Optimizing Transformers with Approximate Computing for Faster, Smaller and more Accurate NLP Models

Amrit Nagarajan, Sanchari Sen[§], Jacob R. Stevens and Anand Raghunathan
Electrical and Computer Engineering, Purdue University
West Lafayette, IN 47906
Email: {nagaraj9, sen9, steven69, raghunathan} @ purdue.edu

Abstract

Transformer models have garnered a lot of interest in recent years by delivering state-of-the-art performance in a range of Natural Language Processing (NLP) tasks. However, these models can have over a hundred billion parameters, presenting very high computational and memory requirements. We address this challenge through Approximate Computing, specifically targeting the use of Transformers in NLP tasks. Transformers are typically pre-trained and subsequently specialized for specific tasks through transfer learning. Based on the observation that pre-trained Transformers are often over-parameterized for several downstream NLP tasks, we propose a framework to create smaller, faster and in some cases more accurate models. The key cornerstones of the framework are a Significance Analysis (SA) method that identifies components in a pre-trained Transformer that are less significant for a given task, and techniques to approximate the less significant components. Our approximations include pruning of blocks, attention heads and weight groups, quantization of less significant weights and a low-complexity sign-matching based attention mechanism. Our framework can be adapted to produce models that are faster, smaller and/or more accurate, depending on the user's constraints. We apply our framework to seven Transformer models, including optimized models like DistilBERT and Q8BERT, and three downstream tasks. We demonstrate that our framework produces models that are up to 4× faster and up to 14× smaller (with less than 0.5% relative accuracy degradation), or up to 5.5% more accurate with simultaneous improvements of up to 9.83× in model size or 2.94× in speed.

I. Introduction

Transformer networks with hundreds of billions of parameters, such as T5 ([17]), Megatron ([22]), BERT ([2]), GPT-2 ([16]) and GPT-3 ([1]), have achieved state-of-the-art performance in several Natural Language Processing tasks. Model sizes are expected to grow further in the future as increasing the number of parameters has been shown to improve performance. For instance, increasing the number of parameters from 1.5B to 175B enabled a reduction in perplexity for Language Modelling (Penn Treebank) from 35.8 in GPT-2 to 20.5 in GPT-3. This makes it computationally challenging to train Transformers as well as perform inference using them. The challenges associated with training these models are alleviated through the (re-)use of pre-trained models that are subsequently fine-tuned for different tasks. Consequently, these models incur a major one-time cost in computational resources, time and energy during the pre-training process, but the repeated fine-tuning for individual downstream tasks is performed at a considerably lower cost.

However, performing inference using fine-tuned Transformer models continues to remain a challenge because of the large amount of storage and compute operations required. Prior research efforts have explored different techniques for improving the efficiency of Transformer inference. However, several of the proposed approaches either require training the network completely from scratch (which is extremely compute and memory-intensive), or cause significant degradation in accuracy on the downstream task. In this work, we overcome these limitations by exploiting the transfer learning step in Transformers to produce individually optimized models for the different downstream tasks, using techniques that do not require training from scratch and maintain or improve accuracy levels.

From the runtime and memory breakdown of Transformers (Fig. 1), we observe that the most time-consuming and memory-intensive operations in a Transformer are the self-attention (ATTN) blocks, which are used to identify and form relationships between the different tokens in text, and the feed-forward neural network blocks (FFN blocks) in the Transformer layers. These blocks together account for more than 85% of the inference time (and more than 75% of the model's parameters). We accordingly optimize the execution of these two components in our approach. The self-attention component dominates the execution time and memory size for longer context lengths as its operation scales quadratically in time and memory with sequence length. Some previous works ([9], [29]) have addressed this issue, accelerating training and inference of Transformers when large context lengths are used. However, they suffer from significant overheads and slowdowns in applications with smaller context lengths, such as question answering, where questions and answers are usually short, in the order of a few hundred

[§]Sanchari Sen is currently a research staff member at IBM T.J. Watson Research Center, Yorktown Heights, NY (sanchari.sen@ibm.com).

tokens. Our approach, on the other hand, performs well across context lengths, size of hidden layers, number of layers and other network characteristics.

The pre-training of Transformer models with some initial objective (most commonly predicting masked words in a large text corpus) and the subsequent fine-tuning on a downstream task leads to highly over-parameterized models for many downstream tasks ([13]), providing ample opportunities for approximations. As these models grow larger, such opportunities are expected to increase even further. We observe that for a given downstream task, some parts of the pre-trained Transformer are more significant to obtain good accuracy, while other parts are less important or unimportant. In order to exploit this observation in a principled manner, we introduce a framework to introduce approximations while fine-tuning a pre-trained Transformer network, optimizing for either size, latency, or accuracy of the final network. We perform and apply significance analysis in a hierarchical manner, first pruning entire blocks, followed by attention heads, and finally pruning weight groups. We achieve further gains by also allowing elements that cannot be pruned to be approximated by other techniques. We specifically apply two forms of approximations, depending on the element type. For weights, we utilize quantization. For the self-attention operation, we replace the scaled dot product attention mechanism with a novel sign matching-based attention mechanism.

We summarize our main contributions as follows:

- We introduce a framework for creating fine-tuned models from pre-trained Transformer models that are optimized for various metrics (size, latency, accuracy).
- We incorporate multiple heuristics in the framework, such as hierarchical processing, model-driven insights, and run-time based ordering of elements.
- We propose a significance analysis technique to identify the importance of each element of the pre-trained Transformer for a given downstream task. We use this technique to prune entire blocks, attention heads, and weight groups and to guide the quantization of low-importance weights.
- We propose a low-complexity attention mechanism, sign matching, in order to approximate dot product attention in the less significant attention layers.
- Across a suite of different Transformer networks, including previously proposed optimized networks, we demonstrate that our techniques produce models that are up to 4× faster and up to 14× smaller (with less than 0.5% relative accuracy degradation), or up to 5.5% more accurate with simultaneous size and latency improvements.

II. Related Work

Given the effectiveness and popularity of Transformer models, several techniques have been proposed to overcome their computational and memory challenges, and to accelerate inference using these models. Most of these works directly pre-train efficient models from scratch. For example, DistilBERT ([20]), MobileBERT ([23]) and TinyBERT ([7]) use knowledge distillation to train smaller and faster networks using the original network as a teacher. LayerDrop ([4]) randomly drops layers during pre-training, thereby enabling their dropping during inference. SchuBERT ([8]) learns the optimal sizes of the BERT elements during pre-training. Lite Transformer ([26]) uses Long-Short Range Attention to speed up the self-attention operation, with different attention heads attending to local and global context. Depth-adaptive Transformer ([3]) and DeeBERT ([27]) modulate Transformer depth depending on the complexity of each input sample using gating functions that are trained along with the model. AlBERT ([10]) uses factorized embeddings and cross-layer parameter sharing. These works are orthogonal to ours, as the models that they produce are still subsequently fine-tuned for downstream tasks. We demonstrate using DistilBERT, AlBERT and LayerDrop as examples that these optimized networks still have significant opportunities that our techniques can take advantage of.

Other works (including ours) aim to improve the inference efficiency of Transformers using techniques that do not require training new models from scratch. Among these, PoWER-BERT ([5]), which eliminates redundant word vectors from the model without removing any parameters, and Q8BERT ([30]), which quantizes all weights and activations in the model to 8-bit integers through the use of Quantization-Aware Training at fine-tuning time, are orthogonal and complementary to our work. Poor Man's BERT ([19]) evaluates several layer-dropping techniques that do not require re-training. Compared to layer-dropping techniques that do not require re-training, our techniques produce models that are up to 20% more accurate at comparable inference speed, and this is especially true when working with highly optimized baselines such as Q8BERT. Our framework can also be adapted to satisfy a wide range of user constraints.

III. Preliminaries

A Transformer (Fig. 1) consists of an embedding layer, followed by multiple transformer layers stacked together, and a task-specific final layer. A transformer layer consists of the multi-headed self-attention operation (ATTN block), followed by a feed-forward neural network (FFN block) with layer norm operations at the input and output of the layer. In this work, we define the *elements* of a Transformer to include different levels of granularity, *i.e.*, ATTN blocks, FFN blocks, Attention Heads and Weight Groups. We define Weight Groups only along dimensions that do not impact the shape of the output of the block when these groups are removed.

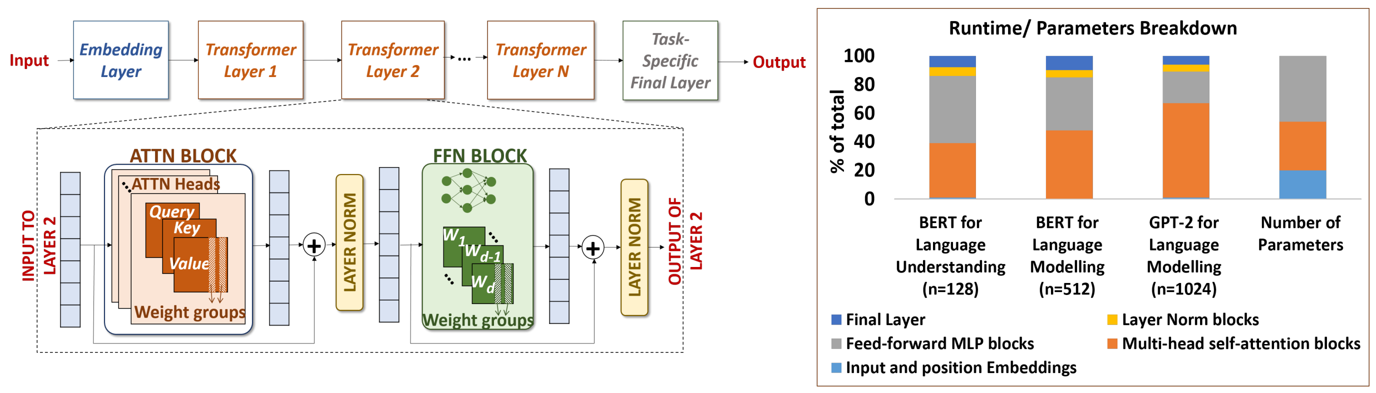


Fig. 1. **[Left] A typical Transformer architecture and its elements. [Right] Runtime/parameter-count breakdown of Transformers.** At smaller context lengths, the feed-forward neural network is the time-dominant operation. As context length increases, self-attention becomes the time-dominant operation. In both cases, ATTN and FFN blocks together account for >75% of total parameters, and >85% of the total runtime on an NVIDIA GTX 1080 Ti GPU.

The self-attention operation takes as input a sequence n of vectors X, and computes three matrices, Query = $X \times W_q$, Key = $X \times W_k$ and Value = $X \times W_v$. Then, the output of the self-attention operation is computed as $Y = softmax((Query \times Key^T) + attention\_mask) \times Value$. For auto-regressive models, tokens are not allowed to attend to future tokens. Hence, an attention mask is applied before the softmax operation, setting attention scores with future tokens to a very large negative number, which becomes zero after the softmax operation. This operation has multiple "attention heads" working in parallel on the input sequence, where each head has its own set of parameters to compute the query, key and value matrices. The independent attention outputs are concatenated and transformed into the expected output dimensions. The self-attention operation scales quadratically in time and memory with sequence length n since $Query \times Key^T$ has n^2 entries.

IV. Design Methodology

We propose a framework for producing fine-tuned Transformer models that are optimized for a specific metric (speed, model size, or accuracy). Fig. 2 presents an overview of the proposed framework. As shown in the figure, the inputs to the framework are a pre-trained Transformer model, the fine-tuning dataset, the goal of optimization (speed, size or accuracy) and acceptable accuracy loss (when optimizing for speed or size). The framework has three major components: (i) a set of heuristics used to build an ordered queue of elements (TransElements) to be considered, (ii) a significance analysis method to identify insignificant elements in a pre-trained Transformer and (iii) a set of techniques to prune or approximate the insignificant elements. The framework proceeds in an iterative manner. That is, we first start with the original Transformer. We then remove an element from the TransElements queue, analyze its significance, and apply pruning/approximation techniques to the element. This results in new Transformer, where the element is replaced by the pruned or approximated version. This modified Transformer is then used as the baseline for the next iteration. After processing all of the identified elements, we fine-tune on the downstream task for the same number of epochs as the baseline model to obtain the final, optimized model. In the following subsections, we further describe our techniques for generating the ordered queue TransElements, followed by the significance analysis method, and finally the pruning and approximation techniques for different Transformer elements.

A. *TransElement Ordered Queue*

In order to optimize a given model, we would ideally want to characterize the significance of each and every parameter in the model, rank them in order of importance, and finally prune/approximate only the least significant parameters, as in [14]. However, Transformers have billions of parameters, making this process computationally infeasible. In addition, previously proposed techniques that can efficiently estimate the importance of each parameter, such as using Taylor expansion, are not applicable. This is because the {approximate, fine-tune, approximate} cycle does not work for Transformers during fine-tuning, since they very quickly overfit the training data for the downstream task (usually within 5 epochs). We take advantage of hierarchical structure of Transformers and consider them in a hierarchical manner, ordered by increasing granularity. Specifically, we place entire FFN and ATTN blocks earlier in the queue, followed by heads, and finally weight groups. Through this ordering, we are able to quickly eliminate large numbers of parameters from further consideration, speeding up future iterations of the framework. For example, eliminating a single FFN block in the BERT-Base model removes 5.6% of all parameters under consideration. To further reduce the number of elements under consideration, we also dynamically remove elements from the queue if they are encompassed by a high-importance block. For example, if a given ATTN block is determined to be of high importance, we remove all heads and weight groups within that block from the TransElement queue.

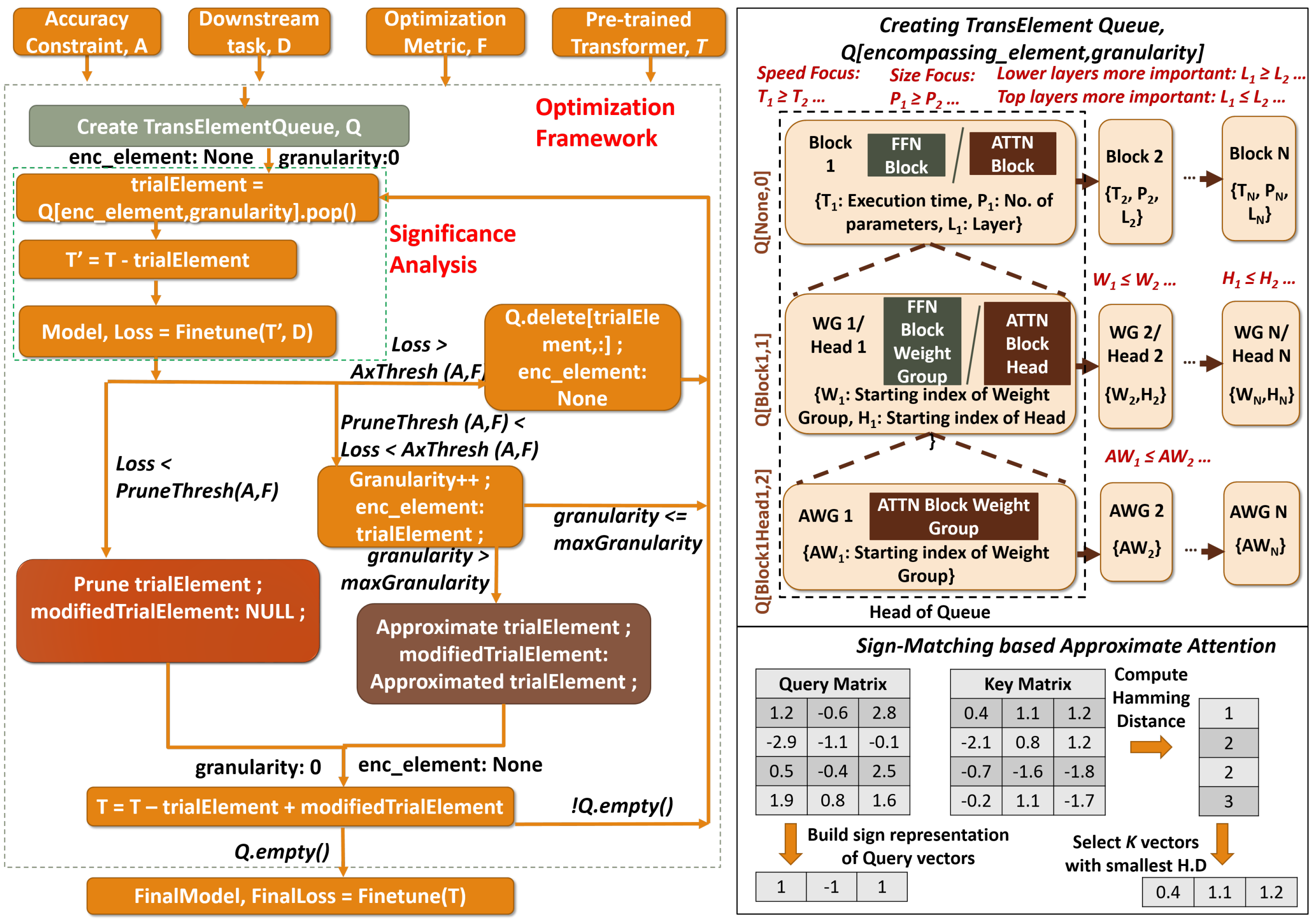

Accuracy Constraint, A
Downstream task, D
Optimization Metric, F
Pre-trained Transformer, T
Optimization Framework
Create TransElementQueue, Q
enc_element: None
granularity:0
trialElement = Q[enc_element,granularity].pop()
T' = T - trialElement
Model, Loss = Finetune(T', D)
Significance Analysis
Loss > AxThresh (A,F)
Q.delete[trialElement,:] ; enc_element: None
PruneThresh (A,F) < Loss < AxThresh (A,F)
Loss < PruneThresh(A,F)
Granularity++ ; enc_element: trialElement ;
granularity <= maxGranularity
granularity > maxGranularity
Prune trialElement ; modifiedTrialElement: NULL ;
Approximate trialElement ; modifiedTrialElement: Approximated trialElement ;
granularity: 0
enc_element: None
T = T – trialElement + modifiedTrialElement
!Q.empty()
Q.empty()
FinalModel, FinalLoss = Finetune(T)
Creating TransElement Queue, Q[encompassing_element,granularity]
Speed Focus: $T_1 \geq T_2$ ...
Size Focus: $P_1 \geq P_2$ ...
Lower layers more important: $L_1 \geq L_2$ ...
Top layers more important: $L_1 \leq L_2$ ...
Q[None,0]
Block 1
FFN Block
ATTN Block
{T_1: Execution time, P_1: No. of parameters, L_1: Layer}
Block 2 {T_2, P_2, L_2}
Block N {T_N, P_N, L_N}
$W_1 \leq W_2$ ...
$H_1 \leq H_2$ ...
Q[Block1,1]
WG 1/ Head 1
FFN Block Weight Group
ATTN Block Head
{W_1: Starting index of Weight Group, H_1: Starting index of Head}
WG 2/ Head 2 {W_2,H_2}
WG N/ Head N {W_N,H_N}
$AW_1 \leq AW_2$ ...
Q[Block1Head1,2]
AWG 1
ATTN Block Weight Group
{AW_1: Starting index of Weight Group}
AWG 2 {AW_2}
AWG N {AW_N}
Head of Queue
Sign-Matching based Approximate Attention
Query Matrix
1.2 -0.6 2.8
-2.9 -1.1 -0.1
0.5 -0.4 2.5
1.9 0.8 1.6
Key Matrix
0.4 1.1 1.2
-2.1 0.8 1.2
-0.7 -1.6 -1.8
-0.2 1.1 -1.7
Compute Hamming Distance
1
2
2
3
Build sign representation of Query vectors
1 -1 1
Select K vectors with smallest H.D
0.4 1.1 1.2

Fig. 2. **Illustration of the Transformer Approximation Methodology. Sign Matching is illustrated for K=1**

Since the framework iterates through the entries of the TransElement queue sequentially, its efficacy is dependent on the ordering of the elements at each level of granularity. In order to minimize the run-time of the framework, we provide two additional heuristics to guide the ordering of elements. First, we use the unique linguistic properties captured by the different Transformer layers ([6]). These properties depend on both the Transformer and the downstream task under consideration, since different tasks require different types of linguistic knowledge. For example, top layers usually have low significance for Language Understanding tasks, since long-range dependency information is not required for most tasks (for example, sentiment analysis requires only local context). Hence, we place the final layer at the front of the queue, and work our way backwards towards the first layer, since blocks in the final layers are more likely to be removed, thereby speeding up future iterations. Second, we use a run-time (or parameter-count) aware ordering of the TransElements, such the most time consuming blocks (or blocks with the most parameters) are likely to be removed earlier in the algorithm. For example, at large context lengths, we start with the ATTN blocks in all layers before moving on to the FFN blocks, and vice-versa at small context lengths. This has the dual benefit of producing highly optimized models for inference, as well as speeding up the significance analysis process by eliminating time-consuming blocks early and making further iterations faster.

B. Significance Analysis

To determine the significance of each Transformer element, we first fine-tune the original Transformer model for the given downstream task to obtain the baseline loss. We then use this baseline loss, along with the provided acceptable accuracy degradation, to generate a set of loss thresholds that determine whether a given element is of low importance and therefore can be pruned/approximated. This is a one-time step and performed globally for all elements in the TransElements queue.

Then, for the element under consideration in each iteration of the framework, we compute the loss of the current Transformer model with the element removed. We then compare this loss to the thresholds determined above. The exact thresholds used are dependent on the optimization metric: speed, size, or accuracy. If we are optimizing the network for speed or size, we prune the element under consideration if the training/validation loss upon removing it from the Transformer is less than the pruning threshold. If we are optimizing for accuracy, we prune the element only if the training/validation loss when it is removed is less the minimum loss seen thus far during the optimization process, since the goal is to find a model with minimum loss. Similarly, we apply approximations if the loss with the element removed from the Transformer is greater than the pruning threshold but lower than the approximation threshold.

C. Pruning and Approximating

As evident from Preliminaries, the structure and functionality of ATTN blocks differ significantly from that of FFN blocks in a Transformer. We accordingly adopt different strategies for approximating them, as described below. But pruning an entire ATTN or FFN block is effectively the same as it simply involves using the skip connection to bypass that block.

Pruning Weight Groups within approximable FFN Blocks. Consider an approximable FFN block that performs the transformation $\mathbb{R}^{n\times d} \times \mathbb{R}^{d\times y} \to \mathbb{R}^{n\times y}$, with weight groups defined along the d dimension ((d/W) weight groups of (Wn) weights each, where W is a hyperparameter that defines the granularity of approximations). When optimizing models for accuracy, we remove weight groups only if doing so results in a reduction in the model loss. When optimizing for size, we remove weight groups that maintain loss within the pruning threshold when removed. When optimizing for speed, however, removing weight groups with low significance from arbitrary locations does not help, since it introduces unstructured sparsity in the weight matrix that can be difficult to exploit to achieve speedups. Instead, we impose structure on our pruning. Specifically, we use a greedy algorithm that finds the largest number of weight groups that can be removed while maintaining loss below the threshold, such that the weight groups that remain in the model form a contiguous block. We first start from the bottom (weight group 0), work our way up and remove as many weight groups as possible while staying within the loss threshold. We then start from the top (weight group d/W), work our way down and remove as many weight groups as possible while staying within the loss threshold. When this process is completed, the weight groups that remain form a contiguous dense block, enabling speedups on all hardware platforms. Since weight groups are removed along the "hidden" dimension d, our methods do not change the shape of the output of this block; instead, we are simply "shrinking" the effective hidden dimension size through structured pruning.

Quantizing Weight Groups within approximable FFN Blocks. When optimizing the Transformer for size, we also quantize weight values within weight groups for which the loss lies between the pruning and approximation thresholds. This reduces the memory requirements of those weight groups but doesn't improve the execution time as the computations are still performed at the baseline precision.

Pruning ATTN heads and Weight Groups within approximable ATTN Blocks. We divide the multi-headed self-attention operation into two main steps. In the first step, we compute the Query, Key and Value matrices by multiplying the input to this layer with the corresponding weight matrices ($\mathbb{R}^{n\times d} \times \mathbb{R}^{d\times y} \to \mathbb{R}^{n\times y}$), and then reshape them into multiple attention heads ($\mathbb{R}^{n\times y} \to \mathbb{R}^{n\times h\times (y/h)}$). Our approach to pruning this step is exactly the same as for the FFN blocks, where we iteratively prune weight groups along the d dimension using our shrinking algorithm. In the second step, we compute the "attention output" as $Y = softmax((Query \times Key^T) + attention\_mask) \times Value$. To optimize this step, we apply two techniques. Firstly, we identify insignificant attention heads, and prune them from the model. However, removing attention heads changes the shape of the output of this layer. We overcome this by keeping track of the pruned heads, and padding the output with zeros in the corresponding locations. In spite of this overhead, we still manage to achieve significant speedup from this approximation technique since pruning heads makes multiple downstream operations (computing the attention scores, applying softmax to the attention scores, and computing the final score) considerably faster. Secondly, we reduce the size of the key and value matrices by pruning weight groups from the same location along the n dimension in both matrices. This again makes multiple downstream operations considerably faster and does not change the shape of the output of the pruned block. In summary, we do not use our hidden dimension shrinking method for structured pruning, but rather rely on unstructured pruning as it allows for greater pruning which further benefits the downstream operations.

Approximating self-attention within approximable ATTN Blocks. We observe that the "attention scores" matrix is highly sparse, especially after the softmax operation. This sparsity implies that most of the dot products between the query and the key are unnecessary. Thus, we would ideally like to perform the attention operations for the query vectors that give highest dot-product with each key vector efficiently without explicitly performing all of the dot products. To this end, we propose replacing the $O(n^2)$ dot product-based attention mechanism with a linear-time sign-matching-based mechanism in approximable ATTN blocks. Sign-matching attention (SM) is based on the idea that key vectors whose signs match with the largest number of query vectors will have high dot-products with maximum number of query vectors. However, it is expensive to compute a sign match for all pairs of query-key vectors, as this will grow quadratically. Instead, we employ a low-cost approximation. For each column of the query matrix, we identify if more number of vectors will have a positive or negative number in that column. This becomes the representative sign in that column for all the query vectors. Each key vector is then scored by how well the sign of each of its elements matches with the sign of the representative query vector by computing the Hamming distance between the two sign vectors. This score is used to select the top K key vectors. As a result, we reduce the number of computations required to score each key vector (and the overall complexity) from O(n^2) to O(n). Sign matching is illustrated in Fig. 2, and explained in detail in Appendix A. As this approximation does not increase the accuracy of the models nor decrease the number of parameters, they are only applied when optimizing the fine-tuned models for speed.

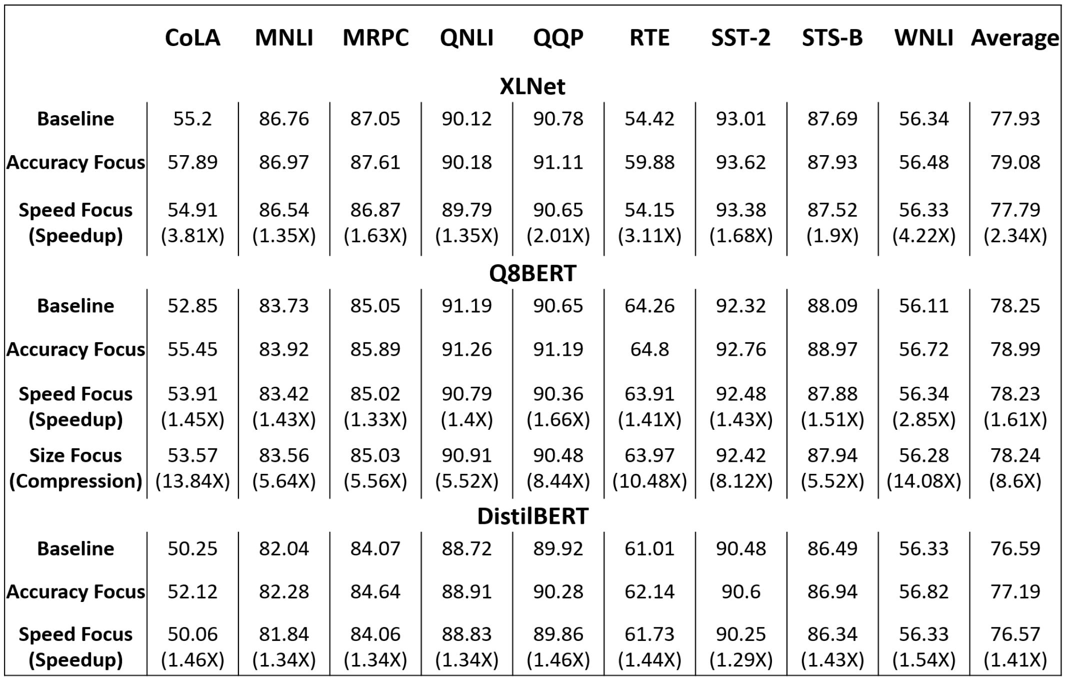
V. Experiments and Results

We implement our techniques within Huggingface's Transformers library in PyTorch ([25]). We use Intel AI's NLP Architect for experiments on Q8BERT. The experiments were performed on a GeForce RTX 2080 Ti GPU with 11GB memory. We present results on several NLP tasks with multiple Transformer architectures. All results reported are the average of 10 runs with random seeds after 3 epochs of fine-tuning on the dev set, unless otherwise specified. Detailed descriptions of the tasks and Transformers used in our experiments is given in Appendix B. Additional results on the GLUE test set are presented in Appendix C.

A. Primary Results

We present results on GLUE ([24]) (Table I), SQUADv1.1 ([18]), and Penn Treebank ([12]) (Table II). When optimizing for speed, we aim to reduce inference time as much as possible while maintaining at least 99.5% of baseline accuracy. While the focus in these experiments is on speed, we find that our framework still leads to models that are `1.29×-10.65×` smaller due to TransElements being dropped from the pre-trained model. On the other hand, when optimizing for size, we focus on reducing the model size as much as possible while maintaining the <0.5% accuracy degradation constraint. We use uniform quantization with Quantization-Aware Training proposed in Q8BERT ([30]) within our hierarchical framework to quantize insignificant blocks, heads and weight groups to lower precisions. This leads to models that are smaller than and at least as fast as a uniform 8-bit integer quantized model such as Q8BERT. Our results are competitive with QBERT ([21]), which uses second order Hessian information to quantize BERT, while maintaining the advantages of uniform 8-bit quantization over the group-wise quantization proposed in QBERT. The compression is lowest for AlBERT since its parameters are shared across layers, and most of the compression is from quantization. While the focus in these experiments is on size, we find that our framework still leads to models that are `1.07×-3.26×` faster due to elements being dropped from the pre-trained model, with potential for much greater speedups on optimized 8-bit integer kernels. When optimizing for accuracy, the goal is to maximize the accuracy of the pre-trained Transformer model for any given downstream task. While the focus in these experiments is on accuracy, we find that our framework still leads to models that are `1.28×-9.83×` smaller and `1.03×-2.94×` faster due to TransElements being dropped from the pre-trained model.

TABLE I
Results on GLUE. We report Matthews correlation for CoLA, Pearson Correlation for STS-B and accuracy for all other tasks. We report only "matched" accuracy for MNLI.



	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI	Average
XLNet										
Baseline	55.2	86.76	87.05	90.12	90.78	54.42	93.01	87.69	56.34	77.93
Accuracy Focus	57.89	86.97	87.61	90.18	91.11	59.88	93.62	87.93	56.48	79.08
Speed Focus (Speedup)	54.91 (3.81X)	86.54 (1.35X)	86.87 (1.63X)	89.79 (1.35X)	90.65 (2.01X)	54.15 (3.11X)	93.38 (1.68X)	87.52 (1.9X)	56.33 (4.22X)	77.79 (2.34X)
Q8BERT										
Baseline	52.85	83.73	85.05	91.19	90.65	64.26	92.32	88.09	56.11	78.25
Accuracy Focus	55.45	83.92	85.89	91.26	91.19	64.8	92.76	88.97	56.72	78.99
Speed Focus (Speedup)	53.91 (1.45X)	83.42 (1.43X)	85.02 (1.33X)	90.79 (1.4X)	90.36 (1.66X)	63.91 (1.41X)	92.48 (1.43X)	87.88 (1.51X)	56.34 (2.85X)	78.23 (1.61X)
Size Focus (Compression)	53.57 (13.84X)	83.56 (5.64X)	85.03 (5.56X)	90.91 (5.52X)	90.48 (8.44X)	63.97 (10.48X)	92.42 (8.12X)	87.94 (5.52X)	56.28 (14.08X)	78.24 (8.6X)
DistilBERT										
Baseline	50.25	82.04	84.07	88.72	89.92	61.01	90.48	86.49	56.33	76.59
Accuracy Focus	52.12	82.28	84.64	88.91	90.28	62.14	90.6	86.94	56.82	77.19
Speed Focus (Speedup)	50.06 (1.46X)	81.84 (1.34X)	84.06 (1.34X)	88.83 (1.34X)	89.86 (1.46X)	61.73 (1.44X)	90.25 (1.29X)	86.34 (1.43X)	56.33 (1.54X)	76.57 (1.41X)

TABLE II
[Left] Results on SQUAD v1.1. We report the Exact Match score. The compression is lowest for AlBERT since parameters are shared across layers, and most of the compression is from quantization. **[Right] Results on Penn Treebank.** We report perplexity.

Network	Baseline	Accuracy Focus	Speed Focus (Speedup)	Size Focus (Compression)
AlBERT	80.97	81.32	80.66 (1.42X)	80.71 (1.28X)
XLNet	81.48	81.89	81.26 (1.39X)	81.24 (7.72X)
Q8BERT	80.28	80.99	80.14 (1.27X)	80.11 (6.61X)

Network	Baseline	Speed Focus (Speedup)
BERT-Large	12.04	12.08 (2.47X)
GPT-2	14.21	14.24 (2.08X)

B. *Tuning the Approximation Knobs*

In this work, we considered a tight accuracy constaint of <0.5% accuracy degradation while optimizing the model, and determined the hyperparameter values (PruneThreshold and ApproxThreshold) empirically for that constraint. However, users of different applications and platforms may be willing relax the accuracy constraint for obtaining faster or smaller models. In view of this, we demonstrate the ability of our framework to operate at different points in the speed-size-accuracy tradeoff curve (Fig. 3) through different values of hyperparameters. We note that directly using optimized pre-trained Transformers for inference works best when there is a need for significant speed/size improvement with negligible loss in accuracy (<2%), or if there is a need for more accurate models. When significant degradation in accuracy (>3%) is acceptable, techniques that distil knowledge into simpler networks that no longer maintain the structure of Transformers may be more beneficial. Even in these situations, our techniques are still useful, since they serve as better teachers/baselines during distillation/architecture search.

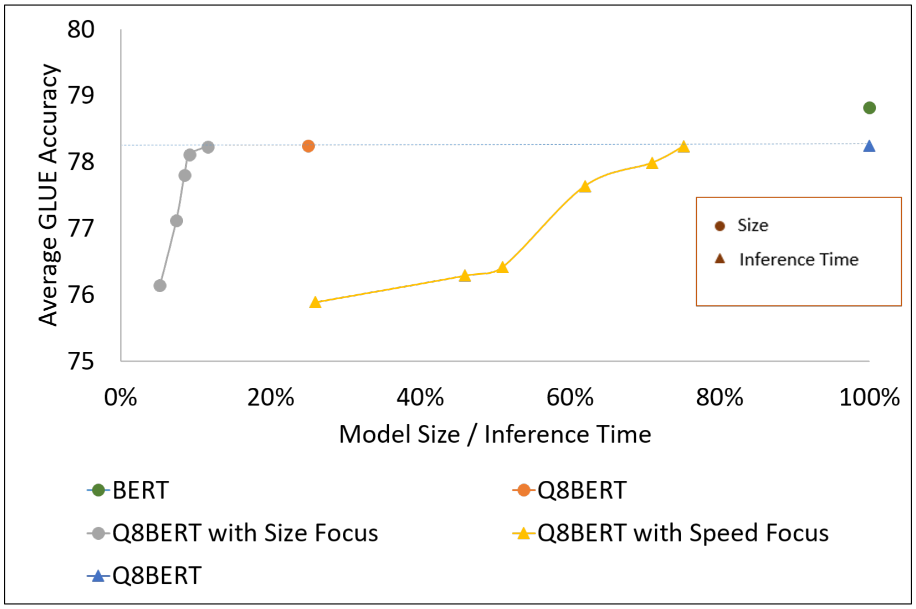


Fig. 3. **Tuning the SA Approximation Knobs with Size and Speed Focus.** The average GLUE scores across the 9 tasks using Q8BERT are reported for different acceptable accuracy loss levels.

C. *Comparison to layer dropping strategies*

Poor Man's BERT ([19]) evaluated several layer dropping strategies that do not require pre-training, and found top-layer dropping to produce least accuracy degradation across tasks. Compared to top-layer dropping, our method produces models that are 1.98×, 3.66× and 1.21× faster and 1.48×, 8.87× and 1.14× smaller on average across 9 GLUE tasks using XLNET, Q8BERT and DistilBERT, respectively while being iso-accurate. The largest benefits are observed on Q8BERT, where the use of quantization greatly reduces the resilience of the network, making it unsuitable for drastic changes such as dropping entire layers. However, by approaching the problem of improving inference efficiency in a hierarchical manner and with finer granularity, we are able to exploit redundancies in the model missed by a layer-only strategy, achieving greater benefits without significant loss in accuracy. In fact, in our experiments, we observe that starting on the layers as a first step leads to worse models than starting with blocks. We find that the effect of removing an ATTN block of relatively high significance may be masked by removing the FFN block of very low significance in the same layer (and vice-versa), leading to low significance for the entire layer. This has consequences further along in the process, since removing a high-significance block greatly reduces further opportunities for pruning and approximating the model.

Layerdrop ([4]) incorporates layer dropping in the pre-training process so that layers can be removed at inference time without additional fine-tuning. We experiment on RoBERTA ([11]) pre-trained with Layerdrop using the fairseq ([15]) model with a layer drop rate of 0.2. We find that on MNLI and QNLI, even after dropping the 6 layers suggested by Layerdrop, we can apply our framework, optimizing for speed, to the remaining layers to get 1.21× and 1.16× additional speedups with only 0.07% and 0.08% further performance degradation, respectively. Similarly, we can instead optimize for the size of the model, achieving 1.44× and 1.42× additional reduction in model size with only 0.06% and 0.09% further performance degradation, respectively. Also, by using SA after pruning 6 layers to approximate the model further, we are able to produce models that are (1.21×, 1.19×) faster and (1.34×, 1.31×) smaller while being iso-accurate to directly pruning the 9 layers suggested by LayerDrop on MNLI and QNLI, respectively.

D. *Analysis of our optimization framework*

We explore the implications of our optimization framework (tailored specifically for Transformers), and compare it to other techniques commonly used in literature (Fig. 4) for pruning and approximating models. ”Oracle” ([14]) refers to computing the significance of each parameter in the model (measured by the effect on training loss when that parameter is removed), and removing those parameters whose significance is less than the baseline loss. Taylor Expansion ([14]) follows a similar method, except that the significance of each parameter is estimated through first-order Taylor Series expansion after a single training iteration, rather than removing them one-by-one, training the model, and computing the effect on training loss. Hence, Taylor Expansion can be considered to be a fast and approximate version of ”Oracle”, and we remove parameters whose effect on training loss is negative when they are removed. We find that these techniques do not work well for Transformers. This is because they are normally used for iterative pruning of CNNs and RNNs, where after pruning a few parameters with least significance, the network is re-trained (or fine-tuned) for a few epochs to regain the lost accuracy, and this process is repeated till the target model size or accuracy is reached. However, this is not possible in Transformers because they are highly overparameterized, and very quickly overfit on the downstream tasks (within 5 epochs). Without this iterative pruning mechanism, these techniques cause severe accuracy degradation. We also find that our method (an iterative greedy algorithm with Transformer-specific heuristics) gives comparable performance to iterative algorithm without heuristics, while being significantly faster.

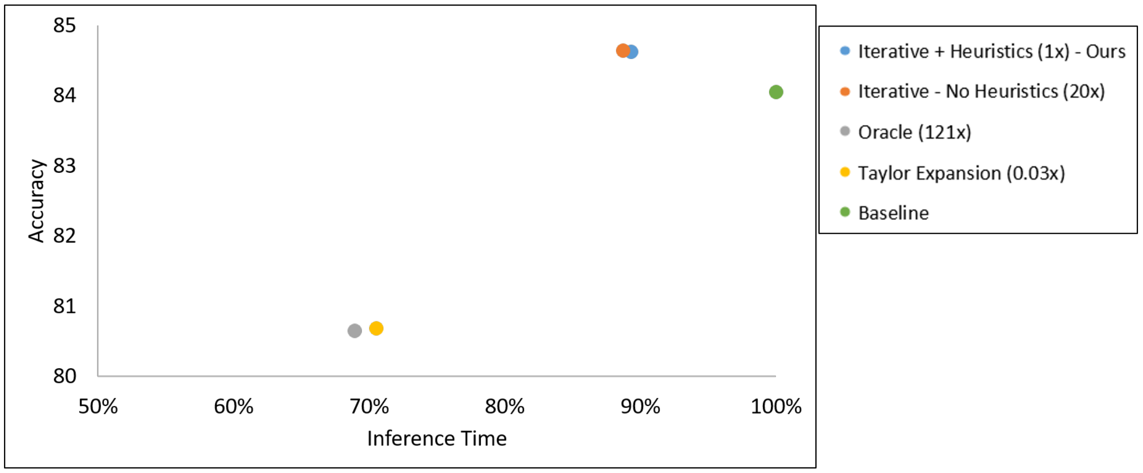


Fig. 4. **Comparison of our framework with commonly used techniques.** We choose to experiment on MRPC using DistilBERT under Accuracy Focus (to provide a fair comparison using only pruning of TransElements and not approximations), since Oracle and Greedy without heuristics are computationally prohibitive for the larger networks and tasks, and MRPC is the most stable (least variation in accuracy across random seeds) among the smaller tasks. The numbers in brackets indicate the (relative) time taken for optimizing the Transormers using each of these methods.

E. *Analysis of blocks pruned and approximated*

We study which blocks are pruned and approximated (Fig. 5). We find that different networks have different distributions of blocks that are more significant. For example, in DistilBERT, the blocks in the top layers (5 and 6) are the least significant across tasks. However, in GPT-2, we find that ATTN Blocks in the top layers and FFN Blocks in the bottom layers are least significant. And in Q8BERT, we find that all ATTN Blocks have high importance, and they can only be approximated and never pruned. In addition, we also find that the distribution of blocks varies across tasks. On GLUE, we find that blocks in top layers are least significant. On SQUAD, we find that blocks in middle and top layers are least significant, and in Language Modelling, there are no clear trends and differs from one network to another. These trends provide search hints to help reduce overheads of SA at fine-tuning time through additional heuristics. Also, at a high level, these trends agree with previous works on understanding how Transformers process language ([6]). For example, on GLUE tasks, we expect that long-range dependency information is not required for most tasks, since most of these tasks depend on local contexts. This is confirmed by the fact that blocks in layers are more likely to be pruned/approximated than earlier layers using our framework. Similarly, we expect that this is not the case for Language Modelling, since long-range dependency information is vital to fully understand the text. This is also confirmed by the observed trends using our framework. We leave it to future work to use this framework to gain deeper understanding of the working of Transformers, especially at finer levels of granularity.

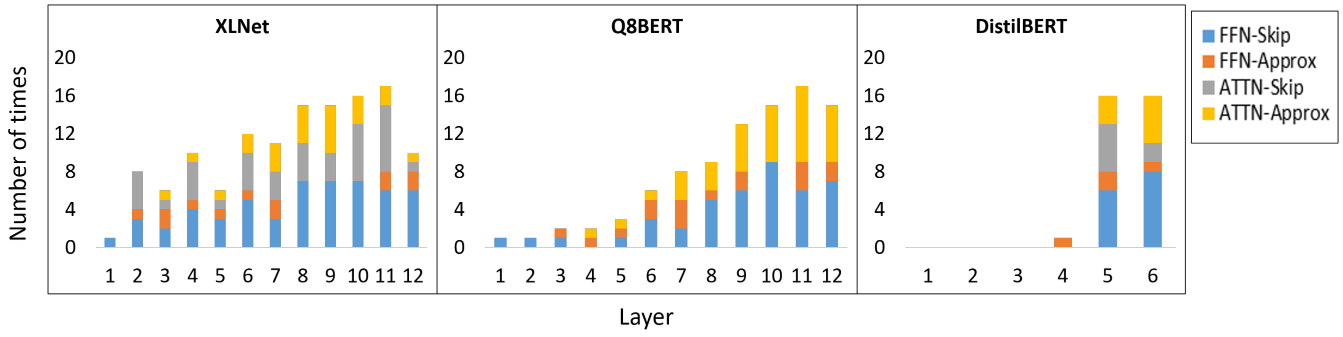


Fig. 5. **Distribution of blocks skipped and approximated.** These numbers are from the 9 GLUE tasks and SQUAD for each of these networks with Speed Focus. We observe that in general, initial layers have higher significance for GLUE tasks, and later layers are more likely to be pruned/approximated.

VI. Conclusion and Future Work

We proposed an approximate computing framework to optimize pre-trained Transformers. The framework identifies elements that are insignificant for the downstream task at hand, and applies techniques to approximate these elements. We demonstrated that this framework can be adapted to produce models that are faster, smaller or more accurate, depending on the user's constraints. Using this framework, we produced models that were up to 4.22× faster, up to 14.08× smaller (with less than 0.5% relative accuracy degradation) and up to 5.46% absolute points more accurate with simultaneous speed and size benefits. A possible direction for future research is to use our framework to gain linguistic insights by studying why specific elements of the pre-trained Transformer are more and less important for a downstream task, and how these relate to the pre-training and fine-tuning processes.

VII. Acknowledgments

This work was supported by C-BRIC, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program, sponsored by DARPA.

References

[1] Tom B. Brown, Benjamin Mann, and Nick Ryder et al. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.
[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.
[3] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.
[4] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.
[5] Saurabh Goyal, Anamitra Roy Choudhury, Venkatesan T. Chakaravarthy, Saurabh ManishRaje, Yogish Sabharwal, and Ashish Verma. Power-bert: Accelerating BERT inference for classification tasks. *CoRR*, abs/2001.08950, 2020.
[6] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. What does BERT learn about the structure of language? In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3651–3657. Association for Computational Linguistics, 2019.
[7] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling BERT for natural language understanding. *CoRR*, abs/1909.10351, 2019.
[8] Ashish Khetan and Zohar S. Karnin. schubert: Optimizing elements of BERT. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2807–2818. Association for Computational Linguistics, 2020.

[9] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[10] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[11] 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. *CoRR*, abs/1907.11692, 2019.

[12] Mitchell P. Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: Annotating predicate argument structure. In *Human Language Technology, Proceedings of a Workshop held at Plainsboro, New Jerey, USA, March 8-11, 1994*. Morgan Kaufmann, 1994.

[13] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 14014–14024, 2019.

[14] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[15] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[16] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[17] 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. *CoRR*, abs/1910.10683, 2019.

[18] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. In Jian Su, Xavier Carreras, and Kevin Duh, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2383–2392. The Association for Computational Linguistics, 2016.

[19] Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. Poor man's bert: Smaller and faster transformer models, 2020.

[20] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[21] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Q-BERT: hessian based ultra low precision quantization of BERT. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8815–8821. AAAI Press, 2020.

[22] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019.

[23] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic BERT for resource-limited devices. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2158–2170. Association for Computational Linguistics, 2020.

[24] 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. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[25] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019.

[26] Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. Lite transformer with long-short range attention. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[27] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating BERT inference. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2246–2251. Association for Computational Linguistics, 2020.

[28] Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. Bert-of-theseus: Compressing BERT by progressive module replacing. *CoRR*, abs/2002.02925, 2020.

[29] Zihao Ye, Qipeng Guo, Quan Gan, Xipeng Qiu, and Zheng Zhang. Bp-transformer: Modelling long-range context via binary partitioning. *CoRR*, abs/1911.04070, 2019.

[30] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8BERT: quantized 8bit BERT. *CoRR*, abs/1910.06188, 2019.

Appendix A
Sign Matching

A. Algorithm

Algorithm 1: Sign Matching

```
Input: Set of query vectors Query = [$q_1, q_2, ..., q_n$], set of key vectors Key = [$k_1, k_2, ..., k_n$], set of value vectors
       Value = [$v_1, v_2, ..., v_n$], number of key vectors to select K
Output: Set of key vectors with highest sign match with query vectors Key' = [$k'_1, k'_2, ..., k'_K$] , set of corresponding
        value vectors Value' = [$v'_1, v'_2, ..., v'_K$]
/* Build sign representation of Query vectors */
$i \leftarrow 1$
$count \leftarrow 0$
while $i <= d$ do
    $j \leftarrow 1$
    while $j <= n$ do
        if $q_{j,i} > 0$ then
            $count[i] \leftarrow count[i] + 1$
        $j \leftarrow j + 1$
    if $count[i] >= (n/2)$ then
        $val[i] \leftarrow 1$
    else
        $val[i] \leftarrow -1$
    $i \leftarrow i + 1$
/* Compute sign matches of Key vectors with the representative Query vector */
$i \leftarrow 1$
$matches \leftarrow 0$
while $i <= n$ do
    $H\_Dist[i] \leftarrow Hamming\_Distance(sign(k_i), val)$
    $i \leftarrow i + 1$
$matches \leftarrow indices(sort\_ascending(H\_Dist))$
$matches \leftarrow matches[1 : K]$
$K' \leftarrow gather(K(matches))$
$V' \leftarrow gather(V(matches))$
```

B. Sign Matching in Auto-Regressive Models

In Auto-regressive models (XL-Net, GPT-2, Transformer-XL, etc.), tokens are not allowed to attend to tokens in the future, and an attention mask is applied to set the corresponding weights to a large negative value. This is a problem because the key vectors selected by SM may be such that vectors at the start of the sequence (first few query vectors) may not be able to attend to any of the key vectors (i.e., their attention outputs will be 0), leading to significant loss of information and degradation in accuracy. We avoid this by selecting the top-scoring $(K/4)$ vectors from the top quarter of the key matrix, and the top-scoring $(3K/4)$ vectors from the overall key matrix and not included in the $(K/4)$ vectors initially selected, instead of directly selecting top K vectors from overall key matrix. This reduces the probability of vectors having no vectors from their past to attend to.

Appendix B
Experiment details and Hyperparameters

A. *Description of tasks and Transformers used in our experiments*

TABLE III
[Left] Transformer models and [Right] downstream tasks used in our experiments and studies.

Transformer	Layers	Parameters (M)	Auto-Regressive?	Optimizations
BERT-Base	12	110	No	None
BERT-Large	24	340	No	None
AlBERT-Base	12	12	No	• Uses factorized embeddings and cross-layer parameter sharing • Faster and has less parameters than BERT.
DistilBERT-Base	6	66	No	• Uses Knowledge Distillation in the pre-training phase • Has half the number of layers with <3% accuracy Prune compared to BERT.
Q8BERT-Base	12	110	No	• Quantizes BERT to 8-bit integer weights and activations • Uses Fake Quantization and Quantization Aware Training in the fine-tuning phase. • 4x smaller and nearly 4x faster than BERT on optimized int8 kernels with <1% accuracy Prune.
XLNET-Base	12	110	Yes	None
GPT-2 Base	12	117	Yes	None

Task	Dataset	Transformers used	Context Length
General Language Understanding	GLUE	Q8BERT-Base, DistilBERT-Base, XLNET-Base	128
Question Answering	SQUAD v1.1	AlBERT-Base	384
Language Modelling	Penn Treebank	BERT-Large, GPT-2 Base	512(BERT-Large)/1024 (GPT-2)

B. *Hyperparameters used in our experiments*

TABLE IV
Hyperparameters used in our experiments.

Hyperparameter	Accuracy ApproxFocus	Speed ApproxFocus	Size ApproxFocus
Fine-tuning Epochs (baseline)	3	3	3
Fine-tuning Epochs (approx)	3	3	3
Learning rate	2.00E-05	2.00E-05	2.00E-05
Warmup steps	0	0	0
Batch size	32 (fine-tuning) / 8 (inference)	32 (fine-tuning) / 8 (inference)	32 (fine-tuning) / 8 (inference)
W (size of one weight group in SA)	256	256	256
K (number of key elements selected in Sign Matching)	N/A	16, if n<=128 64, if 128<n<1024 128, if n>=1024	N/A
Skip_Threshold (<0.5% accuracy loss)	min_loss	1.005*baseline_loss	1.005*baseline_loss
Approx_Threshold (<0.5% accuracy loss)	1.02*min_loss	1.02*baseline_loss	1.02*baseline_loss

TransElement Loss	Number of bits used to represent the TransElement
<1.005*baseline	0
1.005*baseline_loss - 1.006*baseline_loss	4
1.006*baseline_loss - 1.007*baseline_loss	5
1.007*baseline_loss - 1.008*baseline_loss	6
1.008*baseline_loss - 1.009*baseline_loss	7
>1.009*baseline_loss	8

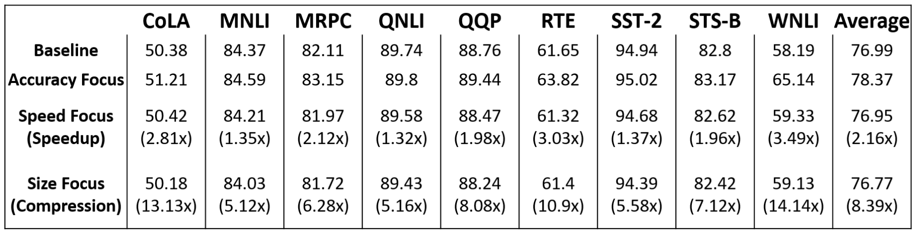
- min_loss refers to the smallest loss seen during the Significance Analysis process. We use this as the threshold with Accuracy ApproxFocus since the goal is to produce a model with the smallest possible loss.
- baseline_loss refers to the loss when the pre-trained transformer is used as-is, without any TransElements skipped or approximated. We use this to compute the threshold with Speed and Accuracy ApproxFocus since small accuracy loss is acceptable.
- When evaluating our techniques using the dev set, we split the training dataset into training and validation (10% of data) sets for the larger datasets (MNLI, QQP, QNLI, SST-2, SQUAD). We skip (approximate) TransElements only when their loss on both sets is less than the skipping (approximation) threshold. This means that for an element to be dropped under Accuracy ApproxFocus, the TransElement loss on the train set must be less than the min loss on the train set ***and*** the TransElement loss on the dev set must be less than the min loss on the dev set. For the smaller datasets, we use the entire training set to train the model, and it is sufficient if the TransElement loss is less than the min loss only on this set, since many of these datasets have very few examples, making it impossible to partition them meaningfully.
- For the larger datasets, to characterize the importance of each TransElement, it is sufficient to compare the loss after 1 epoch of fine-tuning with the relevant TransElements dropped with the baseline loss after 1 epoch. For the smaller datasets, we need to fine-tune for 3 epochs since the datasets are extremely small, making the results unstable.
- When evaluating our techniques on the test set, we use the entire training dataset to train the model, and we use the dev set as the validation set for all datasets, irrespective of their size. We then follow the method described above using training and validation sets.
- Table IV [Right] shows how we quantize insignificant TransElements to lower precisions when operating with Size ApproxFocus.

Appendix C
Additional Experiments

A. *Results on GLUE Test Set*

While our main results are on the dev set following standard practise, we report results on the test set also using the BERT (base) model in Table V. For Size Focus, we start with Q8BERT as the baseline model. In these experiments, we use the entire training set for training and the dev set for validation (see B). We use the GLUE evaluation server to obtain the scores, and make use of code from [28] to prepare the data for submission.

TABLE V
Results on GLUE Test Set. We report Matthews correlation for CoLA, Pearson Correlation for STS-B and accuracy for all other tasks, averaged across 10 random seeds. We report only "matched" accuracy for MNLI.



	CoLA	**MNLI**	**MRPC**	**QNLI**	**QQP**	**RTE**	**SST-2**	**STS-B**	**WNLI**	**Average**
Baseline	50.38	84.37	82.11	89.74	88.76	61.65	94.94	82.8	58.19	76.99
Accuracy Focus	51.21	84.59	83.15	89.8	89.44	63.82	95.02	83.17	65.14	78.37
Speed Focus (Speedup)	50.42 (2.81x)	84.21 (1.35x)	81.97 (2.12x)	89.58 (1.32x)	88.47 (1.98x)	61.32 (3.03x)	94.68 (1.37x)	82.62 (1.96x)	59.33 (3.49x)	76.95 (2.16x)
Size Focus (Compression)	50.18 (13.13x)	84.03 (5.12x)	81.72 (6.28x)	89.43 (5.16x)	88.24 (8.08x)	61.4 (10.9x)	94.39 (5.58x)	82.42 (7.12x)	59.13 (14.14x)	76.77 (8.39x)