

# Hybrid Dynamic Pruning: A Pathway to Efficient Transformer Inference

Ghadeer A. Jaradat, Mohammed F. Tolba, Ghada Alsahl, Hani Saleh, *Member, IEEE*, Mahmoud Al-Qutayri, *Member, IEEE*, Thanos Stouraitis, *Life Fellow, IEEE*, Baker Mohammad, *Member, 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 quadratic computational intensity and memory demands. To overcome these challenges we introduce a novel Hybrid Dynamic Pruning (HDP), an efficient algorithm-architecture co-design approach that accelerates transformers using head sparsity, block sparsity and approximation opportunities 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 row-balanced block pruning to prune unimportant blocks in the attention matrix at run time, 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 and power efficiency, we propose a HDP co-processor architecture.

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

## I. INTRODUCTION

Transformer models, including BERT[1], GPT [2], T5 [3] and others [4] [5] , have transformed Natural Language Processing (NLP) with their attention mechanism, achieving top performance in tasks such as question-answering [6], text classification [3], and machine translation [7]. The transformer architecture uses 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 has led to fast progress in NLP, with models like BERT exceeding human performance in difficult tasks [9] and expanding their application to computer vision, including object recognition and detection [10], image classification [11], and segmentation [12].

Deploying large transformer models on devices with limited resources is challenging due to their high computational and memory requirements. For example, BERT-Base Transformer needs 440 MB of memory and over 176 Giga FLoating Point Operations (GFLOPs) [13] . The computations are particularly difficult because of the complex attention operations and the quadratic computational complexity related to the length of input sequences [14]. Attention operations in transformer models become increasingly dominant as the input sequence length grows. For BERT-Base Transformer deployed on

edge platforms, with a sequence length of 512, 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 [15]. Therefore, finding efficient ways to handle attention operations is crucial for speeding up transformers.

Many studies utilized sparsity to mitigate the quadratic time and space complexity issue. Some techniques save computational effort by using fixed or static sparse attention patterns [16] [14] [17], but their performance is limited [18], since the sparse pattern in attention is naturally dynamic, and depends only on the input. Other techniques focus on dynamic sparsity, meaning there's no fixed pattern for which parts are sparse (zero). For example,  $A^3$  [19] 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 [20] 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 [15] 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 [21] prunes values in all transformer matrix multiplications if they fall below certain predetermined threshold.  $A^3$ , Energon, and AccelTran leverage unstructured sparsity to realize efficient deployment for Transformers which leads to non uniform data access and lower efficiency. It's hard to predict these sparsity patterns, which can slow down performance.

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. Researchers in [22] add trainable gating parameters attached to each head and regularized with  $L_0$  loss. In [23], the importance of each attention head is gauged based on its sensitivity to the overall loss. This sensitivity is utilized as an indirect measure to determine the significance of each head. In [24], a novel approach termed 'single-shot meta-pruner' is presented. This method 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. All of these studies perform pruning at compile time not run time, require retraining to recover the

accuracy drop. SpAtten [20] also performs a cascaded head pruning at run time using the Top-K approach, wherein 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<sup>3</sup>, Energon and AccelTran do not support head pruning.

To overcome the above challenges, 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 employs fine-grained block pruning for removing smaller, less critical blocks and incorporates head pruning to selectively eliminate less important heads, all based on the integer parts of inputs for decision-making. The pruning is done dynamically; applied during inference without relying on fixed patterns for pruning, and without fine-tuning or retraining. Our main contributions are as follows.

- 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 for enhanced efficiency.
- 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 the method in [20] which performs pruning after all computations.
- 3) We approximate attention calculation by breaking down the multiplication of Q and K into three components: *integerQ × integerK*, *integerQ × fractionalK*, and *fractionalQ × integerK*. By summing these components together, we not only approximate the attention outcome but also achieve near-zero pruning, as the *fractionalQ – fractionalK* multiplication is omitted. This method effectively reduces computational complexity while maintaining model accuracy during inference.
- 4) We design and implement an ASIC-based architecture to efficiently execute HDP, utilizing encoder-only models to reduce the critical path to the half and enhance throughput and hardware utilization. Our architecture functions as a co-processor, compatible with existing neural network accelerators. Through carefully designed pipelines and architectural optimizations, we aim to significantly boost performance and reduce energy consumption.

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 HDP co-processor. Section V outlines the experimental setup, the baselines used for comparison and discusses the results. Finally Section VI concludes the article.

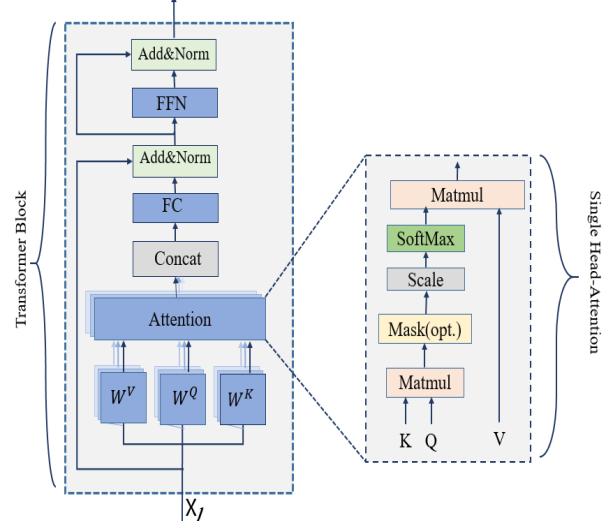


Fig. 1: Transformer Block

## II. BACKGROUND AND MOTIVATION

### A. Transformer Algorithm

Transformers have shown state-of-the-art performance in natural language processing field since their initial introduction in 2017 [8]. They are often regarded as alternatives to classic CNNs and RNNs in various situations in real-world due to their exceptional efficiency and generality. Transformers consist of two essential components: the encoder and the decoder, which are formed by layering many transformer blocks. As depicted in Fig. 1, a 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 (FFN), and the projection layer. The processes starts with transforming the input vectors into embedding vectors, which are then sequentially passed through a series of processing blocks. Within each block, the input is projected through the projection layer to *Query* ( $Q$ ), *Key* ( $K$ ) and *Value* ( $V$ ) features using the weights  $W^Q$ ,  $W^K$  and  $W^V$ . 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  $Q$ ,  $K$  and  $V$  matrices into multiple smaller sets  $Q_h$ ,  $K_h$ , and  $V_h$  equivalent to the number of heads  $H$ . Each of these sets forms an "attention head", see Fig. 1 right side. Inside each head the output is computed as follows:

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

The process starts with the computation of the dot product between  $Q_h$  and  $K_h$ , 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

**Algorithm 1:** Attention

---

```

0 Input:  $\{K, Q, V\} \in \mathbb{R}^{l \times d}$ 
1  $l$ : sequence length,  $d$ : input feature dimension
2 Number of heads:  $H$ ;
3  $Q_h, K_h, V_h \leftarrow$  Split  $Q, K, V$  into  $H$  chunks;
4  $d_h = d/H$ ;
5 for  $head_id \leftarrow 0$  to  $H$  do
6    $attention\_score = Q_h K_h^T$  ;
7    $attention\_score \in \mathbb{R}^{l \times l}$  ;
8    $attention\_score = \frac{attention\_score}{\sqrt{d_h}}$  ;
9   for  $row_id \leftarrow 0$  to  $l$  do
10    |  $attention\_prob =$ 
11    |    $\text{softmax}(attention\_score[row_id])$  ;
12   end
13    $result[head_id] = attention\_prob \cdot V_h$  ;
14 end
15 attention_out = concat( $result[head_id]$ ) ;
Output:  $attention\_out \in \mathbb{R}^{l \times d}$  ;

```

---

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 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 [25], text summarization [26], 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. The function of the FFN can be mathematically expressed as:

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

where  $X, W_1, b_1, W_2$ , 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, providing non-linear transformation to the output of the first layer before it is passed on to the second layer [27], where it is widely used in Transformers. Our focus in this work is the attention layer as it is the bottleneck for

Transformers models.

**B. Complexity of Attention Layers**

The primary computational requirements of attention layers in a neural network model are pairwise dot-products performed on a set of  $l$  vectors. As a consequence, the complexity is  $O(l^2d)$ . The computational cost associated with attention operations escalates as the length of the input sequence  $l$  increases, while  $d$  is usually fixed. By measuring the time consumed in attention layer for Bert-base model [1], on both embedded GPU and CPU platforms, it's observed that attention operations consume half of the total computational time for input sequences longer than 512. Furthermore, these operations consume almost 70% of computational time for input sequences reach 768 [15]. Therefore, in scenarios where processing long input sequences is necessary, attention operations emerge as the primary computational bottleneck. Also because of the advancements in linear layers through weight pruning [28], quantization [29], and specialized accelerators [30] [31], there is a need to optimize the attention mechanism to ensure balanced computational efficiency in transformer models.

**C. Dynamic sparsity in Multi Head Self Attention**

Sparse attention methods are driven by the recognition that not all attention probabilities are significant. After applying a row-wise *Softmax* function to get the *attention\_prob*, most likely a small subset of the scores in each row will impact the *attention\_out* results. Similarly, sparse multi-head attention strategies are typically motivated by the recognition that not all heads in the attention mechanism have an equal impact on the final output. This suggests that 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 attention probabilities in the attention architecture, indicating that the model's performance could be enhanced by focusing on the most crucial heads and removing unimportant *attention\_scores*.

By examining the attention weights matrix (*attention probability*) generated from different heads across various layers and inputs, we can say that the significance of individual heads in a transformer's multi-head attention mechanism depends only on the input data. Varying not only with the input data but also with their position within the model's layered architecture. Fig. 2 presents the attention weights of various heads across different layers and inputs in a BERT-base model fine tuned on the SST-2 dataset [32]. As illustrated in Fig. 2a, there is notable variation in the attention weights for the same head across different layers. For example, the attention values for the eleventh head, highlighted by the red box, show considerable fluctuation across Layers 9, 10, and 11. Additionally, the attention patterns of the same head within the same

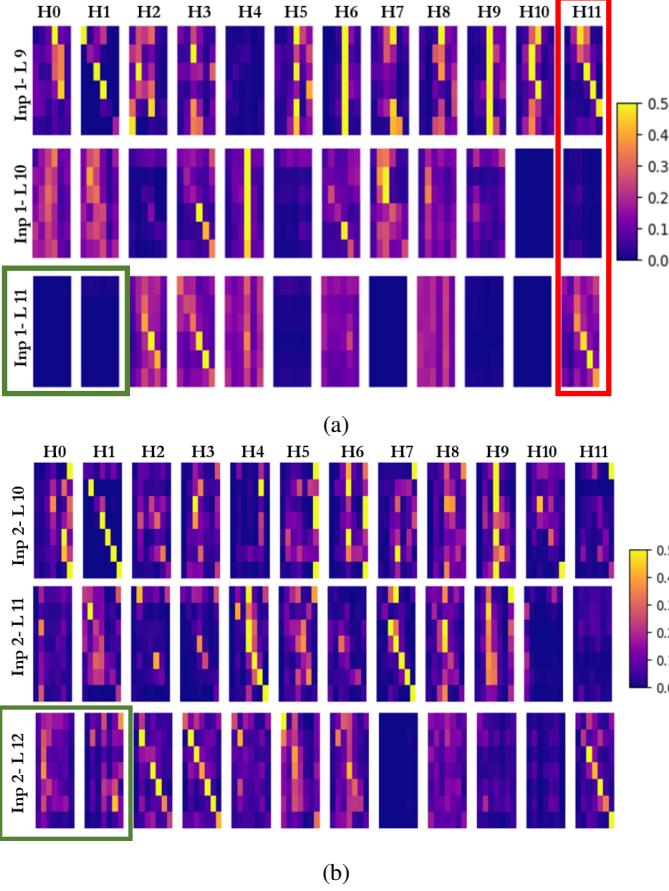


Fig. 2: Attention Probability Analysis in BERT-Base Model for (a) Input 1, (b) Input 2. The red box indicates the variability of attention probabilities across layers for the same input, for a single head (Head11) across different layers (Layers 9, 10, and 11). The green boxes highlight the contrast in the attention probabilities for the same head and layer with two distinct inputs. Head 0 and Head 1 in Layer 11 show lower values for Input 1, while Head 1 and Head 2 in the same layer exhibit significantly higher values for Input 2.

layer also exhibit significant differences when compared across various inputs. This is clearly depicted in Fig. 2a and Fig. 2b, where the green boxes depict the differences in attention values for the same heads and layers when subjected to different inputs. Specifically, Head 0 and Head 1 in Layer 11 exhibit notably low attention weights for Input 1. In contrast, for Input 2, Head 1 and Head 2 in Layer 11 display significantly higher attention values, highlighting the data-dependent nature of attention mechanisms in the model. Furthermore, it is clear for most of the heads that it is only a subset of the attention weights which have high magnitude, where there is no fixed pattern for the important weights.

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 role of a head may differ across layers. Furthermore, the high magnitude attention weights subset is dynamic depending on the input sequence, and different heads have different subsets. This data-dependent behavior of attention heads is a critical consideration in model analysis and optimization and motivates us to explore effective methods to eliminate unimportant heads and query-key relations and save computations.

### III. ALGORITHMIC OPTIMIZATION

In this section, we discuss the algorithmic optimizations that enhance the transformer model's efficiency and performance, focusing on block, head pruning and approximation techniques. These optimizations are key to reducing computational complexity and memory access.

#### A. Block Pruning

For the attention score matrix, most of the query-key relations are not important and can be pruned safely, many methods have been used to prune these relations. Top- $K$  pruning method [20] is used to prune the attention weights where a whole row can be pruned, but this requires a retraining to recover the accuracy, also it requires a specialized hardware to get the  $k$  most significant attention weights. Energon [15] avoided the Top- $K$  selection and used the mean filtering as a practical approximation instead, but it still has a separate unit to perform this operation, also faces data duplication overhead due to the multi-round filtering method employed. In Energon and AccelTran [21] the 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 an irregular memory access and stalls in the hardware.

To address these challenges we propose integer-based block pruning, where pruning decision is exclusively determined by the integer parts of the numbers. We employ a small block size for pruning to eliminate the necessity for retraining and to guarantee a more organized and hardware-compatible sparsity pattern.

Initially, multiplication is conducted only on the integer parts of  $Q$  and  $K$  to obtain *Integer\_atten*. For each  $2 \times 2$  block, we calculate its importance,  $\theta$ , as the absolute sum of the values within the block. For each row of blocks, we determine the block pruning ratio,  $\Theta$ , using a method similar to that in Energon, which involves calculating the minimum, maximum, and mean importance values, along with a predefined pruning ratio,  $\rho_B$ , as shown in Algorithm 2 line 15. Blocks with importance,  $\theta$ , falls below the row-specific threshold,  $\Theta$ , are pruned and the mask value for the block is assigned to 0. When a block is pruned, subsequent computations for that block are omitted. Conversely, if a

block is retained (mask is 1), the final attention result is approximated, with the approximation technique detailed in B. The block pruning mechanism is expressed in details in Algorithm 2 lines 6 to 17 and in Fig. 3.

### B. Approximation

For blocks with mask value equals 1,  $\theta$  exceeds  $\Theta$ , we proceed to approximate the final attention outcome after obtaining the *Integer\_att* result. This involves calculating two fractions: the product of  $Q$ 's fractional component with  $K$ 's integer component, and vice versa, yielding *Frac1\_atten* and *Frac2\_atten*, respectively. The ultimate attention score is obtained by summing these two fractions with the *Integer\_att*. This method not only approximates the attention outcome but also enables near-zero pruning, which has minimal impact on model accuracy during inference [33]. Specifically, when two numbers are close to zero, their integer parts are zero, leading all three components to be zero. Consequently, the multiplication of fractional parts is omitted, resulting in effective near-zero pruning. This approximation process is highlighted within the black box in Fig. 3 and in Algorithm 2 lines 19 to 28.

### C. Head Sparsity

Not all heads are essential, and many can be pruned without affecting the overall performance. Unlike the method in [20], where head importance is assessed after completing all computations for the head, we introduce an early head pruning approach. To evaluate head importance,  $\theta_{Head}$ , we compute the absolute summation of all values in *Integer\_att*. Heads with  $\theta_{Head}$  below a predefined threshold  $\tau_H$  are completely pruned and the rest of computations of this head are skipped. The threshold  $\tau_H$  is a parameter that will be profiled. The head pruning process is visually indicated by the red box in Fig 3 and described in Algorithm 2 at line 33.

## IV. HARDWARE ARCHITECTURE

Current attention accelerators and traditional CPUs/GPUs lack the capability to execute the proposed hybrid dynamic sparse attention technique efficiently. To address this gap, we are introducing a novel HDP accelerator. HDP is developed to function as a co-processor and is compatible with a variety of neural network accelerators for easy integration.

### A. Architecture Overview

The HDP architecture, depicted in Fig. 4, compromises multiple cores, the architecture of individual core is 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 *attention\_prob*  $\times V$ , also calculating importance values. The SE is tasked with identifying which blocks to prune and deciding whether a head should be pruned or not.

---

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

---

```

0 Input: Quantized  $\{K, Q, V\} \in \mathbb{R}^{l \times d}$ 
1 Block size:  $2 \times 2$ 
2 Block Pruning Ratio:  $\rho_B$ 
3 Head Pruning Threshold:  $\tau_H$ 
4 Integer part of Q:  $IQ$ , Fractional part of Q:  $FQ$ 
5 Integer part of K:  $IK$ , Fractional part of K:  $FK$ 
6  $Integer\_atten \leftarrow IQ \cdot IK^T ;$ 
   /* For the Integer_atten matrix with
       $2 \times 2$  block size, iterate over the
       $l/2$  rows of blocks */ 
7 for  $i \leftarrow 0$  to  $l/2$  do
8   /* Process each  $2 \times 2$  block within
      the row */ 
9   for  $j \leftarrow 0$  to  $l/2$  do
10     $\theta_j \leftarrow \sum_{x \in block_{ij}} |x| ;$ 
11     $\theta_{Head} \leftarrow \theta_{Head} + \theta_j ;$ 
12  end
13   $min_i \leftarrow \min(\theta_j) \quad \forall \theta_j ;$ 
14   $max_i \leftarrow \max(\theta_j) \quad \forall \theta_j ;$ 
15   $mean_i \leftarrow \frac{\sum_0^{l/2} \theta_j}{(l/2)} ;$ 
16
 $\Theta_i = \begin{cases} \rho_B \times max_i + (1 - \rho_B) \times mean_i & \text{if } 0 \leq \rho_B < 1 \\ -\rho_B \times min_i + (1 + \rho_B) \times mean_i & \text{if } -1 < \rho_B < 0 \end{cases}$ 
17   $Mask_j^i \leftarrow (\theta_j < \Theta_i)?0:1 ;$ 
18   $Integer\_atten \leftarrow Integer\_atten \odot Mask_j^i ;$ 
19 end
20 if  $\theta_{Head} > \tau_H$  then
21   for  $i \leftarrow 0$  to  $l/2$  do
22     for  $j \leftarrow 0$  to  $l/2$  do
23       if  $Mask_j^i == 1$  then
24          $Frac1\_atten_j^i \leftarrow IQ_i \cdot FK_j^T ;$ 
25          $Frac2\_atten_j^i \leftarrow FQ_i \cdot IK_j^T ;$ 
26       end
27     end
28   /* approximated attention score */ 
29    $approximation \leftarrow Integer\_atten + Frac1\_atten + Frac2\_atten ;$ 
30    $attention\_score \leftarrow approximation / \sqrt{d} ;$ 
31    $attention\_prob \leftarrow softmax(attention\_score) ;$ 
32    $result \leftarrow attention\_prob \cdot V ;$ 
33 else
   /* Prune the whole head */
34    $result = 0;$ 
35 end
36 Output:  $result \in \mathbb{R}^{l \times d} ;$ 

```

---

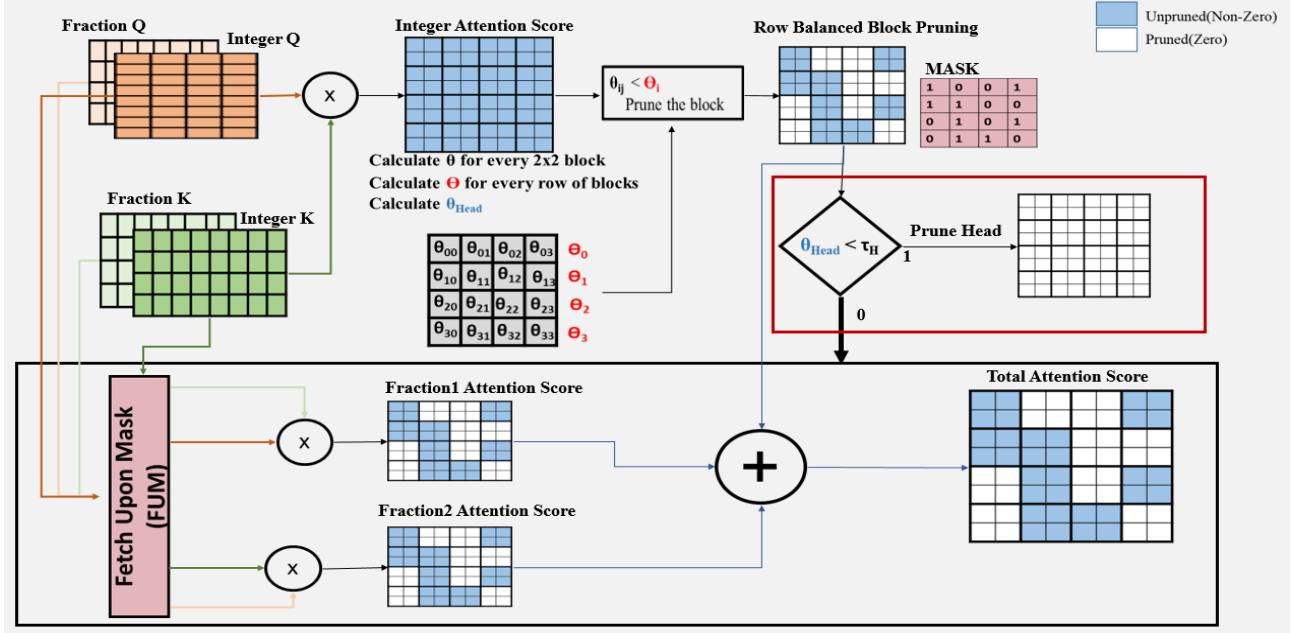


Fig. 3: Block Pruning, Head Pruning, and Approximation in Integer-Based Row-Balanced Block Sparsity: Pruning is applied to the result of  $\text{Integer}_Q \times \text{Integer}_K$  based on the comparison between the importance  $\theta$  of each block in a row and the row's threshold  $\Theta$ . Head pruning, highlighted by a red box in the image, is conducted for heads with  $\theta_{\text{Head}}$  below the predefined threshold  $\tau_H$ . The approximation process, highlighted by a black box in the image, involves generating fractional components and adding them to the pruned integer results to obtain the final output. This image represents an example with  $Q$  of size  $8 \times 4$ ,  $K$  of size  $4 \times 8$ , and a block size of  $2 \times 2$ .

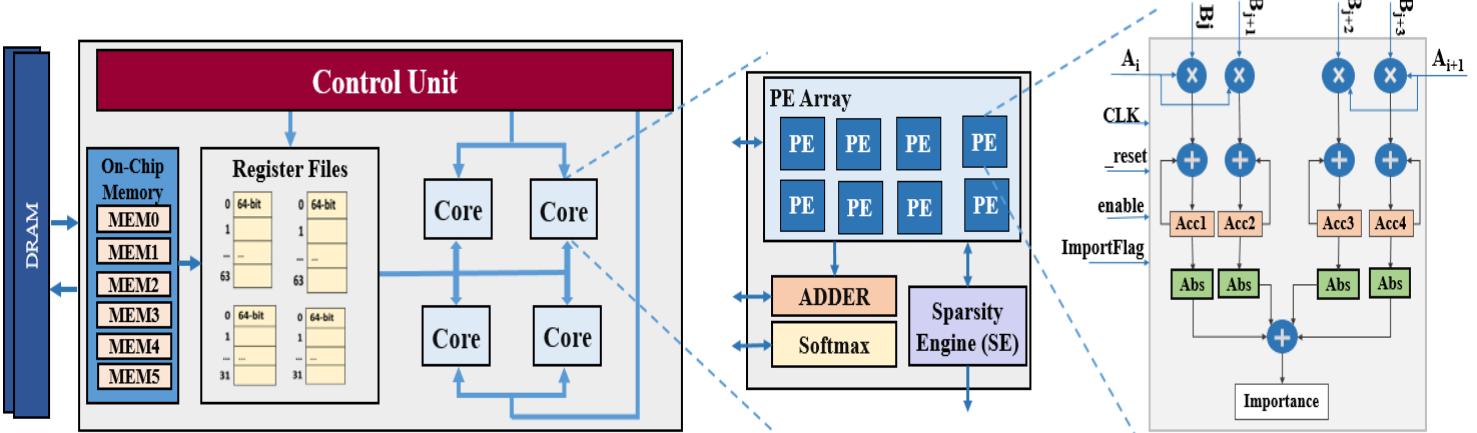


Fig. 4: HDP Architecture Overview.

**Workflow:** Once  $Q$ ,  $K$ , and  $V$  are generated and quantized by another processor in fixed point 16 bit format and stored in memory, HDP 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  $\text{Integer}_Q \times \text{Integer}_K$  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  $\text{Integer}_Q \times \text{Integer}_K$  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 proceeds to the next head. For heads that remain unpruned, a Fetch Upon Mask (FUM) strategy is utilized. 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 processing element (PE) array calculates the two remaining fractions ( $\text{Integer}_Q \times \text{Frac}_K$  and

$Frac\_Q \times Integer\_K$ ) simultaneously. These results, along with the integer results from the previous step, will be added together using an ADDER module to obtain the total *attention\_score*. After performing all  $Q \times K$ , a row-wise softmax will be applied to the *attention\_score*. Then, the PEs will be utilized to perform the calculation  $attention\_score \times v$ . Specifically, the first and second PEs in the first row calculate  $Integer\_attention\_score \times Integer\_v$ , while the third and fourth PEs in the first row compute  $Integer\_attention\_score \times Frac\_v$ . The first and second PEs in the second row calculate  $Frac\_attention\_score \times Integer\_v$ , and the third and fourth PEs in the second row calculate  $Frac\_attention\_score \times Frac\_v$ . Finally, all these results will be summed using the ADDER module to obtain the final output. The attention results for each tile will be immediately stored in DRAM memory upon completion. Consequently, host DNN accelerators can access these results to perform subsequent computations. In the sections that follow, we provide a more in-depth exploration of each module and the optimizations we have proposed to enhance their functionality.

## B. Tiling and Dataflow

A significant portion of the computational workload in transformer models is attributed to matrix multiplication, necessitating optimization to boost accelerator performance. We propose the implementation of tiled matrix multiplication, a technique primarily employed in GPUs [34], 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  $4 \times 4$  tile from matrix  $A$  is multiplied by the first  $4 \times 8$  tile from matrix  $B$ , with the partial results stored in a  $4 \times 8$  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 sums in  $C$ . During this stage, an output stationary dataflow approach is employed, facilitating the reuse of partial sum 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 [35]. Next, the process proceeds along ③ in all matrices, followed by a moving along ④.

## C. Processing Elements

The processing element, shown in Fig. 4 (right), serves as a fundamental computational unit within the accelerator handling all matrix multiplication operations. Operating in an output stationary mode, and behaving similar to systolic array PE; 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 sums in accumulators until the entire row from the first matrix has been

multiplied by the corresponding column from the second matrix. At this point, the accumulators hold the final results for the tile of the result matrix. In the case of  $Integer\_Q \times Integer\_K$  multiplication, these results are also utilized to determine the block's importance, as the output from a processing element corresponds to a block in the result matrix. The importance of the block, as illustrated in Fig. 4, is equal to the absolute sum of the accumulators.

## 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. Upon receiving the *END\_R*, which signals 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  $Integer\_Q \times Integer\_K$  multiplication, the engine calculates the block pruning threshold  $\Theta$  for that specific row. This calculation is based on the equation provided in line 15 of Algorithm 2. Additionally, the engine generates the *Mask* for the row by subtracting  $\Theta$  from the importance values of the blocks. If the result is negative, the block falls below  $\Theta$  and is marked for pruning.

Furthermore, when the *END\_H* is received, signaling the completion of the  $Integer\_Q \times Integer\_K$  multiplication, the engine utilizes the  $\theta_{Head}$  value it has computed representing the total sum of all importance values across the entire head. Upon receiving this flag, the engine compares it with  $\tau_H$ , an input parameter denoting the head pruning threshold. If  $\theta_{Head}$  falls below  $\tau_H$ , 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, a row-wise softmax function is applied defined as  $e^{s_i} / \sum_j e^{s_j}$ . For every input received to the module, the exponent is approximated using 2<sup>nd</sup> order polynomial. The exponent results are stored in internal memory, and the sum of these exponent results 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 the reciprocal to generate the softmax result.

## V. EVALUATION

### A. Algorithm Evaluation

1) *Evaluation Models and Datasets:* To validate the efficacy of our proposed method, HDP, we focused on

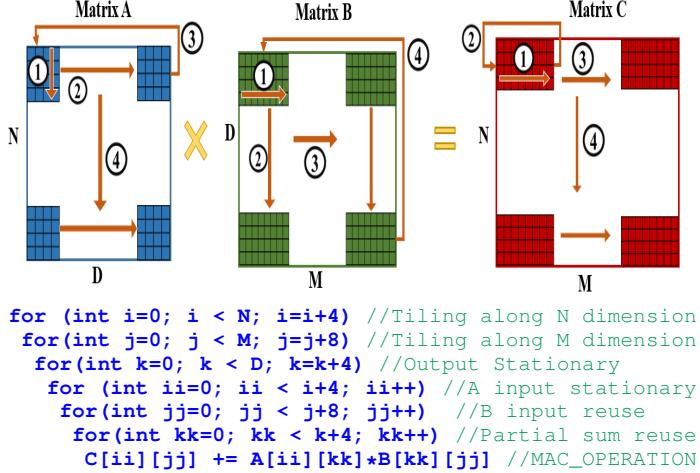


Fig. 5: Matrix Multiplication Tiling with the Dataflow.

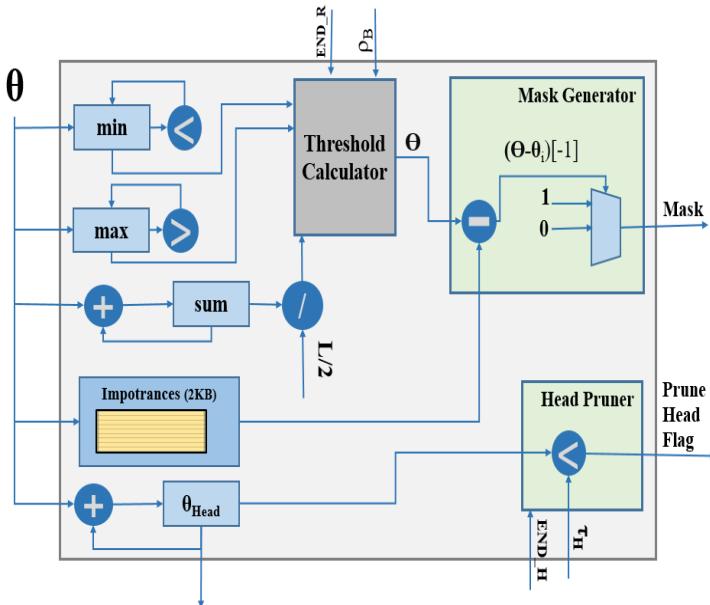


Fig. 6: Internal architecture of Sparsity Engine (SE).

evaluating encoder-only models. For this purpose, we utilized BERT-Tiny [36] 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. These encoder-only models are particularly promising for applications in areas such as machine translation [37] and language generation [38], where their efficiency and scalability can be leveraged for improved performance. Our evaluation was conducted on two benchmark tasks: SST-2 and COLA, both sourced from the GLUE benchmark [9].

2) *Dynamic Inference with Transformer:* we present the results of our experiments, 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 size of  $2 \times 2$ . As depicted in Fig. 7, 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%, HDP no longer serves as a reliable approximation to the Top-K method. This discrepancy is evident in the figure, where the accuracy of HDP is significantly higher than that of Top-K, indicating that the model is unable to accurately determine the correct block pruning threshold,  $\Theta$ . This issue arises because the model incorrectly assumes that it is pruning a high percentage of blocks, when in fact it is not, and this attributed to the assumption in that the mean divides the data into equal halves.

Both models exhibit an initial improvement in accuracy followed by a decline as the level of sparsity increases. This may be linked to the over-parameterization of the BERT model [39], and it can be thought as in NLP, if unimportant tokens are removed, the model can focus more on the important token resulting in better accuracy.

(b) **Head Pruning:** In HDP, the impact of head pruning is depicted in Fig. 8, 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, head pruning is particularly critical for BERT-Tiny, as illustrated in Fig. 8c and Fig. 8d. 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, with only 4 in total. Consequently, removing even a single head amounts to a significant coarse-grained pruning of one-fourth of all heads, and this is a very large coarse grained pruning to be done without any retraining. On the other hand, head pruning in the BERT-Base model yields more favorable pruning ratios due to its larger number of heads, totaling 144. As depicted in Fig. 8a and Fig. 11b, the model can prune approximately 13 – 17% of its heads with only a 1% decrease in accuracy.

(c) **Approximation:** To assess the effectiveness of the proposed approximation method, we will examine its impact on the models' accuracy. Fig. 9 illustrates the accuracy of models employing block pruning with and without the approximation technique. For the BERT-Base model, as depicted in Fig. 9a and Fig. 9b, the model's performance remains almost the same, suggesting that the approximation does not negatively affect the model while providing benefits in terms of computational efficiency. In contrast, for the BERT-Tiny model, as shown in Fig. 9c and Fig. 9d, the model is more

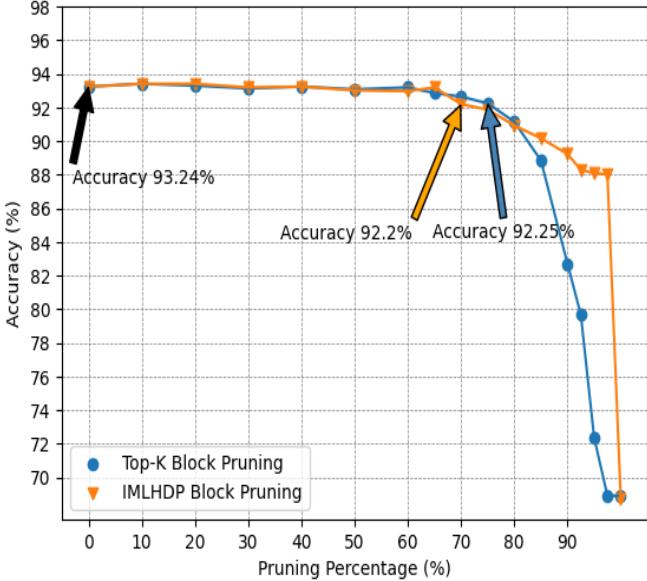


Fig. 7: Top-k VS HDP block pruning.

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, which allocates higher softmax values to unpruned elements, allows the model to focus more on crucial  $Q - K$  relations, thereby enhancing its concentration on important components. However, in some instances, the approximation may lead to reduced accuracy. This could be attributed to the nature of the approximation itself, as the fraction removed is not uniform across all values. Consequently, this could inadvertently lower the attention score for important  $Q - K$  relations in certain scenarios.

- (d) Net Pruning: Fig. 10 demonstrates the combined effect of block pruning, head pruning, and approximation techniques on the model's overall sparsity. 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. For BERT-Base on the CoLA dataset, the net sparsity is 65%. By leveraging net sparsity, we were able to attain a pruning ratio comparable to that of the Top-K method. In the Top-K approach, even within an unimportant head, certain blocks remain unpruned due to their significance within that specific head. However, the entire head may not be crucial overall. By implementing head pruning, we successfully removed such unimportant heads, thereby achieving a higher overall pruning ratio.

## B. Comparisons with Related Work

1) *Comparison with SpAtten:* To evaluate our proposed head pruning technique, we will conduct a comparison with SpAtten [20], the only study to date that dynamically applies head pruning directly on hardware platforms. As documented in SpAtten's findings for the BERT-Base model applied to the CoLA dataset, it achieved up to 17% head pruning with no loss in the accuracy. To ensure a fair comparison with SpAtten, we quantized our model to 12 bits and adhered to the identical fine-tuning protocol outlined in their study. The fine-tuning of the BERT-Base model on the CoLA dataset was completed on an average GPU in under two hours, employing various combinations of block pruning ratios and head pruning thresholds without pruning any thing from the first 30% of the layers. Fig. 11 shows these values. After fine-tuning, our method was able to prune approximately 17% of the heads same as SpAtten. It's worth noting that for higher pruning ratios for example 35% pruning percentage(1.55x pruning ratio), although there is a significant drop in accuracy, the decrease in accuracy is less pronounced in our model which is 7.5% compared to SpAtten which equal 10%. SpAtten employs a cascading head pruning approach, where once a head is pruned from one layer, it is also pruned from all subsequent layers. This is in contrast to findings that suggest head importance is only data-dependent, indicating that a head may be important in one layer but not in another, as demonstrated in Fig. 2.

Table I compares HDP with popular transformer accelerators.

TABLE I: Comparison of HDP with related works along different aspects.

Work	$A^3$ [19]	SpAtten [20]	Energon [15]	AccelTran [21]	<b>HDP (Ours)</b>
Head Pruning		✓			✓
Block Pruning					✓
Approximation	✓				✓
Tiled Mat. Mul.				✓	✓
Sparsity-aware		✓	✓	✓	✓
Dynamic Infer- ence	✓	✓	✓	✓	✓

## VI. CONCLUSION

In this work we presented HDP, a novel algorithm-architecture co-design to efficiently run dynamic sparse attention models. We first proposed a novel integer-based row-balanced block pruning to prune unimportant blocks in attention matrix and integer based head pruning to prune unimportant heads. Moreover, we propose an approximation method that reduces the computations and performs near-zero pruning. We also implemented this method in 2 co-processors architecture, HDP-Edge and HDP-Server, to accelerate algorithm on mobile and sever platforms.

## ACKNOWLEDGMENT

This work was supported by the Khalifa University of Science and Technology under SOCL.

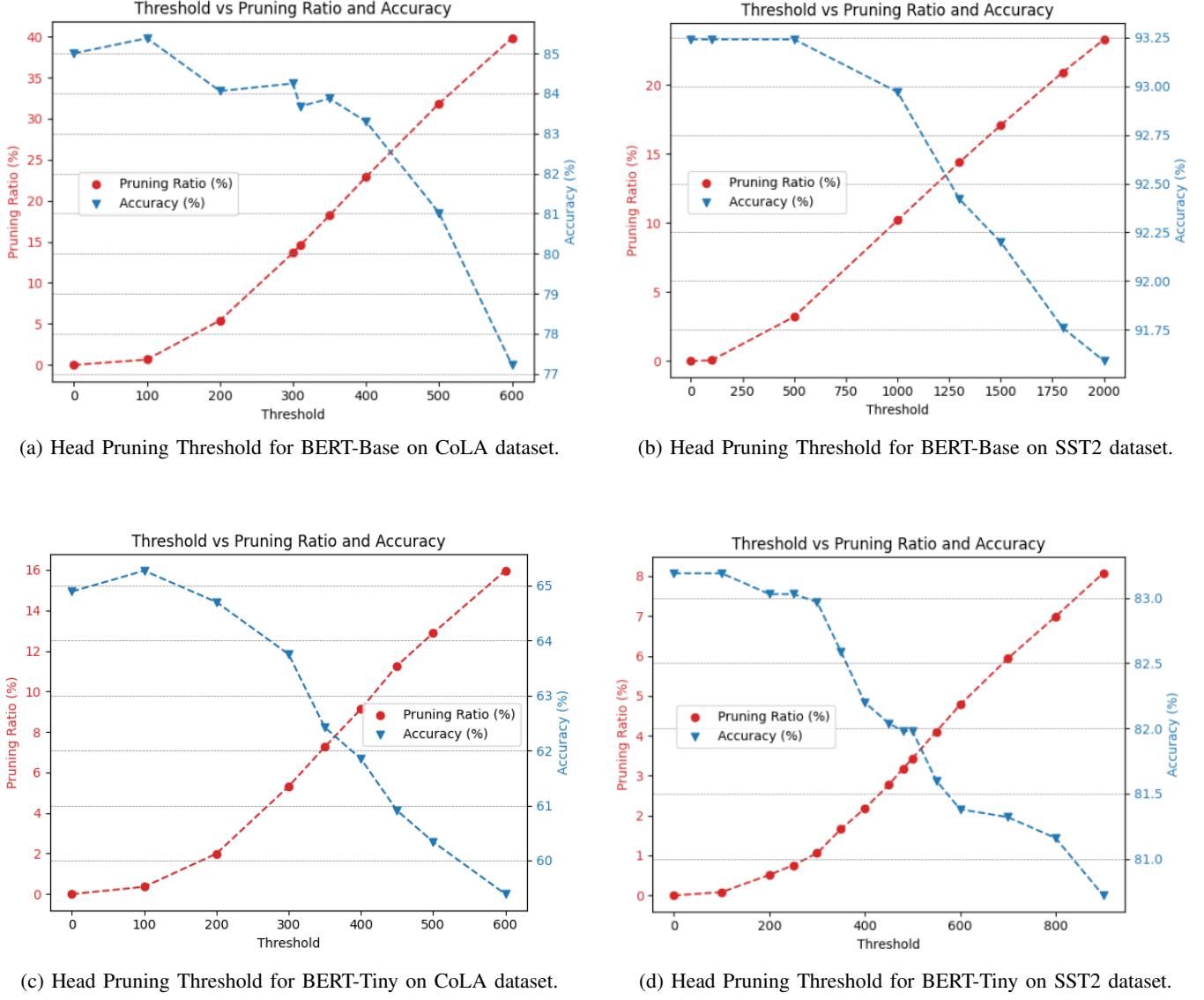
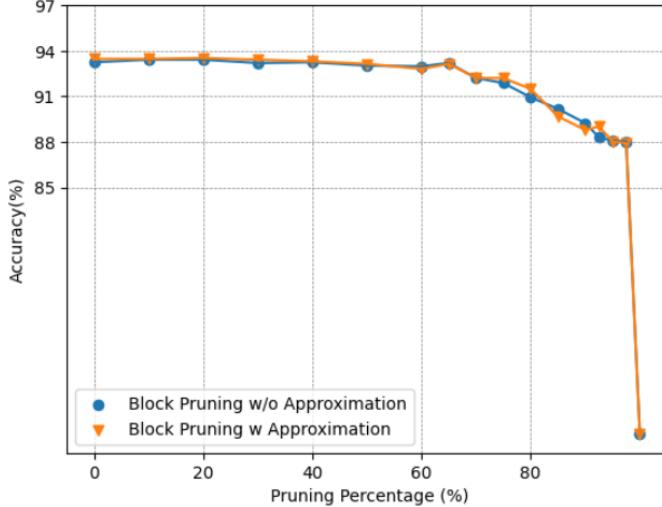


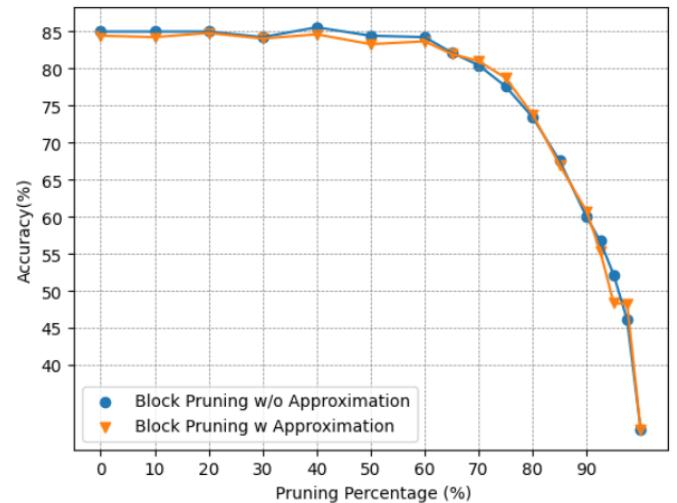
Fig. 8: Head Pruning Threshold for BERT-Base and BERT-Tiny on SST2 and CoLA.

## REFERENCES

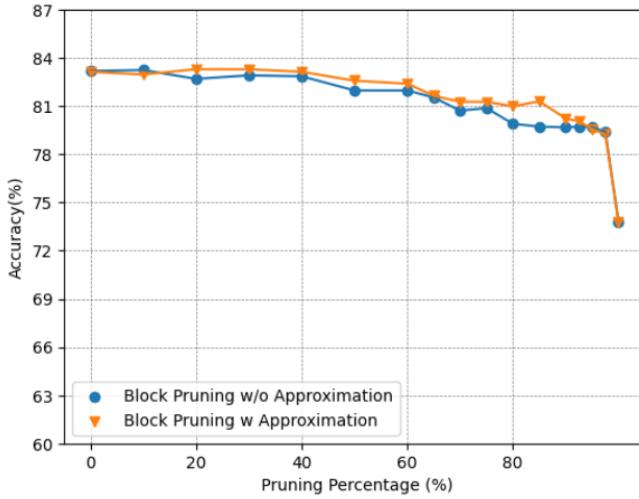
- [1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [2] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding 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] 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.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] Chao Fang, Aojun Zhou, and Zhongfeng Wang. An algorithm–hardware co-optimized framework for accelerating n: M sparse transformers.



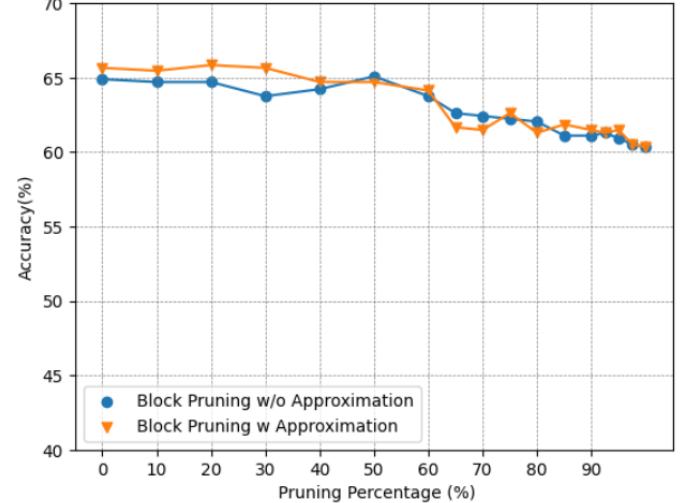
(a) Accuracy of BERT-Base on SST2.



(b) Accuracy of BERT-Base on CoLA.



(c) Accuracy of BERT-Tiny on SST2.

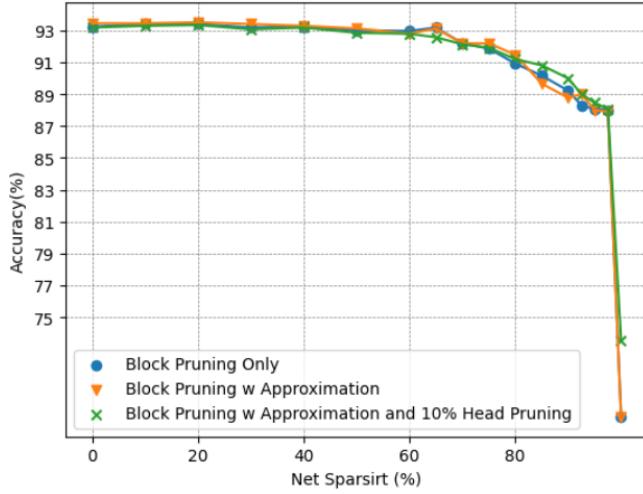


(d) Accuracy of BERT-Tiny on CoLA.

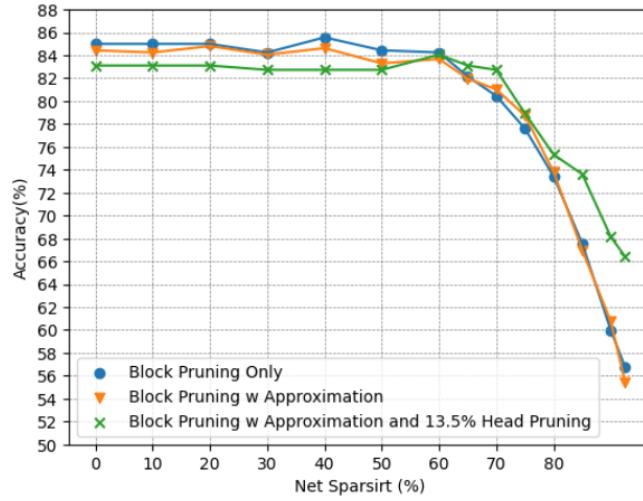
Fig. 9: Accuracy Comparison of BERT-Base and BERT-Tiny Block Pruning Method with and without Approximation on SST2 and CoLA.

*IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(11):1573–1586, 2022.

- [14] 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.
- [15] 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.
- [16] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- [17] 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.
- [18] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6), dec 2022.
- [19] 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. A<sup>3</sup>: Accelerating attention mechanisms in neural networks with approximation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 328–341. IEEE, 2020.
- [20] 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.
- [21] 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.
- [22] 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.
- [23] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? may 2019.
- [24] 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.



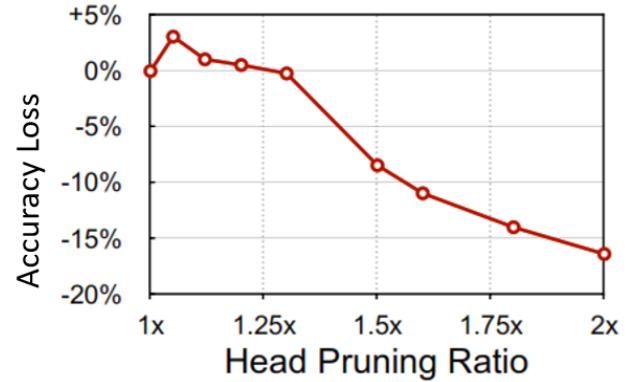
(a) BERT-Base on SST2.



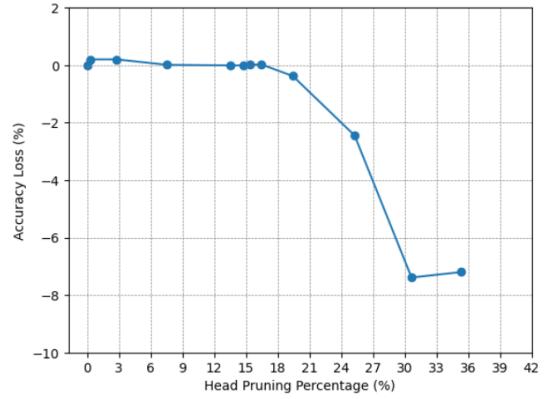
(b) BERT-Base on CoLA.

Fig. 10: Comparison of Accuracy vs. Net Pruning Ratio for BERT-Base Using Block Pruning, Head Pruning with and with Approximation on SST2 and CoLA Datasets.

- [25] 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.
- [26] Tinghai 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.
- [27] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.
- [28] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.
- [29] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. Adaptive quantization for deep neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- [30] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News*, 42(1):269–284, 2014.
- [31] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH computer architecture news*, 44(3):367–379, 2016.



(a) SpAtten Head Pruning Results[20].



(b) Proposed Fine Tuned Head Pruning Method.

Fig. 11: Accuracy of BERT-Base after dynamic head pruning in (a) SpAtten, (b) Our method fine tuned.

- [32] 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.
- [33] 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.
- [34] Yuyao Niu, Zhengyang Lu, Haonan Ji, Shuhui Song, Zhou Jin, and Weifeng Liu. Tilespgemm: A tiled algorithm for parallel sparse general matrix-matrix multiplication on gpus. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 90–106, 2022.
- [35] Ranaghan 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.
- [36] 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.
- [37] Jinhua Zhu, Yingce Xia, Lijun Wu, Di He, Tao Qin, Wengang Zhou, Houqiang Li, and Tie-Yan Liu. Incorporating bert into neural machine translation. *arXiv preprint arXiv:2002.06823*, 2020.
- [38] Sascha Rothe, Shashi Narayan, and Aliaksei Severyn. Leveraging pre-trained checkpoints for sequence generation tasks. *Transactions of the Association for Computational Linguistics*, 8:264–280, 2020.
- [39] Yaru Hao, Li Dong, Furu Wei, and Ke Xu. Visualizing and understanding the effectiveness of bert. *arXiv preprint arXiv:1908.05620*, 2019.