computations. In the sections that follow, we provide an in-depth exploration of each module and the proposed optimizations that enhance their functionality.

This article has been accepted for publication in IEEE Transactions on Artificial Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAI.2025.3563144

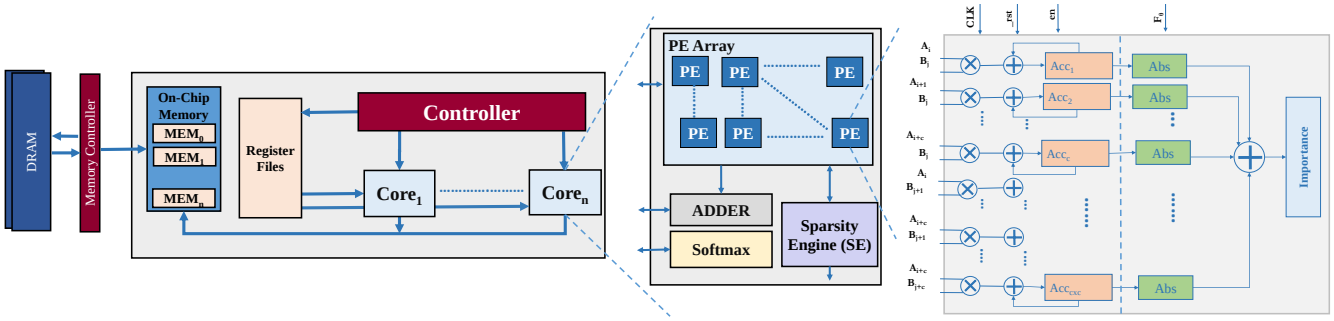


Fig. 4: HDPA architecture overview.

B. Matrix Multiplication

A significant portion of the computational workload in transformer models is attributed to matrix multiplication, necessitating optimization to boost accelerator performance. We implement tiled matrix multiplication to enhance the efficiency of these operations in our accelerator design. Tiling enhances resource efficiency and enables parallel computation, as depicted in Fig. 5. The first $H \times H$ tile from matrix A is multiplied by the first $H \times W$ tile from matrix B, with the partial results stored in an $H \times W$ tile in matrix C, denoted by (1) in all matrices. Subsequently, the process advances along (2) in the tiles of A and B to accumulate additional partial products in C. During this stage, an output-stationary-dataflow approach is employed, facilitating the reuse of partial product outputs in the accumulator. Additionally, a local A-stationary strategy is implemented, meaning that while outputs are reused in the outer loop, inputs from matrix A are retained and reused in the inner loop [33]. Next, the process proceeds along (3) in all matrices, followed by a moving along (4).

C. Processing Elements

The PE, shown in Fig. 4 (right), serves as a fundamental computational unit within the accelerator, handling all matrix multiplications. Operating in an output-stationary systolic computations; it receives rows from tiles of the first matrix and columns from tiles of the second matrix as inputs, one input at a time. It multiplies these values and stores the intermediate products in accumulators until an entire row of the first matrix has been multiplied by the corresponding column of the second matrix. At this point, the accumulators hold the final results for a tile of the product matrix. Each PE contains $c \times c$ multipliers, matching the size of the block in Algorithm 2. In the case of $Q_I \times K_I$ multiplication, and upon receiving F_θ signal from the controller, which initiates the calculation of the block's importance, the results in the accumulators are used to evaluate the importance.

D. Sparsity Engine

The Sparsity Engine is responsible for determining the sparsity pattern at both block and head levels. Illustrated in Fig. 6, the Sparsity Engine's internal architecture takes in importance scores from the PE and stores them in its internal memory. Additionally, it keeps track of the minimum, maximum, and total sum of these importance values for every row of blocks. The completion of a full row in the result matrix or equivalently, the multiplication of a row from the first matrix by all columns in the second matrix in $Q_I \times {K_I}^T$ multiplication, is indicated by the signal END_R. Then, the engine calculates the block pruning threshold Θ for that specific row using the equation provided in line 15 of Algorithm 2. Additionally, the engine generates the $Mask$ for this row by subtracting Θ from the importance values of the blocks. If the result is negative, the block falls below Θ and is marked for pruning.

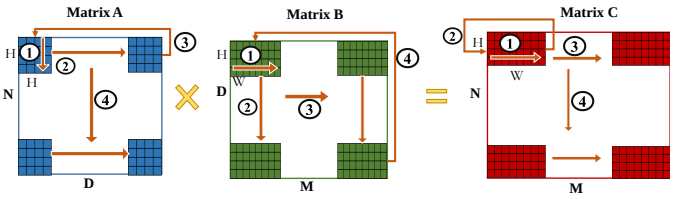
Furthermore, when the signal END_H is received, signaling the completion of the $Q_I \times {K_I}^T$ multiplication, the engine utilizes the computed θ_H value, which represents the total sum of all importance values across the entire head. Upon receiving this flag, the engine compares it with τ_H, an input parameter denoting the head pruning threshold. If θ_H falls below τ_H, the head is assumed redundant and is thus excluded, allowing the accelerator to bypass any further calculations for this head and proceed to processing the subsequent head.

E. Softmax Module

Once the attention scores are obtained, the row-wise softmax function $e^{s_i} / \sum_j e^{s_j}$ is applied. For every input received by the module, the exponent is approximated using a piecewise second-order polynomial. The exponents are stored in internal memory, and their sum is calculated. By the end of every row, the reciprocal of the sum is computed using a linear approximation. Then the exponent values are multiplied by this reciprocal to generate the softmax.

F. Data Layout and Fetching

The proposed hardware architecture features four SRAM banks, each designated for storing the integer and fraction values of Q and K. Given a sequence length of up to 1024 tokens and an input feature dimension d= 64 for each head, a single row in the SRAM bank accommodates 1024 bits, representing 8-bit features from 2 tokens. Once




```
for (int i=0; i < N; i=i+H) //Tiling along N dimension
 for(int j=0; j < M; j=j+W) //Tiling along M dimension
  for(int k=0; k < D; k=k+H) //Output Stationary
   for (int ii=0; ii < i+H; ii++) //A input stationary
    for(int jj=0; jj < j+W; jj++)  //B input reuse
     for(int kk=0; kk < k+H; kk++) //Partial sum reuse
      C[ii][jj] += A[ii][kk]*B[kk][jj] //MAC_OPERATION
```

Fig. 5: Matrix multiplication tiling with the dataflow.

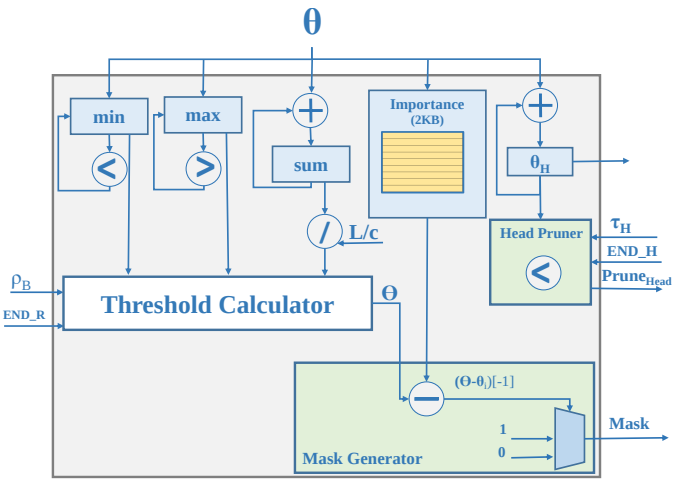


Fig. 6: Internal architecture of Sparsity Engine.

the Q_I and K_I matrices are fetched from DRAM into the on-chip memory, one or more rows from the Q_I SRAM bank and multiple columns from the K_I SRAM bank are loaded into the register files. The processing cores then initiate parallel multiplications of tiles from Q_I rows and K_I columns. During this computation, the register file for K_I is continuously updated with subsequent columns of K_I, while the values in the Q_I register file are reused for all multiplications. Once all K_I columns have been multiplied by the rows in the Q_I register file, the results are stored in a fifth SRAM bank with dimensions 1024×1024. Simultaneously, the SE unit continuously monitors each row of blocks, tracking the minimum, maximum, and sum values to compute the threshold value, Θ, and the head importance, θ_H. Also, a corresponding mask value is generated for each row of blocks and stored in an on-chip mask memory, with a size of $1024/c \times 1024/c$ bits, where c is the block width.

If θ_H falls below τ_H, then the subsequent computations for this head are skipped. Then, the corresponding Q_I and K_I values for the next head are promptly loaded into the on-chip memory. Conversely, if the head is not pruned, then the FUM strategy will selectively load only the required Q_F rows and K_F columns, based on the mask values, to perform the remaining attention computations. For the V matrix, it will be loaded into the same SRAM banks as Q. Since our design is not pipelined, the SRAM banks for Q and K can be reused to store V.

V. Evaluation

A. Algorithm Evaluation

1) Evaluation Models and Datasets: To validate the efficacy of our proposed HDP method, we focused on implementing encoder-only models. We utilized BERT-Tiny [34] and BERT-Base [1], two well-established pre-trained models. BERT-Tiny consists of two encoder layers, each with a hidden dimension of 128 and two attention heads, while BERT-Base contains 12 encoder layers, each with a hidden dimension of 768 and 12 attention heads. Our evaluation was conducted on two benchmark tasks: SST-2 and COLA, both sourced from the GLUE benchmark [35].

2) Dynamic Pruning with Transformer: We present experimental results, encompassing various aspects of our study. These results provide a comprehensive overview of the performance and efficacy of the proposed HDP algorithm, from exploring various block pruning ratios and profiling head thresholds to conducting thorough comparisons with the established Top-K block pruning method.

(a) Block Pruning: Our baseline for comparison is the Top-K block pruning method with a block width $c = 2$, as this was the largest block size achievable without sacrificing accuracy or requiring retraining. The selection of this block size is based on Fig. 7, where we evaluated various block sizes across different pruning ratios for BERT-base model on the CoLA dataset. As observed in Fig. 7, the highest accuracy was achieved at $c = 1$. However, this results in unstructured pruning, meaning no coherent block structure is maintained. In contrast, $c = 2$ provided the optimal balance, achieving the highest pruning ratio with minimal accuracy degradation while preserving structured sparsity. As the block size increases beyond $c = 2$, the accuracy drop becomes more pronounced, eventually requiring retraining to recover lost performance.

The comparison with Top-K block pruning is depicted in Fig. 8, the Top-K method can prune up to 75% of all blocks with 1% accuracy loss, whereas HDP can achieve a pruning ratio of 70%. However, for pruning ratios exceeding 80%, the accuracy of HDP is significantly higher than that of Top-K, indicating that the HDP method is unable to accurately determine the correct block pruning threshold, Θ. This issue arises because the model incorrectly assumes that mean divides the data into equal halves, and therefore, it assumes pruning a high percentage of blocks, when in fact it is not.

Both models exhibit an initial improvement in accuracy, followed by a decline as the level of sparsity increases. This improvement is linked to the over-parameterization of the BERT model [36]. In NLP, this effect is similar

This article has been accepted for publication in IEEE Transactions on Artificial Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAI.2025.3563144

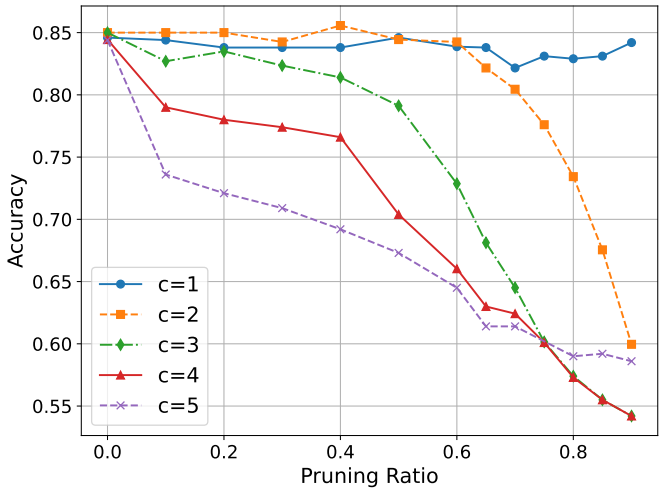


Fig. 7: Accuracy vs. Pruning Ratio for different block sizes (c values).

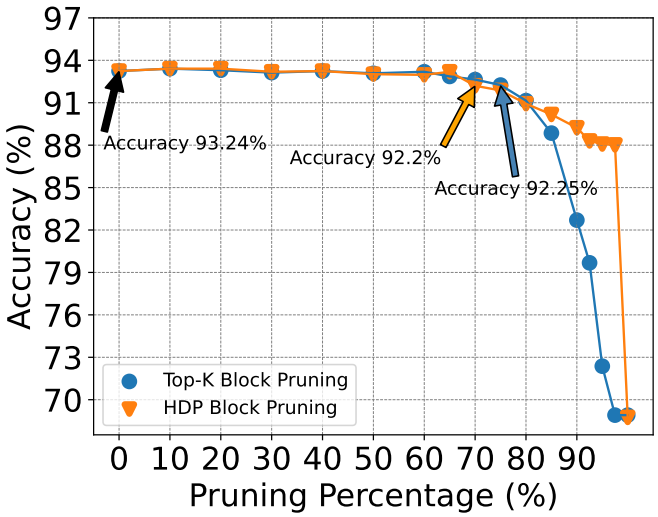
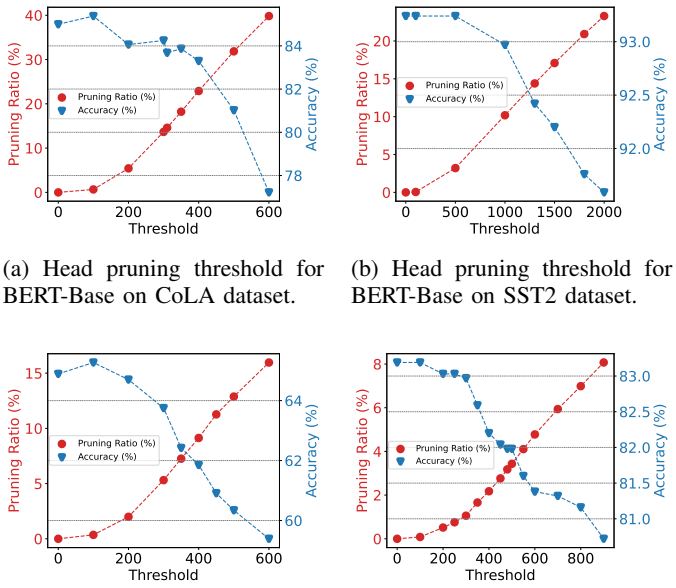


Fig. 8: Top-K vs. HDP block pruning.

to removing unimportant tokens allowing the model to focus on important tokens and thereby enhancing accuracy.

(b) Head Pruning: In HDP, the impact of head pruning is depicted in Fig. 9, which demonstrates the threshold profiling and the corresponding accuracies after applying head pruning to the BERT-Base and BERT-Tiny models on the SST2 and CoLA datasets. As anticipated, BERT-Tiny is sensitive to head pruning, as illustrated in Fig. 9c and Fig. 9d. These figures reveal that less than 2% of the model's heads can be pruned without affecting the accuracy. This sensitivity is due to the model's limited number of heads (only 4 in total). Removing a single head amounts to the significant coarse-grained pruning of one-fourth of all heads. This is a large coarse-grained pruning to be done without retraining. Head pruning in the BERT-Base model yields more favorable pruning ratios due to its larger number of heads, totaling 144. As illustrated in Fig. 9a and Fig. 9b, the model can prune approximately $13 - 17\%$ of the heads with only a 1% decrease in accuracy.



(a) Head pruning threshold for BERT-Base on CoLA dataset.

(b) Head pruning threshold for BERT-Base on SST2 dataset.

(c) Head pruning threshold for BERT-Tiny on CoLA dataset.

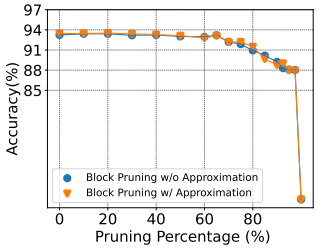
(d) Head pruning threshold for BERT-Tiny on SST2 dataset.

Fig. 9: Head pruning threshold for BERT-Base and BERT-Tiny on SST2 and CoLA.

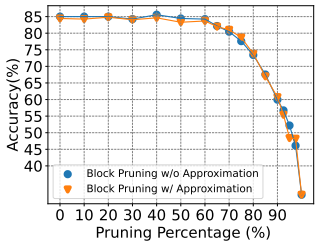
(c) Approximation: To assess the effectiveness of the proposed approximation method, we examine its impact on the models' accuracy. Fig. 10 illustrates the accuracy of models employing block pruning with and without the approximation technique. For the BERT-Base model, as depicted in Fig. 10a and Fig. 10b, the model's performance remains almost the same, suggesting that the approximation does not negatively affect the model, while providing higher computational efficiency. In contrast, as shown in Fig. 10c and Fig. 10d, the BERT-Tiny model is more sensitive to the approximation, experiencing a greater effect on its performance. While both BERT-Base and BERT-Tiny have an identical hidden size of 64 for each head, the reduced number of heads in BERT-Tiny amplifies the impact of pruning within a head on its overall performance. The near-zero pruning strategy, allocates higher softmax values to unpruned elements, and allows the model to focus more on crucial $Q - K$ relations, thereby concentrating on important components.

(d) Net Pruning: Fig. 11 demonstrates the combined effect of block pruning, head pruning, and approximation techniques on the model's overall accuracy. With a 1% reduction in accuracy, the net sparsity achieved by BERT-Base on the SST2 dataset is 75%, matching the pruning percentage attained through the Top-K method, but with significantly less computational overhead. For BERT-Base on CoLA dataset, the net pruning is 65%.

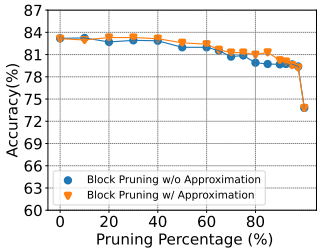
In the Top-K block pruning approach, even within an unimportant head, certain blocks remain unpruned due to



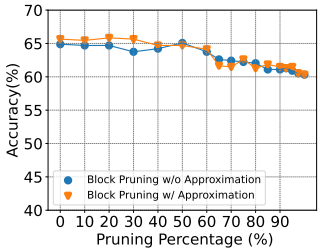
(a) Accuracy of BERT-Base on SST2.



(b) Accuracy of BERT-Base on CoLA.



(c) Accuracy of BERT-Tiny on SST2.



(d) Accuracy of BERT-Tiny on CoLA.

Fig. 10: Comparison of accuracy for BERT-Base and BERT-Tiny block pruning on SST2 and CoLA, with and without approximation.

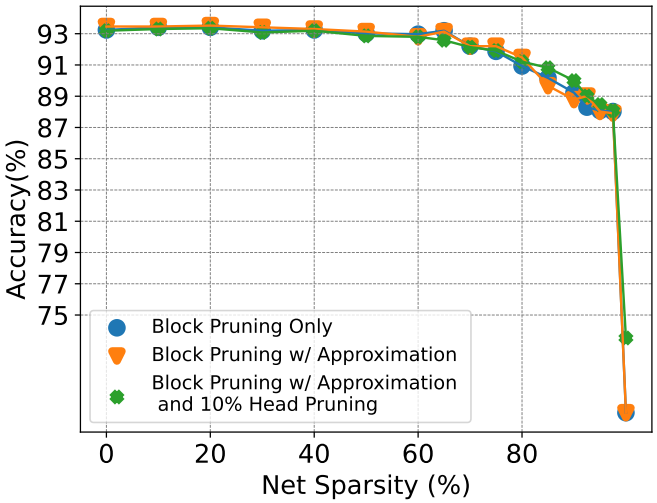
their relative significance within that specific head. However, the entire head may not be crucial. By implementing head pruning, we successfully remove such unimportant heads, thereby achieving a higher overall pruning ratio.

B. Architecture Evaluation

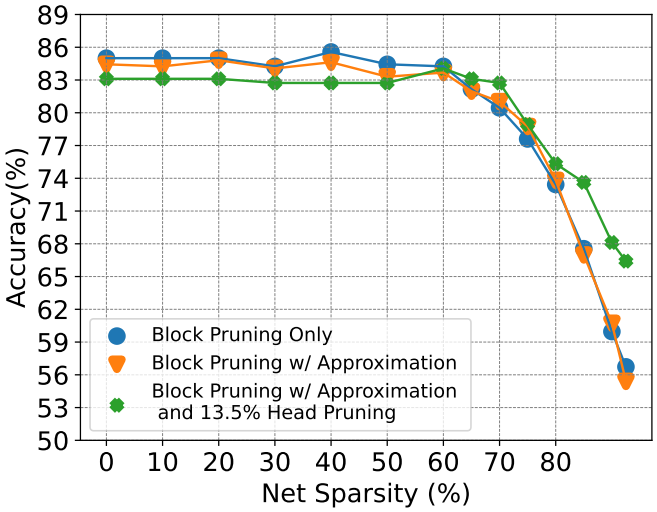
1) Evaluation Methodology: We implemented HDPA architecture at Register-Transfer Lever (RTL) using Verilog. We synthesized it using Synopsys Design Compiler [37] with the GF FD-SOI 22 nm standard cell library. On-chip SRAMs were compiled using Synopsys Memory Compilers, and RTL simulation was performed with Modelsim.

Two configurations of HDPA, HDPA-edge and HDPA-server, were synthesized to meet the requirements of mobile and server platforms. Hardware and architecture parameters are detailed in Table I. Both configurations operate at a clock frequency of 1 GHz and a supply voltage of $0.72V$. We used the NVIDIA T4 GPU and Intel CPU (i7-1185G7 @1.80 GHz) as baselines to evaluate HDPA performance. Attention operations were run on the CPU and GPU using PyTorch, with MKL and cuDNN, respectively. GPU latency was measured using `torch.cuda.Events`, and CPU latency was recorded with `time.time`. Power consumption was monitored using `nvidia-smi` for the GPU and HWiNFO64 [38] for the CPU. For latency measurements, the input sequence length was set to the average of the development set. We conducted 1000 iterations, discarding the highest and lowest 15% of values to remove outliers, and averaged the remaining results for accurate assessment.

2) Experimental Results:



(a) BERT-Base on SST2.



(b) BERT-Base on CoLA.

Fig. 11: Comparison of accuracy vs. net pruning ratio for BERT-Base using block pruning, head pruning and with approximation on SST2 and CoLA datasets.

(a) Power and Area Evaluation: Table II presents the area and power consumption of the HDPA architectures, while Fig.12 provides a detailed breakdown of the area and power consumption for the computation logic in HDPA-edge. Notably, the sparsity engine accounts for only 3.1% of the computation unit's area and 0.6% of its power, equating to just 0.079% and 0.002% of the total architecture. This demonstrates the efficiency of the proposed pruning method, with minimal overhead confirming the effectiveness of the pruning approach, which achieves significant computational savings with negligible impact on resources. Additionally, the use of the mask substantially reduces switching power, thereby lowering dynamic power consumption. According to synthesis results, a 0.75 pruning ratio leads to a 38.04% reduction in switching power due to the mask's impact. We further evaluated the energy consumption of the

This article has been accepted for publication in IEEE Transactions on Artificial Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAI.2025.3563144

TABLE I: Hardware and Architecture Parameters

Accelerator	Module	Configuration
HDPA-Edge	Cores	1
	PEs/Core	8
	Softmax	4
	Memory	128KB query/ key SRAM, 2MB output SRAM, 32KB Mask SRAM, 5.53KB Register files
HDPA-Server	Cores	4
	PEs/Core	8
	Softmax	16
	Memory	128KB query SRAM 128KB key SRAM per core, 2MB output SRAM, 32KB Mask SRAM, 5.53KB Register files per core

HDPA architectures by calculating $latency \times power$. The server version of HDPA is formed by integrating multiple HDPA-edge cores, resulting in a throughput increase approaching $4\times$ under ideal workload distribution and parallel execution.

Fig. 13a shows the energy savings of both architectures compared to a CPU at 10nm technology node and a GPU at 12nm technology node. Energy savings is defined as the percentage reduction in total energy consumption relative to the baseline system. The results indicate that the HDPA-edge and HDPA-server architectures achieve significant energy reductions over these baseline systems. These energy savings are primarily due to efficient fetch reduction strategies, which minimize unnecessary data retrieval by $1.13\times$, and head pruning techniques that reduce memory access by $1.1\times$. Combined, these strategies result in a total $1.25\times$ reduction in memory access, contributing to overall energy efficiency.

(b) Latency Evaluation: Fig. 13b shows the processing speedup for attention layers compared to baseline systems. Speedup is defined as the ratio of execution time between the baseline system and the proposed architecture. The HDPA architectures exhibit significantly reduced latency compared to the baseline systems, primarily due to the approximation method, which reduces the computation in the attention layer by $1.15\times$. Additionally, the FUM strategy contributes an additional $1.1\times$ speedup.

C. Comparisons with Existing Accelerators

We compare our approach to SOTA sparse attention accelerators, including A^3 [18], SpAtten [19], Energon [14], DOTA [21], Acceltran [20] and acceleration using block pruning in [22]. In line with the settings from [19] [39] [40], and to ensure a standardized and fair comparison, we follow their adopted methodology for evaluating accelerator designs, which assumes a 1 GHz clock frequency and 128 multipliers across all architectures. AccelTran operates at 700 MHz; therefore, to match the computational capacity of the other models, it is

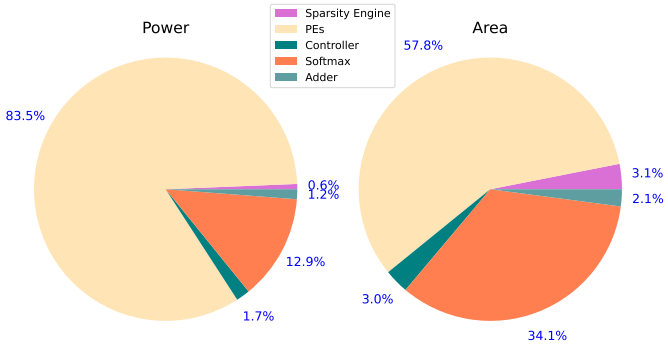
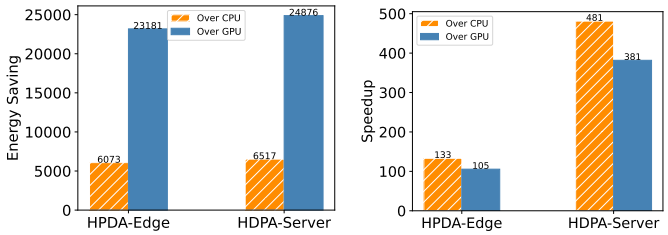


Fig. 12: Area and power breakdown for HDPA-edge computation logic.



(a) Energy Saving. (b) Speedup.

Fig. 13: Energy savings and speedup of HDPA architectures over baselines on attention layers.

configured with 183 multipliers ($183 \times 700M = 128$ GOP/s). This adjustment ensures that its theoretical peak performance remains equivalent to that of a design operating at 1 GHz. Table III shows the comparison with the SOTA accelerators. Throughput, measured in GOP/s (Giga Operations Per Second), represents the total number of Multiply-Accumulate (MAC) operations executed per second. We also evaluate MAC efficiency (GOP/s per MAC unit), which measures throughput relative to the number of MAC units. Furthermore, we assess energy efficiency (GOP/J), which measures the number of operations performed per unit of energy consumed. In comparison, HDPA achieves a $2.08\times$ higher throughput than A^3 and demonstrates $2.1\times$ higher MAC efficiency and $1.5\times$ improved energy efficiency. Additionally, A^3 requires loading all data from DRAM to execute its approximation and incurs a preprocessing overhead for sorting keys. In contrast, our proposed head pruning technique reduces DRAM access by approximately $1.1\times$, thereby enhancing memory efficiency and overall performance. SpAtten employs a Top-K approach to progressively eliminate less important tokens and heads. However, this method results in a significant accuracy drop, requiring model retraining to restore performance. Moreover, the Top-K unit incurs a substantial overhead, consuming approximately 0.08W at the 40nm technology node. When scaled down to the 22nm node [41] and using a nominal voltage of 1.1V for TSMC's 40nm technology node [42], this overhead reduces to 0.034W. In contrast, HDPA's sparsity engine introduces a minimal overhead of just 0.28mW. Additionally, HDPA achieves $1.28\times$ higher throughput, $2.1\times$ greater MAC efficiency, and $1.1\times$ improved energy efficiency, further

TABLE II: Area and Power of HDPA Architectures

	Area(mm^2)			Power(mW)		
	computation Logic	SRAM	Total	computation Logic	SRAM	Total
HDPA-Edge	0.091	3.46	3.55	47.03	91.27	138.3
HDPA-Server	0.355	4.401	4.756	165	300	465

TABLE III: Comparison of HDPA with related works along different aspects. (Results in Brackets Scaled to $22nm$)

	A^3 [18][a]	SpAtten [19][a]	Energon [14][b]	DOTA [21]	AccelTran [20][c]	ISQED'21 [22]	**HDPA (Ours)**
Technology	ASIC(40nm)	ASIC(40nm)	ASIC(45nm)	ASIC(22nm)	ASIC(14nm)	FPGA(16nm)	ASIC(22nm)
Sparse Matrix	unstructured	structured	unstructured	unstructured	unstructured	structured	structured
Head Pruning		✓					✓
Block Pruning						✓	✓
Approximation	✓						✓
Tiled Mat. Mul.					✓		✓
# of Multipliers	128				183	$\sim$ 3368	128
Frequency	1GHz				700 MHz	-	1GHz
Throughput (GOP/s)	221	360	612	128	364	-	460
MAC Effciency (GOP/s/MAC)	1.7	2.8	4.8	1	2.5	0.20	3.6
Energy Effciency (GOP/j)	269 (631)	382 (891)	233 (450)	305	158 (194)	-	991

[a] Scaling made with $1.1V$.
[b] Scaling made with $1V$.
[c] Scaling made with $0.8V$.

highlighting its performance advantages. AccelTran is a highly extensive design, with its impressive throughput primarily attributed to the large number of multipliers it utilizes. However, it struggles to achieve a high pruning ratio. AccelTran delivers $12.6\times$ higher throughput than SpAtten with 262,144 multipliers. However, when scaled down to 183 multipliers, this throughput drops significantly, as $262,144/183 = 1,432$, leading to a much lower normalized throughput of $0.009\times$. On the other hand, HDPA achieves $1.26\times$ higher throughput along with $1.44\times$ higher MAC efficiency and $5.1\times$ better energy efficiency, further highlighting its improved performance. HDPA achieves a $3.6\times$ increase in throughput over DOTA, along with a $3.6\times$ improvement in MAC efficiency and a $3.2\times$ enhancement in energy efficiency. The block pruning method proposed in [22] does not specify the operating frequency, which prevents us from normalizing the number of MAC units for a direct comparison. However, it achieves a MAC efficiency of 0.2, underperforming HDPA by $18\times$. Additionally, due to the coarse granularity of the block pruning approach, it necessitates retraining to recover the accuracy loss, as the large block size introduces significant accuracy degradation when applied without finetuning. Energon approximates the Top-k method through mean filtering, utilizing a mixed-precision, multi-round filtering unit to identify important query-key pairs. This approach introduces significant overhead, requiring data duplication and a dedicated unit that consumes approximately 30% of the total power and area in Energon-edge. HDPA, achieving 991 GOP/J, delivers $2.2\times$ higher energy efficiency compared to Energon, which operates at 450 GOP/J [40]. When comparing power consumption between HDPA-edge and Energon-edge, normalized to a 22nm technology node with a nominal voltage of 1V [43], HDPA-edge achieves a 21% reduction in power consumption.

VI. Conclusion

We presented HDP, a novel algorithm architecture co-design to efficiently run dynamic, sparse attention models. We first proposed a novel integer-based block pruning technique to prune unimportant blocks in the attention matrix and integer-based head pruning to prune unimportant heads. We also proposed an approximation method that reduces the computations and performs near-zero pruning. We implemented this method in two co-processor architectures, HDPA-edge and HDPA-server, to accelerate algorithm on mobile and sever platforms. HDPA-server achieves speedups of $481\times$ and $381\times$ over CPU and GPU, respectively, our design achieves $1.26\times$ to $2.08\times$ higher throughput , along with $1.3\times$ to $18\times$ higher MAC efficiency and $1.1\times$ to $5.1\times$ greater energy efficiency compared to SOTA accelerators, when normalized to the same computational load. We also provided a performance analysis breakdown for each technique.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
[2] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.
[3] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[5] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

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

[7] Alexis Conneau and Guillaume Lample. Cross-lingual language model pretraining. *Advances in neural information processing systems*, 32, 2019.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[9] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[11] Jianyuan Guo, Kai Han, Han Wu, Yehui Tang, Xinghao Chen, Yunhe Wang, and Chang Xu. CMT: Convolutional neural networks meet vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12175–12185, 2022.

[12] Chao Fang, Aojun Zhou, and Zhongfeng Wang. An algorithm–hardware co-optimized framework for accelerating n: M sparse transformers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(11):1573–1586, 2022.

[13] Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen-tau Yih, Sinong Wang, and Jie Tang. Blockwise self-attention for long document understanding. *arXiv preprint arXiv:1911.02972*, 2019.

[14] Zhe Zhou, Junlin Liu, Zhenyu Gu, and Guangyu Sun. Energon: Toward efficient acceleration of transformers using dynamic sparse attention. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(1):136–149, 2022.

[15] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

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

[17] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6), dec 2022.

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

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

[20] Shikhar Tuli and Niraj K Jha. Acceltran: A sparsity-aware accelerator for dynamic inference with transformers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[21] Zheng Qu. 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*. 2022.

[22] Hongwu Peng, Shaoyi Huang, Tong Geng, Ang Li, Weiwen Jiang, Hang Liu, Shusen Wang, and Caiwen Ding. Accelerating transformer-based deep learning models on fpgas using column balanced block pruning. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pages 142–148. IEEE, 2021.

[23] Jun-Woo Jang, Sehwan Lee, Dongyoung Kim, Hyunsun Park, Ali Shafiee Ardestani, Yeongjae Choi, Channoh Kim, Yoojin Kim, Hyeongseok Yu, Hamzah Abdel-Aziz, et al. Sparsity-aware and re-configurable npu architecture for samsung flagship mobile soc. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 15–28. IEEE, 2021.

[24] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 58–70. IEEE, 2020.

[25] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

[26] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? may 2019.

[27] Zhengyan Zhang, Fanchao Qi, Zhiyuan Liu, Qun Liu, and Maosong Sun. Know what you don't need: Single-shot meta-pruning for attention heads. *AI Open*, 2:36–42, 2021.

[28] Yachao Li, Junhui Li, Jing Jiang, Shimin Tao, Hao Yang, and Min Zhang. P-transformer: Towards better document-to-document neural machine translation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 31:3859–3870, 2023.

[29] Tinghuai Ma, Qian Pan, Huan Rong, Yurong Qian, Yuan Tian, and Najla Al-Nabhan. T-bertsum: Topic-aware text summarization based on bert. *IEEE Transactions on Computational Social Systems*, 9(3):879–890, 2022.

[30] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[31] Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 455–465, 2013.

[32] Tianchu Ji, Shraddhan Jain, Michael Ferdman, Peter Milder, H. Andrew Schwartz, and Niranjan Balasubramanian. On the Distribution, Sparsity, and Inference-time Quantization of Attention Values in Transformers. In *Findings of the Association for Computational Linguistics: ACL 2021*, Online, August 2021. Association for Computational Linguistics.

[33] Rangharajan Venkatesan, Yakun Sophia Shao, Miaorong Wang, Jason Clemons, Steve Dai, Matthew Fojtik, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. Magnet: A modular accelerator generator for neural networks. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[34] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: The impact of student initialization on knowledge distillation. *arXiv preprint arXiv:1908.08962*, 13, 2019.

[35] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[36] Yaru Hao, Li Dong, Furu Wei, and Ke Xu. Visualizing and understanding the effectiveness of bert. *arXiv preprint arXiv:1908.05620*, 2019.

[37] SynopsysDesignCompiler(2022).[Online].Available:https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html. [Accessed 28-03-2024].

[38] https://www.hwinfo.com/. Accessed: 2024-9-23.

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

[40] Hongxiang Fan, Thomas Chau, Stylianos I Venieris, Royson Lee, Alexandros Kouris, Wayne Luk, Nicholas D Lane, and Mohamed S Abdelfattah. Adaptable butterfly accelerator for attention-based nns via hardware and algorithm co-design. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 599–615. IEEE, 2022.

[41] Jan M. Rabaey. *Integrated Circuits: A Design Perspective*. Prentice Hall, 1996.

[42] TSMC and EUROPRACTICE. Tsmc - europractice information, 2023. Accessed: 2024-09-13.

[43] North Carolina State University. Freepdk45 - a 45nm open-source pdk, 2024. Accessed: 2024-09-13.

Ghadeer A. Jaradat received the B.S. and M.S. degree in Computer Engineering from Yarmouk University, Irbid, Jordan (2015, 2022). She is currently pursuing the Ph.D. degree in the Department of Electrical and Computer Engineering at Khalifa University. Her research interests include: AI Accelerator design, Digital ASIC Design and Computer Architecture.

Mohammed F. Tolba received the B.Sc. degree in electronics and communications engineering from Fayoum University, Faiyum, Egypt, in 2014, and the M.Sc. degree in electronic and electrical engineering from Micro-electronics System Design, Nile University, Giza, Egypt, in 2018. He is currently a Research Associate with SOC, Khalifa University. He authored or coauthored more than 28 journal and conference papers. His research interests include digital design and implementation of deep learning, convolution neural network, lightweight encryption, low-power approximation techniques, Graphics processing unit architectures, computer arithmetic, fractional order circuits, Memristor, and chaotic circuits. Mr. Tolba was the recipient of the Best Paper Award in Modern Circuits and Systems Technologies 2017 and Best master's Thesis Award in July 2018.

Ghada Alsuhli earned her B.S. and M.S. in Electronics and Communication Engineering from Damascus University, Syria (2009, 2015), and completed her Ph.D. in Electronics and Communication Engineering at Cairo University, Egypt (2019). Her academic journey was enriched by roles at esteemed research centers including the National Research Center, The American University in Cairo, Egypt, and the Khalifa University SoC Center, UAE. Currently, she serves as a Post-Doctoral Researcher at Khalifa University, leading several projects focused on efficient hardware implementation for AI and post-quantum cryptography. Her research encompasses embedded systems, energy-efficient IoT solutions, edge computing, efficient hardware implementation, and AI applications for wireless communications and biomedical engineering. She is the primary author of numerous papers in reputable journals and conferences, she has also authored a book on efficient DNN hardware implementation. Her contributions extend to reviewing for esteemed journals and committee memberships for international conferences.

Hani Saleh (Senior Member, IEEE) is an associate professor of ECE at Khalifa University since 2017, he joined Khalifa on 2012. He is a co-founder of Khalifa University Research Center 2012-2018, and the System on Chip Research Center (SoC 2019-present). Hani has a total of 19 years of industrial experience in ASIC chip design, Microprocessor/Microcontroller design, DSP core design, Graphics core design and embedded systems design. Hani a Ph.D. degree in Computer Engineering from the University of Texas at Austin. Prior to joining Khalifa University he worked for many leading semiconductor design companies including Apple, Intel, AMD, Qualcomm, Synopsys, Fujitsu and Motorola Australia. Hani has published more than 48 journal papers, more than 119 conference papers, more than 6 books and 7 book chapters. Hani research areas includes but not limited to: AI Accelerator design, Digital ASIC Design, Digital Design, Computer Architecture and Computer Arithmetic.

Mahmoud Al-Qutayri (Senior Member, IEEE) is Professor of Electrical and Computer Engineering at Khalifa University (KU), UAE. He is also affiliated with KU System-on-Chip Center. Prior to joining Khalifa University, he worked at De Montfort University, UK, and the University of Bath, UK. He also worked at Philips Semiconductors, Southampton, UK. Dr. Al-Qutayri has authored or coauthored numerous technical papers in peer-reviewed journals and international conferences. His current research interests include embedded systems, in-memory computing and emerging memory technologies, energy-efficient IoT systems, efficient edge computing and artificial intelligence hardware implementation, application of AI to wireless communication systems, wireless sensor networks, and cognitive wireless networks. He received a number of awards during his education and professional career. His professional services include serving on the editorial board of some journals as well as membership of the steering, organizing, and technical program committees of a number of international conferences.

Thanos Stouraitis (IEEE Life Fellow) is Professor of EECS at Khalifa University, UAE, and Professor Emeritus of the University Patras. He has also served on the faculties of Ohio State University, University of Florida, New York University, and University of British Columbia. He holds a Ph.D. from the University of Florida. His current research interests include AI hardware systems, signal and image processing systems, computer arithmetic, and design and architecture of optimal digital systems with emphasis on cryptographic systems. He has authored about 200 technical papers, several books and book chapters, and holds patents on DSP processor design. He received the IEEE Circuits and Systems Society Guillemin-Cauer Award. He has served as Editor or Guest Editor for numerous technical journals, as well as General Chair and/or Technical Program Committee Chair for many international conferences. He was President (2012-13) of IEEE Circuits and Systems Society.