Latency Adjustable Transformer Encoder for Language Understanding

Sajjad Kachuee, Mohammad Sharifkhani
Department of Electrical Engineering
Sharif University of Technology

Abstract—Adjusting the latency, power, and accuracy of natural language understanding models is a desirable objective of an efficient architecture. This paper proposes an efficient Transformer architecture that adjusts the inference computational cost adaptively with a desired inference latency speedup. In fine-tuning phase, the proposed method detects less important hidden sequence elements (word-vectors) and eliminates them in each encoder layer using a proposed Attention Context Contribution (ACC) metric. After the fine-tuning phase, with the novel offline-tuning property, the inference latency of the model can be adjusted in a wide range of inference speedup selections without any further training. Extensive experiments reveal that most word-vectors in higher Transformer layers contribute less to subsequent layers, allowing their removal to improve inference latency. Experimental results on various language understanding, text generation, and instruction tuning tasks and benchmarks demonstrate the approach’s effectiveness across diverse datasets, with minimal impact on the input’s global context. The technique improves Time-to-First-Token (TTFT) of Llama3 by up to 2.9x, with minor performance drop. The suggested approach posits that in Large Language Models (LLMs), although the complete network is necessary for training, it can be truncated during the fine-tuning phase.

Index Terms—Word-vector elimination profile, attention context contribution (ACC), offline-tuning, latency adjustable.

I. INTRODUCTION

Recently, Transformer [1] based architectures achieved remarkable success in various Natural Language Processing (NLP) tasks. However, the training difficulty, inference latency, and small size of available datasets are the main concerns with using these models in real applications [2]. Fine-tuning a pre-trained language model on downstream tasks mitigates some of these concerns, including training effort and dataset size. BERT [3], XLNet [4], GPT [5], Llama [6], and Gemma [7] represent a spectrum of pre-trained language models, ranging from earlier developments to the latest advancements. These models vary significantly in size, ranging from hundreds of millions to hundreds of billions of parameters. Hence, the computational complexity bears trillions of Floating Point Operations (FLOPs) for processing a single sample. In some cases, the inference phase computational effort is beyond the ability of resource-limited systems such as edge devices. Therefore, baseline Transformer models cause unacceptable latency and energy usage limitations in such devices.

BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) are two landmark architectures in the development of large language models (LLMs). BERT, introduced by Google, uses a bidirectional training approach, allowing it to capture context from both directions of a sentence simultaneously. This makes it particularly effective for tasks like sentence classification, question answering, and named entity recognition, where understanding relationships between words is crucial [3]. On the other hand, GPT, developed by OpenAI, takes an autoregressive approach, generating text word by word in a unidirectional manner. Starting with GPT-1 and evolving through GPT-4, GPT has proven its strength in creative and coherent text generation, conversational AI, and text summarization, making it a highly flexible model for both content generation and interactive applications [5], [8].

More recent models like Meta’s LLaMA (Large Language Model Meta AI) and Gemma are pushing the boundaries of LLM development. LLaMA is designed to be highly efficient in terms of computational resources, offering a smaller model that still performs well on a wide range of natural language processing tasks. Its streamlined design makes it accessible for research and industrial applications where resources might be limited [6]. Gemma, an emerging model family, is expected to further refine the efficiency and adaptability of LLMs, with an emphasis on real-time performance and scalability. Gemma is particularly well-suited for industry-specific applications, aiming to bridge the gap between general-purpose LLMs and more specialized use cases [7]. Together, these model families reflect the rapid progression and specialization in the field of large language models, each offering distinct strengths based on the needs of users and developers.

In the realm of Deep Neural Networks (DNNs) optimization, diverse methods have been proposed to enhance speed and reduce power consumption. This is crucial for applications such as extending battery life and enabling powerful tasks on small devices. The approaches can be categorized into pruning, removing unnecessary parts; quantization, simplifying numerical usage; Knowledge Distillation, simplifying based on a complex model; parameter-sharing, using the same information across components; tensor decomposition, breaking down data structures; and sub-quadratic complexity transformers [9].

This study introduces an innovative attention layer designed to enhance the efficiency of inference stages in Transformer-based models with an adjustable approach. The suggested solution involves a sorting and elimination process for hidden sequence elements (word-vectors), resulting in a reduction of effective word-vectors in each layer and a decrease in the number of FLOPs. The primary contributions of this research...
can be summarized as follows:

- The novel structure and tuning policy are proposed to speed up the inference times with negligible accuracy degradation.
- A new Attention Context Contribution (ACC) metric is proposed to observe the context contribution in each attention layer. It is shown that the context contribution is decreased at the last encoder layers significantly.
- An accurate analytical inference speedup estimation is presented for the proposed approach that helps a system designer determine the hyper-parameter selection at the fine-tuning phase.
- Offline-tuning property that adjusts the inference latency after the fine-tuning phase. This property controls the trade-off between latency and accuracy in a wide range of inference speedup selections without requiring additional training.
- The proposed approach can be applied to a variety of Transformer-based architectures, while preserving the global context of the input.

II. RELATED WORKS

Pruning-based methodologies have been extensively explored in the existing body of literature, with categorizations based on distinct pruning approaches. In weight pruning methods, the model’s weights are systematically pruned according to a predetermined threshold [10]–[13]. Head and channel pruning methods extend their scope by entirely excising a channel or head of the network [14]–[24]. Layer and block pruning strategies [25]–[27], involve the targeted removal of entire layers or blocks within the network. Token and word-vector pruning methods, adopting a more fine-grained approach, focus on the selective removal of specific tokens or word-vectors within the model [28]–[30]. This classification clarifies the various approaches taken in the pruning paradigm, each playing a unique role in achieving the overall goal of optimizing networks.

A. Weight Pruning

In [31], the magnitude weight pruning method is applied to BERT and shows that the model can be pruned once during pre-training rather than separately for each task without affecting performance. Reweighted Proximal Pruning (RPP) has introduced a pruning method specifically designed for a large-scale language representation model that is much better than typical iterative pruning. This approach yields a sparse version of BERT [32]. Transformer mixed-length vector pruning (TF-MVP) examines the sparsity patterns in pruned Transformers and introduces the TF-MVP architecture using the mixed-length vector pruning (MVP) technique [33]. While TF-MVP achieves a pruning ratio of approximately 70%, there is a notable decline in accuracy. In [34], the model weights are encouraged to be zero with the L0 regularization term.

B. Head/Channel Pruning

A parameter reduction framework was proposed by identifying the most critical heads in each encoder layer using layer-wise relevance propagation and pruning the redundant heads [35]. It showed that the earlier encoder layer’s heads are much more critical than the last. Moreover, in [36], it is shown that around 17% of attention heads can be removed at the test time without significantly impacting inference performance. In [37], a rapid post-training pruning framework tailored for Transformers is introduced. This framework involves three key fine-tuning stages: mask search, mask rearrangement, and mask tuning. Notably, the method obviates the necessity for any retraining, presenting an efficient approach to achieve model compression in the Transformer architecture.

C. Layer/Block Pruning

A progressive layer-dropping at the pre-training phase of BERT architecture offers up to 2.5 times speedup at the inference phase with more knowledge transferability and better generalization on downstream tasks [38]. FastBERT is proposed with adaptive inference time in which the model uses a sufficient number of encoder layers based on the inference certainty of a sample [39]. It dynamically adjusts the number of active layers to reduce the overall computational effort. ELBERT proposed the Confidence-Window Based Early Exit mechanism, improving the ALBERT inference speed [40]. In [41], it is experimentally shown that using all layers of a pre-trained model in downstream tasks is unnecessary. Studying various layer-dropping strategies shows that many model weights are not necessary. LayerDrop introduced a type of structured dropout that imparts regularization benefits during the training process and enables effective pruning at inference time, resulting in the removal of approximately half of the Transformer layers [42].

D. Token/Word-Vector Pruning

PoWER-BERT proposes a word-vector elimination method that eliminates redundant word-vectors [28]. A supplement learnable layer is added between each encoder self-attention and feed-forward layer, and an additional loss function term is used to train these supplement parameters. PoWER-BERT training consists of 3 steps “fine-tuning, elimination configuration search, and re-training” [28]. Funnel-Transformer is inspired by U-net from computer vision [43]. It improves the Transformer architecture by inserting a pooling layer in each self-attention layer [29]. Funnel-Transformer compresses the sequence of hidden states to a shorter one and reduces the computation cost. Learned Token Pruning (LTP) adaptively removes unimportant tokens as an input sequence passes through transformer layers [30]. LTP is applied to RoBERTa [44], which results in up to 2.0 times throughput improvement with less than 1% accuracy drop.

In STTABT [45], the impact of token pruning on later layers’ attentions and on final predictions is studied. The study shows that the proposed attention back-tracking allows the model to better retain the full model’s performance even at high sparsity rates. Conversely, since in the self-attention architecture all input elements contribute to the formation of each attention output, it can be stated that in the word-vector pruning procedure, no independent concept is discarded. In the subsequent layers, every concept retains the ability to expand
and influence the outputs and model predictions. Additionally, STTABT prunes tokens based on a lightweight attention approximation network trained with knowledge distillation, which adds extra training and inference overhead.

In [46], a learnable mechanism is employed to determine which uninformative tokens can be dropped from the context at any point during the generation process. In [47], the model leverages an asymmetric attention mechanism to iteratively distill inputs into a tight latent bottleneck, allowing it to scale to handle very large inputs.

E. Non-Pruning Methods

ALBERT is a lighter version of BERT model [48]. It uses cross-layer weight sharing and a modified embedding layer structure, making the embedding size independent of the encoder’s hidden size. Moreover, the new Sentence Order Prediction (SOP) task is utilized in the pre-training phase instead of the Next Sentence Prediction task. ALBERT reports state-of-the-art results with its parameter reduction and modified pre-training policy. ALBERT is nine times smaller and 1.2 times faster than BERT-base, with less than 2.5% on average accuracy drop. ALBERT is an example of the parameter-sharing technique.

DistilBERT [49], TinyBERT [50] and MobileBERT [51] use the knowledge distillation technique during the pre-training and offer up to 5.5 times faster inference on downstream tasks than BERT-base. In [52], a Self-Distillation (SD) mechanism obtains high-accuracy models directly without going through an assistive model. The main concerns with these techniques are the pre-training effort of the model from scratch, extra training steps, and significant accuracy degradation.

In [53], the authors propose sufficient conditions under which they prove that a sparse attention model can universally approximate any sequence-to-sequence function. They show that sparse Transformers with only \( O_n \) connections per attention layer can approximate the same function class as the dense model with \( n^2 \) connections. Consequently, in recent years, many works [54]–[62] have proposed methods that aim to reduce the complexity of the attention mechanism.

Low-Rank Adaptation (LoRA) involves the freezing of weights in the pre-trained model, and during the fine-tuning phase, it introduces trainable rank decomposition matrices into each layer of the Transformer architecture [63]. LoRA enhances the efficiency of memory usage during the training phase by up to 3 times, but it does not confer any advantages during the inference phase. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes [64]. QLoRA uses an iterative algorithm to decompose each pre-trained matrix into a high-precision low-rank component and a memory-efficient quantized component [65].

The previous works are trying to reduce the model size and inference latency. These works show that significant part computations in the models appear unnecessary, and models with less computational complexity can be obtained from the original ones. The main drawbacks of these methods are pre-training from scratch, multi-phase fine-tuning, and significant inference accuracy drop. In these methods, the inference speedup is set in the fine-tuning phase, and the achieved speedup is fixed in the inference phase. The proposed solution tries to mitigate the previous drawbacks. It presents the powerful offline-tuning mechanism that controls the inference speedup of the model after the fine-tuning phase.

The impact of the speedup methods on the model stability and the input’s global context understanding is an important concern that must be studied in solutions evaluations. Backpack learns multiple non-contextual sense vectors for each word in a vocabulary and represents a word in a sequence as a context-dependent, non-negative linear combination of sense vectors in this sequence [66]. The Backpack Model shows that each word in the input sequence can interpreted as a sense vector that encodes different aspects of a word which is very sensitive. A good practice for assessing the performance of a method is evaluating it in a more complex task, like text generation tasks that inspect the model proficiency. Therefore the suggested method is applied to the several autoregressive models and evaluated on several text generation tasks.

Caching methods focus on optimizing the efficiency and speed of inference in autoregressive large language models (LLMs). KV-Runahead enhances prompt phase parallelization by using multiple processes to manage KV-cache population, thereby reducing the Time-to-First-Token (TTFT) [67]. CacheGen speeds up context loading with a custom tensor encoder that compresses KV-cache into compact bitstream representations, minimizing decoding overhead [68]. CacheBlend introduces a method for reusing pre-computed KV-caches and selectively updating a small subset of tokens, optimizing cache reuse [69]. Prompt Cache accelerates inference by precomputing and storing attention states for frequently occurring text segments, enabling efficient reuse across different prompts [70]. Lastly, CachedAttention employs a hierarchical KV-caching system using cost-effective memory and storage solutions to manage KV-caches for all requests [71]. Together, these methods contribute to faster and more efficient LLM inference by improving cache management and state reuse.

III. PROPOSED METHOD

This section presents details of the proposed method and the analysis and reasoning behind the approach. Section III-A introduces the preliminary background of the Transformer architecture, while Section III-B analyzes the latency and TTFT across various Transformer-based models. The proposed encoder layer is based on an insight into the amount of the contribution of word-vectors at different layers, which is considered in Section III-C. Section III-D explores the proposed architecture in detail. Additionally, Section III-E extends the proposed method to autoregressive architectures and applies it to several newly developed models.

A. Background

Despite the current progress and advancements in mainstream LLM architectures such as GPT, Gemma, Mistral [72], and Llama, these models still follow the foundational Transformer architecture [1]. The primary differences among them
lie in the number of parameters, layers, heads, hidden states, and specific modifications to normalization layers, residual paths, and attention implementations. For clarity and a deeper understanding of the suggested method, we first focus on the BERT [48] and GPT-2 [73] models before extending the approach to other large language models (LLMs).

Fig. 1 shows BERT-base model architecture and output classifier. The input sentence goes through the tokenizer layer and embedding layer. Then the context is fed into 12 cascaded encoder layers to be processed. The pooler and classifier layers are added in the last layer to perform proper output for the desired task. In the figure, $T$ and $H$ are the numbers of input word-vectors and the hidden state vector size, respectively. Each encoder consists of a Bidirectional Multi-Head Self-Attention layer and a Feed-Forward layer with several tensor multiplications. The residual branches and normalization layers increase the training stability.

B. Transformer Architecture Latency and TTFT Analysis

Table I presents an analysis of the latency and TTFT share across the embedding, attention, and feed-forward layers in several Transformer-based models. As shown in the table, regardless of model size, the latency of the embedding layer is minimal, with the majority of latency attributed to the attention and feed-forward layers. Attention share ranges from around 20% to 47% of the total. In neural networks, latency and computational effort are closely related to the number of FLOPs. Consequently, reducing the number of FLOPs in the encoder layers, which account for most of the latency, significantly decreases both total latency and TTFT.

C. Attention Context Contribution (ACC) metric

The context contribution of word-vectors at each self-attention layer is studied in this Section to find more important word-vectors at each encoder layer. For a given encoder layer, the Attention Context Contribution (ACC) metric measures the contribution of word-vectors in the layer’s output. This metric is derived from a Score Vector (SV) to find more important word-vectors in each layer. For a given layer, the SV indicates the contribution of each input word-vector to the output of the attention layer.

Fig. 2 illustrates the calculation of the Score Vector based on the attention probability matrix. Fig. 2 (left) shows the attention probability matrix of BERT-base model. The rows of this matrix are normalized to one. By averaging the attention probability matrix over the heads, the aggregated probability matrix is resulted, Fig. 2 (center). The SV results from the sum of this matrix along the unnormalized dimension as shown in Fig. 2 (right). The SV elements with higher score values correspond to the word-vectors that contribute more to the layer output.

Fig. 3 illustrates BERT-base SV results on the IMDB sentiment analysis task [77] for an input sample. As shown in earlier encoder layers, the SV values approximately have uniform distributions, and in the last encoder layers, the distribution of SV values changes to approximately $\delta$ distribution. The distribution of SV values in the last layers indicates that the concentration of scores is high on a few word-vectors and others have less contribution to the self-attention layer output.

The SV elements are positive and with an average of one. So, the median of SV is bounded. Suppose the median of the SV is lower than its average. In that case, more than half of the elements are lower than the average value, and only a few are higher than the average value. It suggests that most of the word-vectors do not bear enough contribution to the layer output, and a few word-vectors influence the final result of the layer. The median of the Score Vector is interpreted as the proposed Attention Context Contribution metric.

The ACC metric calculation is explained in Algorithm 1. The algorithm inputs are BERT encoder word-vectors and the model parameters. The algorithm output is the ACC metric vector that suggests the contribution of the word-vectors in each encoder self-attention layer. For encoder layer $l$, the first step is to prepare the pre-computed self-attention probability matrix; $M^{l}_{At}$. Then the average of this matrix over its heads is computed; $M^{l}_{H}$. In the next step, the resulting matrix’s summation over its unnormalized dimension is computed; $M^{l}_{SV}$. This matrix is the Score Vector. Attention Context Contribution of the layer ($E^{l}_{ACC}$) is the median of the Score Vector.

Fig. 4 shows the ACC metric for BERT-base encoder layers on IMDB [77] dataset and several General Language Understanding Evaluation (GLUE) [78] benchmark tasks. The fitted curve to the ACC metric results shows that the ACC metric is reduced at later layers gradually. It indicates that the fraction of the word-vectors that contribute more in the last layers is less in those layers; hence more word-vectors can be eliminated. By eliminating less important word-vectors in each
TABLE I
LATENCY AND TIME-TO-FIRST-TOKEN (TTFT) ANALYSIS FOR BERT [3], GPT-2 [73], T5 [74], GEMMA2 [75], MISTRAL [72], AND LLAMA3 [76] MODELS

| Model         | Architecture Type | Parameters | Layers | Attention Heads | Latency/TTFT Share\((a)\) (%) |
|---------------|-------------------|------------|--------|----------------|-------------------------------|
|               |                   |            |        |                | Embedding | Attention | Feed-Forward |
| BERT-base     | Encoder           | 110M       | 12     | 12             | 0.58      | 34.94     | 64.48       |
| GPT-2         | Decoder           | 124M       | 12     | 12             | 0.06      | 34.94     | 65.00       |
| T5-base       | Encoder-Decoder   | 220M       | 12     | 12             | 0.35      | 47.00     | 52.65       |
| T5-large      | Encoder-Decoder   | 770M       | 24     | 16             | 0.06      | 40.46     | 59.47       |
| GPT-2-large   | Decoder           | 774M       | 36     | 20             | 0.01      | 32.25     | 67.74       |
| Gemma2        | Decoder           | 2B         | 26     | 8              | 0.01      | 20.36     | 79.63       |
| Mistral       | Decoder           | 7B         | 32     | 32             | 0.01      | 20.34     | 79.65       |
| Llama3        | Decoder           | 8B         | 32     | 32             | 0.01      | 19.55     | 80.44       |

\(a\) Latency share for BERT model and Time-to-First-Token (TTFT) for other autoregressive models

Algorithm 1 Attention Context Contribution (ACC) metric

**Input:** Encoder inputs \(\in R^{LTTH}\) & fine-tuned BERT

\(\{T = \text{number of input word-vectors}, \ L = \text{number of encoder layers}, \ H = \text{hidden state size}\}\)

**Output:** ACC vector: \(E_{\text{ACC}} \in R^L\)

**Initialisation:**
1. \(E_{\text{ACC}} = [0]\)

**LOOP Process**
2. for encoder \(l\) in Encoders do
3. \(M_{l}^{\text{Att}} \leftarrow \text{Self-Attention Probability Matrix} \in R^{HTT}\)
4. \(M_{l}^{H} \leftarrow \text{Mean of } M_{l}^{\text{Att}} \text{ over heads} \in R^{TT}\)
5. \(M_{l}^{S} \leftarrow \text{Sum of } M_{l}^{H} \text{ over unnormalized dimension} \in R^{L}\)
6. \(E_{l}^{\text{ACC}} \leftarrow \text{Median of } M_{l}^{S} \text{ (Score Vector)} \in R\)
7. Append \(E_{l}^{\text{ACC}}\) to \(E_{\text{ACC}}\)
8. end for
9. return \(E_{\text{ACC}}\)

layer, the processed word-vectors are decreased gradually, and the inference time of the model is decreased.

As shown in Fig. 4, the behavior of the ACC metric strongly correlates with the intricate specifications inherent in each task. The ACC results consistently exhibit a monotonic decrease with the layer number, except in the case of the SST-2 [79] dataset, where unexpected behavior is observed. Notably, the SST-2 dataset, centered around sentiment analysis of movie reviews, shares a strong resemblance to the IMDB dataset. A comparative analysis of the ACC curves for SST-2 and IMDB reveals a notable disparity, primarily associated with the length of input sequences. Specifically, in the IMDB dataset, the majority of samples consist of around 512 tokens; conversely, in the SST-2 dataset, the majority of samples feature fewer than 50 tokens. It can be inferred that the limited number of input tokens leads to an undesired sensitivity to word-vectors, and in such cases, some word-vectors are revived through residual branches.

**D. Proposed Architecture**

According to the discussion in Section III-B, attention and feed-forward layers are major contributors of the total latency. In addition, according to the observation in Section III-C, the word-vector contribution in the encoder decreases at later
ACC behavior on subsequent layers. This adaptive approach guarantees the preservation of a monotonic descending shape of this hyper-parameter \( \alpha \). In (2), the \( \alpha \) word-vectors. In (2), the \( \alpha \) word-vectors that is eliminated in each encoder layer. For layer \( l \), \( \alpha_{ER} \) is the word-vector Elimination Rate obtained from the pre-computed Elimination-Profile (\( \alpha_{EP} \)) vector and a pre-specified Speedup-Coefficient hyper-parameter (\( \alpha_{SC} \)). \( \alpha_{SC} \) adjusts the trade-off between accuracy and speed in fine-tuning and offline-tuning phases.

\[
\alpha_{ER}^l = \alpha_{EP}^l \times \alpha_{SC}
\]

(1)

\( \alpha_{EP} \) is obtained from the fitted curve to the ACC metric (red lines in Fig. 4) at layer \( l \). In some tasks, the ACC values of layers are not smooth. To increase stability and prevent applying stress to the model in fine-tuning phase, in the proposed formulation, instead of ACC metric values \( (E_{ACC}) \), the fitted curve to the ACC metric \( (P_{ACC}) \) is used that presents a more smooth elimination trend. It is important to note that when an increasing behavior of the ACC value occurs in a layer, the elimination process is halted, and the remaining layers proceed using a fixed number of retained word-vectors. In (2), the \( \alpha_{EP} \) lower bound is set to one to ensure a monotonic descending shape of this hyper-parameter curve. This adaptive approach guarantees the preservation of crucial information and mitigates the impact of unexpected ACC behavior on subsequent layers.

\[
\alpha_{EP}^l = \min(1, \frac{P_{ACC}^l}{P_{ACC}^{l-1}}) \quad (2)
\]

The number of remaining word-vectors in each encoder layer is calculated according to the number of the layer input word-vectors and the \( \alpha_{ER} \) hyper-parameter. Equation (3) presents the number of remaining word-vectors \( (T_l) \) at the output of encoder layer \( l \). The minimum number of remaining word-vectors at each encoder output is limited to one.

\[
T_l = \max(1, [\alpha_{ER}^l \times T_{l-1}]) \quad (3)
\]

The effective number of processed word-vectors can be estimated using the number of word-vectors before and after the eliminate layer. Based on Table I in BERT-base, around 65% of latency occur after the self-attention layer. Equation (4) presents the effective number of Processed word-vectors \( (PW_l) \) in layer \( l \).

\[
P_W \approx \frac{7T_{l-1} + 13T_l}{20} \quad (4)
\]

Hence,

\[
P_W \approx T \left[ \frac{7}{20} + \sum_{i=1}^{L-1} \prod_{j=1}^{i} \alpha_{EP}^j + \frac{13}{20} \prod_{j=1}^{L} \alpha_{EP}^j \right] \quad (5)
\]

Processed word-vectors in the original BERT-base is:

\[
P_W^{BERT-base} = T \times L \quad (6)
\]

In these equations, \( L \) is the number of BERT encoder layers, and \( T \) is the number of encoder input word-vectors. Since almost all inference latency occurs in the encoder, the overall model computation effort is almost equal to that of the encoder stage. Equation (7) presents the proposed method inference time speedup to BERT-base.

\[
K_{speedup} \approx \frac{20L}{7 + 20 \sum_{i=1}^{L-1} \prod_{j=1}^{i} \alpha_{EP}^j + 13 \prod_{j=1}^{L} \alpha_{EP}^j} \quad (7)
\]

The above equation assumes that the number of remaining word-vectors in each encoder layer is more than a single word-vector. In other words, (3) returns the second argument, which is valid for most cases. For example, if \( L \) is 12 and \( \alpha_{EP} \) is constant and equal to 0.8 for all layers, the final inference speed is around 2.99 times faster than BERT-base.

The proposed Sort and Elimination layers are included in the fine-tuning phase. Given that the eliminated word-vectors constitute a relatively small portion within each layer during the fine-tuning process, we can assume that the gradients effectively propagate through the remaining word vectors. Additionally, the residual paths stay functional, contributing to the updates in the initial layers of the network. This design ensures a robust flow of information throughout the fine-tuning process.
Fig. 4. BERT-base [3] Encoder Attention Context Contribution (ACC) metric results (blue bars) and second degree fitted curves (red lines) on SST-2 [79], MRPC [80], QQP [81], QNLI [82], RTE [83], MNLI [84], STS-B [85] and IMDB [77] tasks.

Fig. 5. The proposed self-attention layer

To enhance the speed of the suggested self-attention layer, the Sort and Eliminate layers can be positioned after the self-attention concatenation layer. This adjustment results in a reduction of per-head calculations, and for a given layer, the eliminated word-vectors play a role in shaping the layer’s output. Furthermore, in accordance with the suggested word-vector elimination strategy, the pruned word-vectors efficiently accelerate the inference phase, leading to an anticipated close alignment between the reduction in Floating Point Operations (FLOPs) and the speedup in inference.

E. Extending Proposed Architecture to Autoregressive Models

The suggested solution can also be applied to decoder architectures, including GPT, Gemma, Mistral, and Llama Large Language Models (LLMs). This section outlines the key considerations for implementing the method on the GPT-2 model, as a representative example of autoregressive models.

The foundational GPT-2 model comprises 12 decoder layers, with each decoder featuring a causal attention layer equipped with 12 heads. The total number of parameters in the GPT-2 model is approximately 124 million. A key distinction between the architectures of GPT-2 and the BERT-base model lies in the design of their attention layers. As
explained in Section III-A, the attention layers of BERT-base encoders are bidirectional, involving all input word-vectors in generating each output word-vector. Conversely, the GPT-2 model, tailored for text generation tasks, employs causal attention layers, where only the preceding word-vectors contribute to the attention layer output at each time stamp.

In a causal attention layer, the attention probability matrix, represented as $M_{att}$, adopts the form of a triangular matrix. To extend the proposed architecture outlined in Section III-D to causal attention layers, an additional normalization step needs to be included after the fifth step in Algorithm 1. This supplementary step normalizes the Score Vector $M^l_{S_i}$ based on the count of non-zero elements in the probability matrix $M^l_{att}$. Equation (8) articulates this normalization process in layer $l$, wherein the $i$-th element of the Score Vector is adjusted relative to the number of non-zero elements in the $i$-th column of the probability matrix. Through this normalization step, the resulting score values become comparable.

$$M^l_{S_i} = M^l_{S_i} \div N^l_{M_{att}} \quad (8)$$

IV. EXPERIMENTS

In this section, we implement and evaluate the suggested method on BERT-base, GPT-2, Flan-T5 [80], Gemma2 [75], Mistral, and Llama3 [76], exploring its effectiveness across various natural language understanding tasks, text generation datasets, and instruction-tuning benchmarks. Toward the conclusion of this section, we examine the offline-tuning property and conduct an ablation study on the proposed attention layer.

A. Datasets

The performance of the proposed method on natural language understanding tasks is evaluated using the IMDB dataset [77], which focuses on sentiment analysis, as well as the General Language Understanding Evaluation (GLUE) benchmark [78]. The GLUE benchmark includes two single-sentence classification tasks: SST-2 [79] and CoLA [87], two multi-sentence similarity classification tasks: MRPC [80] and QQP [81]; one binary question answering task: QNLI [82]; three natural language inference tasks: RTE [83], MNLI-M, and MNLI-MM [84]; and one multi-sentence similarity regression task: STS-B [85].

To assess the performance of the proposed approach on text generation tasks, we conduct evaluations on several datasets. The WikiText-103 [88] language modeling dataset, comprising over 100 million tokens extracted from verified Good and Featured articles on Wikipedia, serves as a primary evaluation ground. Additionally, the PTB [89] dataset, featuring a million words from 1989 Wall Street Journal material, and the 1BW [90] dataset, established as a benchmark corpus for measuring progress in statistical language modeling, are included in our evaluation. Moreover, we assess the approach on The LAMBADA [91] dataset, a collection of narrative passages characterized by the intriguing feature that human subjects can predict their last word when exposed to the entire passage. Extracted from BookCorpus, The LAMBADA dataset comprises 10,022 passages, contributing to a comprehensive evaluation of the proposed method’s performance in decoder models.

To further demonstrate the suggested approach’s performance in language understanding, it is evaluated on instruction-tuning tasks and assessed using the Massive Multitask Language Understanding (MMLU) benchmark. This benchmark includes exam questions from 57 diverse tasks, covering subjects such as mathematics, history, law, and medicine [92].

B. Models Settings

In the assessment of the method on BERT-base, the encoder self-attention layer undergoes substitution with the proposed encoder self-attention layer. The hyper-parameters are the same as those in BERT-base with extra $\alpha_{EP}$, and $\alpha_{ER}$ hyper-parameters are added to set the word-vector elimination rate in each encoder layer. The hyper-parameters are: learning rate - [1.5 x 10^{-5}, 4.5 x 10^{-5}]; batch size - [4, 8, 16, 32, 64, 128]; number of epochs - [2, 3, 4, 5]; Dropout - [0.1, 0.15], $\alpha_{ER}$ - [0.77, 0.97], and $\alpha_{EP}$ vector that is obtained from (2). The cross-entropy loss function is used for classification tasks and the Mean Square Error (MSE) loss function for regression tasks. The linear learning rate scheduler is used for all tasks. Initial linear learning warm-up is used for some tasks. In all tasks, AdamW optimizer with $\epsilon = 1 \times 10^{-8}$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ is used. The model input sequence length is delicately fixed so that less than 1% of samples are truncated. The selected input sentence length for each dataset is specified in Table II.

In the case of the GPT-2 model, the attention layer within each decoder layer undergoes modification with the proposed attention layer, which is described in Section III-E. Introducing the hyper-parameters $\alpha_{EP}$ and $\alpha_{ER}$ becomes essential to determine the word-vector elimination rate in each encoder layer. The specified hyper-parameters for the evaluation are as follows: a learning rate of $1 \times 10^{-6}$, a batch size of 8, a three-epoch training duration, a dropout rate of 0.1, $\alpha_{ER}$ values ranging between 0.9 and 1.2, and an $\alpha_{EP}$ vector derived from equation (2). These parameters collectively define the experimental setup for a comprehensive evaluation of the method’s impact on the GPT-2 model.

For the instruction-tuning evaluation, the Flan-T5 [80] Gemma2, Mistral, and Llama3 models are targeted. In these models, the self-attention layers are replaced with the proposed self-attention layer, and other layers are adjusted to accommodate a dynamic number of word vectors in each encoder and decoder layer. The models use their default weights without any tuning or fine-tuning. The parameter $\alpha_{ER}$ is selected within the range [0.8, 1.0], and all other evaluation hyperparameters remain consistent with those of the default models.

C. Implementation Details

The initial models utilized for this study are the pre-trained versions of BERT-base, GPT-2, Flan-T5, Gemma2, Mistral, Llama3, and Flan-T5 is an enhanced version of T5 [74]
and Llama3 from the Hugging Face implementations. The implementation is carried out in the PyTorch library, utilizing an Nvidia Tesla P100 GPU for BERT-base, an Nvidia Tesla T4 GPU for GPT-2, and an Nvidia Tesla A100 GPU for Flan-T5, Gemma2, Mistral, and Llama. The datasets are sourced from the Hugging Face Datasets library, and the GLUE results are validated through the official GLUE service. The instruction tuning assessment is done with help of the INSTRUCTEVAL method. This framework ensures consistency and reliability in the assessment of the proposed solution’s performance on the specified models.

D. Experimental Results on Language Understanding Tasks

The experimental results of the proposed method applied to BERT-base for a wide range of tasks are presented in Table II. The proposed approach is 2.9 times faster than BERT-base, with only a 0.7% accuracy drop on the test set. Additionally, the model’s inference time is up to 4.8 times faster than BERT-base, with just a 0.9% reduction in accuracy. These results demonstrate that the suggested method significantly enhances the performance and resource efficiency of Transformer-based models while maintaining minimal accuracy loss.

The table shows the experimental speedup, and the expected speedup predicted using \((7)\). The slight differences are due to the computational overheads, including the computational effort of the tokenizer and output classifier layers that are not considered in \((7)\). The consistency between the experimental speedup, the expected speedup, approves the hypothesis discussed in Section III-D.

Table II compares the proposed approach and the previously reported speedup methods. BERT-base model \((L=12, h=12, d=768)\) is used as the basis for comparison. The test results, the average accuracy drop compared to BERT-base, the average number of FLOPs, and the average inference speedup are reported for each model. As shown in the table, the proposed approach achieves 2.9 times speedup with 0.73% on average accuracy drops compared to BERT-base model.

Compared to other works, DistilBERT model achieves 1.7 times speedup with around 2.74% accuracy degradation on average. In DistilBERT, a pre-trained general-purpose language presentation model must be trained from scratch with significant effort. The TinyBERT model speedup is 2.0 times with 0.4% accuracy drops on average. TinyBERT uses a two-stage learning framework that performs Transformer distillation at pre-training and fine-tuning phases with considerable extra training effort. The MobileBERT model speedup is 5.5 times with around 1.19% accuracy drop, but the starting point of this model is BERT\(_{\text{large}}\). The PoWErBERT model speedup is 2.9 times with around 0.92% accuracy drop. However, the training of PoWErBERT consists of 3 phases: "fine-tuning, elimination configuration search, and retraining" \((8)\) that impose extra training effort. LayerDrop and FPT speedups are 1.7 and 1.5 times, respectively, with corresponding average accuracy drops of 1.47% and 0.93%.

The table shows that the results of the suggested technique are competitive and more effective than previous baseline methods across a wide range of experimental tasks. Furthermore, compared to recently reported methods, our approach is applied only during the fine-tuning phase and offers the offline-tuning property. Blank fields in the table, indicate instances where some works did not report their assessments on certain datasets.

To assess the impact of the proposed method on GPT-2 model, the training process involves utilizing 1% of the OpenWebText \((94)\) dataset with a stride of 128 for three epochs. Table IV exhibits the classification accuracy findings of the proposed method in contrast to GPT-2 and DistilGPT2. Notably, the proposed method attains an inference latency of 3.7x and 2.4x, respectively, with an average accuracy drop of 1.47% and 0.93%.

The table shows that the results of the suggested technique are competitive and more effective than previous baseline methods across a wide range of experimental tasks. Furthermore, compared to recently reported methods, our approach is applied only during the fine-tuning phase and offers the offline-tuning property. Blank fields in the table, indicate instances where some works did not report their assessments on certain datasets.

E. Experimental Results on Text Generation Tasks

\(^2\)https://huggingface.co/

\(^3\)https://pytorch.org/

\(^4\)https://gluebenchmark.com/
shown, the suggested method achieves up to a 3-fold reduction in the MMLU benchmark. Table VI presents the experimental results for the proposed architecture enabling efficient adaptation with just a few small epochs. Furthermore, the proposed technique exhibits superior performance when compared to the DistilGPT2 and ZipGPT2 models in the evaluation.

Table V presents the results of the zero-shot perplexity comparison between GPT-2 and DistilGPT2. This Work Pruning achieves an accuracy of 53.4 on SST-2 with a speedup of 2.9x compared to GPT-2. Table IV presents the accuracy comparison between GPT-2 and DistilGPT2 with latency-adjustable GPT-2 (This Work).

Table V presents the perplexity comparison between GPT-2 and DistilGPT2. This Work Pruning achieves a perplexity of 49.3 on Wiki Text-103 with a speedup of 3.5x compared to GPT-2. Table III presents the comparison of results between BERT-base, DistilBERT, TinyBERT, MobileBERT, PoWER-BERT, LayerDrop, FPT, and latency-adjustable BERT-base (This Work).

### F. Experimental Results on Instruction Tuning Tasks

For the instruction-tuning evaluation, the Flan-T5, Gemma2, Mistral, and Llama3 models are selected. In these models, the self-attention layers are replaced with the proposed encoder self-attention layer, and the default weights are used without any additional tuning or fine-tuning. The method’s performance is evaluated using various prompting types on the MMLU benchmark. Table VI presents the experimental results for 5-shot direct prompting on the MMLU benchmark. As shown, the suggested method achieves up to a 3-fold reduction in TTFT with only a marginal decrease in score.

#### G. Offline-Tuning Property

After the fine-tuning phase, the trade-off between inference accuracy and speed of the model can be adjusted with the offline-tuning property. This property controls each encoder layer’s word-vector elimination rate by adjusting the Speedup-Coefficient hyper-parameter ($\alpha_{SC}$) value. Fig. 6 presents the offline-tuning property results on BERT-base. In the following experiments, the model first fine-tuned with $\alpha_{SC}$ equal to one for a given task, and then in the offline-tuning phase, the $\alpha_{SC}$ value changes from 0.85 to 1.2 to observe its effect on inference speedup and accuracy.

Fig. 6a shows the model inference speedup for a range of $\alpha_{SC}$ hyper-parameter values on multiple tasks. Fig. 6b shows the model inference speedup increased more than three times compared to BERT-base, with the $\alpha_{SC}$ value decreasing to 0.85. Fig. 6c presents the accuracy stability on multiple tasks. The model inference results are almost unchanged even if the complexity of the model reduces by 70% according to the proposed method. In these experiments, the model’s accuracy on validation sets is
Fig. 6. The proposed offline-tuning results, (a) Inference speedup compare to BERT-base versus \( \alpha_{SC} \), (b) Validation set results versus speedup compare to BERT-base for MRPC \([90]\), SST-2 \([79]\), QNLI \([82]\), IMDB \([77]\) and MNLI-M/MM \([81]\) tasks.

**TABLE VI**
TIME-TO-FIRST-TOKEN (TTFT) SPEEDUP RESULTS OF THE PROPOSED APPROACH ON FLAN-T5 \([86]\), GEMMA2 \([75]\), MISTRAL \([72]\), AND LLAMA3 \([76]\) EVALUATED WITH THE MMLU \([92]\), INSTRUCTION TUNING DATASET.

| Architecture Name | Model     | Model Size | MMLU Score \(^{a}\) | TTFT Speedup \(^{b}\) |
|-------------------|-----------|------------|----------------------|-----------------------|
| T5                | Flan-T5-base | 220M       | 34.05                | 1.00x                  |
|                   | Flan-T5-large | 770M       | 41.95                | 1.00x                  |
| Gemma             | Gemma2    | 2B         | 56.74                | 1.00x                  |
|                   | Mistral   | 7B         | 41.85                | 1.00x                  |
|                   | Llama3    | 8B         | 65.69                | 1.00x                  |

\(^{a}\) MMLU includes exam questions from 57 tasks, uses 5-shot direct prompting, and measure the exact-match score.

\(^{b}\) The proposed attention layer is applied to the models, and the Time-to-First-Token (TTFT) speedup is reported.

reported because of the GLUE test set verification limitations.

Two approaches can be offered to select the proper value of the \( \alpha_{SC} \) hyper-parameter in the fine-tuning phase. The first approach is using equation \((7)\) and calculating the approximate value of the \( \alpha_{SC} \) based on the desired inference speedup. The second approach is to select the \( \alpha_{SC} \) value using suggested Fig.\((6)\) (c) curves. After the fine-tuning phase, the speedup can be accurately adjusted with the offline-tuning property.

The offline perplexity (PPL) stability of the proposed method for text generation tasks is evaluated by training the model with a specific speedup coefficient and then modifying this value during inference, while measuring the model’s zero-shot perplexity. Fig. \(7\) illustrates the method’s offline perplexity results on GPT-2 across various datasets, including WikiText-103, LAMBADA, PTB, and 1BW. The graph demonstrates the model’s robust stability, showing a smooth perplexity curve across a wide range of speedup values, which highlights the solution’s effectiveness in maintaining consistent performance under varying conditions.

**H. Ablation Studies**

Table \([VII]\) presents the ablation study on the proposed self-attention layer. The first row of the table presents the original BERT-base results, and the subsequent rows are the results of the self-attention layer with different types of word-vector sorting strategies.

The first approach involves removing the concluding word-vectors from the contextual sequence without the use of a sorting layer. The second approach employs the Random-Sort-Layer, which shuffles word-vectors and randomly eliminates them. The final and better strategy is the proposed strategy that incorporates the SV-Sort-Layer that arranges word-vectors according to their Score values, subsequently removing the concluding word-vectors.
As shown in the table, the first strategy shows a significant accuracy drop compared to the original BERT-base and the proposed method. The second strategy has a destructive effect on the model results, and the fine-tuning process becomes completely unstable in CoLA, QQP, QNLI, and RTE datasets not presented in the table. The third strategy is superior to other investigated strategies and proves that the proposed self-attention Sort layer is effective. In these experiments, the hyper-parameters values and the fine-tuning conditions are the same.

V. CONCLUSION

This paper presents a latency-adjustable Transformer architecture applicable to a wide range of models, with a marginal decrease in accuracy. We introduce a novel Attention Context Contribution (ACC) metric to evaluate the context contribution of each Transformer encoder layer, demonstrating experimentally that word-vector contributions diminish in the last encoder layers. Extensive experiments on language understanding, text generation, and instruction-tuning tasks utilizing experimentally that word-vector contributions diminish in the last encoder layers. Extensive experiments on language understanding, text generation, and instruction-tuning tasks validate the suggested technique’s effectiveness across diverse datasets, with minimal impact on the input’s global context. The paper also provides an accurate analytical estimation of speedup, assisting system designers with hyper-parameter selection. Additionally, the method’s powerful offline-tuning capability allows for model speedup adjustments after the fine-tuning phase. The proposed approach highlights that, while the complete network is essential for training, it can be truncated during the fine-tuning phase in Large Language Models (LLMs).

ACKNOWLEDGMENT

Thanks to Mr. Seyed Mahdi Hosseini for his assistance and comments that greatly improved our work.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[2] E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for modern deep learning research,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 09, 2020, pp. 13693–13696.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[4] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, “Xlnet: Generalized autoregressive pretraining for language understanding,” *Advances in neural information processing systems*, vol. 32, 2019.

[5] T. B. Brown, “Language models are few-shot learners,” *arXiv preprint arXiv:2005.14165*, 2020.

[6] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar et al., “Llama: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.

[7] G. Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Riviè re, M. S. Kale, J. Love et al., “Gemma: Open models based on gemini research and technology,” *arXiv preprint arXiv:2403.08295*, 2024.

[8] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever et al., “Improving language understanding by generative pre-training,” 2018.

[9] M. Gupta and P. Agrawal, “Compression of deep learning models for text: A survey,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 16, no. 4, pp. 1–55, 2022.

[10] S. Wiedemann, K.-R. Müller, and W. Samek, “Compact and computationally efficient representation of deep neural networks,” *IEEE transactions on neural networks and learning systems*, vol. 31, no. 3, pp. 772–785, 2019.

[11] Y. Jiang, S. Wang, V. Valls, B. J. Ko, W.-H. Lee, K. K. Leung, and L. Tassiulas, “Model pruning enables efficient federated learning on edge devices,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[12] Z. Tang, L. Luo, B. Xie, Y. Zhu, R. Zhao, L. Bi, and C. Lu, “Automatic sparse connectivity learning for neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[13] M. R. Ganesh, D. Blanchard, J. J. Corso, and S. Y. Sekeh, “Slimming neural networks using adaptive connectivity scores,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[14] Y. Ma, X. Yao, R. Chen, R. Li, X. Shen, and B. Yu, “Small is beautiful: Compressing deep neural networks for partial domain adaptation,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[15] Y. Chen, R. Li, W. Li, J. Wang, and R. Li, “Three-stage global channel pruning for resources-limited platform,” *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[16] Y. Guan, N. Liu, P. Zhao, Z. Che, K. Bian, Y. Wang, and J. Tang, “Dais: Automatic channel pruning via differentiable annealing indicator search,” *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[17] C. Peng, Y. Li, R. Shang, and L. Jiao, “Recnas: Resource-constrained neural architecture search based on differentiable annealing and dynamic pruning,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[18] Y. Qian, Z. He, Y. Wang, B. Wang, X. Ling, Z. Gu, H. Wang, S. Zeng, and W. Swaileh, “Hierarchical threshold pruning based on uniform response criterion,” *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[19] G. Liu, K. Zhang, and M. Lv, “Soks: Automatic searching of the optimal kernel shapes for stripe-wise network pruning,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[20] Y.-J. Zheng, S.-B. Chen, C. H. Ding, and B. Luo, “Model compression based on differentiable network channel pruning,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[21] H. Salehninejad and S. Valaee, “Edropout: Energy-based dropout and pruning of deep neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 10, pp. 5279–5292, 2021.
