EdgeBERT: Sentence-Level Energy Optimizations for Latency-Aware Multi-Task NLP Inference

Thierry Tambe
Harvard University
ttambe@g.harvard.edu

Coleman Hooper
Harvard University
chooper@college.harvard.edu

Lillian Pentecost
Harvard University
lillian_pentecost@g.harvard.edu

Tianyu Jia
Harvard University
tjia@g.harvard.edu

En-Yu Yang
Harvard University
enyu_yang@g.harvard.edu

Marco Donato
Tufts University
marco.donato@tufts.edu

Victor Sanh
Hugging Face
victor@huggingface.co

Paul N. Whatmough
Arm Research
paul.whatmough@arm.com

Alexander M. Rush
Cornell University
arush@cornell.edu

David Brooks
Harvard University
dbrooks@g.harvard.edu

Gu-Yeon Wei
Harvard University
gywei@g.harvard.edu

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 algorithm-hardware co-design for latency-aware energy optimizations for multi-task NLP. EdgeBERT employs entropy-based early exit predication in order to perform dynamic voltage-frequency scaling (DVFS), at a sentence granularity, for minimal energy consumption while adhering to a prescribed target latency. Computation and memory footprint overheads are further alleviated by employing a calibrated combination of adaptive attention span, selective network pruning, and floating-point quantization.

Furthermore, in order to maximize the synergistic benefits of these algorithms in always-on and intermediate edge computing settings, we specialize a 12nm scalable hardware accelerator system, integrating a fast-switching low-dropout voltage regulator (LDO), an all-digital phase-locked loop (ADPLL), as well as, high-density embedded non-volatile memories (eNVMs) wherein the sparse floating-point bit encodings of the shared multi-task parameters are carefully stored. Altogether, latency-aware multi-task NLP inference acceleration on the EdgeBERT hardware system generates up to 7×, 2.5×, and 53× lower energy compared to the conventional inference without early stopping, the latency-unbounded early exit approach, and CUDA adaptations on an Nvidia Jetson Tegra X2 mobile GPU, respectively.

MICRO '21, October 18–22, 2021, Virtual Event, Greece

ACM ISBN 978-1-4503-8557-2/21/10…$15.00
https://doi.org/10.1145/3466752.3480095

CCS CONCEPTS

• **Computer systems organization** → *Application specific integrated circuits.*

KEYWORDS

natural language processing, software and hardware co-design, latency-aware, embedded non-volatile memories

ACM Reference Format:
Thierry Tambe, Coleman Hooper, Lillian Pentecost, Tianyu Jia, En-Yu Yang, Marco Donato, Victor Sanh, Paul N. Whatmough, Alexander M. Rush, David Brooks, and Gu-Yeon Wei. 2021. EdgeBERT: Sentence-Level Energy Optimizations for Latency-Aware Multi-Task NLP Inference. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21), October 18–22, 2021, Virtual Event, Greece.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3466752.3480095

1 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 [16], 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 [58, 78], and it is now applied at a global scale in web search engines [52] with marked improvements in the quality of query results.

Advances in NLP models are also fueling the growth of intelligent virtual assistants, which leverage NLP to implement interactive voice interfaces. Currently, these applications are offloaded from the edge device to the cloud. However, they are naturally better suited

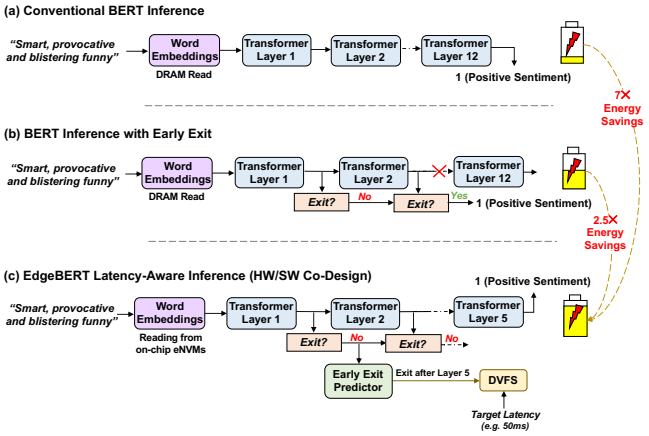


Figure 1: (a) Conventional BERT inference, (b) Conventional latency-unbounded BERT inference with early exit. (c) Proposed latency-bounded inference. The entropy result from the first layer is used to auto-adjust the accelerator supply voltage and clock frequency for energy-optimal operation while meeting an application end-to-end latency target.

to deployment on edge devices, where personal data can be kept private and the round trip latency to the cloud is removed. However, the impressive performance of BERT comes with a heavy compute and memory cost, which makes on-device inference prohibitive. Most significantly, the BERT base model consumes a staggering 432 MB of memory in native 32-bit floating-point (FP32).

Therefore, the goal of deploying BERT on edge/mobile devices is challenging and requires tight co-design of the BERT model optimizations with dedicated hardware acceleration and memory system design. The constraints on mobile can be quite different to the datacenter scenario, where BERT has been mainly deployed to date. Firstly, since we are dealing with user input, we need to meet real time throughput requirements to prevent a noticeable lag to the user. Secondly, energy consumption is a critical concern on mobile devices, both for the model inference and also the associated data movement cost. A number of prior works have been proposed to reduce BERT storage and computation overheads [21]. In fact, most of the compression techniques (weight pruning [49], distillation [63], quantization [68, 89]) originally proposed for convolutional and recurrent neural networks (CNNs, RNNs) have been independently applied to Transformer-based DNNs.

In this work, we present *EdgeBERT*, a principled latency-driven approach to accelerate NLP workloads with minimal energy consumption via early exit prediction, dynamic voltage-frequency scaling (DFVS), and non-volatile memory bitmask encoding of the shared word embeddings.

In conventional BERT inference (Fig. 1a), the final classification result is generated by the last Transformer layer. Early exit mechanisms [65, 73, 87, 90] (Fig. 1(b)) have been proposed to reduce the average energy and latency. The early exit entropy, which is a probabilistic measure of the classification confidence, is evaluated at the output of each computed Transformer layer and the inference exits when the entropy value falls below a pre-defined threshold. While this approach can appreciably reduce computation and energy costs, the achieved latency can vary drastically from one input sentence to another, potentially violating the strict real time latency constraint of the application. In contrast, EdgeBERT uses this upper-bound latency and the target entropy as optimization constraints, and then dynamically auto-adjusts the accelerator supply voltage and clock frequency to minimize energy consumption (Fig. 1(c)), while meeting the real time throughput requirement. Since energy scales quadratically with V_{DD} and linearly with the number of computation cycles, our DVFS algorithm finds the lowest possible frequency/voltage, while also minimizing the total number of FLOPs via adaptive attention span predication.

While the benefits of early exit and attention predications can be reaped on commodity GPUs, we unlock additional energy savings by co-designing the hardware datapaths. Specifically, we exploit these algorithmic optimizations in the EdgeBERT accelerator system, which integrates a fast-switching low-dropout (LDO) voltage regulator and an all-digital phase-locked loop (ADPLL) for DVFS adjustments. The EdgeBERT accelerator uses bit-mask encoding for compressed sparse computations, while optimizing key operations (entropy assessment, layer normalization, softmax and attention masking) for numerical stability and energy efficiency.

Furthermore, edge/IoT devices operate intermittently which motivates powering down as much as possible. The model's weights, typically stored in on-chip SRAMs, either have to be reloaded from DRAM each wake up cycle or the on-chip SRAMs storing the weights must be kept on, wasting leakage power [39]. Embedded non-volatile memories (eNVMs), which have shown considerable progress in recent years, offer great promise, if used judiciously, to eliminate the power penalty associated with intermittent operation. For this purpose, we perform monte-carlo fault injection simulations to identify robust and viable eNVM structures for storing the shared NLP multi-task parameters with bitmask encoding. Our resulting eNVM configuration significantly alleviates the energy and latency costs associated with multi-task intermediate computing by as much as 66,000× and 50×, respectively.

Altogether, EdgeBERT generates on average up to 7×, and 2.5× per-sentence energy savings compared to the conventional BERT inference, and latency-unaware early exit approaches, respectively.

This paper therefore makes the following contributions:

- We propose EdgeBERT, a novel algorithm-hardware co-design approach to enable latency-bound NLP workloads on resource-constrained embedded devices.
- Recognizing that BERT word embeddings are shared across NLP tasks, we significantly alleviate off-chip communication costs by identifying viable and robust multi-level eNVM structures for storing the multi-task word embeddings.
- Leveraging the insights from this broad analysis, we propose and design a 12nm accelerator that integrates a fast-switching LDO, an ADPLL, and a compressed sparse hardware accelerator that efficiently computes the DVFS, entropy, and adaptive attention span predication algorithms and other key Transformer operations using specialized datapaths.
- We evaluate the energy consumption of latency-bound inference on four NLP tasks, and find that the EdgeBERT hardware accelerator system generates up to 7×, 2.5×, and 53× lower energy compared to the unoptimized baseline inference without early exit, the conventional latency-unaware

early exit approach, and CUDA adaptations on an Nvidia Jetson Tegra X2 mobile GPU respectively.

2 BACKGROUND

2.1 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 [78]. 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).

2.2 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 [44] 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 [63] and MobileBERT [70] leverage knowledge distillation to reduce BERT size by 1.7× and 4.3×, respectively, with iso-accuracy. SqueezeBERT [29] substitutes several operations in the Transformer encoder with 1D grouped convolutions achieving 4× speedup while being 2× smaller. Q8BERT [89] 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 [37] yields the smallest footprint to date for a compressed BERT variant with only 12M parameters, with 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 has the same architecture as the legacy Transformer network [75], 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 variant as an efficient baseline. This work further pursues strategies to reduce latency and storage requirements to suit embedded hardware platform constraints.

3 ALLEVIATING TRANSFORMER MEMORY AND COMPUTATION COSTS

An accelerator's energy consumption can be abstracted as:

$$Energy \propto \alpha C V_{DD}^2 N_{cycles}$$

where α, C, V_{DD} and N_{cycles} are the switching activity factor, the effective wire and device capacitance, the supply voltage, and the required number of clock cycles to complete the inference,

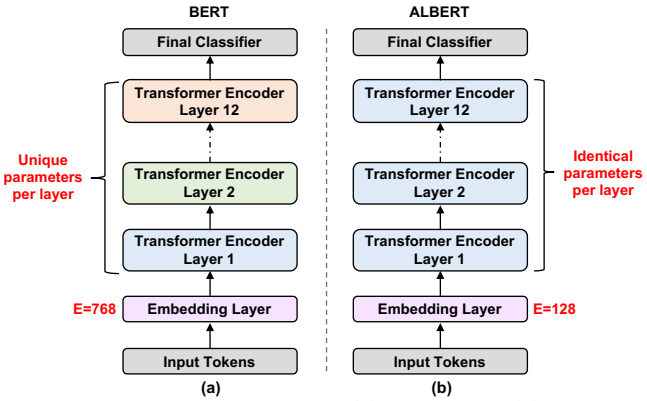


Figure 2: Comparison between (a) BERT, and (b) ALBERT base models. ALBERT uses a smaller embedding size and its Transformer encoder layers share the same parameters.

respectively. While the DVFS algorithm (Sec. 5.2) lowers the energy quadratically by bringing V_{DD} down to the lowest optimal voltage, in this section, we explore avenues to further reduce the energy by minimizing α, C, and N_{cycles}.

For this purpose, we carefully incorporate into the multi-task ALBERT inference: 1) adaptive attention span predication and early exit which reduce N_{cycles}; 2) network pruning, which ultimately reduces α; and 3) floating-point quantization helping decrease C, altogether with minimal accuracy degradation. While briefly describing these optimizations individually in this section, we provide a reasoned methodology for applying them to the ALBERT model, as shown in Fig. 3.

3.1 Entropy-based Early Exit

The motivation behind early exit (EE) is to match linguistically complex sentences with larger (or deeper) models and simple sentences with smaller (or shallower) models [13, 87]. This is typically done by adding a lightweight classifier at the output of the 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, E_T, in order to assess whether the prediction should exit or continue inference in the next Transformer encoder layer. The entropy metric quantifies the amount of uncertainty in the data. Smaller entropy values at a Transformer layer output implies greater confidence in the correctness of the classification result. The entropy H on sample x is estimated as:

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

The early exit condition is met when $H(x) < E_T$. Therefore, the larger E_T becomes, the earlier the sample will exit (i.e. N_{cycles} becomes smaller) with potentially lower accuracy.

In this work, we modify the conventional EE inference approach by predicting the early exit layer from the output of the first Transformer layer in order to run the rest of the network computation in an energy-optimal and latency-bounded manner (Sec. 5).

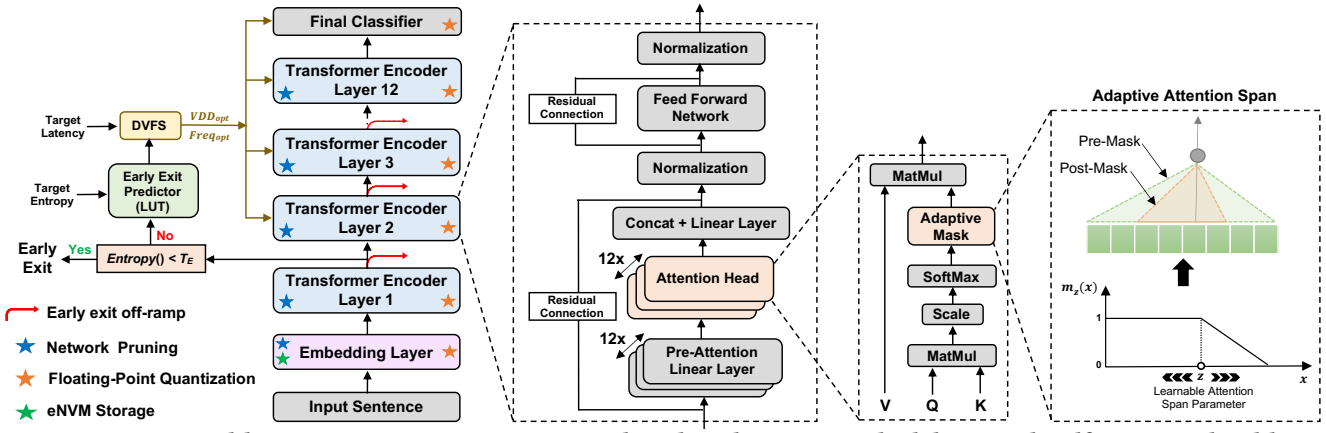


Figure 3: Memory and latency optimizations incorporated in the EdgeBERT methodology. Each self-attention head learns its own optimal attention span. Network pruning is performed on all Transformer encoders. The embedding layer is stored in non-volatile memory. Floating-point quantization is applied to all weights and activations. During real-time on-device execution, DVFS is performed for latency-bounded inference.

3.2 Adaptive Attention Span

The attention mechanism [8] is a powerful technique that 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 [51]. 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 reach of every attention head.

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 [69]. 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 in order to re-modulate their saliencies. The optimal span is automatically learned during the fine-tuning process by adding back the average loss from the reduced span to the training cross-entropy loss.

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 1 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 (which linearly correlates with N_{cycles}) 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. Notably, all the computations inside any of the twelve attention head units can be effectively skipped in case its associated attention span mask is 100% null. The EdgeBERT accelerator takes advantage of this observation in a proactive manner during inference in the custom hardware (Sec. 7.4.1).

Table 1: Learned spans of every attention head in ALBERT. Baseline Acc: MNLI=85.16, QQP=90.76, SST-2=92.20, QNLI=89.48

	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

3.3 Network Pruning

The EdgeBERT hardware accelerator (Sec. 7) executes sparse computations and saves energy by gating MACs whenever input operands are null. Therefore, the extent to which we can prune the ALBERT model, without appreciable accuracy loss, determines the overall accelerator energy efficiency.

In this work, we consider both movement pruning [64] and the well-known magnitude pruning [25] methods. Movement pruning is a first-order pruning technique that is applied during model fine-tuning which eliminates weights that are dynamically shrinking towards 0 (i.e., according to the movement of the values). In some cases, magnitude pruning may be a sub-optimal method to use during transfer learning, as pre-trained weights closer to zero may have a high chance of being eliminated regardless of the fine-tuning requirement. We observe that 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. Therefore, choosing between the two pruning techniques would depend on the per-task tolerance to increasing sparsity levels. We note that magnitude pruning is always applied to the ALBERT embedding layer in order to enforce uniformity in the data during multi-domain on-chip acceleration – as using movement pruning on the embedding layer would make its weights unique for each NLP domain, thereby forgoing opportunities for data reuse in hardware.

3.4 Floating-Point Quantization

DNN algorithmic resilience allows for parameters to be represented in lower bit precision without accuracy loss. Fixed-point or integer quantization techniques, commonly adopted in CNN models, suffer from limited range and may be inadequate for NLP models, whose weights can be more than an order of magnitude larger [72]. This phenomenon is owed to layer normalization [7], which is commonly adopted in NLP models and has invariance properties that do not reparameterize the network – unlike batch normalization [30], which produces a weight normalization side effect in CNNs.

In this work, we employ floating-point based quantization, which provides 2× higher dynamic range compared to integer datatypes [32]. Both weights and activations are quantized across ALBERT layers to 8-bit precision. We also performed a search on the optimal exponent bit width to satisfy the dynamic range requirements of the ALBERT model. Setting the floating-point exponent space to 4 bits within the 8-bit word size, with the exponent being scaled at a per-layer granularity, provided the best accuracy performance across NLP tasks.

4 NON-VOLATILE MEMORY STORAGE OF SHARED PARAMETERS

In contrast to task-specific encoder weights, word embedding parameters are deliberately fixed during fine-tuning and reused across different NLP tasks. We seek to avoid the energy and latency costs of reloading the word embeddings from off-chip memory for different tasks by storing these shared parameters in embedded non-volatile memories (eNVMs). eNVM storage also enables energy-efficient intermittent computing because the embedding weights will be retained if and when the system-on-chip powers off between inferences. However, despite their compelling storage density and read characteristics, eNVMs exhibit two main drawbacks: potentially high write cost (in terms of energy and latency) and decreased reliability, particularly in multi-level cell (MLC) configurations [15]. Fortunately, the word embeddings are 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 DNN task accuracy [56]. On the other hand, encoder weights need to be updated when switching across different NLP tasks. To prevent the energy and latency degradation that would follow from updating the encoder weight values in eNVMs, we map the natural partition of shared and task-specific parameters to eNVMs and SRAMs, respectively [17].

4.1 eNVM Modeling Methodology

This work specifically considers dense, energy-efficient Resistive RAM (ReRAM) arrays [10, 43] as an on-chip storage solution for shared embedding parameters. 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 [14], but is beyond the scope of this work.

We evaluate the robustness of storing the 8-bit quantized word 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 [15], and evaluate 100 fault injection trials per storage configuration to identify robust eNVM storage solutions. We leverage and extend Ares [59], which is an existing open-source fault injection framework for quantifying the resilience of DNNs.

Table 2: Results of fault injection simulations modeling impact of ReRAM embedding storage on 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	

Algorithm 1: Conventional early exit inference

Input: E_T := target entropy
for *input sentence* $i = 0$ *to* n **do**
  for *encoder layer* $l = 1$ *to* 12 **do**
    $z_l = f(x; \theta | VDD_{nom}, Freq_{max})$
    if $entropy(z_l) < E_T$ **then**
      exit inference

After pruning, we store non-zero 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 SLC devices, while we experiment with MLC storage for the non-zero data values [56].

4.2 Optimal eNVM Configuration

Table 2 uncovers exceptional resilience to storing word embeddings in MLC ReRAM. Across many fault injection trials, we observe that MLC2 (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 system leverages MLC2 ReRAMs for word embedding storage (Sec.7).

5 EDGEBERT'S LATENCY-AWARE INFERENCE

The conventional BERT inference (Algorithm 1) with early exit (EE) can significantly reduce BERT inference latency. To further reduce the energy consumption for NLP inference, a latency-aware inference scheme leveraging the EE predictor and dynamic voltage and frequency scaling (DVFS) is proposed to minimize end-to-end per-sentence energy consumption while satisfying the real-time latency target.

Algorithm 2: EdgeBERT latency-aware inference. Computations exit at the predicted exit layer or earlier.

Input: T := per-sentence latency target, E_T := entropy target

for *input sentence* $i = 1$ *to* n **do**
  for *encoder layer* $l = 1$ **do**
    $z_l = f(x; \theta | VDD_{nom}, Freq_{max})$
    if $entropy(z_l) < E_T$ **then**
      exit inference
    else
      $L_{predict} = LUT(entropy(z_1), E_T)$
      $VDD_{opt}, Freq_{opt} = DVFS(L_{predict}, T)$
  for *encoder layer* $l = 2$ *to* $L_{predict}$ **do**
    $z_l = f(x; \theta | VDD_{opt}, Freq_{opt})$
    if $entropy(z_l) < E_T$ **then**
      exit inference
  exit inference

5.1 Methodology

DVFS is a widely used technique to dynamically scale down the voltage and frequency for less computationally intensive workloads. In the past, DVFS has been widely deployed in commercial CPUs [74], [28] and GPUs [50]. However, these schemes typically adjust the voltage and frequency at a coarse granularity at workload-level. In the era of AI, DVFS has started to be explored for DNN accelerators [38]. For example, a recent state-of-the-art AI chip has reported per-layer DVFS to save energy [3]. In this work, we explore a fine-grained sentence-level DVFS to reduce the energy consumption for NLP inference while meeting the latency target.

The proposed early exit -based latency-aware inference methodology is illustrated in Algorithm 2. The inference of a sentence starts at nominal voltage and maximum frequency, and the entropy value is calculated at the output of the first Transformer encoder layer. The entropy result is then sent to a trained classifier (EE predictor) to predict which following encoder layer should early exit (e.g. early exit at encoder layer 6 before the final encoder layer 12). Based on the predicted early exit layer, the voltage and frequency is scaled down to proper energy-optimal setting for the rest of encoder layers (e.g. encoder layer 2 to 6) while meeting the latency target for each sentence. This scheme produces a quadratic reduction in the accelerator power consumption.

In our work, the EE predictor is a ReLU-activated five-layer perceptron neural network with 64 cells in each of the hidden layers. It takes the entropy of encoder layer 1 as input and forecasts the early exit Transformer layer which has an entropy below the desired threshold. The neural network architecture of the EE predictor was empirically searched with the goal of minimizing the difference between the predicted and the true entropy-based exit layer. For this purpose, we constructed parallel training and test datasets containing the entropy values at the output of the 12 Transformer layers during evaluation on the GLUE benchmarks.

The EE predictor is distilled as a lookup table (LUT) leading to negligible one-time (per-sentence) computational overhead. Furthermore, implementing the EE predictor as a LUT simplifies its hardware operation. As the neural network based LUT is error-prone, it may predict a higher exit layer than necessary. Therefore, during the inference, the entropy is checked after each encoder layer for early stopping until the predicted layer. If the computed entropy becomes lower than the exit threshold before the predicted encoder

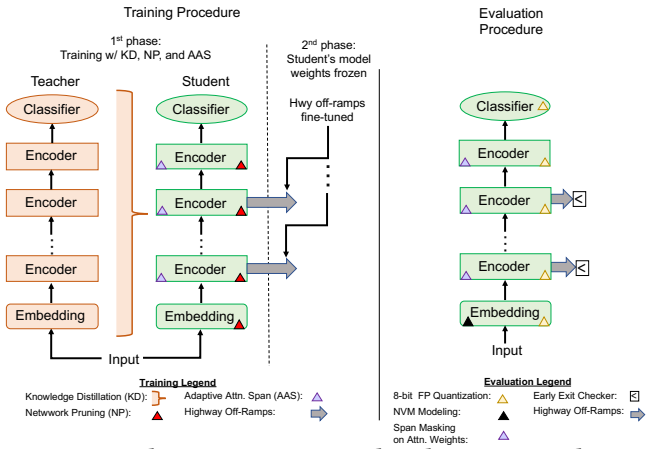


Figure 4: EdgeBERT training and evaluation procedure.

layer, the inference will terminate at that early exit condition point. In case the inference reaches the predicted layer, termination occurs even if the entropy at that layer is still higher than the exit threshold in order to not violate timing constraints.

When assessing the impacts of using entropy prediction instead of traditional EE methods, we set a fixed accuracy degradation threshold of 1%, 2%, or 5% (relative to the inference accuracy of the full ALBERT model) and increased the entropy threshold until the accuracy dropped to the desired threshold. This allowed us to compare energy savings between entropy prediction and conventional EE for a fixed accuracy target. For the same accuracy threshold, the entropy threshold for entropy prediction was lower than the entropy threshold for conventional EE, leading to a slightly later average exit layer during inference. However, entropy prediction allows for DVFS since the maximum exit layer is known after the first layer, whereas with the conventional EE appproach, the maximum exit layer is always the final encoder layer. EdgeBERT latency-aware inference therefore achieves greater energy savings than the conventional EE approach by facilitating DVFS (Sec. 8.2.2).

5.2 On-chip DVFS system

To realize fast per-sentence DVFS, the on-chip DVFS system is developed and integrated within EdgeBERT. The DVFS system includes a DVFS controller, an on-chip synthesizable linear voltage regulator (LDO), and an all-digital PLL (ADPLL). Compared with the conventional workload-level DVFS [74], the proposed scheme adjusts voltage and frequency at a finer-grained sentence-level granularity. Based on the predicted early exit layer from the EE predictor, the required run cycles, N_{cycles}, for the rest of the encoder layers before early exit can be known. And, knowing the frontend elapsed time $T_{elapsed}$ up to the EE predictor within the per-sentence latency target T, the optimal running frequency can be calculated as follows:

$$Freq_{opt} = N_{cycles}/(T - T_{elapsed})$$

Meanwhile, the corresponding energy-optimal supply voltage, VDD_{opt}, is selected by the DVFS controller to achieve the lowest operational voltage value at $Freq_{opt}$. In the EdgeBERT accelerator system, this is done via indexing the look-up table containing

Table 3: Summary of optimization results in terms of achievable sparsity, attention span with early exit performance and accuracy implications. Baseline Acc: MNLI=85.16, QQP=90.76, SST-2=92.20, QNLI=89.48

					Conventional EE Approach		EdgeBERT Latency-Aware Inference		
	Embedding Sparsity (%)	Encoder Sparsity (%)	Avg. Attn. Span	Pct. Pt. Acc. Drop	Entropy Threshold	Avg. Exit Layer	Entropy Threshold	Avg. Predicted Exit Layer	Avg. Actual Exit Layer
MNLI	60	50	12.7	1%	0.4	8.55	0.31	11.00	8.91
				2%	0.49	8.00	0.34	10.52	8.61
				5%	0.65	6.89	0.47	8.37	7.34
QQP	60	80	11.3	1%	0.25	5.84	0.12	8.88	6.41
				2%	0.32	5.28	0.15	7.65	5.84
				5%	0.43	4.31	0.26	5.94	4.76
SST-2	60	50	18.4	1%	0.23	4.30	0.09	7.78	5.25
				2%	0.28	3.94	0.16	4.91	3.90
				5%	0.46	2.70	0.28	3.65	3.05
QNLI	60	60	21.5	1%	0.18	8.46	0.13	12	9.07
				2%	0.29	7.38	0.15	10.22	8.32
				5%	0.44	5.89	0.25	8.01	6.85

the ADPLL frequency/voltage sweep coordinates. The DVFS is performed for each real-time sentence inference due to its fast response time; the implementation details are shown in Sec. 7.4.3.

6 ALGORITHMIC SYNERGY

In order to quantify the different tradeoffs, and evaluate the synergistic impact on the model accuracy from the memory and latency optimizations, the eNVM modeling, and the EE predictor, we implemented the training and evaluation procedures illustrated in Fig. 4 on the base of HuggingFace's Transformers infrastructure [85].

6.1 Training and Evaluation Procedure

The training methodology consists of two phases. In the first phase, the model is pruned during fine-tuning: magnitude pruning is applied to the embedding layer and either movement or magnitude pruning is applied to the Transformer 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 learn their optimal spans. In the second training phase, we freeze the model's parameters prior to fine-tuning the early exit highway off-ramps.

At evaluation time, 8-bit floating-point quantization is applied on all the weights and activations. The quantized 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. Entropy prediction is then deployed along with early exit during inference according to Algorithm 2.

6.2 Impact on Model Accuracy, Computation, and Storage

Using the multi-step procedure illustrated in Fig. 4, we amalgamate into ALBERT the various memory and latency reduction techniques at training and evaluation times. Table 3 summarizes the generated benefits of the synergistic inference with the following main observations:

- EdgeBERT latency-aware inference provides comparable average exit layer for the same accuracy threshold as the conventional EE approach, while allowing the DVFS algorithm to reduce the frequency and voltage in accordance with the predicted exit layer.
- The EdgeBERT approach requires a lower entropy threshold than the conventional EE approach for the same accuracy target; this demonstrates that the we must predict conservatively due to the classification error introduced by the neural network-based entropy predictor.
- 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.

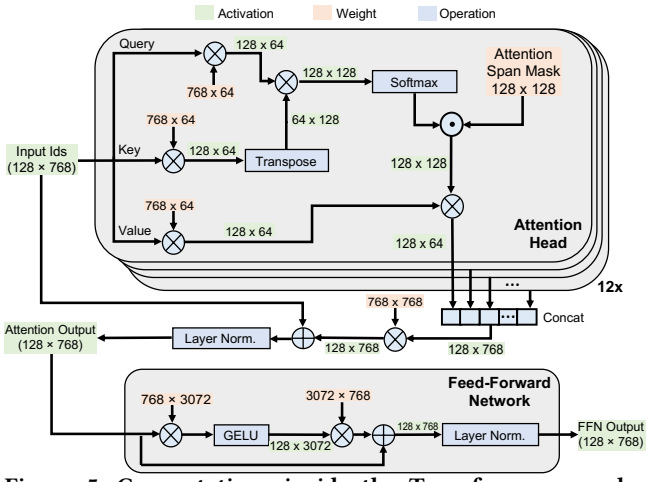


Figure 5: Computations inside the Transformer encoder with attention span modulation. Here, the input sequence is composed of 128 tokens. To simplify the computational diagram, the bias layers are not included.

7 THE EDGEBERT HARDWARE ACCELERATOR SYSTEM

7.1 Required Computations in ALBERT

The Transformer encoder is the backbone of ALBERT/BERT, consuming more than 95% of inference computations. Fig. 5 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 head units can be effectively skipped in case its associated attention span mask is 100% null. The EdgeBERT accelerator reaps this benefit by enforcing adaptive attention span masking during fine-tuning.

7.2 The EdgeBERT Accelerator System

In order to maximize the benefits of the latency and memory reduction techniques during latency-aware inference, we designed a scalable accelerator system that exploits these algorithms for compute and energy efficiency with the following key highlights:

- Specialized datapath support for *(i)* early exit assessment, *(ii)* softmax and attention span masking, and *(iii)* layer normalization. We notably reformulate their mathematical definitions in order to avoid numerical instability, and where possible, hardware components with long cyclic behaviors such as divisions.

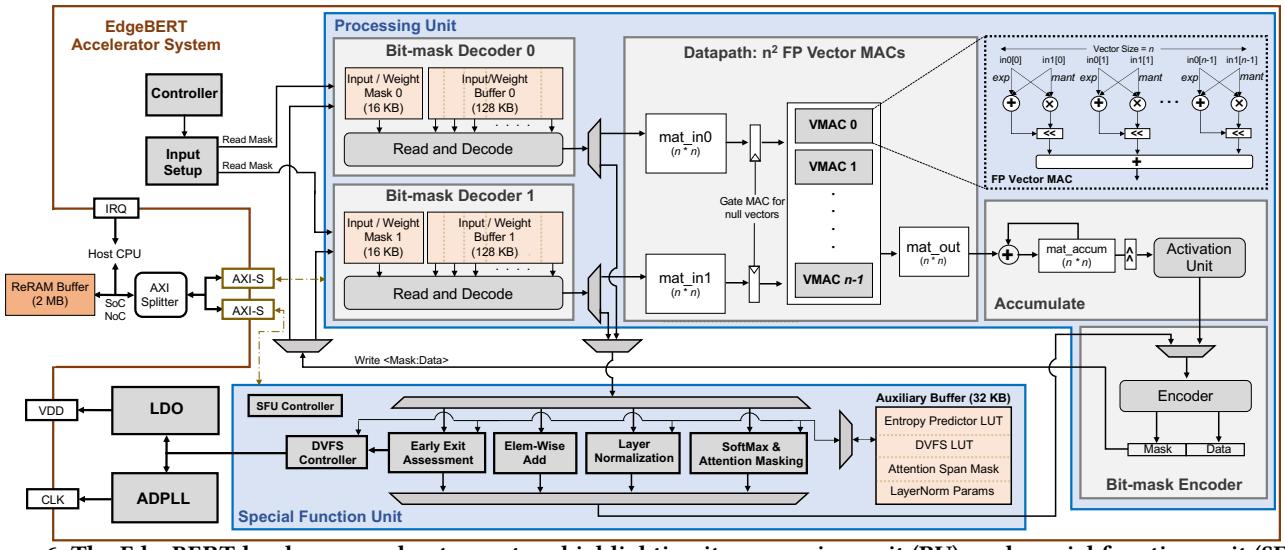


Figure 6: The EdgeBERT hardware accelerator system highlighting its processing unit (PU), and special function unit (SFU). A fast-switching LDO and fast-locking ADPLL are also integrated for latency-driven DVFS.

- Non-volatile and high density storage of the shared multi-task parameters substantially improves the accelerator's energy and area efficiency (Sec. 8.3).
- On-demand DVFS aided by the integration of a fast-locking ADPLL and a fast-switching LDO regulator.
- Compressed sparse execution via bitmask encoding.

The EdgeBERT hardware accelerator, illustrated in Fig. 6, consists of a processing unit (PU), a special function unit (SFU), a LDO and ADPLL for latency-bounded DVFS. The communication between the PU and SFU 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 SFU AXI-slave partitions. The multi-task embedding pruned weights and corresponding bitmask are stored in a 2MB ReRAM NVM buffer in order to avoid reloading them when powered on. Specifically, the bitmask embedding values are stored in a single-level cell (SLC) ReRAM configuration while the nonzero embedding parameters are kept in a 2-bit per cell (MLC2) ReRAM structure, according to the learnings from the NVM studies (Sec. 4).

7.3 Processing Unit

The processing unit (PU) is designed to execute matrix-matrix multiplications in linear layers and attention heads of ALBERT.

In the PU datapath in Fig. 6, 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 $n * n$ 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, 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 and decoding for compressing and decompressing the sparse matrix, and (2) implement skipping 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 the decoder 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 from the data vector before sending the compressed mask and data vector to one of the PU decoder blocks. To save energy, the PU datapath skips 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 1.65× in energy consumption (Sec. 8.2).

7.4 Special Function Unit

The special function unit (SFU) contains specialized datapaths that compute the EE assessment, DVFS control, element-wise addition, layer normalization, and softmax, all of which get invoked during the latency-aware EdgeBERT inference. The SFU also integrates a 32KB auxiliary buffer to house the EE and DVFS LUTs, the layer normalization parameters, and the multi-head attention span masks learned during the fine-tuning process. All the computations in the SFU are in 16-bit fixed-point format.

7.4.1 *Computing the Multi-Head 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 SFU softmax unit.

First, prior to computing an attention head, the SFU controller inspects its associated attention span mask in the auxiliary buffer. In case the attention span mask for an attention head is null, the SFU controller proactively cancels and skips entirely the sequence of computations required for that head, and directly writes zero in

Algorithm 3: Computing Softmax and Attn. Span Masking

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**
      $max = max(vec)$
  // Step 2: compute log-exponential-sum
  $sum_{exp} = 0$
  for $j = 0$ *to* $T - 1$ **do**
    $vec <= load(A_{[i][n*j:n*j+n-1]})$
    $sum_{exp} += sum(exp(vec - max))$
  $logsum_{exp} = ln(sum_{exp})$
  // Step 3: Get softmax and modulate with attn span mask
  for $j = 0$ *to* $T - 1$ **do**
    $vec <= load(A_{[i][n*j:n*j+n-1]})$
    $mask <= load(A_{M[i][n*j:n*j+n-1]})$
    $vec = exp(vec - max - logsum_{exp})$
    $vec = vec * mask$
    $store(vec) => A_{O[i][n*j:n*j+n-1]}$

the corresponding logical memory for its context vector stored in one of the PU decoder blocks.

In case the attention span mask for a head contains non-zero elements, the softmax unit takes advantage of the *LogSumExp* [19] and *Max* [48] tricks to vectorize the computation of the softmax function $SM()$ as:

$$SM(A_k) = exp[A_k - MAX_k(A) - ln(\sum_{k=1}^{K} exp(A_k - MAX_k(A)))] \quad (2)$$

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 softmax unit then performs element-wise multiplication between the resulting attention scores and the attention span mask as described in Algorithm 3.

7.4.2 *Performing Early Exit Assessment.*

The EE assessment unit computes the numerically-stable version of the entropy function from equation 1 as follows:

$$H(x_k) = \ln(\sum_{k=1}^{n} e^{x_k - MAX_k(x)}) - MAX_k(x) - \frac{\sum_{k=1}^{n} x_k e^{x_k - MAX_k(x)}}{\sum_{k=1}^{n} e^{x_k - MAX_k(x)}} \quad (3)$$

The EE assessment unit then compares the result with the register value for the entropy threshold. If the EE condition is met, the unit then triggers the accelerator's interrupt (IRQ). Otherwise, the SFU controller initiates the computation of the next Transformer encoder. In the case of latency-aware inference in intermittent mode, the EE assessment unit also indexes the EE predictor LUT stored in the auxiliary buffer in order to acquire the predicted exit layer value, which is then passed on to the DVFS controller.

7.4.3 *DVFS System.*

During each sentence inference, the DVFS FSM algorithm keeps track of the EE predictor result and manages the operating voltage and frequency accordingly. Based on the predicted early exit layer, the DVFS controller indexes the logical memory for the V/F LUT table in the auxiliary buffer and extracts the lowest corresponding supply voltage value, VDD_{opt}. At the same time, the DVFS controller simultaneously updates the ADPLL and LDO configuration registers with settings for $Freq_{opt}$ and VDD_{opt}, respectively.

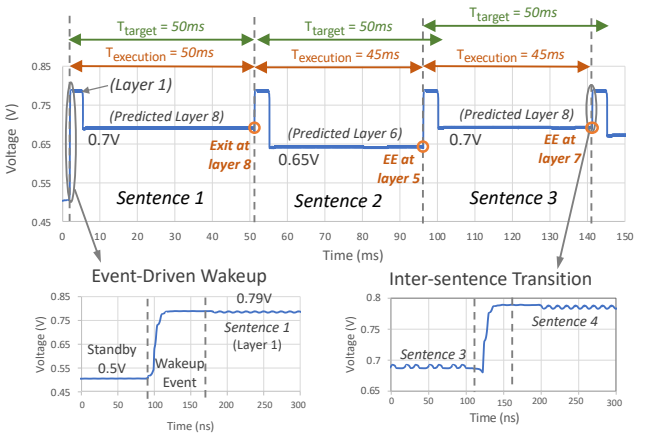


Figure 7: Spice simulations of LDO dynamic voltage adjustments. The LDO stabilizes voltage transitions within 100ns.

The synthesizable LDO is implemented using standard power header cells [9], and evenly distributed across the EdgeBERT accelerator. The LDO is able to scale the accelerator voltage from 0.5V to 0.8V with a 25mV step. With careful power header selection and layout resistance optimization, the LDO can achieve nearly linear scaled power efficiency and a fast response time of 3.8ns/50mV. The ADPLL is also implemented using all-synthesizable approach with the PLL architecture from the FASoC open-source SoC design framework [4]. Following a frequency update request, the all-digital PLL can relock the frequency in a fast speed with low power consumption. The 12nm performance specs of the LDO and ADPLL are shown in Table 4.

Table 4: Performance specs of LDO and ADPLL

LDO response time	$3.8ns/50mV$
LDO peak current efficiency	99.2% @ $I_{load,max}$
LDO $I_{load,max}$	$200mA$
ADPLL power	$2.46mW@1GHz$

Fig. 7 show the spice-level simulation of the DVFS for a consecutive sequence of sentence inference. For each sentence, the entropy is calculated after the computation of Encoder 1 and sent to the EE predictor to forecast the early exit layer. Based on the predicted early exit encoder and latency requirement for the sentence, the DVFS controller select the lowest voltage level and proper frequency to meet the latency requirement T_{target}. Therefore, the remaining encoder stages will compute at a lower voltage level to save energy. For example, the sentence 1 of Fig. 7, the early exit layer is predicted as 8. Therefore, the rest Encoders (i.e encoder 2-8) in sentence 1 are computed under a lower voltage 0.7V.

After the inference of the first sentence, the voltage level ramps back to nominal 0.8V for the computation of layer 1 in the following sentence. As on-chip integrated LDO is used, the transition and settling time is optimized to be within 100ns, which is negligible

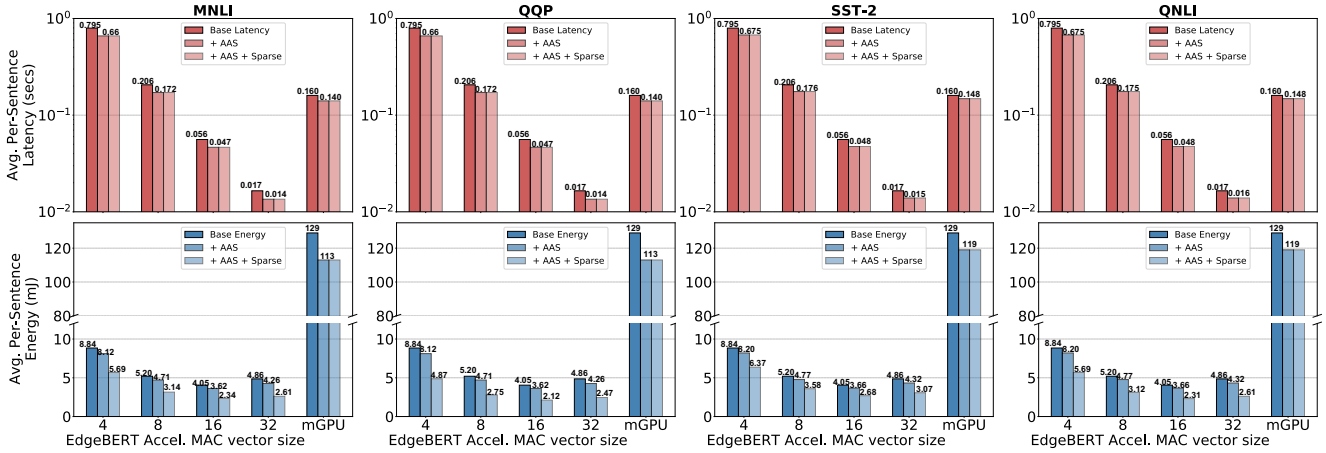


Figure 8: Average latency (top row) and energy (Bottom row) per sentence as the PU MAC vector size scales at max frequency (1GHz) and nominal voltage (0.8V), highlighting impact of adaptive attention span (AAS), and sparsity in weights and activations (Sparse) on the EdgeBERT accelerator and TX2 mGPU. MAC size of 16 yields the most energy efficient design.

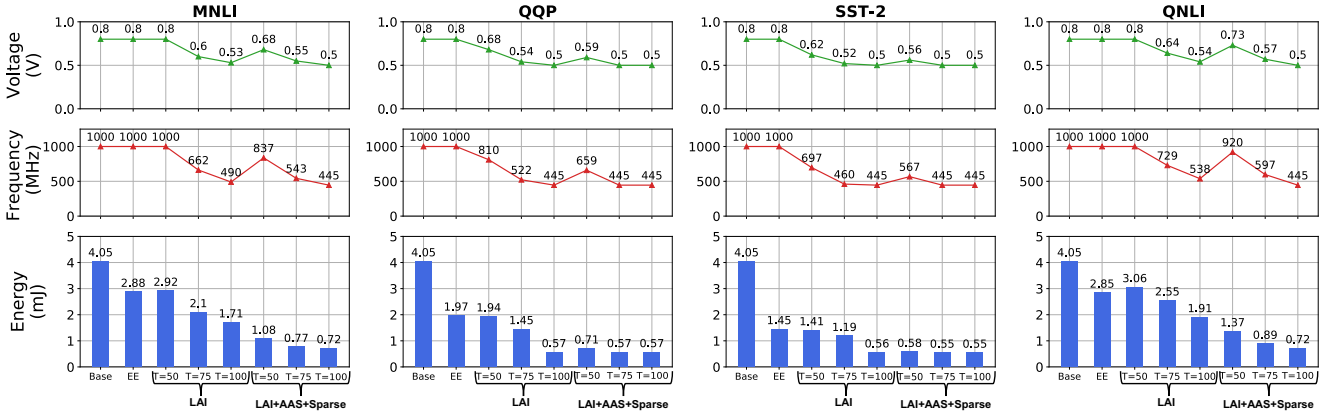


Figure 9: Average DVFS-driven supply voltage (top row) and clock frequency (middle row), as well as, generated energy expenditures (bottom row) of the EdgeBERT accelerator system with $n = 16$ during latency-aware inference (LAI), and latency-aware inference further improved with adaptive attention span and sparse execution (LAS+AAS+Sparse). Different latency targets of 50ms (T=50), 75ms (T=75), and 100ms (T=100) are used for LAI executions. Results are compared with the baseline 12-layer inference (Base) and the conventional early exit inference (EE).

considering the 50ms latency target. The computation of the next sentence starts once the voltage transition is settled. During idle times, EdgeBERT stays at standby 0.50V to save leakage energy.

8 HARDWARE EVALUATION

8.1 Design and Verification Methodology

The EdgeBERT accelerator is designed in synthesizable SystemC with the aid of hardware components from the MatchLib [34] and HLSLibs [27] open-source libraries. Verilog RTL is auto-generated by the Catapult high-level synthesis (HLS) tool [1] 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 decoder and auxiliary buffers are mapped to synthesized memories from a 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 [18] and its read latency, energy, and area are back-annotated into the accelerator results after scaling to a 12nm F^2 cell definition in order to match the process node used in the rest of the system.

To quantify the benefits of non-volatility (Sec. 8.3), we quantify the alternative cost of loading embeddings from off-chip using DRAMsim3 [40] to extract cycle-accurate LPDDR4 DRAM energy and latency metrics. 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].

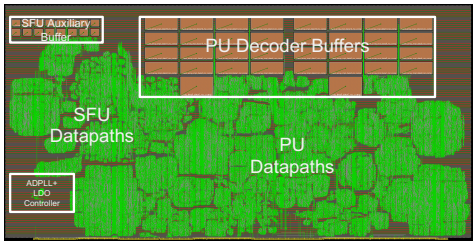
8.2 Performance, Energy and Area Analyses

8.2.1 *Design Space Exploration via MAC scaling.*

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 adaptive attention span mechanism by using the attained optimization results (i.e. embedding and encoder sparsity percentage, and attention span) reported in Table 3 in which the accuracy drop was at 1%-pt of the baseline. Adaptive

(a)

	PU Datapaths			SFU Datapaths			
	MACs	Bitmask Encoding	Bitmask Decoding	Softmax & Attn. Masking	Normalization	Element-Wise Addition	Early Exit Assessment
Latency	90.7%	3.2%	3.2%	1.1%	1.2%	0.13%	0.40%
Energy	98.8%	0.42%	0.33%	0.22%	0.14%	0.003%	0.04%



(b)

Blocks	Area (mm^2)	Power (mW)
PU Datapaths	0.52	36.9
SFU Datapaths	0.21	9.44
SRAM Buffers	0.50	33.6
ReRAM Buffers	0.15	3.48
ADPLL	0.01	2.46
Total	1.39	85.9

Figure 10: (a) Breakdown of latency and energy consumption in PU and SFU datapaths, and (b) 12nm physical layout, and area and power (@ 0.8V/1GHz) breakdown of the energy-optimal EdgeBERT accelerator (MAC size=16).

adaptive span is also applied to the mGPU platform in order to quantify and compare the extent of these benefits.

Fig. 8 shows that the per-sentence 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 increase in the datapath power consumption with $n = 32$ starts to subdue throughput 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. Compressed sparse execution in the PU datapath amounts to an additional 1.4–1.7× energy savings with QQP receiving the benefit the most. The EdgeBERT accelerator starts to outperform the mGPU processing time with $n = 16$. This energy-optimal design generates up 53× lower energy compared to the mGPU when all the optimizations are factored in.

Fig. 10 breaks down the latency, energy, area and power contributions inside the placed-and-routed, energy-optimal (n=16) EdgeBERT accelerator system which occupies 1.4mm^2 while consuming an average power of 86mW.

8.2.2 DVFS-based Latency-Aware Inference.

Fig. 9 shows the DVFS-controlled supply voltage and clock frequency, and the energy savings of the latency-aware inference (LAI) on the energy-optimal accelerator design (i.e. with MAC vector size $n = 16$) using latency targets between 50ms and 100ms (common latency thresholds for real-time human perception [57])). The results show that EdgeBERT optimized LAI achieves up to 7×, and 2.5× per-inference energy savings compared to the conventional inference (Base), and latency-unbounded early exit (EE) approaches, respectively, as seen in the SST-2 case. As AAS further cuts the number of computation cycles, we observe further relaxation of the supply voltage and clock frequency. At some latency targets (e.g., 75ms and 100ms in QQP and SST-2), further energy savings are not possible as V/F scaling bottoms out. To underscore the different contributions to energy savings, at 75ms latency target for example in the case of MNLI, early exit prediction, adaptive attention span, DVFS, sparse execution, and eNVMs account for 21%, 12%, 23%, 39%, and 5%, respectively, of the total accelerator energy reduction.

For stricter latency targets (e.g. < 20ms), the proposed DFVS-based scheme can be used by scaling up to even higher MAC vector sizes (i.e. $n \geq 32$).

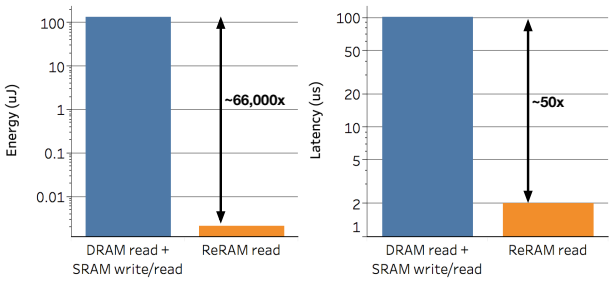


Figure 11: Costs of reading all embedding weights after system power-on. Storing embeddings in ReRAMs gives EdgeBERT significant energy and latency advantages compared to the conventional approach requiring DRAM read followed by SRAM write/read.

8.3 Benefits of NVM Embeddings Storage

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

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 preloaded. 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 can further scale with the frequency of power cycles.

9 RELATED WORK

Over the last decade, there has been extensive research on the design of high-performance and energy-efficient DNN hardware accelerators [5, 6, 11, 12, 22, 24, 26, 31, 33, 35, 36, 41, 42, 45, 47, 53, 54, 60–62, 66, 67, 71, 82–84, 86]. As these accelerators are increasingly deployed at all computing scales, there is additional interest in the hardware community to automatically generate designs [76, 77, 80, 81]. However, most of these works focus on CNN and RNN [20] 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 [23], which proposed a hardware architecture that reduces the number of computations in attention mechanisms via approximate and iterative candidate search. However, the A^3 scheme fetches the full and uncompressed data from DRAM before dynamically reducing computations in the hardware. In contrast, EdgeBERT learns the optimal attention search radius during the finetuning process and then leverages its very sparse mask to avoid unnecessary matrix multiplications. Therefore, our approach substantially eliminates DRAM accesses as the computation and memory optimizations are pre-learned before hardware acceleration. GOBO [88] focuses on

		GOBO [88]	Optimus [55]	A^3 [23]	SpAtten [79]	EdgeBERT (This work)
Model Compression	Pruning	✗	✓	✓	✓	✓
	Quantization	✓	✗	✗	✓	✓
	Knowledge distillation	✗	✗	✗	✗	✓
Computation Optimizations	Optimal attention span is computed during	inference	inference	inference	inference	finetuning
	Early exit assessment	✗	✗	✗	✗	✓
	Compressed sparse execution	✗	✓	✗	✓	✓
	eNVM storage for embeddings	✗	✗	✗	✗	✓

Figure 12: Comparison of EdgeBERT with prior work accelerating Transformer-based NLP models.

BERT quantization only via 3-bit clustering on the majority of BERT weights while storing the outlier weights and activations in full FP32 precision. Although this scheme significantly reduces DRAM accesses, it requires a mixed-precision computational datapath and a non-uniform memory storage. In contrast, EdgeBERT adopts uniform 8-bit data storage in SRAM and eNVMs memories. Lu *et al.* [46] proposes a dense systolic array accelerator for the Transformer's multi-head attention and feed-forward layers and optimizes Transformers' computations via matrix partitioning schemes. The EdgeBERT accelerator executes compressed sparse inference for higher energy efficiency. OPTIMUS [55] looks to holistically accelerate Transformers with compressed sparse matrix multiplications and by skipping redundant decoding computations. FlexASR [71] accelerates attention-based RNNs in a specialized attention datapath and only saves energy by gating the MAC when decoder RNN inputs are null. SpAtten [79] accelerates Transformer-based models via progressive cascade token and attention head pruning. The importance of each attention head is determined during the computation via a top-k ranking system. In contrast, EdgeBERT opts to learn the important attention heads during the fine-tuning process by activating adaptive attention spanning. The optimized and sparse attention spans are then used by the EdgeBERT accelerator to predicate the NLP computation.

Finally, all the aforementioned NLP accelerators stores the embedding weights in traditional volatile SRAM memories. By contrast, this work recognizes that embedding weights do not change across NLP tasks. Therefore, EdgeBERT statically stores the word embeddings in high density eNVMs, generating substantial energy and latency benefits (Sec. 8.3). Fig. 12 qualitatively contrasts some of the prior work with EdgeBERT.

10 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 edge platforms. EdgeBERT provides an in-depth and principled latency-driven methodology to alleviate these computational challenges in both the algorithm and hardware architecture layers. EdgeBERT adopts first-layer early exit prediction in order to perform dynamic voltage-frequency scaling (DVFS), at a sentence granularity, for minimal energy consumption while adhering to a prescribed target latency. Latency and memory footprint overheads are further alleviated by employing a balanced combination of adaptive attention span, selective network pruning, floating-point quantization. We further exploit and optimize the structure of eNVMs in order to store the shared multi-task parameters, granting EdgeBERT significant performance and energy savings from system power-on. Sentence-level, latency-aware inference on the EdgeBERT accelerator notably consumes 7× and 2.5× lower energy than the conventional full-model inference, and the latency-unbounded early exit approach, respectively.

ACKNOWLEDGMENTS

This work was supported in part by the Center for Applications Driving Architectures (ADA), one of six centers of JUMP, a Semiconductor Research Corporation (SRC) program co-sponsored by DARPA; DARPA's DSSoC program; NSF Awards 1704834 and 1718160; Intel Corp.; and Arm Inc.

A ARTIFACT APPENDIX

A.1 Abstract

Our artifact provides the software and hardware modelings behind *EdgeBERT*. It includes the following:

- The Python and Pytorch scripts used for entropy prediction and ALBERT finetuning including support for the various latency and memory alleviating optimizations.
- The SystemC source codes of the hardware accelerator and C++ testbenches.

A.2 Artifact check-list (meta-information)

- **Run-time environment:** Our evaluation scripts assume a Unix environment. Our hardware evaluation workflow has been validated with the following tool and package versions:
  - `catapult`: 10.5a
  - `gcc`: 4.9.3 with C++11
  - `boost`: 1.55.0
  - `systemc`: 2.3.1

  Our software evaluation workflow has been validated with the following tool and package versions:
  - `Anaconda3`: 5.0.1
  - `Cuda`: 10.0.130
  - `Cudnn`: 7.4.1.5
  - `Python`: 3.7.10
- **Metrics:** SW: task accuracy, HW: simulated cycle counts, post-HLS power and area.
- **Experiments:** We provide the scripts and procedures for running the software and hardware experiments.
- **How much disk space required (approximately)?:** 50 GB.
- **How much time is needed to prepare workflow (approximately)?:** SW: 12-24 hours, HW: 0.5-1 hours.
- **How much time is needed to complete experiments (approximately)?:** SW: 12-24 hours, HW: 0.5-1 hours.
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** Yes
- **Archived (provide DOI)?:** 10.5281/zenodo.5138730

A.3 Description and Installation

A.3.1 How to access. The artifact is available at the following links having identical contents:

- https://doi.org/10.5281/zenodo.5138730
- https://github.com/harvard-acc/EdgeBERT

A.3.2 Hardware dependencies. In order to complete the software experiments in a reasonable amount of time, GPU or TPU resources with at least 16GB of memory are necessary.

A.3.3 Data sets. We used the open-source NLP GLUE datasets (MNLI, QQP, SST-2, QNLI) during finetuning. Additionally, we have also open-sourced the custom entropy datasets used for training the MLP-based entropy predictor.

A.3.4 Models. The experiments are carried out specifically on the ALBERT model. However, they could also be applied successfully to other pre-trained BERT variants such as `BERT-base`.

A.4 Experiment workflow

A.4.1 Software Workflow. The `INSTALL.md` and `README.md` files in the sw directory contains the steps required to set up the conda environment and run the software evaluation workflow.

A.4.2 Hardware Workflow. The `README.md` contains all the necessary steps to compile and run the EdgeBERT accelerator. It is critical to have all the hardware dependencies installed prior to simulating the accelerator.

A.5 Methodology

Submission, reviewing and badging methodology:

- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html
- https://www.acm.org/publications/policies/artifact-review-badging

REFERENCES

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

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

[3] A. Agrawal, S. Lee, J. Silberman, M. Ziegler, M. Kang, S. Venkataramani, N. Cao, B. Fleischer, M. Guillorn, M. Cohen, S. Mueller, J. Oh, M. Lutz, J. Jung, S. Koswatta, C. Zhou, V. Zalani, J. Bonanno, R. Casatuta, C. Chen, J. Choi, H. Haynie, A. Herbert, R. Jain, M. Kar, K. Kim, Y. Li, Z. Ren, S. Rider, M. Schaal, K. Schelm, M. Scheuermann, X. Sun, H. Tran, N. Wang, W. Wang, X. Zhang, V. Shah, B. Curran, V. Srinivasan, P. Lu, S. Shukla, L. Chang, and K. Gopalakrishnan. 2021. 9.1 A 7nm 4-Core AI chip with 25.6 TFLOPS hybrid FP8 training, 102.4 TOPS INT4 inference and workload-Aware throttling. In *2021 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC).*

[4] T. Ajayi, S. Kamineni, Y. Cherivirala, M. Fayazi, K. Kwon, M. Saligane, S. Gupta, C. Chen, D. Sylvester, D. Dreslinski, B. Calhoun, and D. Wentzloff. 2020. An Open-source Framework for Autonomous SoC Design with Analog Block Generation. In *020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SoC).*

[5] Vahideh Akhlaghi, Amir Yazdanbakhsh, Kambiz Samadi, Rajesh K. Gupta, and Hadi Esmaeilzadeh. 2018. SnaPEA: Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks. In *Proceedings of the 45th Annual International Symposium on Computer Architecture.* 662–673.

[6] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. In *Proceedings of the 43rd International Symposium on Computer Architecture.*

[7] L. J. Ba et al. 2016. Layer Normalization. *ArXiv* abs/1607.06450 (2016).

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations, ICLR 2015.* http://arxiv.org/abs/1409.0473

[9] S. Bang, W. Lim, C. Augustine, A. Malavasi, M. Khellah, J. Tschanz, and V. De. 2020. 25.1 A Fully Synthesizable Distributed and Scalable All-Digital LDO in 10nm CMOS. In *2020 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC).*

[10] 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. 2014. 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).*

[11] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. 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 (ASPLOS '14).* ACM, New York, NY, USA, 269–284.

[12] Y. Chen, J. Emer, and V. Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA).* 367–379.

[13] J. Choi, Z. Hakimi, P. W. Shin, J. Sampson, and V. Narayanan. 2019. Context-Aware Convolutional Neural Network over Distributed System in Collaborative Computing. In *2019 56th ACM/IEEE Design Automation Conference (DAC).* 1–6.

[14] 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. 2013. A 256-Mcell Phase-Change Memory Chip Operating at 2+ Bit/Cell. *IEEE Transactions on Circuits and Systems I: Regular Papers* 60, 6 (2013), 1521–1533. https://doi.org/10.1109/TCSI.2012.2220459

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

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[17] M. Donato, L. Pentecost, D. Brooks, and G. Wei. 2019. MEMTI: Optimizing On-Chip Nonvolatile Storage for Visual Multitask Inference at the Edge. *IEEE Micro* 39, 6 (2019), 73–81. https://doi.org/10.1109/MM.2019.2944782

[18] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. 2012. NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 7 (2012), 994–1007. https://doi.org/10.1109/TCAD.2012.2185930

[19] Robert Eisele. 2016. The log-sum-exp trick in Machine Learning. https://www.xarg.org/2016/06/the-log-sum-exp-trick-in-machine-learning/

[20] Igor Fedorov, Marko Stamenovic, Carl Jensen, Li-Chia Yang, Ari Mandell, Yiming Gan, Matthew Mattina, and Paul N. Whatmough. 2020. TinyLSTMs: Efficient Neural Speech Enhancement for Hearing Aids. In *Proc. Interspeech 2020.* 4054–4058. https://doi.org/10.21437/Interspeech.2020-1864

[21] Prakhar Ganesh, Yao Chen, Xin Lou, Mohammad Haris Ali Khan, Y. Yang, D. Chen, M. Winslett, Hassan Sajjad, and Preslav Nakov. 2020. Compressing Large-Scale Transformer-Based Models: A Case Study on BERT. *ArXiv* abs/2002.11985 (2020).

[22] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. 2019. TANGRAM: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19).* Association for Computing Machinery, New York, NY, USA, 807–820. https://doi.org/10.1145/3297858.3304014

[23] Tae Jun Ham, S. J. Jung, Seonghak Kim, Young H. Oh, Yeonhong Park, Yongchan Song, Junghun Park, Sang-Hee Lee, K. Park, J. Lee, and Deog-Kyoon Jeong. 2020. A^3: Accelerating Attention Mechanisms in Neural Networks with Approximation. *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2020), 328–341.

[24] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. *SIGARCH Comput. Archit. News* 44, 3 (June 2016).

[25] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *CoRR* abs/1510.00149 (2015).

[26] Kartik Hegde, Jiyong Yu, Rohit Agrawal, Mengjia Yan, Michael Pellauer, and Christopher W. Fletcher. 2018. UCNN: Exploiting Computational Reuse in Deep Neural Networks via Weight Repetition. In *Proceedings of the 45th Annual International Symposium on Computer Architecture.* 674–687.

[27] HLSLibs. [n. d.]. *Open-Source High-Level Synthesis IP Libraries.* Technical Report. https://github.com/hlslibs

[28] B. Huang, E. Fang, S. Hsueh, R. Huang, A. Lin, C. Chiang, Y. Lin, W. Hsieh, B. Chen, Y. Zhuang, C. Wu, J. Chen, Y. Chen, C. Wan, E. Wang, A. Chiou, P. Kao, Y. Tsai, H. Chen, and S. Hwang. 2021. 35.1 An octa-core 2.8/2GHz dual-gear sensor-assisted high-speed and power-efficient CPU in 7nm FinFET 5G smartphone SoC. In *2021 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC).*

[29] Forrest N. Iandola, Albert Eaton Shaw, R. Krishna, and K. Keutzer. 2020. SqueezeBERT: What can computer vision teach NLP about efficient neural networks? *ArXiv* abs/2006.11316 (2020).

[30] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR* abs/1502.03167 (2015). arXiv:1502.03167 http://arxiv.org/abs/1502.03167

[31] Animesh Jain, Amar Phanishayee, Jason Mars, Lingjia Tang, and Gennady Pekhimenko. 2018. Gist: Efficient Data Encoding for Deep Neural Network Training. In *Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA '18).* 776–789.

[32] Jeff Johnson. 2018. Rethinking floating point for deep learning. *CoRR* abs/1811.01721 (2018). arXiv:1811.01721 http://arxiv.org/abs/1811.01721

[33] 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. 2017. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 1–12. https://doi.org/10.1145/3079856.3080246

[34] Brucek Khailany, Evgeni Khmer, Rangharajan Venkatesan, Jason Clemons, Joel S. Emer, Matthew Fojtik, Alicia Klinefelter, Michael Pellauer, Nathaniel Pinckney, Yakun Sophia Shao, Shreesha Srinath, Christopher Torng, Sam (Likun) Xi, Yanqing Zhang, and Brian Zimmer. 2018. A Modular Digital VLSI Flow for High-productivity SoC Design. In *Proceedings of the 55th Annual Design Automation Conference (DAC '18)*. ACM, New York, NY, USA, Article 72, 6 pages. https://doi.org/10.1145/3195970.3199846

[35] Glenn G. Ko, Yuji Chai, Marco Donato, Paul N. Whatmough, Thierry Tambe, Rob A. Rutenbar, David Brooks, and Gu-Yeon Wei. 2020. A 3mm² Programmable Bayesian Inference Accelerator for Unsupervised Machine Perception using Parallel Gibbs Sampling in 16nm. In *2020 IEEE Symposium on VLSI Circuits*. 1–2. https://doi.org/10.1109/VLSICircuits18222.2020.9162784

[36] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. *SIGPLAN Not.* 53, 2 (March 2018), 461–475. https://doi.org/10.1145/3296957.3173176

[37] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *ArXiv* abs/1909.11942 (2020).

[38] Sae Kyu Lee, Paul N. Whatmough, David Brooks, and Gu-Yeon Wei. 2019. A 16-nm Always-On DNN Processor With Adaptive Clocking and Multi-Cycle Banked SRAMs. *IEEE Journal of Solid-State Circuits* 54, 7 (2019), 1982–1992. https://doi.org/10.1109/JSSC.2019.2913098

[39] Haitong Li, Mudit Bhargav, Paul N. Whatmough, and H.-S. Philip Wong. 2019. On-Chip Memory Technology Design Space Explorations for Mobile Deep Neural Network Accelerators. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[40] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob. 2020. DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator. *IEEE Computer Architecture Letters* 19, 2 (2020), 106–109. https://doi.org/10.1109/LCA.2020.2973991

[41] Daofu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Teman, Xiaobing Feng, Xuehai Zhou, and Yunji Chen. 2015. PuDianNao: A Polyvalent Machine Learning Accelerator. *SIGPLAN Not.* 50, 4 (March 2015), 369–381. https://doi.org/10.1145/2775054.2694358

[42] Shaoli Liu, Zidong Du, Jinhua Tao, Dong Han, Tao Luo, Yuan Xie, Yunji Chen, and Tianshi Chen. 2016. Cambricon: An Instruction Set Architecture for Neural Networks. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. 393–405.

[43] 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. 2013. A 130.7mm2 2-layer 32Gb ReRAM memory device in 24nm technology. In *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*.

[44] Y. Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *ArXiv* abs/1907.11692 (2019).

[45] Zhi-Gang Liu, Paul N. Whatmough, and Matthew Mattina. 2020. Systolic Tensor Array: An Efficient Structured-Sparse GEMM Accelerator for Mobile CNN Inference. *IEEE Computer Architecture Letters* 19, 1 (2020), 34–37. https://doi.org/10.1109/LCA.2020.2979965

[46] Siyuan Lu, Meiqi Wang, S. Liang, J. Lin, and Z. Wang. 2020. Hardware Accelerator for Multi-Head Attention and Position-Wise Feed-Forward in the Transformer. *ArXiv* abs/2009.08605 (2020).

[47] D. Mahajan, J. Park, E. Amaro, H. Sharma, A. Yazdanbakhsh, J. K. Kim, and H. Esmaeilzadeh. 2016. TABLA: A unified template-based framework for accelerating statistical machine learning. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 14–26. https://doi.org/10.1109/HPCA.2016.7446050

[48] James McCaffrey. 2016. The Max Trick when Computing Softmax. https://jamesmccaffrey.wordpress.com/2016/03/04/the-max-trick-when-computing-softmax/

[49] J. Scott McCarley. 2019. Pruning a BERT-based Question Answering Model. *ArXiv* abs/1910.06360 (2019).

[50] P. Meinerzhagen, C. Tokunaga, A. Malavasi, V. Vaidya, A. Mendon, D. Mathaikutty, J. Kulkarni, C. Augustine, M. Cho, S. Kim, G. Matthew, R. Jain, J. Ryan, C. Peng, S. Paul, S. Vangal, B. Esparza, L. Cuellar, M. Woodman, B. Iyer, S. Maiyuran, G. Chinya, C. Zou, Y. Liao, K. Ravichandran, H. Wang, M. Khellah, J. Tschanz, and V. De. 2018. 2.3 An energy-efficient graphics processor featuring fine-grain DVFS with integrated voltage regulators, execution-unit turbo, and retentive sleep in 14nm tri-gate CMOS. In *2018 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*.

[51] Paul Michel, Omer Levy, and Graham Neubig. 2019. Are Sixteen Heads Really Better than One? *ArXiv* abs/1905.10650 (2019).

[52] Pandu Nayak. 2019. *Understanding searches better than ever before.* Technical Report. https://blog.google/products/search/search-language-understanding-bert/

[53] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 27–40.

[54] Eunhyeok Park, Dongyoung Kim, and Sungjoo Yoo. 2018. Energy-Efficient Neural Network Accelerator Based on Outlier-Aware Low-Precision Computation. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*. 688–698.

[55] Junki Park, Hyunsung Yoon, Daehyun Ahn, Jungwook Choi, and Jae-Joon Kim. 2020. 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.). Vol. 2. 363–378.

[56] Lillian Pentecost, Marco Donato, Brandon Reagen, Udit Gupta, Siming Ma, Gu-Yeon Wei, and David Brooks. 2019. 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 (MICRO '52)*.

[57] PubNub. 2015. How Fast is Real-time? Human Perception and Technology. https://www.pubnub.com/blog/how-fast-is-realtime-human-perception-and-technology/

[58] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100, 000+ Questions for Machine Comprehension of Text. *ArXiv* abs/1606.05250 (2016).

[59] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G. Wei. 2018. Ares: A framework for quantifying the resilience of deep neural networks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC.2018.8465834

[60] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G. Wei, and D. Brooks. 2016. Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 267–278. https://doi.org/10.1109/ISCA.2016.32

[61] Marc Riera, Jose-Maria Arnau, and Antonio González. 2018. Computation Reuse in DNNs by Exploiting Input Similarity. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*. 57–68.

[62] Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2020. A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 58–68. https://doi.org/10.1109/ISPASS48437.2020.00016

[63] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *ArXiv* abs/1910.01108 (2019).

[64] Victor Sanh, Thomas Wolf, and Alexander M. Rush. 2020. Movement Pruning: Adaptive Sparsity by Fine-Tuning. In *34th Conference on Neural Information Processing Systems (NeurIPS)*. http://arxiv.org/abs/2005.07683

[65] Roy Schwartz, Gabi Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. 2020. The Right Tool for the Job: Matching Model and Instance Complexities. In *ACL*.

[66] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. 2019. Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 14–27. https://doi.org/10.1145/3352460.3358302

[67] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh. 2016. From high-level deep neural models to FPGAs. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12. https://doi.org/10.1109/MICRO.2016.7783720

[68] Sheng Shen, Zhen Dong, J. Ye, L. Ma, Zhewei Yao, A. Gholami, M. Mahoney, and K. Keutzer. 2020. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT. In *AAAI*.

[69] Sainbayar Sukhbaatar, E. Grave, P. Bojanowski, and Armand Joulin. 2019. Adaptive Attention Span in Transformers. In *ACL*.

[70] Zhiqing Sun, H. Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. In *ACL*.

[71] Thierry Tambe, En-Yu Yang, Glenn G. Ko, Yuji Chai, Coleman Hooper, Marco Donato, Paul N. Whatmough, Alexander M. Rush, David Brooks, and Gu-Yeon Wei. 2021. 9.8 A 25mm2 SoC for IoT Devices with 18ms Noise-Robust Speech-to-Text Latency via Bayesian Speech Denoising and Attention-Based Sequence-to-Sequence DNN Speech Recognition in 16nm FinFET. In *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, Vol. 64. 158–160. https://doi.org/10.1109/ISSCC42613.2021.9366062

[72] Thierry Tambe, En-Yu Yang, Zishen Wan, Y. Deng, V. Reddi, Alexander M. Rush, D. Brooks, and Gu-Yeon Wei. 2019. AdaptivFloat: A Floating-point based Data Type for Resilient Deep Learning Inference. *ArXiv* abs/1909.13271 (2019).

[73] S. Teerapittayanon, B. McDanel, and H. T. Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. 2464–2469. https://doi.org/10.1109/ICPR.2016.7900006

[74] Z. Toprak-Deniz, M. Sperling, J. Bulzacchelli, G. Still, R. Kruse, S. Kim, D. Boerstler, T. Gloekler, R. Robertazzi, K. Stawiasz, T. Diemoz, G. English, D. Hui, P. Muench, and J. Friedrich. 2014. 5.2 Distributed system of digitally controlled microregulators enabling per-core dvfs for the Power8 tm microprocessor. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*.

[75] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017). arXiv:1706.03762 http://arxiv.org/abs/1706.03762

[76] Swagath Venkataramani, Ashish Ranjan, Subarno Banerjee, Dipankar Das, Sasikanth Avancha, Ashok Jagannathan, Ajaya Durg, Dheemanth Nagaraj, Bharat Kaul, Pradeep Dubey, and Anand Raghunathan. 2017. ScaleDeep: A Scalable Compute Architecture for Learning and Evaluating Deep Networks. *SIGARCH Comput. Archit. News* (2017).

[77] B. Venkatesan et al. 2019. MAGNet : A Modular Accelerator Generator for Neural Networks. In *ICCAD*.

[78] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *CoRR* abs/1804.07461 (2018). arXiv:1804.07461 http://arxiv.org/abs/1804.07461

[79] Hanrui Wang, Zhekai Zhang, and Song Han. 2021. SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning. *2021 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2021).

[80] Jian Weng, Sihao Liu, Vidushi Dadu, Zhengrong Wang, Preyas Shah, and Tony Nowatzki. 2020. DSAGEN: Synthesizing Programmable Spatial Accelerators. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA '20)*.

[81] Paul N. Whatmough, Marco Donato, Glenn G. Ko, Sae Kyu Lee, David Brooks, and Gu-Yeon Wei. 2020. CHIPKIT: An Agile, Reusable Open-Source Framework for Rapid Test Chip Development. *IEEE Micro* 40, 4 (2020), 32–40. https://doi.org/10.1109/MM.2020.2995809

[82] Paul N. Whatmough, Sae Kyu Lee, David Brooks, and Gu-Yeon Wei. 2018. DNN Engine: A 28-nm Timing-Error Tolerant Sparse Deep Neural Network Processor for IoT Applications. *IEEE Journal of Solid-State Circuits* 53, 9 (2018), 2722–2731. https://doi.org/10.1109/JSSC.2018.2841824

[83] Paul N. Whatmough, Sae Kyu Lee, Marco Donato, Hsea-Ching Hsueh, Sam Xi, Udit Gupta, Lillian Pentecost, Glenn G. Ko, David Brooks, and Gu-Yeon Wei. 2019. A 16nm 25mm2 SoC with a 54.5x Flexibility-Efficiency Range from Dual-Core Arm Cortex-A53 to eFPGA and Cache-Coherent Accelerators. In *2019 Symposium on VLSI Circuits*. C34–C35. https://doi.org/10.23919/VLSIC.2019.8778002

[84] Paul N. Whatmough, Chuteng Zhou, Patrick Hansen, Shreyas Kolala Venkataramanaiah, Jae sun Seo, and Matthew Mattina. 2019. FixyNN: Efficient Hardware for Mobile Computer Vision via Transfer Learning. In *Proceedings of the 2nd SysML Conference, Palo Alto, CA, USA*.

[85] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv* abs/1910.03771 (2019).

[86] Sam (Likun) Xi, Yuan Yao, Kshitij Bhardwaj, Paul Whatmough, Gu-Yeon Wei, and David Brooks. 2020. SMAUG: End-to-End Full-Stack Simulation Infrastructure for Deep Learning Workloads. *ACM Trans. Archit. Code Optim.* 17, 4, Article 39 (Nov. 2020), 26 pages. https://doi.org/10.1145/3424669

[87] J. Xin, Raphael Tang, J. Lee, Y. Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference. *ArXiv* abs/2004.12993 (2020).

[88] Ali Hadi Zadeh and A. Moshovos. 2020. GOBO: Quantizing Attention-Based NLP Models for Low Latency and Energy Efficient Inference. In *53rd IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[89] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8Bit BERT. In *33rd Conference on Neural Information Processing Systems (NeurIPS)*.

[90] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. BERT Loses Patience: Fast and Robust Inference with Early Exit. *ArXiv* abs/2006.04152 (2020).