EdgeBERT: Optimizing On-Chip Inference for Multi-Task NLP

Thierry Tambe[1], Coleman Hooper[1], Lillian Pentecost[1], En-Yu Yang[1],
Marco Donato[2], Victor Sanh[3], Alexander M. Rush[4], David Brooks[1], Gu-Yeon Wei[1]
[1]Harvard University, [2]Tufts University, [3]Hugging Face, [4]Cornell University

Abstract—**Transformer-based language models such as BERT provide significant accuracy improvement to a multitude of natural language processing (NLP) tasks. However, their hefty computational and memory demands make them challenging to deploy to resource-constrained edge platforms with strict latency requirements. We present EdgeBERT, an in-depth and principled algorithm and hardware design methodology to achieve minimal latency and energy consumption on multi-task NLP inference. Compared to the ALBERT baseline, we achieve up to 2.4× and 13.4× inference latency and memory savings, respectively, with less than 1%-pt drop in accuracy on several GLUE benchmarks by employing a calibrated combination of 1) entropy-based early stopping, 2) adaptive attention span, 3) movement and magnitude pruning, and 4) floating-point quantization.**

Furthermore, in order to maximize the benefits of these algorithms in always-on and intermediate edge computing settings, we specialize a scalable hardware architecture wherein floating-point bit encodings of the shareable multi-task embedding parameters are stored in high-density non-volatile memory structures. Altogether, EdgeBERT enables fully on-chip inference acceleration of NLP workloads with 5.2× and 163× lower energy than that of an un-optimized accelerator and CUDA adaptations on an Nvidia Jetson Tegra X2 mobile GPU, respectively.

I. Introduction

Transformer-based networks trained with large multi-domain datasets have unlocked a series of breakthroughs in natural language learning and representation. A major catalyst of this success is the *Bidirectional Encoder Representations from Transformers* technique, or BERT [12], which substantially advanced nuance and context understanding. Its pre-training strategy, which consists of learning intentionally hidden sections of text, have proven beneficial for several downstream natural language processing (NLP) tasks. BERT has sparked leading-edge performance in NLP leaderboards [42], [56], and it is now applied at a global scale in web search engines [37] with marked improvements in the quality of query results.

Advances in NLP models are also fueling the growth of intelligent virtual assistants, which leverage natural language understanding and question answering capabilities to implement interactive voice interfaces. A need for more secure and privacy-guarding platforms, combined with the advantage of eliminating additional communication latency, motivates running these inference tasks directly on edge devices. However, BERT's impressive performance comes with heavy compute and memory costs that make on-device operation challenging. Notably, the BERT base model consumes 432 MB of memory in native 32-bit floating-point (FP32) precision, which limits its adoption on low capacity mobile/embedded systems with real-time constraints.

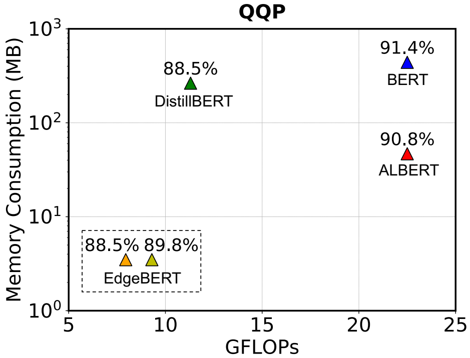

Fig. 1: Memory footprint and computation requirements of prominent BERT-based models on the QQP GLUE task. EdgeBERT significantly lowers storage and compute consumption while sustaining less than 1%-pt accuracy loss compared to the base ALBERT model.

A number of prior works have been proposed to reduce BERT storage and computation overheads [16]. In fact, most of the compression techniques (weight pruning [35], distillation [46], quantization [49], [63]) originally proposed for convolutional and recurrent neural networks (CNNs, RNNs) have been independently applied to Transformer-based networks.

In this work, we present *EdgeBERT*, a principled approach to balance and apply a comprehensive set of algorithmic optimizations and enable highly efficient hardware acceleration of NLP tasks. The optimization techniques we consider have disparate effects on model properties and performance, and our methodology navigates the complex interactions between them to expose opportunities for increased performance and power efficiency for resource-constrained, on-device execution. Namely, we intensively evaluate and curate the accuracy, performance, energy-efficiency, and storage impacts of *(i)* early exit [60], *(ii)* adaptive attention span [50], *(iii)* network pruning (movement [47] and magnitude [19]), and *(iv)* cross-model adaptive floating-point quantization [52] – towards the ultimate goal of exacting minimal latency and energy consumption, and culminating in specialized hardware acceleration.

Fig. 1 compares the raw compute and memory requirements of prominent BERT-based models as well as their attained accuracy across a set of NLP tasks. EdgeBERT per-task model compression generates models that are up to 2.4× and 13.4× less computational and memory demanding, respectively, while remaining within 1%-pt of the baseline ALBERT accuracy.

While some of these algorithmic benefits can be reaped on commodity GPUs, we unlock additional latency (1.1×) and energy (2.7×) savings by co-designing the hardware datapaths.

Specifically, we exploit these algorithmic optimizations in a specialized and modular hardware architecture, the EdgeBERT accelerator, which employs bit-mask encoding for compressed sparse computations while optimizing key operations for numerical stability and energy efficiency.

Moreover, the *ad hoc* usage of a multi-task NLP edge/IoT device may impose further latency and energy overheads due to high-frequency fetching of domain-specific parameters from off-chip memory. In EdgeBERT, we not only significantly reduce the memory footprint of domain-specific parameters, but also establish a compact set of baseline parameters that are shared across tasks and stored in embedded non-volatile memories (eNVMs) in order to alleviate the costs associated with multi-task intermediate computing.

This paper makes the following contributions:

- We propose EdgeBERT, a carefully curated approach to optimize multi-task NLP inference for resource-constrained embedded devices by simultaneously leveraging several latency and memory alleviation techniques, namely, entropy-based early exit, adaptive attention span, movement pruning, magnitude pruning, and adaptive floating-point quantization.
- We evaluate the effectiveness of the EdgeBERT optimizations on natural language inference (MNLI), sentence similarity (QQP), sentiment analysis (SST-2), and question answering (QNLI) tasks revealing up to 2.4× latency savings, and 13.4× memory savings with less than 1% accuracy loss compared to the base ALBERT model.
- Leveraging the insights from this broad analysis, we propose, design, and evaluate a 12nm ASIC hardware accelerator that exploits EdgeBERT algorithmic optimizations and further alleviates communication costs via shared parameter storage in eNVM in order to perform on-chip inference acceleration with up to 5.2× and 163× lower energy consumption than that of an un-optimized accelerator and an Nvidia Jetson TX2 GPU, respectively.

II. Background and Related Work

A. Benchmarks

The General Language Understanding Evaluation (GLUE) benchmark is the most widely used tool to evaluate NLP performance. It consists of nine English sentence understanding tasks covering three categories: Single-Sentence, Similarity and Paraphrase, and Inference [56]. This collection of datasets is specifically designed to favor models that can adapt to a variety of NLP tasks. To validate the robustness and generalization performance of the EdgeBERT methodology, we conduct our evaluation on the four GLUE tasks with the largest corpora, which cover all three GLUE categories: SST-2 (Single-Sentence), QQP (Similarity and Paraphrase), and QNLI and MNLI (Inference).

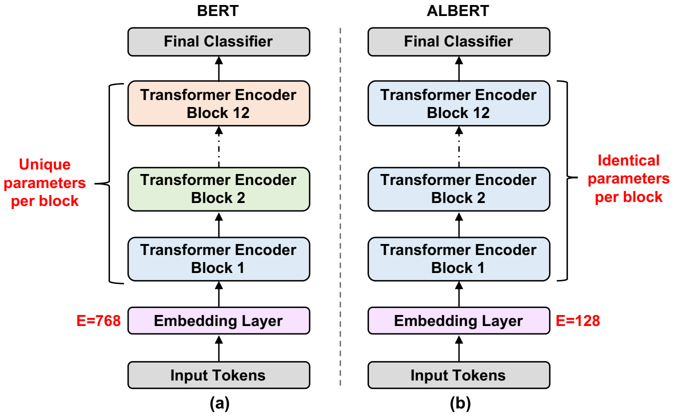
BERT
Final Classifier
Transformer Encoder Block 12
Transformer Encoder Block 2
Transformer Encoder Block 1
Unique parameters per block
E=768
Embedding Layer
Input Tokens
(a)
ALBERT
Final Classifier
Transformer Encoder Block 12
Transformer Encoder Block 2
Transformer Encoder Block 1
Identical parameters per block
Embedding Layer
E=128
Input Tokens
(b)

Fig. 2: Comparison between (a) BERT, and (b) ALBERT base models. ALBERT uses a smaller embedding size and its Transformer encoder layers share the same parameters.

B. Variations of BERT

Since the advent of BERT with 110M parameters, a number of variants were proposed to alleviate its memory consumption or to further improve its prediction metrics. RoBERTa [32] generalizes better on several GLUE tasks by training on significantly more data, and for a longer amount of time, but remains as computationally intensive as BERT. DistilBERT [46] and MobileBERT [51] leverage knowledge distillation to reduce BERT size by 1.7× and 4.3×, respectively, with iso-accuracy. SqueezeBERT [22] substitutes several operations in the Transformer encoder with 1D grouped convolutions achieving 4× speedup while being 2× smaller. Q8BERT [63] employs a symmetric linear quantization scheme for quantizing both weights and activations into 8-bit integers. In contrast, in this work we leverage the higher dynamic range of floating-point encodings for greater quantization resilience.

ALBERT [28] yields the smallest footprint to date for a compressed BERT variant with only 12M parameters, while maintaining competitive accuracy on the GLUE benchmarks. Fig. 2 summarizes the key differences between the ALBERT model and the base BERT model. While each of BERT's twelve encoder layers have a unique set of weights, ALBERT's encoder layers instead share and reuse the same parameters – resulting in significant compression. The encoder block in both models is the same architecture as the legacy Transformer network [53], but with twelve parallel self-attention heads. Moreover, ALBERT employs a smaller embedding size (128 vs. 768) thanks to factorization in the embedding layer. In this work, we adopt the ALBERT [28] variant as an efficient baseline. This work further pursues strategies to reduce latency and storage requirements to suit embedded hardware platform constraints.

C. Hardware Acceleration of Transformer-based models

Over the last decade, there has been vigorous research on the design of high-performance and energy-efficient deep neural network (DNN) hardware accelerators [3], [4], [8], [9], [18], [20], [24], [26], [30], [38], [39], [44], [45]. As these accelerators are increasingly deployed at all computing scales, there is

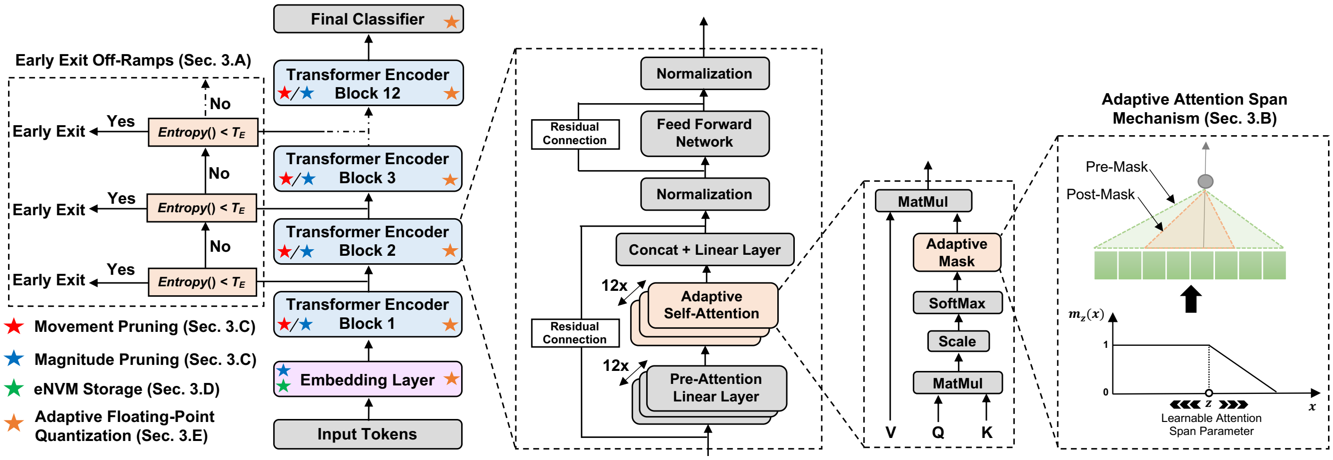
Early Exit Off-Ramps (Sec. 3.A)
Early Exit
Yes
No
Entropy() < T_E
Final Classifier
Transformer Encoder Block 12
Transformer Encoder Block 3
Transformer Encoder Block 2
Transformer Encoder Block 1
Embedding Layer
Input Tokens
Movement Pruning (Sec. 3.C)
Magnitude Pruning (Sec. 3.C)
eNVM Storage (Sec. 3.D)
Adaptive Floating-Point Quantization (Sec. 3.E)
Normalization
Feed Forward Network
Residual Connection
Concat + Linear Layer
12x
Adaptive Self-Attention
Pre-Attention Linear Layer
MatMul
Adaptive Mask
SoftMax
Scale
V
Q
K
Adaptive Attention Span Mechanism (Sec. 3.B)
Pre-Mask
Post-Mask
$m_z(x)$
1
0
z
x
Learnable Attention Span Parameter

Fig. 3: Memory and latency optimizations incorporated in the EdgeBERT methodology. Movement or magnitude pruning is performed on all Transformer encoders. The embedding layer is exclusively pruned by magnitude-based threshold, and stored in non-volatile memory. Adaptive floating-point quantization is applied to all weights and activations. Highway off-ramp layers are attached to the output of the first eleven encoder blocks in order to evaluate early exit confidence. Finally, each self-attention head learns its own optimal attention span.

additional interest in the hardware architecture community to automatically generate designs [54], [55], [58]. However, most of these works focus on CNN and RNN computations, and not as much scrutiny has been given to accelerating Transformer-based networks with self-attention mechanisms.

Recent work in accelerating Transformer-based NLP includes A^3 [17], which proposed a hardware architecture that reduces the number of computations in attention mechanisms via approximate and iterative candidate search. In contrast, this work learns the optimal attention search radius during the finetuning process and stores it in accelerator registers as a mask whose sparsity is then exploited to avoid unnecessary matrix multiplications. OPTIMUS [40] looked to holistically accelerate the Transformer network with compressed sparse matrix multiplications and by skipping redundant decoding computations. [33] proposes a dense systolic array accelerator for the Transformer's multi-head attention and feed-forward layers. GOBO [62] performs 3-bit clustering on the majority of BERT weights while storing the outlier weights and activations in full FP32 precision, thereby requiring a mixed-precision hardware architecture and non-uniform memory storage. In this work, we quantize both weights and activations in 8-bit adaptive floating-point format without accuracy loss.

Noteworthily, HAT [57] conducts hardware-aware neural architecture search to extract smaller, efficient Transformer spin-offs optimized for different hardware targets. In this work, we present a hardware architecture that not only accommodates all the BERT operations with sparse-compressed computations, but also offers specialized and efficient datapath support for the EdgeBERT latency-alleviating optimizations.

III. Memory and Latency Optimizations on ALBERT

In an effort to minimize latency and memory requirements, our EdgeBERT methodology carefully incorporates 1) movement pruning, 2) magnitude pruning, 3) adaptive attention span, 4) adaptive floating-point quantization, and 5) early stopping. While briefly describing them individually in this section, we provide a reasoned methodology for methodically applying them on ALBERT, as shown in Fig. 3, in order to enforce minimal memory and latency overheads during inference.

A. Entropy-based Early Exit

The motivation behind early exit is to match linguistically complex sentences with larger (or deeper) models and simple sentences with smaller (or shallower) models [48], [60]. This is done by adding a lightweight classifier at the output of the shared Transformer layer so that a given input can exit inference earlier or later in the stack, depending on its structural and contextual complexity. The classifier computes and compares the entropy of an output distribution with a preset "confidence" threshold, T_E, in order to assess whether the prediction should exit or continue inference in the next Transformer encoder block. The entropy H on sample x is estimated as:

$$H(x) = -\sum p(x)\log p(x) = \ln\left(\sum_{k=1}^{n} e^{x_k}\right) - \frac{\sum_{k=1}^{n} x_k e^{x_k}}{\sum_{k=1}^{n} e^{x_k}} \quad (1)$$

The early exit condition is met when $H < T_E$. Therefore, the larger T_E becomes, the earlier the sample will exit with potentially lower accuracy.

B. Adaptive Attention Span

The attention mechanism [6] is a powerful technique which allows neural networks to emphasize the most relevant tokens of information when making predictions. The base ALBERT model contains up to twelve parallel attention heads – each learning their own saliency weights on the full length of the encoder input. However, depending on the complexity of the task, many heads can be redundant and can be safely removed without impacting accuracy [36]. Furthermore, the cost of computing the attention mechanism scales quadratically with the sequence length. Therefore, there is potentially a meaningful amount of computations and energy to be saved in optimizing the inspection scope of every attention head.

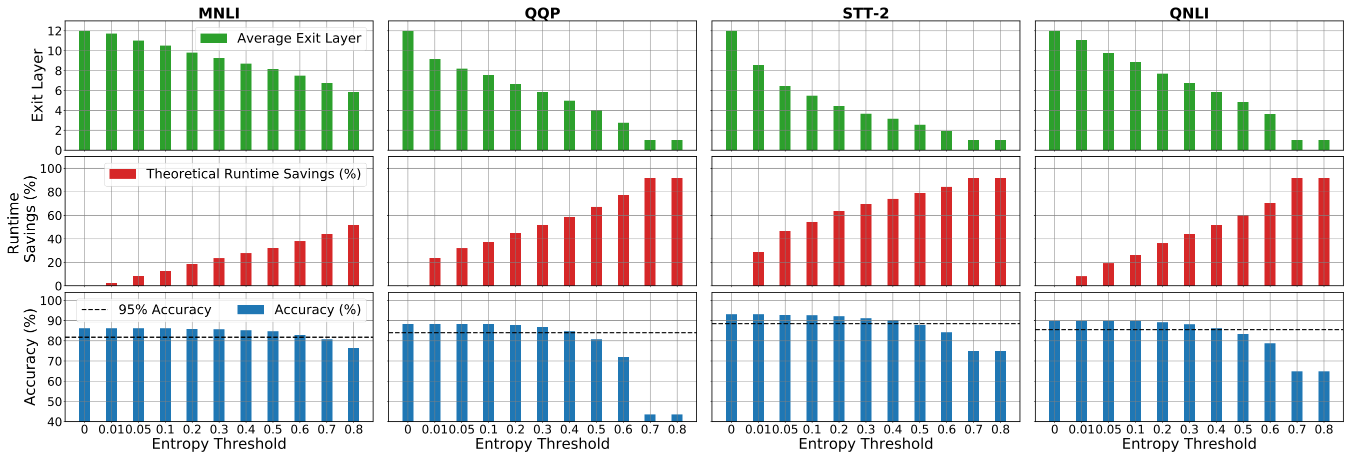

Fig. 4: Early exit results on ALBERT, highlighting accuracy, runtime savings, and average exit layer.

In the quest to avoid needless attention computations in ALBERT, a learnable parameter z is introduced in the datapath of each self-attention head in order to find its own optimal attention span [50]. The parameter z is mapped to a masking function with a [0, 1] output range, as shown in Fig. 3. The masked span is then applied on the attention weights to re-modulate their saliencies. The optimal span is learned during the fine-tuning stage by adding back the average loss from the reduced span to the training cross-entropy loss.

C. Network Pruning

To quantify the benefits of network pruning on the ALBERT model, we evaluate both movement [47] and the more common magnitude-based [19] pruning.

Movement pruning is a simple first-order pruning technique that is applied during model fine-tuning. The authors observed that during fine-tuning, weights stay close to their pre-trained values. Therefore, magnitude pruning may be intuitively a sub-optimal method to use during transfer learning, as pre-trained weights closer to zero have a high chance of being eliminated regardless of the fine-tuning requirement. Instead, movement pruning removes weights that are dynamically shrinking towards 0 (i.e., according to movement of the values). In the EdgeBERT methodology, movement pruning is applied on the Transformer encoder of ALBERT as the parameters in this layer are set to auto-update during the fine-tuning process.

Magnitude pruning is a well-known neural network compression technique, which preserves weights whose absolute values are larger than a defined threshold. In this work, we apply it on the Transformer encoder as well as on the embedding layer of ALBERT. The pruned embedding weights are shared across tasks, and are frozen during the fine-tuning process. EdgeBERT submits all the NLP tasks to the same embedding magnitude pruning threshold in order to enforce uniformity in the data during multi-domain on-chip acceleration.

We note that using movement pruning on the embedding layer would require unfreezing its weights and would therefore make them unique for each NLP domain, forgoing opportunities for data reuse in hardware.

Movement vs. Magnitude pruning for sparsifying the Transformer encoder. Movement pruning particularly outperforms magnitude-based pruning in high sparsity regimes, as each individual remaining weight becomes more important to learn the task at hand [47]. Therefore, choosing between the two techniques would depend on the per-task tolerance to increasing sparsity. Magnitude-based pruning would be the preferred solution in case the model exhibits greater sensitivity in low to medium sparsity thresholds.

D. Embedded non-volatile memory (eNVM) embedding storage

In contrast to task-specific encoder weights, embedding parameters are deliberately frozen during EdgeBERT fine-tuning, and reused across different NLP tasks. We seek to avoid the energy and latency costs of reloading embeddings from off-chip memory for different tasks by storing these shared parameters in eNVMs. eNVM storage also enables energy-efficient intermittent computing because the eNVM will retain embedding values if and when the system-on-chip powers off between inferences.

Despite their compelling storage density and read characteristics, ReRAM exhibits two main drawbacks: potentially high write cost (in terms of energy and latency) and decreased reliability, particularly in multi-level cell (MLC) configurations [11]. Fortunately, the ALBERT embeddings will be acting as read-only parameters on-chip, which makes them highly suitable for eNVM storage, but previous work highlights the need to study the impacts of faulty, highly-dense ReRAM storage on NLP task accuracy [41]. On the other hand, encoder weights need to be updated when switching across different NLP tasks. To prevent energy and latency degradation that would follow from updating the encoder weight values in eNVM, we map the natural partition of shared and task-specific parameters to eNVM and SRAM respectively [13].

This work will specifically consider dense, energy-efficient Resistive RAM (ReRAM) arrays [7], [31] as an on-chip storage solution for shared embedding parameters. As we seek to

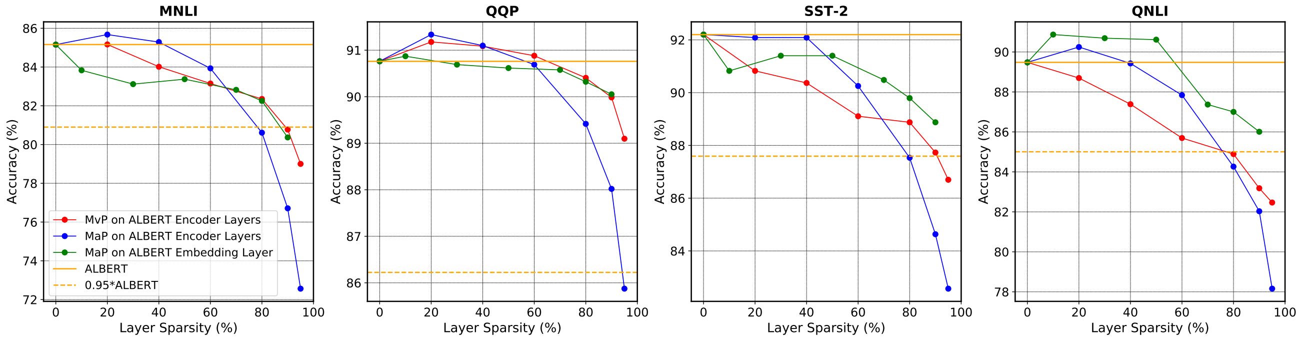

Fig. 5: Outcome of movement pruning (MvP) and magnitude pruning (MaP) on Transformer encoders and magnitude pruning (MaP) on the embedding layer of ALBERT. Movement pruning starts to outperform magnitude pruning in high sparsity regimes (≥ 70%). Overall, there is significant resilience in pruning the embeddings.

maximize area efficiency, we evaluate the impact of MLC ReRAM reliability on task accuracy by running fault injection simulations to identify viable and robust eNVM storage configurations. We leverage and extend Ares, an existing, open-source fault injection framework for quantifying the resilience of DNNs [43].

E. Adaptive Floating-Point Quantization

DNN algorithmic resilience allows for parameters to be represented in lower bit precision without accuracy loss. Fixed-point or integer-based quantization techniques, commonly adopted in CNN models, suffer from dynamic range deficits and may be inadequate for NLP models whose weights can be more than an order of magnitude larger [52]. This phenomenon is owed to layer normalization [5], which is commonly adopted in NLP models and has invariance properties that do not reparameterize the network – unlike batch normalization [23], which produces a weight normalization side effect in CNNs.

As part of the EdgeBERT methodology, we employ floating-point based quantization, which provides 2× higher dynamic range compared to integer techniques [25]. We specifically adopt the AdaptivFloat [52] encoding scheme, which adapts at a per-layer granularity to the statistical distribution of the DNN parameters. In this work, both weights and activations are quantized with AdaptivFloat across all the layers of ALBERT.

IV. Experimental Results of the Algorithms

We evaluate the effectiveness of the optimizations on the ALBERT-v2 version using four common NLP benchmarks [56]: natural language inference (MNLI), sentence similarity (QQP), sentiment analysis (SST-2), and question answering (QNLI). We note the MNLI results show the accuracy under the "match" category. We first report the impact of each individual optimization and then the synergistic impact when they are all applied.

A. Individual Impact of the Latency and Memory Optimizations

Early Exit: We analyzed the performance of ALBERT with early exit for a range of entropy thresholds. The average exit layer, accuracy, and run-time savings per entropy threshold are shown in Figure 4. Employing early exit allows for 30%, 45%, 54%, and 36% in theoretical runtime savings, and an average exit layer of 8.69, 6.60, 5.47, and 7.68 on MNLI, QQP, SST-2 and QNLI benchmarks respectively, with minimal accuracy loss (i.e., less than 1%-pt). Latency savings increase to 44%, 62%, 78%, and 53%, respectively, with 5%-pt accuracy loss.

TABLE I: Learned spans of every attention head in ALBERT. Baseline Acc: MNLI=85.16, QQP=90.76, SST-2=92.20, QNLI=89.48

Task	Attention Head												Avg. Span	Acc.	Diff.
	1	2	3	4	5	6	7	8	9	10	11	12			
MNLI	20	0	0	0	0	0	36	81	0	0	0	10	12.3	85.11	-0.05
QQP	16	0	0	0	0	0	40	75	0	0	0	2	11.0	90.80	0.04
SST-2	31	0	0	0	0	101	14	5	0	36	0	0	15.6	91.99	-0.21
QNLI	39	0	0	0	0	105	22	19	0	51	0	0	19.6	88.92	-0.56

To maximize this benefit in hardware, the EdgeBERT accelerator contains datapath specialization to efficiently compute the entropy for early exit assessment. As the tolerance to accuracy loss is subjective, the EdgeBERT accelerator allows the entropy threshold value to be programmed (Sec. V-D2). Moreover, an energy-constrained system-on-chip (SoC) may use the entropy threshold as a power management knob by slightly penalizing the prediction accuracy in order to save energy.

Adaptive Attention Span: The maximum sentence length for fine-tuning the GLUE tasks is 128. As a result, shorter sentences are typically zero-padded to 128 during the tokenization pre-processing. Table I shows the final attention span learned by each self-attention head when fine-tuning with the adaptive attention span technique. Strikingly, the twelve parallel self-attention heads in ALBERT do not need to inspect their inputs at maximum span. In fact, more than half of the attention heads, 8 for MNLI and QQP and 7 for SST-2 and QNLI, can be completely turned off with minimal accuracy loss. This amounts to a 1.22× and 1.18× reduction, respectively, in the total number of FLOPS required for single-batch inference.

The twelve attention spans, learned during fine-tuning, are written to registers in the EdgeBERT accelerator in the form of a 128-wide vector – in order to predicate on the inference computation of the multi-head attention (Sec. V-D1).

Sparsity studies: The first application of movement pruning on BERT shows that the model reaches 95% of its original performance on MNLI and QQP when zero-ing 95% of its

TABLE II: Impact of both weight and activation bit compression on ALBERT after post-finetuning floating-point quantization. Exponent field is fixed to 3-bit.

Bit Widths	MNLI	QQP	SST-2	QNLI
FP32	85.16	90.76	92.20	89.48
8	85.24	90.82	92.32	89.88
7	85.30	91.00	92.43	88.87
6	85.74	90.77	92.20	88.68
5	81.41	87.50	88.53	81.93
4	48.06	47.16	52.87	49.86

TABLE III: Results of fault injection simulations modeling impact of ReRAM embedding storage on ALBERT task accuracy. SLC=single-level cell (1 bit per cell). MLC2= 2 bits per cell. MLC3 = 3 bits per cell.

	SLC		MLC2		MLC3	
	MEAN	MIN	MEAN	MIN	MEAN	MIN
MNLI	85.44	85.44	85.44	85.44	85.42	85.25
QQP	90.77	90.77	90.77	90.77	90.75	90.61
SST-2	92.32	92.32	92.32	92.32	91.86	90.83
QNLI	89.53	89.53	89.53	89.53	**88.32**	**53.43**
Area Density (mm^2/MB)	0.28		0.08		0.04	
Read Latency (ns)	1.21		1.54		2.96	

encoder weights [47]. Fig. 5 shows that this observation is maintained for MNLI and QQP in our approach with ALBERT even though the latter is already substantially compressed via cross-layer parameter sharing. Furthermore, movement pruning is shown to outperform magnitude pruning at high sparsity levels ($\geq 70\%$) – although the accuracy benefits diminish more steeply, especially for smaller datasets (SST-2, QNLI). We also find significant resilience in pruning ALBERT's embeddings where 95% of its weights can be eliminated while maintaining at least 95% of ALBERT accuracy across the four tasks.

To exploit the benefits of these insights on hardware, the EdgeBERT accelerator employs bit-mask encoding and zero-skipping for energy and memory efficient compressed sparse computations. (Sec. V-C).

Quantization studies: We applied post-finetuning adaptive floating-point quantization [52] uniformly on all layers, compressing both weight and activation precisions. We first performed a search on the optimal exponent bit width to satisfy the dynamic range requirements of ALBERT. Adjusting the floating-point exponent space to 3 bits yielded the best inference performance across tasks. This exponent setting is, therefore, used in the bit compression results reported in Table II.

We observe that near FP32 performance is maintained with 8-bit adaptive floating-point quantization across the four benchmarks. In fact, the regularization effect of the quantization error pushes 7-bit and 8-bit accuracies above the FP32 baseline.

For robust computational accuracy, the processing units in the EdgeBERT accelerator are implemented with 8-bit floating-point datapaths (Sec. V-C).

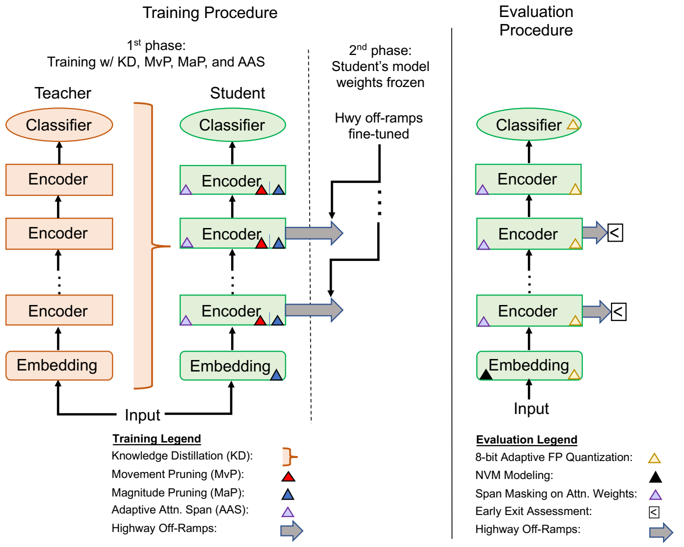

Fig. 6: EdgeBERT training and evaluation procedure.

Impact of eNVM storage: We evaluate the robustness of storing the 8-bit floating-point encodings of ALBERT embeddings in eNVM storage. In order to quantify the trade-offs between storage density and task accuracy, we use cell characteristics of 28nm ReRAM programmed with varying number of bits per cell [11] and launch 100 fault injection trials per storage configuration. Our base ALBERT model is pruned such that 60% of embedding weights are zero-valued, so we store compressed embedding weights using a bitmask-style sparse encoding. Previous work demonstrates that DNN weight bitmask values are vulnerable to MLC faults, so the bitmask is protectively stored in lower-risk single-level cells, while we experiment with MLC storage for the non-zero data values [41]. Fault injection and accuracy evaluation were performed using a previously-validated, open-source fault injection framework and MLC eNVM modeling on sparse-encoded embedding parameters [41], [43].

Table III uncovers exceptional resilience to storing embeddings in MLC ReRAM. Across many fault injection trials, we observe that MLC2 embedding storage (ReRAM programmed at 2 bits-per-cell) does not degrade accuracy across multiple tasks, while MLC3 exhibits potentially catastrophic degradation in minimum accuracy and an appreciable decline in average accuracy for the QNLI task, highlighted in bold. Based on this observation, the EdgeBERT accelerator leverages MLC2 ReRAMs for storage of nonzero embedding weights (Sec. V-D). We selected ReRAMs for their relative maturity and demonstrated read characteristics. However, we note that there is a larger design space of opportunities to be explored with other emerging MLC-capable NVM technologies such as PCM [10], which is beyond the scope of this work.

B. *Synergistic Impact of Latency and Memory Optimizations*

1) Training and Evaluation Procedure:

We implemented the EdgeBERT training and evaluation procedures, illustrated in Fig. 6, on the base of HuggingFace's Transformers infrastructure [59].

The training methodology consists of two phases. In the first phase, ALBERT is pruned during fine-tuning: magnitude

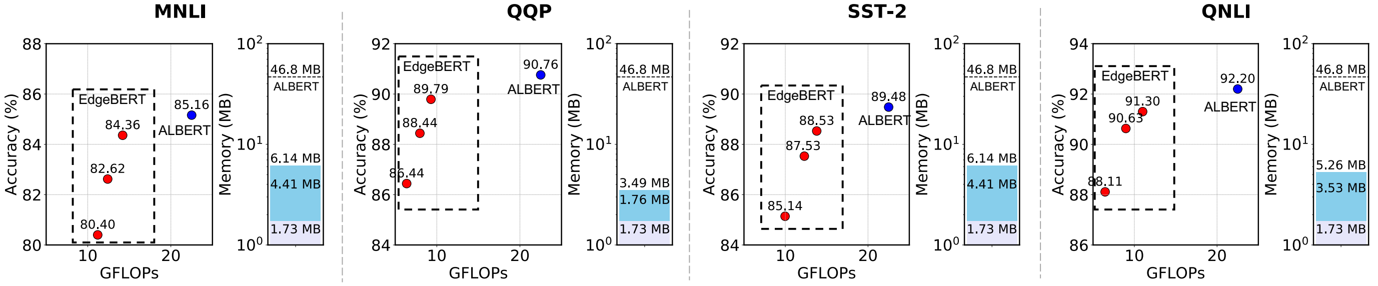

Fig. 7: Accuracy, performance and memory footprint attained after a combined application of the optimizations. With less than 1%-pt accuracy drop, up to 2.4× and 13.4× reduction in latency and storage, respectively, is achieved. Embedding weights (lavender color) were uniformly reduced to 1.73MB across tasks.

TABLE IV: Summary of optimization results in terms of achievable sparsity, attention span, and early exit parameters, with performance and accuracy implications shown in Figure 7.

	Encoder Sparsity (%)	Embedding Sparsity (%)	Enc. Pruning Method	Avg. Attn. Span	Early Exit Threshold	Avg. Exit Layer
Baseline	0	0	-	128	-	12
MNLI	50	60	MaP	12.7	0.4	9.18
					0.6	8.02
					0.7	7.25
QQP	80	60	MvP	11.3	0.3	5.99
					0.4	5.12
					0.5	4.08
SST-2	50	60	MaP	18.4	0.05	6.93
					0.1	5.62
					0.2	4.02
QNLI	60	60	MaP	21.5	0.2	8.69
					0.3	7.77
					0.4	6.26

pruning is applied to the embedding layer and either movement or magnitude pruning is applied to the encoder layer. An additional loss term comes from knowledge distillation using the base ALBERT model fine-tuned on the target task as a teacher. The embeddings and the encoder layer are subject to separate pruning schedules. At the same time, the attention heads are learning their optimal spans. In the second training phase, we followed the DeeBERT approach [60] to freeze the model's parameters prior to fine-tuning the early exit highway off-ramp.

At evaluation time, 8-bit adaptive floating-point quantization is applied on all the weights and activations. The quantized and pruned embedding weights are modeled according to a 2-bit per cell multi-level (MLC2) ReRAM NVM configuration. The learned attention span mask is element-wise multiplied with the attention weights to re-modulate their saliencies. Early exit is assessed at the output of the encoder layer by computing the entropy of samples.

2) Computation and Memory Benefits:

Using the multi-step procedure illustrated in Fig. 6, we amalgamate into ALBERT the various memory and latency reduction techniques at training and evaluation time. By virtue of the achieved optimizations results reported in Table IV, Fig. 7 summarizes the generated benefits of the EdgeBERT methodology with the following main observations:

- With less than 1%-pt accuracy drop, efficient variants that are up to 13.4× more memory-efficient and 2.4× faster compared to the ALBERT-base model are derived. Further speedups up to 3.53× are possible under 5%-pt accuracy degradation.
- Across the four corpora, a uniform 40% density in the embedding layer is achieved, establishing a compact memory baseline of 1.73MB to be stored in eNVMs.
- As QQP exhibits strong resiliency in high sparsity regimes, movement pruning proved to be the superior pruning technique, allowing up to 80% of its encoder weights to be zero-valued with less than 1%-pt accuracy drop. On the other hand, magnitude pruning produces higher accuracy on MNLI, SST-2, and QNLI tasks as between 50% and 60% of encoder weights could be pruned with less than 1%-pt accuracy penalty.
- As a side effect of network pruning, the average attention span slightly increases. For example in MNLI and SST-2 benchmarks, the multi-head attention expands its inspection span from an average of 12.3 and 15.6 (as shown in Table I) to 12.7 and 18.4, respectively. Intuitively, pruning forces the adaptive attention span mechanism to "cast a wider net" in order to compensate for the loss of contextual information. More importantly, despite the slight uptick in the average span, the attention heads that were disabled (i.e. with span=0) pre-pruning still remain disabled post-pruning. Therefore, even with significant weight sparsity, the latency and energy reduction benefits from optimizing the attention span are marginally affected.
- We also notice a moderate increase in the average exit layer between pre- (Fig. 4) and post- network pruning. Among the four NLP tasks under evaluation, SST-2 exhibits the greatest percentage increase (27%) in the average exit layer for a fixed 1%-pt accuracy decay. Prior work demonstrated that pruning may cause confidence loss in certain DNNs and exacerbate prediction latencies [61], so the EdgeBERT methodology deliberately avoids steeper latency degradation by not sparsifying the layer normalization parameters, the early exit off-ramp layer, and the final classifier. Formulating early exit performance as:

$$EE_{perf} = \frac{\text{Model Accuracy}}{1 - \text{Theoretical Runtime Savings}} \quad (2)$$

EE_{perf} further deteriorates by 3.2× on SST-2 when the layer normalization, off-ramp, and final classifier parameters are 50% zero-valued. Moreover, these parameters

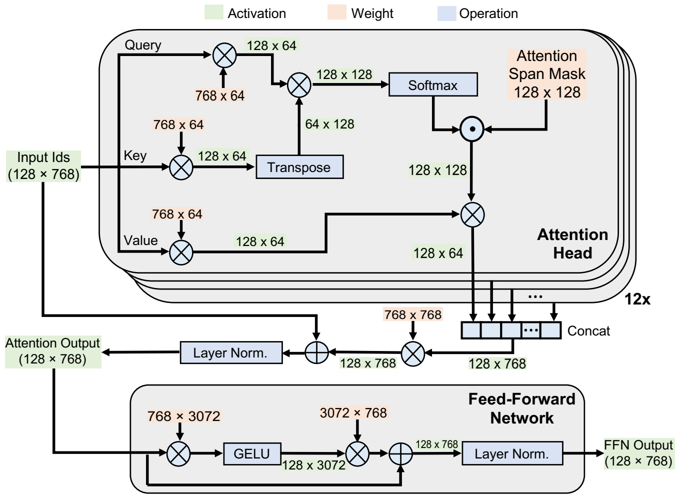

Fig. 8: Computations inside the ALBERT Transformer encoder with attention span modulation. Here, the input sequence is composed of 128 tokens. To simplify the computational diagram, the bias layers are not included.

tally to only 0.6MB of memory in FP8 precision. The savings to be made with pruning them would therefore be relatively small.

3) Memory Overheads of the Optimizations:

The memory consumption reported in Fig. 7 includes the cost of supporting bitmask sparse encoding which adds a 12% overhead on top of the storage requirement for the non-zero values. The off-ramp classifier used for early exit is composed of a 768×768 linear layer, which adds an additional 0.59MB as previously mentioned. And finally, the attention span mask accounts for a small 1.53KB. Despite these additional memory overheads, the final compression ratio over the base ALBERT model is still large enough for the entire compressed model to be stored on a reasonably-sized SoC or on modern FPGAs. EdgeBERT further optimizes the accelerator area efficiency by storing embeddings in high-density 2-bits per cell (MLC2) eNVMs (Sec. V-D).

V. The EdgeBERT Hardware Architecture

A. ALBERT Operations

The Transformer encoder is the backbone of ALBERT, as more than 95% of inference computations are spent there. Fig. 8 summarizes the computations required in this unit. Assuming a sentence length of 128, the transformer encoder requires 1.9GFLOPs to compute matrix multiplications, layer normalizations, element-wise operations (add, mult.), and softmax. The attention span mask learned during fine-tuning is element-wise multiplied with the softmax output. Notably, all the computations inside any of the twelve attention units can be effectively skipped in case its associated attention span mask is 100% null.

B. The EdgeBERT Accelerator

In order to maximize the benefits of the latency and memory reduction techniques, we specialized a scalable hardware accelerator that exploits and further optimizes these algorithms for compute and energy efficiency as well as numerical stability. The EdgeBERT hardware accelerator, illustrated in Fig. 9 (a), consists of a processing unit (PU) containing MAC arrays, and a multi-function global buffer unit (GB). The communication handshaking between the PU and GB occurs via a custom-built bi-directional streaming channel. An AXI splitter arbitrates the CPU-controlled flow of instructions and data bound for the PU and GB AXI-slave partitions. The SRAM and ReRAM buffers of the EdgeBERT accelerator are sized reasonably so that the optimally-compressed and finetuned ALBERT model fits entirely on-chip.

C. Processing Unit

The processing unit (PU) in Fig. 9 (b) is designed for performing matrix-matrix multiplications (MATMUL) in linear layers and attention heads of ALBERT.

In the datapath, n defines the number of parallel floating-point vector MACs (VMAC) and the vector size of each VMAC. So, there are n^2 MAC units in total. The PU datapath takes two $n \times n$ matrices as input and computes $n*n*n$ MAC operations in n clock cycles. We use 8-bit floating point as the input and weight data type as no accuracy degradation was observed (Table II), and 32-bit fixed-point during accumulation. The PU accumulator sums activation matrices and quantizes the final matrix back to 8-bit floating-point.

To exploit sparsity in both input and weight matrices, we (1) adopt bit-mask encoding/decoding for compressing/decompressing the sparse matrix, and (2) implement skipping hardware logic in the datapath. Bit-masks are binary tags to indicate zero and non-zero entries of a matrix so that only non-zero entries are stored in SRAM scratchpads. For every cycle during decoding, a size n vector is fetched and decoded. The decoder first reads a n-bit mask from the single-banked mask buffer to figure out what bank in the n-banked input can be neglected, and inserts zero values back to the original zero entries. The encoder also takes a similar approach. It creates a bit mask vector and removes zero entries before sending the compressed output vector to the global buffer. To save energy, we implement logic to skip the computation of a VMAC product-sum if one of the operand vectors contains only zero values. Although the cycle-behavior of the datapath is not affected by the sparsity of inputs due to the fixed scheduling of data accesses and computations, skipping VMAC operations saves up to 2.6× in energy consumption (Sec. VI-B).

D. Muti-function Global Buffer Partition

The global buffer (GB) unit, in Fig. 9 (c), streams activation tiles to the PU, and then collects and unifies partial MATMUL output blocks in its unified activation buffer (UAB). Moreover, the GB is augmented with peripheral specialized units that compute layer normalization, softmax, and element-wise addition, all of which get invoked during the ALBERT inference. The accelerator further exploits the latency reduction techniques (adaptive attention span and early exit) by specializing them inside the GB peripherals. The GB also contains a 64KB

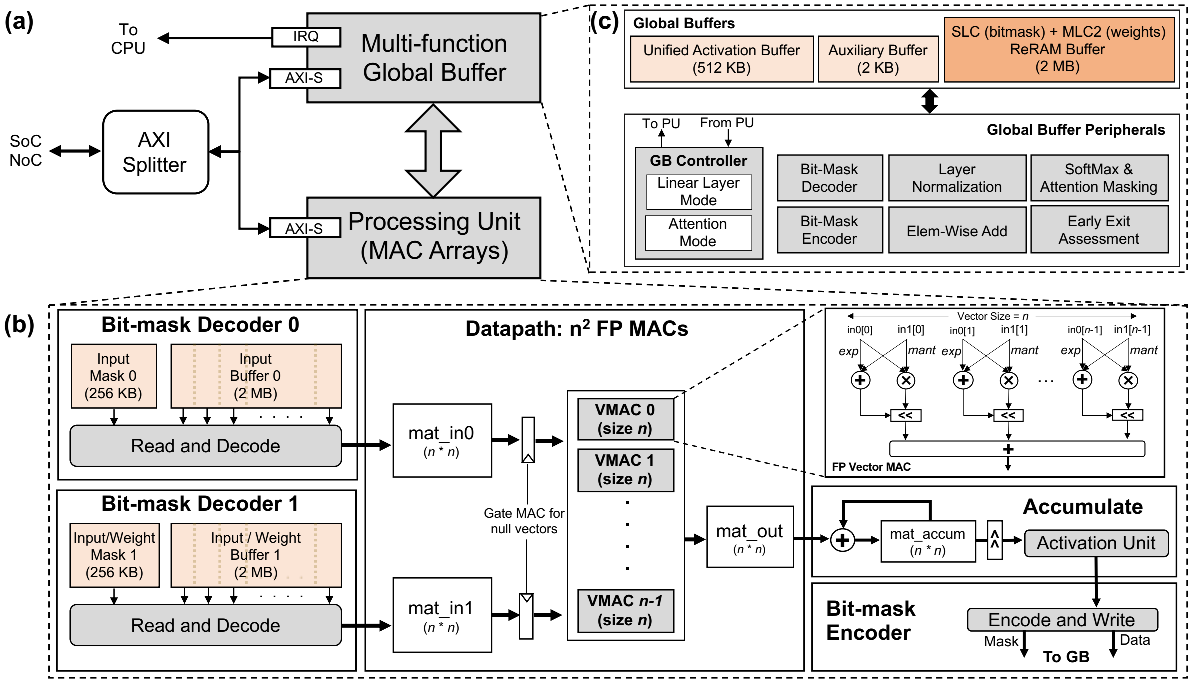

Fig. 9: (a) The EdgeBERT hardware accelerator highlighting (b) its processing unit, and (c) multi-function global buffer.

auxiliary buffer to house the parameters of layer normalization and the multi-head attention span masks learned during the fine-tuning process. The multi-task embedding pruned weights and corresponding bitmask are stored in a separate 2MB ReRAM NVM buffer in order to avoid reloading them when powered on. A bit-mask encoder and decoder are also instantiated in order to decompress and re-compress the hidden states before and after computation in the GB peripherals.

1) Computing the Multi-Head Self-Attention:

While the linear layers for the attention query, key and value tensors are computed in the PU, the proceeding softmax operation is optimized in the GB softmax unit.

First, prior to computing an attention head, the GB controller inspects its associated attention span mask in the auxiliary buffer. In case the attention span mask for an attention head is null, the GB controller proactively cancels and skips entirely the sequence of computations required for that head, and directly writes zero in the corresponding UAB logical memory for its context vector. In case the attention span mask for a head contains non-zero elements, the softmax unit takes advantage of the *LogSumExp* [15] and *Max* [34] tricks in order to vectorize (Algorithm 1) the computation of the softmax function $SM()$ as:

$$SM(A_k) = exp[A_k - MAX_k(A) - ln(\sum_{i=1}^{K} exp(A_k - MAX_k(A)))] \tag{3}$$

By doing so, the hardware prevents numerical instability stemming from exponential overflow, and avoids the computationally intensive division operation from the original softmax function. Upon completing the softmax operation, the GB softmax unit then performs element-wise multiplication between the resulting attention scores and the attention span mask.

```
Input: attention matrix $A$, and mask $A_M$ of size $(T * T)$
Output: masked softmax output matrix $A_O$
$T$ := number of tokens; $n$ := tile size;
for $i = 0$ to $T - 1$ do
    // Step 1: compute max value
    $max = -\infty$
    for $j = 0$ to $T - 1$ do
        $vec <= load(A_{[i][n*j:n*j+n-1]})$
        if $max < max(vec)$ then
            $max = max(vec)$
        end
    end
    // Step 2: compute log-exponential-sum
    $sum_{exp} = 0$
    for $j = 0$ to $T - 1$ do
        $vec <= load(A_{[i][n*j:n*j+n-1]})$
        $sum_{exp} += sum(exp(vec - max))$
    end
    $logsum_{exp} = ln(sum_{exp})$
    // Step 3: Get softmax and modulate with attn span mask
    for $j = 0$ to $T - 1$ do
        $vec <= load(A_{[i][n*j:n*j+n-1]})$
        $mask <= load(A_{M[i][n*j:n*j+n-1]})$
        $vec = exp(vec - max - logsum_{exp})$
        $vec = vec * mask$
        $store(vec) => A_{O[i][n*j:n*j+n-1]}$
    end
end
```

Algorithm 1: Computing Softmax and Attn. Span Masking

2) Computing the early exit entropy:

The early exit assessment unit is invoked right after the GB collects the activations of the second layer normalization operation in the encoder. Similarly to algorithm 1, it computes, in a vectorized fashion, the numerically-stable version of the

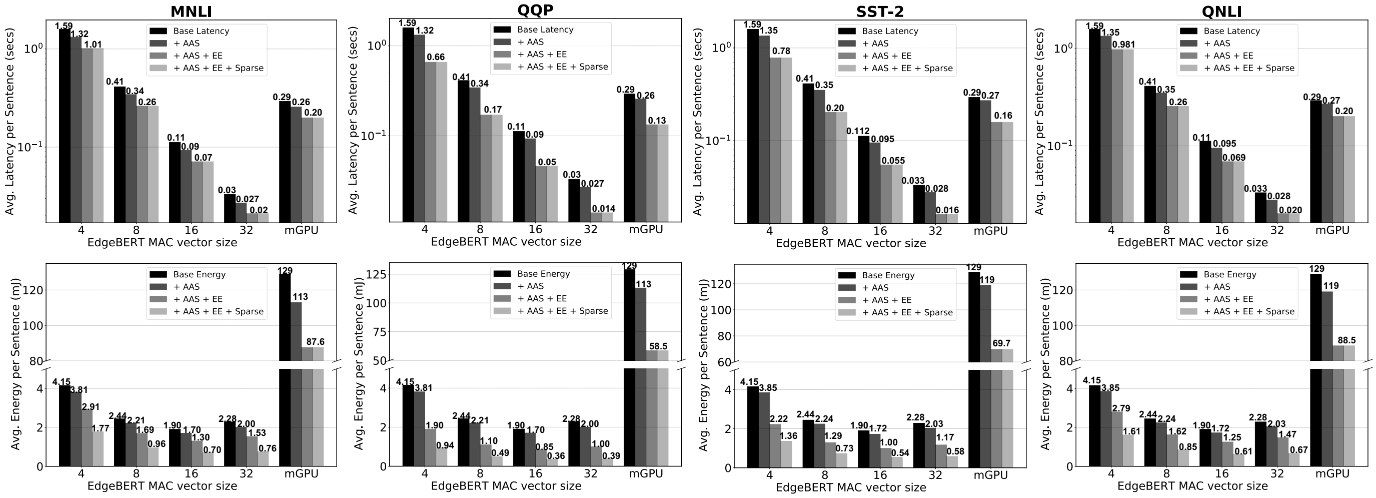

Fig. 10: Average latency (Top row) and energy (Bottom row) per sentence as the PU MAC vector size scales, highlighting impact of adaptive attention span (AAS), early exit (EE), and sparsity in weights and activations (Sparse) on the EdgeBERT accelerator and TX2 mGPU. The MAC vector size of 16 yields the most energy efficient design. Inference on QQP produces the lowest latency and energy after activating the latency and memory reduction optimizations.

entropy function from equation 1 as follows:

$$H(x_k) = \ln\left(\sum_{k=1}^{n} e^{x_k - MAX_k(x)}\right) - MAX_k(x) - \frac{\sum_{k=1}^{n} x_k e^{x_k - MAX_k(x)}}{\sum_{k=1}^{n} e^{x_k - MAX_k(x)}} \tag{4}$$

After completing the entropy calculation, the early exit unit then compares the result with the register value for the entropy threshold. If the early exit condition is met, the unit then triggers the accelerator's interrupt. Otherwise, the GB controller sends the activations back to the PU to execute the next Transformer encoder sequence.

3) Computing LayerNorm:

Normalization is commonly adopted in NLP seq2seq networks in order to speed up the training process. While batch normalization [23] uses the dimension of the mini-batch as a regularizer during training, layer normalization [5] instead uses the dimension of the hidden states in the same layer. During inference, the activation is normalized as:

$$X_{norm} = \frac{X - \mathrm{E}[X]}{\sqrt{\mathrm{Var}[X]}} * \gamma + \beta \tag{5}$$

where γ and β are learnable parameters obtained after training and stored in the GB auxiliary buffer. The EdgeBERT accelerator first computes the mean, $\mathrm{E}[X]$, using a running average over the number of hidden states, then evaluates the variance, $\mathrm{Var}[X]$, as: $\mathrm{E}[X^2] - \mathrm{E}[X]^2$. This process gets repeated for all the needed token steps.

VI. Hardware Evaluation

A. Design and Verification Methodology

The EdgeBERT accelerator is designed in synthesizable SystemC with the aid of hardware components from the MatchLib [27] and HLSLibs [21] open-source libraries. Verilog RTL is auto-generated by the Catapult high-level synthesis (HLS) tool [1] at 500MHz using a commercial 12nm process node. HLS constraints are uniformly set with the goal to achieve maximum throughput on the pipelined design. During the bottom-up HLS phase, the PU and GB SRAM buffers are mapped to synthesized memories from the foundry memory compiler, while the rest of the registers are mapped to D-latches.

The energy, performance, and area results are reported on the post-HLS Verilog netlists by the Catapult tool at the 0.8V/25c/typical corner. The 28nm ReRAM cells are characterized in NVSIM [14] and its read latency, energy, and area are back-annotated into the accelerator results after scaling to a 12nm F^2 cell definition.

To quantify the benefits of non-volatility (Sec. VI-C), we quantify the alternative cost of loading embeddings from off-chip using DRAMsim3 [29] to extract thermally-aware and cycle-accurate LPDDR4 DRAM energy and latency metrics. Finally, GPU results are obtained from CUDA implementations on an Nvidia TX2 mobible GPU (mGPU), whose small form-factor SoC targets embedded edge/IoT applications [2].

B. Performance, Energy, and Area Analyses

We start by measuring the energy-performance trade-offs of the EdgeBERT accelerator by scaling its PU MAC vector size. Simultaneously, we further quantify the benefit of bitmask encoding and the predicating logic of the early exit and adaptive attention span mechanisms by using the attained optimization results (i.e. embedding and encoder sparsity percentage, average exit layer and average attention span) reported in Table IV in which the achieved accuracy was within 1%-pt that of the base ALBERT model. The latency reduction techniques are also applied to the mGPU platform in order to quantify and compare the extent of these benefits.

Fig. 10 shows that the per-sentence end-to-end processing latency decreases by roughly 3.5× as the vector size doubles. Across the four tasks, the energy-optimal accelerator design is obtained with a MAC vector size, n, of 16. This is because the

TABLE V: Breakdown of area and power consumption in the EdgeBERT accelerator design with MAC size = 16

Blocks	Area (mm^2)	Power (mW)
PU datapaths	0.45	40.26
GB Peripherals	0.41	6.13
SRAM Buffers	4.10	60.67
ReRAM Buffers	0.15	3.48
Total	5.11	110.5

increase in the datapath power consumption with $n = 32$ starts to subdue the latency reduction gains. The predication/skipping mechanism of adaptive attention span reduces the accelerator processing time and energy consumption by up to 1.2× and 1.1×, respectively. Early stopping accrues both the latency and energy benefits in the EdgeBERT accelerator by an additional 1.3–2.0× across the four corpora. Additionally, compressed sparse execution in the PU and GB datapaths amounts to a larger 1.9–2.6× energy savings.

The EdgeBERT accelerator starts to outperform the mGPU processing time with n =16. This energy-optimal design generates 163× lower energy compared to the mGPU when all the optimizations are factored in. Furthermore, by specializing the predicating logic of the adaptive attention span and early exit mechanisms, greater overall latency reduction gains are achieved in the EdgeBERT hardware (2.42×) compared to the mGPU (2.2×) which is forced to rely on the embedded CPU to compute conditional and serial logic. We further highlight that the time spent evaluating the entropy inside the EdgeBERT accelerator is a paltry 0.02%–0.78% of a fully optimized inference pass, as the vector MAC scales. Therefore, early stopping assessment is performed with negligible additional latency overhead. When the ALBERT inference is fully optimized, the EdgeBERT hardware reaps up to 2.7× greater energy saving advantages compared to those of the mGPU.

Table V breaks down the area and power contributions inside the EdgeBERT accelerator which occupies 5.11mm^2 of area while consuming an average power of 110.5mW. Despite accounting for more than a quarter of the total accelerator storage capacity, the ReRAM NVMs for embedding storage consume only 3.5% and 5.7% of the memory area and power, respectively.

C. Benefits of Non-Volatile Embeddings Storage

BERT word embeddings make a natural fit for non-volatile storage given that the EdgeBERT methodology freezes them during fine-tuning and reuses them during inference across NLP domains. By virtue of this scheme, we have established a compact 1.73MB baseline wherein the bitmask of ALBERT embeddings is stored in a SLC ReRAM buffer while the nonzero parameters are stored in a 2-bit per cell (MLC2) ReRAM buffer.

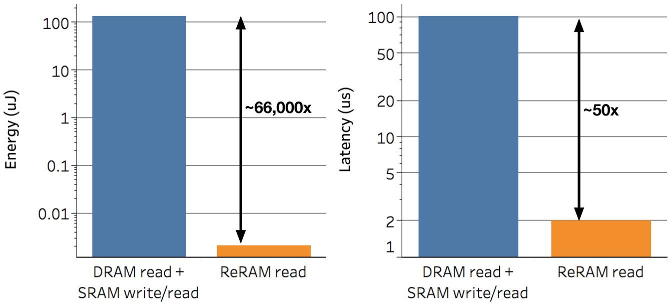

Fig. 11: Energy and latency costs of reading all embedding weights after system power-on. Storing embeddings on ReRAMs gives EdgeBERT significant energy and latency advantages compared to the conventional approach requiring DRAM read followed by SRAM write/read.

Fig. 11 illustrates the immense gains of leveraging this eNVM configuration during single-batch inference after SoC power-on. In EdgeBERT, ALBERT embeddings would only need to be read from the integrated ReRAM buffers due to being statically pre-loaded. The conventional operation dictates reading the embedding weights from off-chip DRAM, then writing them to dedicated on-chip volatile SRAM memories so they can be reused for future token identifications. The EdgeBERT approach enforces a latency and energy advantage that is, respectively, 50× and 66,000× greater than the overhead costs in the conventional operation. The non-volatility of this embedded storage means that these benefits are bound to further scale with the frequency of power cycles.

VII. Conclusion

As newer Transformer-based pre-trained models continue to generate impressive breakthroughs in language modeling, they characteristically exhibit complexities that levy hefty latency, memory, and energy taxes on resource-constrained embedded platforms. EdgeBERT provides an in-depth and principled methodology to alleviate these computational challenges in both the algorithm and hardware architecture layers.

To the best of our knowledge, this is the first work to curate, combine, and carefully balance a rich set of latency and memory reduction techniques for highly efficient ALBERT inference, namely, entropy-based early stopping, adaptive attention span, movement and magnitude pruning, and adaptive floating-point quantization – achieving up to 2.4× and 13.4× inference latency and memory savings, respectively, with less than 1%-pt drop in accuracy. EdgeBERT decouples the tensions between these algorithms to unlock impressive performance and efficiency advantages via their careful application in different parts of the model and application to different extents. We further exploit and optimize the structure of eNVMs in order to store the shareable multi-task embedding parameters, granting EdgeBERT significant performance and energy savings from system power-on.

To maximize on-chip performance, we prototype a hardware accelerator, augmented with ReRAM storage, that provides specialized and efficient datapath support for the latency-alleviating optimizations while simultaneously reducing energy expenditures with compressed sparse executions. The EdgeBERT accelerator notably consumes 5.2× and 163× lower energy consumption than that of an un-optimized accelerator baseline and an Nvidia TX2 mobile GPU, respectively.

ACKNOWLEDGEMENT

This work was supported in part by the Application Driving Architectures (ADA) Research Center, a JUMP Center cosponsored by SRC and DARPA, DSSoC programs, NSF Awards 1704834 and 1718160, Intel Corp., and Arm Inc.

REFERENCES

[1] *Catapult High-Level Synthesis*, accessed Oct 1, 2020. [Online]. Available: https://www.mentor.com/hls-lp/catapult-high-level-synthesis

[2] *Jetson TX2 Module*, accessed Oct 1, 2020. [Online]. Available: https://developer.nvidia.com/embedded/jetson-tx2

[3] V. Akhlaghi, A. Yazdanbakhsh, K. Samadi, R. K. Gupta, and H. Esmaeilzadeh, "Snapea: Predictive early activation for reducing computation in deep convolutional neural networks," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, 2018, p. 662–673.

[4] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proceedings of the 43rd International Symposium on Computer Architecture*, 2016.

[5] L. J. Ba *et al.*, "Layer normalization," *ArXiv*, vol. abs/1607.06450, 2016.

[6] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015*, 2015. [Online]. Available: http://arxiv.org/abs/1409.0473

[7] M. Chang, J. Wu, T. Chien, Y. Liu, T. Yang, W. Shen, Y. King, C. Lin, K. Lin, Y. Chih, S. Natarajan, and J. Chang, "19.4 embedded 1mb reram in 28nm cmos with 0.27-to-1v read using swing-sample-and-couple sense amplifier and self-boost-write-termination scheme," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014.

[8] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 269–284.

[9] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 367–379.

[10] G. F. Close, U. Frey, J. Morrish, R. Jordan, S. C. Lewis, T. Maffitt, M. J. BrightSky, C. Hagleitner, C. H. Lam, and E. Eleftheriou, "A 256-mcell phase-change memory chip operating at 2+ bit/cell," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 6, pp. 1521–1533, 2013.

[11] Cong Xu, Dimin Niu, N. Muralimanohar, N. P. Jouppi, and Yuan Xie, "Understanding the trade-offs in multi-level cell reram memory design," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–6.

[12] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[13] M. Donato, L. Pentecost, D. Brooks, and G. Wei, "Memti: Optimizing on-chip nonvolatile storage for visual multitask inference at the edge," *IEEE Micro*, vol. 39, no. 6, pp. 73–81, 2019.

[14] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[15] R. Eisele. (2016) The log-sum-exp trick in machine learning. [Online]. Available: https://www.xarg.org/2016/06/the-log-sum-exp-trick-in-machine-learning/

[16] P. Ganesh, Y. Chen, X. Lou, M. H. A. Khan, Y. Yang, D. Chen, M. Winslett, H. Sajjad, and P. Nakov, "Compressing large-scale transformer-based models: A case study on bert," *ArXiv*, vol. abs/2002.11985, 2020.

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

[18] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *SIGARCH Comput. Archit. News*, vol. 44, no. 3, Jun. 2016.

[19] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[20] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. W. Fletcher, "Ucnn: Exploiting computational reuse in deep neural networks via weight repetition," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, 2018, p. 674–687.

[21] HLSLibs, "Open-source high-level synthesis ip libraries," Tech. Rep. [Online]. Available: https://github.com/hlslibs

[22] F. N. Iandola, A. E. Shaw, R. Krishna, and K. Keutzer, "Squeezebert: What can computer vision teach nlp about efficient neural networks?" *ArXiv*, vol. abs/2006.11316, 2020.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[24] A. Jain, A. Phanishayee, J. Mars, L. Tang, and G. Pekhimenko, "Gist: Efficient data encoding for deep neural network training," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18, 2018, p. 776–789.

[25] J. Johnson, "Rethinking floating point for deep learning," *CoRR*, vol. abs/1811.01721, 2018. [Online]. Available: http://arxiv.org/abs/1811.01721

[26] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, June 2017, pp. 1–12.

[27] B. Khailany, E. Khmer, R. Venkatesan, J. Clemons, J. S. Emer, M. Fojtik, A. Klinefelter, M. Pellauer, N. Pinckney, Y. S. Shao, S. Srinath, C. Torng, S. L. Xi, Y. Zhang, and B. Zimmer, "A modular digital vlsi flow for high-productivity soc design," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: ACM, 2018, pp. 72:1–72:6. [Online]. Available: http://doi.acm.org/10.1145/3195970.3199846

[28] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *ArXiv*, vol. abs/1909.11942, 2020.

[29] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

[30] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon: An instruction set architecture for neural networks," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16, 2016, p. 393–405.

[31] T. Liu, T. H. Yan, R. Scheuerlein, Y. Chen, J. K. Lee, G. Balakrishnan, G. Yee, H. Zhang, A. Yap, J. Ouyang, T. Sasaki, S. Addepalli, A. Al-Shamma, C. Chen, M. Gupta, G. Hilton, S. Joshi, A. Kathuria, V. Lai, D. Masiwal, M. Matsumoto, A. Nigam, A. Pai, J. Pakhale, C. H. Siau, X. Wu, R. Yin, L. Peng, J. Y. Kang, S. Huynh, H. Wang, N. Nagel, Y. Tanaka, M. Higashitani, T. Minvielle, C. Gorla, T. Tsukamoto, T. Yamaguchi, M. Okajima, T. Okamura, S. Takase, T. Hara, H. Inoue, L. Fasoli, M. Mofidi, R. Shrivastava, and K. Quader, "A 130.7mm2 2-layer 32gb reram memory device in 24nm technology," in *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, 2013.

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

[33] S. Lu, M. Wang, S. Liang, J. Lin, and Z. Wang, "Hardware accelerator for

multi-head attention and position-wise feed-forward in the transformer," *ArXiv*, vol. abs/2009.08605, 2020.

[34] J. McCaffrey. (2016) The max trick when computing softmax. [Online]. Available: https://jamesmccaffrey.wordpress.com/2016/03/04/the-max-trick-when-computing-softmax/

[35] J. S. McCarley, "Pruning a bert-based question answering model," *ArXiv*, vol. abs/1910.06360, 2019.

[36] P. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?" *ArXiv*, vol. abs/1905.10650, 2019.

[37] P. Nayak, "Understanding searches better than ever before," Tech. Rep., 2019. [Online]. Available: https://blog.google/products/search/search-language-understanding-bert/

[38] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 27–40.

[39] E. Park, D. Kim, and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, 2018, p. 688–698.

[40] J. Park, H. Yoon, D. Ahn, J. Choi, and J.-J. Kim, "Optimus: Optimized matrix multiplication structure for transformer neural network accelerator," in *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds., 2020, vol. 2, pp. 363–378.

[41] L. Pentecost, M. Donato, B. Reagen, U. Gupta, S. Ma, G.-Y. Wei, and D. Brooks, "Maxnvm: Maximizing dnn storage density and inference efficiency with sparse encoding and error mitigation," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52, 2019.

[42] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100, 000+ questions for machine comprehension of text," *ArXiv*, vol. abs/1606.05250, 2016.

[43] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[44] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 267–278.

[45] M. Riera, J.-M. Arnau, and A. González, "Computation reuse in dnns by exploiting input similarity," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, 2018, p. 57–68.

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

[47] V. Sanh, T. Wolf, and A. M. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," in *34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020. [Online]. Available: http://arxiv.org/abs/2005.07683

[48] R. Schwartz, G. Stanovsky, S. Swayamdipta, J. Dodge, and N. A. Smith, "The right tool for the job: Matching model and instance complexities," in *ACL*, 2020.

[49] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. Mahoney, and K. Keutzer, "Q-bert: Hessian based ultra low precision quantization of bert," in *AAAI*, 2020.

[50] S. Sukhbaatar, E. Grave, P. Bojanowski, and A. Joulin, "Adaptive attention span in transformers," in *ACL*, 2019.

[51] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: a compact task-agnostic bert for resource-limited devices," in *ACL*, 2020.

[52] T. Tambe, E.-Y. Yang, Z. Wan, Y. Deng, V. Reddi, A. M. Rush, D. Brooks, and G.-Y. Wei, "Adaptivfloat: A floating-point based data type for resilient deep learning inference," *ArXiv*, vol. abs/1909.13271, 2019.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[54] S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey, and A. Raghunathan, "Scaledeep: A scalable compute architecture for learning and evaluating deep networks," *SIGARCH Comput. Archit. News*, 2017.

[55] B. Venkatesan *et al.*, "Magnet : A modular accelerator generator for neural networks," in *ICCAD*, 2019.

[56] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," *CoRR*, vol. abs/1804.07461, 2018. [Online]. Available: http://arxiv.org/abs/1804.07461

[57] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, "Hat: Hardware-aware transformers for efficient natural language processing," in *ACL*, 2020.

[58] J. Weng, S. Liu, V. Dadu, Z. Wang, P. Shah, and T. Nowatzki, "Dsagen: Synthesizing programmable spatial accelerators," in *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ser. ISCA '20, 2020.

[59] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv*, vol. abs/1910.03771, 2019.

[60] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin, "Deebert: Dynamic early exiting for accelerating bert inference," *ArXiv*, vol. abs/2004.12993, 2020.

[61] R. Yazdani, M. Riera, J. Arnau, and A. González, "The dark side of dnn pruning," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 790–801.

[62] A. H. Zadeh and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *53rd IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020.

[63] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8bert: Quantized 8bit bert," in *33rd Conference on Neural Information Processing Systems (NeurIPS)*, 2019.