ET: An Energy Efficient Edge Transformer Architecture

SHASHANK NAG, Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, United States
ALAN BACELLAR, The University of Texas at Austin, Austin, United States
ANSHUL JHA, The University of Texas at San Antonio, San Antonio, United States and University of the Pacific, Stockton, United States
EUGENE JOHN, The University of Texas at San Antonio, San Antonio, United States
KRISHNAN KAILAS, Independent Researcher, N/A, United States and IBM TJ Watson Research Center, Yorktown Heights, United States
PRISCILA LIMA, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
NEERAJA YADWADKAR, The University of Texas at Austin, Austin, United States
FELIPE FRANÇA, Google LLC, Mountain View, United States and Instituto de Telecomunicações, Porto, Portugal
LIZY K. JOHN, The University of Texas at Austin, Austin, United States

Transformers are increasingly being deployed at the edge to support agentic AI, real-time processing, and privacy-preserving applications. However, their substantial computational and memory requirements often exceed the energy and latency constraints of edge devices. Through detailed characterization, we observe that the majority of computations and parameters in transformer models are concentrated in the feed-forward Multi-Layer Perceptron (MLP) blocks, identifying them as a key target for optimization. In this work, we introduce ET, a novel transformer architecture tailored for energy-efficient edge inference. By integrating shared Look Up Table (LUT) based feed-forward blocks with traditional self-attention modules, ET significantly reduces computational load, memory footprint, and energy usage, aligning closely with the operational and resource constraints of edge devices. We present the design, implementation, and evaluation of ET, and demonstrate that it delivers comparable accuracy to traditional transformers while achieving 2.1× improvement over a baseline

This research was supported by Semiconductor Research Corporation (SRC) Task 3148.001, National Science Foundation (NSF) Grants #2326894, #2425655 (supported in part by the federal agency and Intel, Micron, Samsung, and Ericsson through the FuSe2 program), NVIDIA Applied Research Accelerator Program, and compute resources on the Vista GPU cluster through CGAI and TACC at UT Austin. Any opinions, findings, conclusions, or recommendations are those of the authors and not of the funding agencies.
Authors' Contact Information: Shashank Nag (corresponding author), Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, Texas, United States; e-mail: shashanknag@utexas.edu; Alan Bacellar, The University of Texas at Austin, Austin, Texas, United States; e-mail: alanbacellar@utexas.edu; Anshul Jha, The University of Texas at San Antonio, San Antonio, Texas, United States and University of the Pacific, Stockton, California, United States; e-mail: anshul.jha@my.utsa.edu; Eugene John, The University of Texas at San Antonio, San Antonio, Texas, United States; e-mail: eugene.john@utsa.edu; Krishnan Kailas, Independent Researcher, N/A, United States and IBM TJ Watson Research Center, Yorktown Heights, New York, United States; e-mail: kailas@acm.org; Priscila Lima, Federal University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil; e-mail: priscilamvl@cos.ufrj.br; Neeraja Yadwadkar, The University of Texas at Austin, Austin, Texas, United States; e-mail: neeraja@austin.utexas.edu; Felipe França, Google LLC, Mountain View, California, United States and Instituto de Telecomunicações, Porto, Portugal; e-mail: felipe@ieee.org; Lizy K. John, The University of Texas at Austin, Austin, Texas, United States; e-mail: ljohn@ece.utexas.edu.

ACM 1544-3973/2026/07-ART92
https://doi.org/10.1145/3828726

ASIC accelerator. It also achieves a 1.12× efficiency on GPU acceleration, and about 1.34× improvement over a prior BinaryBERT FPGA accelerator. We further report 1.2× energy efficiency over recent multiplication-free models, such as Bitnet, showing that ET offers a compelling and practical solution for deploying transformers effectively at the edge, whether on FPGAs, ASICs, or edge GPUs.[1]

CCS Concepts: • **Computer systems organization → Neural networks**; **Reconfigurable computing**; • **Computing methodologies → Machine learning**; • **Hardware → Hardware accelerators**

Additional Key Words and Phrases: Energy efficient, transformers, FPGA, ASIC, GPU

ACM Reference Format:
Shashank Nag, Alan Bacellar, Anshul Jha, Eugene John, Krishnan Kailas, Priscila Lima, Neeraja Yadwadkar, Felipe França, and Lizy K. John. 2026. ET: An Energy Efficient Edge Transformer Architecture. *ACM Trans. Arch. Code Optim.* 23, 3, Article 92 (July 2026), 25 pages. https://doi.org/10.1145/3828726

1 Introduction

Transformer-based models have rapidly become ubiquitous across numerous AI tasks, excelling notably in language modeling, reasoning, and computer vision. Recently, these models have evolved into autonomous AI agents capable of independent reasoning, decision-making[32], and execution, significantly shifting AI deployment paradigms from cloud-centric to edge-based platforms. Deploying **transformers directly at the edge**—on devices such as smartphones, IoT sensors, drones, and embedded systems offers compelling advantages: (i) Enhanced privacy and security: By keeping sensitive data within the local device or network, minimizes the risk of data breaches while complying with privacy regulations [27]. (ii) Reduced bandwidth usage and cost savings: Data movement is often expensive, in terms of cost, latency and energy, and processing data locally reduces the need to transmit large volumes of data to the cloud [8]. (iii) Offline functionality: Even with unreliable or no internet connectivity, edge-deployed transformers can continue to function autonomously, making them suitable for applications in remote areas or during network outages [12].

Relatively **small language models** such as GPT-2 [35] and BERT [10] could be targeted towards mobile applications, and advancements in model optimization techniques like quantization, and pruning could make it feasible to run these on resource-constrained edge devices [15, 16, 23]. However, large memory footprint, high computational costs, and the, consequently, large energy consumption per inference in these is still an impediment. With stringent power and latency budgets, this leaves optimized deployment of transformers at the edge still an open challenge. In this work, we explore incorporating paradigms alternative to typical neural network architectures, especially those based on RAM or **Look Up Table** (**LUT**) neurons into transformers to achieve neuromorphic energy efficiencies.

Most transformer models are mainly composed of a stack of encoder or decoder blocks, each composed of a self-attention block and a feed-forward block [9, 10, 44] (more details in Section 2). Prior studies show that the feed-forward blocks, or the **Multi-Layer Perceptrons** (**MLPs**) in these models, are the key behind the *knowledge* learnt in them—with the attention block catering to the context between tokens [11].

We characterize common transformers to identify opportunities to improve their inference efficiency, by analyzing the number of parameters and computations performed in the feed-forward and

[1]Extension of Conference Paper: This work is an extension of our prior work [29], which was designed specifically targeting the deployment of vision transformers on edge FPGAs, using LUT based neurons. In this work, we propose a generic LUT based transformer architecture, that is, applicable to a wide range of language models, and propose techniques for their acceleration across GPUs, ASICs, and FPGAs. Further, we incorporate algorithmic and architectural co-design techniques, such as using shared LUT blocks, to create area-efficient designs while targeting ASICs.

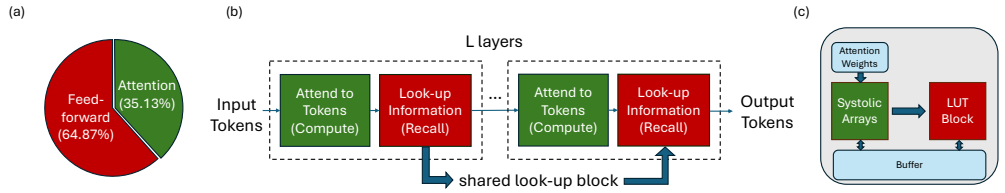

Fig. 1. (a) Distribution of multiply-add computations between attention and feed-forward (recall) layers in the GPT-2 model (a typical transformer with Multi-Head Self Attention). More than half of all multiplications are attributed to the feed-forward layer (more details in Figure 3). (b) L layers of Attention and Recall in a transformer. Recognizing that feed-forward layer implements "information-retrieval", ET, our edge-optimized transformer "looks-up" information in the recall block, and employs a traditional self-attention block. (c) A custom accelerator where the attention (compute) block is scheduled on a systolic array, and the recall is scheduled on a LUT block.

attention layers, across typical transformer models. Interestingly, we observe that **a major fraction of computations and weights is in the feed-forward layers** of the model (Figure 1(a)). However, this is counterintuitive to how the human brain works, where information retrieval from the memory should be simpler, and consequently, computationally inexpensive as compared to reasoning.

As the feed-forward MLP layers dominate the computations in current transformer models, an efficient alternative for the MLP layer would result in significant overall energy savings. We see an opportunity here by tapping into **LUT based neural networks**—involving only *lookup* operations instead of the typical **multiply-accumulate** (**MAC**) operations. In this realm, models such as LogicNets [45], PolyLUT [2], NeuraLUT [3], and **Differentiable Weightless Neural Networks** (**DWNs**) [4], have demonstrated significant latency and energy advantages against iso-accuracy MLP models, with designs matching the underlying logic fabric on FPGAs. Drawing inspiration from these, we design edge transformers with *lookup* based blocks for knowledge retrieval, potentially offering efficient models that are well-suited for deployment on edge devices. Though the *lookup* blocks are energy efficient, when implemented on hardware, they are stationary and consume chip area, unlike weights in typical feedforward layers that can be fetched from off-chip memory. To ensure that our models fit the on-chip resource budgets of edge devices, we draw inspiration from MobiLlama [43], and the *lookup* blocks are shared across the decoder layers of the transformer.

Based on the aforementioned observations, we propose **ET–an edge-optimized transformer** architecture and accelerator design. ET consists of a stack of decoder (or encoder) blocks, each consisting of a self-attention module, similar to typical transformer models, and a LUT-based feedforward layer, that is, shared across the blocks (Figure 1(b)). We define gradients of LUTs based on DWN [4], integrate the *lookup* block with the compute-based attention layers and train the entire model end-to-end. In contrast to techniques such as LUT-NN [41], Affine [21], and LUT-GEMM [31] where MACs/neurons in the models are converted to LUTs post-training, our approach of constructing and training LUT-based transformers from scratch enables us to build smaller, efficient models, while utilizing the higher expressive power of LUTs[2]. When deployed on a GPU, we demonstrate that owing to a smaller model size, ET offers an improved throughput. We also develop a custom accelerator targeting FPGAs and ASICs, involving systolic array blocks to compute self-attention, and dedicated LUT blocks to handle the *lookup* based feed-forward layers (Figure 1(c)).

[2]1. An n-input LUT can be programmed to represent any one of 2^{2^n} non-linear functions, i.e., 16 for LUT2 and 2^{64} (18 quintillions) for LUT6.

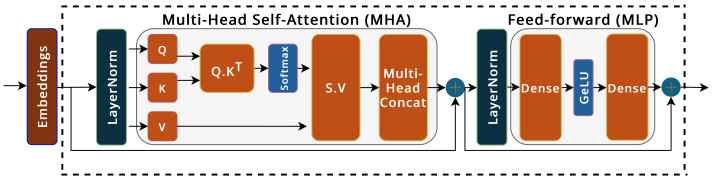

Fig. 2. A typical transformer model architecture with an encoder-only or decoder-only stack of layers. In the case of decoder-only models, the MHA shown here is a masked self-attention, with future tokens masked during attention score computation. MHA is followed by Feed Forward MLP layers.

The LUT blocks map perfectly to the underlying 6-input (LUT6) logic slices on modern FPGAs, without involving any memory overhead. With this, we offer an inference-efficient edge solution that delivers the model performance advantages of transformers, while offering hardware performance benefits of LUT-based networks. Summarizing, we make the following contributions in this paper:

— We present an energy-efficient transformer architecture that involves LUT-based feed-forward blocks for efficient knowledge retrieval, and conventional self-attention blocks that cater to the interaction between tokens.
— We train these models and develop optimized GPU kernels for their inference, and demonstrate 1.3–1.6× throughput improvements under memory constrained environments such as edge. Use of LUT neurons in the feed-forward layers leads to 1.12× improvements in latency and energy on an NVIDIA T4 GPU. A GPT2-medium model which cannot fit on an edge constrained GPU can now fit on it with the optimizations in ET.
— We propose a dedicated hardware accelerator design for ET, and evaluate its performance on FPGA and ASIC against baseline accelerator and software implementations. On FPGA models, ET achieves 1.34× improved efficiency (prompts/Joule) compared to the best prior work on binary BERT [15] inference. On ASIC, we achieve 2.1× prompts/joule compared to baseline ASIC. In comparison to bitnet-1.58 model, ET still achieves 1.2× better energy efficiency (prompts/joule).

2 Background and Motivation

2.1 Transformer-Based Models

The transformer model architecture is based on the principle of deriving context or relationships on a series of input tokens, and using it to predict a likely output token. The transformer design forms the bedrock of most of the current day language and reasoning models [47]. Figure 2 shows a typical transformer model architecture.

Several **small-scale transformer models** have been developed, targeting applications at the edge [49], also highlighted in a recent market survey [13]. One of the most popular among them is BERT [10], a model with a stack of encoders, designed for language understanding. GPT2 is another model, focused on generative tasks, with a stack of decoders [35]. Most of the large language models being used in the current day, are also architecturally similar to these models [30, 37, 44]. While language understanding models like BERT have a classification head at the end to infer a class based on the input sentence, in generative models like GPT, inference serving comprises of two phases—prefill and decode. In the **prefill** phase, the model takes in the entire input prompt and processes it through the model to produce the first output token. In the **decode** phase, the output token is incorporated into the input context, and the next token is generated iteratively. As the K-V pairs are typically computed and cached in the prefill phase, the decode phase leverages this to

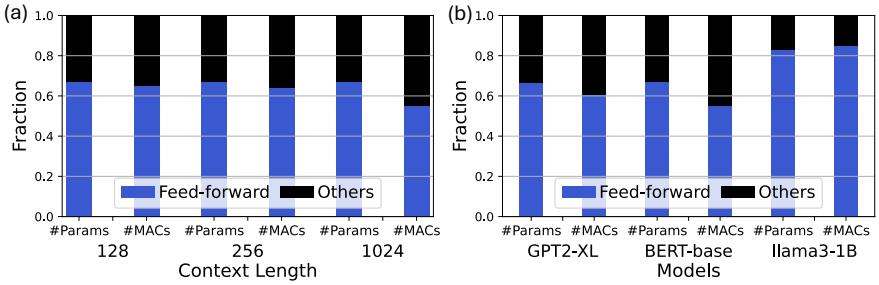

Fig. 3. Fraction of parameters and computations in feed-forward block vs. others for (a) GPT-2 with varying context lengths and (b) for different models.

perform fewer computations by reusing these. Splitwise [33] is a prior work that leveraged this observation to propose an inference serving platform that handles these two phases on separate systems.

2.2 Algorithmic and Architectural Optimizations

In order to address the issues with increasing model sizes of transformers and the consequent deployment implications, quantization of such models has been investigated, which helps reduce the memory requirement and compute precision. SmoothQuant [54], Q8BERT [57], and FQ-BERT [23] have proposed different quantization techniques, including quantization-aware training, and post-training quantization upto 8-bits. BinaryBERT [5] furthers this to perform a quantization aware training, that reduces the precision for weights upto 1-bit. However, in all of these works, the computational complexity of transformers remains unresolved, with all of them still involving several MAC operations. On the implementation front, there have been optimizations targeting the deployed devices. For software implementations, such as on GPUs, cache and memory management is key to ensuring computational efficiency in inference serving, with KV caching widely employed [34]. While longer context lengths require more memory for KV cache, memory availability also dictates the number of inference requests that can be batched concurrently. vLLM [22] is a popular framework that integrates this concept alongwith the idea of PagedAttention, enabling efficient inference serving of requests both in offline and online serving scenarios. It optimizes the achieved throughput through efficient memory management and scheduling techniques. Similarly, custom hardware accelerator designs have also been developed, with optimized dataflow schemes that exploit the regularity of operations in transformers [18].

2.3 Workload Analysis

We characterize transformers in terms of their constituent computations and parameters to identify potential for energy savings. Figure 3(a) shows the fraction of parameters and computations in the feed-forward and other layers of a GPT-2 model, with varying context lengths relevant for edge scenarios. As could be seen, the overall computations (and consequently, the energy consumption) is largely dominated by the feed-forward (MLP) blocks in the network. Similarly, even in terms of model parameters (and consequently, the model size/memory consumption), the feedforward blocks contribute the most. These trends hold true even as we consider different model variants (Figure 3(b)) for a context length of 1,024. In all these cases, roughly 50–60% of the overall model parameters and MAC operations are attributed to the feed-forward blocks.

As noted earlier, MLPs are considered to be the knowledge house of transformer models [11]. However, MLPs being the most compute intensive layers of the transformers is counterintuitive

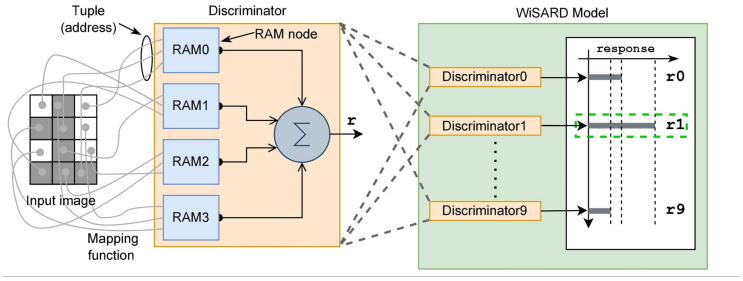

Fig. 4. An example **Weightless Neural Network** (**WNN**). The patterns in the inputs are observed, and the RAM (LUT) nodes in the discriminator corresponding to the correct class update their values during training [39]. The discriminator that presents the highest score at inference is chosen as the predicted class. Figure adapted from [40]

to the idea that information retrieval in the human brain is a much simpler process than logical reasoning which derives context. These observations suggest that there is a large room for improvement in the design. As the MLP block is the most-significant energy contributor, an energy-efficient alternative for it would also result in considerable overall savings. With that background, we now turn our focus towards LUT-based models, in an effort to identify potential energy-efficient alternatives to implement the information retrieval.

2.4 LUT-Based Neural Networks

Of late, several LUT based models have been proposed to improve neural network efficiency, which eliminate the need for power and resource-hungry MAC operations in conventional neurons. LogicNets [45] trains sparse binary-activation **Deep Neural Networks** (**DNNs**) and converts neurons into LUTs by enumerating all input combinations. However, this approach does not exploit the representative capacity of LUTs. An n-input LUT is a highly expressive structure, which can inherently represent any one of 2^{2^n} possible non-linear functions as the output. However, by converting a trained DNN neuron to a LUT, the LUT's representative ability is reduced to that of a DNN neuron, resulting in larger and less efficient models.

PolyLUT [2] and NeuraLUT [3] attempt to address this limitation by using richer input feature mappings, and capturing more complex patterns within LUTs. **Despite these improvements, these methods still fall short of fully exploiting the computational power of LUTs, as they generally treat LUTs as post hoc accelerators instead of learning them. LUT-based transformer models are also plagued by the same inefficiencies.** Works like LUT-GEMM [31] and LUT-NN [41] attempt to use LUTs to accelerate transformer inference, but these are done at the level of replacing each multiplication with a LUT—leaving a high scope for improvement.

In contrast, WNNs are neural networks inspired by the dendritic trees of biological neurons, comprising of learnable LUT based neurons (Figure 4). Recent models such as BTHOWeN [39] and ULEEN [40] have demonstrated significantly lower compute and memory footprints compared to iso-accuracy MLPs and CNNs for image classification tasks. These developments have revived interest in WNNs, especially for low-latency, energy-efficient inference on FPGAs and ASICs. Though model training on GPUs is a slight challenge due to the bit-wise and sparse nature of the operations, since our primary focus is inference efficiency, training complexity is a secondary concern.

DWNs [4], addresses the above limitations of LUT representations by proposing a differentiable training framework for LUT-based model architectures. DWNs use an **Extended Finite Difference**

Table 1. Comparison of Logic and LUT-Based Networks

Work	Learnable LUTs	Supports Transformers	Work	Learnable LUTs	Supports Transformers
LogicNets [45]	×	×	PolyLUT [2]	×	×
AmigoLUT [52]	×	×	NeuraLUT [3]	×	×
ULEEN [40]	✓	×	DWN [4]	✓	×
LUT GEMM [31]	×	✓	**ET (Ours)**	✓	✓

(**EFD**)-based approximation technique to enable backpropagation through LUT entries and binary inputs. Compared to LogicNets, PolyLUT, and NeuraLUT, DWNs fully leverage the expressive capacity of LUTs during training rather than post-training treating them as inference-only modules. In iso-accuracy scenarios, DWNs outperform quantized networks like FINN [46] by up to 2000x in area-delay product. However, like other WNNs, they still struggle with tasks requiring spatial and temporal feature invariance due to architectural limitations in handling positional information. Table 1 summarizes the above discussed LUT-based models. Our contribution is to incorporate LUT blocks into the recall MLP layers of the transformer.

2.5 Design Intuition

Prior research in the field of WNNs demonstrates that these networks can efficiently learn patterns represented by MLP layer, with iso-accuracy performance reported to MLP-only models [4, 39, 40]—while involving reduced computations, energy consumption and latency. We also note that since the LUTs can inherently learn non-linearities, these do not require additional non-linear activation layers that would be otherwise required within an MLP layer. This motivates us to implement a LUT-based feed-forward block to realize the information retrieval phase of our transformer design.

Transformers utilize self-attention to learn positional dependence, and derive context across tokens, that enable it to perform well on language modeling and reasoning—a key factor lacking in current WNNs. We noted in Section 2.3 that self-attention layers consume a smaller fraction of model weights and overall computations. Furthermore, several implementational optimizations, such as KV-caching, have been performed that reduces the computational complexity of these layers. On the contrary, there are not many optimizations geared towards the MLP layers in these models. We therefore use the typical self-attention layer for realizing the compute block in ET in order to leverage their efficient implementations.

3 Our Proposal ET

We propose ET, an algorithm hardware co-design solution for energy efficient transformers to enable intelligence at the edge. We develop a design that integrates the efficiency of LUT-based networks, implement optimized software kernels for inference on GPUs, and design a hardware accelerator for inference targeting ASICs and FPGAs.

3.1 Design Overview

Figure 5 shows an overview of our design. ❶ In order to attend to other tokens and get context, ET employs the typical self-attention mechanism. ❷ For the information retrieval and to attend to information across feature channels, ET employs a LUT-based feedforward layer.

❶ **Attention** ET uses the self-attention layer for attending to tokens and gathering context. As described in Section 2.1, this involves the input tokens being passed through the embeddings,

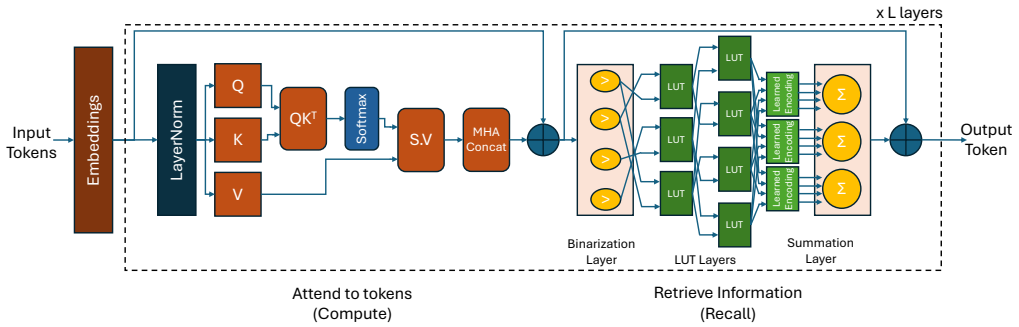

Fig. 5. Proposed design for ET. The input tokens are passed through a typical self-attention block, where computations are performed to attend to the tokens. The output activations are then passed to the recall block. Here the activations are first binarized based on learned thresholds, and the outputs with a learned mapping are used to index into LUTs. The output from the LUTs are passed into the summation layer, where the learned encodings are accumulated.

generating Q, K, and V matrices for each head, computing QK^T, applying softmax, multiplying it by V to produce $S.V$, and weighting and concatenating attention scores across heads.

❷ **Recall** We utilize LUT-based layers to build the recall block of ET. The activations from the attention block (following a skip connection) are first passed into a binarization layer, where feature-wise learned thresholds are used to encode the activations. Note that the number of bits to which each element in the activation matrix are encoded, is configurable. The resultant output is a sequence of bits, which are mapped (learnable) as inputs to the first layer of LUTs. This mapping could be one-to-many, based on the number of bits required as inputs to all the LUTs. Each LUT presents a binary (0 or 1) output, and the sequence of bits at the output are fed into the next layer of LUTs based on a static mapping. After multiple such layers of LUTs, finally the output from each LUT is used to multiplex into a learned encoding block—if the output from LUT is 1, an encoded value is passed, while a 0 is passed otherwise. These blocks correspond to each of the feature channels (D) of the model. For each channel, the learned encodings are effectively a vector, with one element corresponding to each LUT in the previous layer. The layer aggregates the learned encodings received corresponding to all the LUTs in the final layer for each encoding. These adaptations ensure that the resultant output from this layer is of dimension D, consistent with the activations from the skip connections.

Recall Block Operation. Important to note is that the recall block incurs no multiplication overhead during inference, as it merely adds an encoded value based on which LUTs participate in the summation—while still maintaining the full-precision required by the model. It therefore, efficiently realizes the recall block with minimal computations. To emulate the independent processing of tokens typical to a feedforward block, the recall block in ET operates row-wise on the activations. The output layer contains D summation units, thus preserving the network's latent dimensionality. For smoother gradients during training, the encoded values use fp32 precision. However, post-training these values are quantized to a lower precision, consistent to that of the activations in the rest of the network.

Learning the Recall Block. Figure 6 summarizes the learned parameters and configured hyperparameters in the LUT-based recall block. Let $\mathbf{h} \in \mathbb{R}^D$ denote the row-wise activation vector entering the recall block. The Binarization Layer first converts each feature into a configurable number of

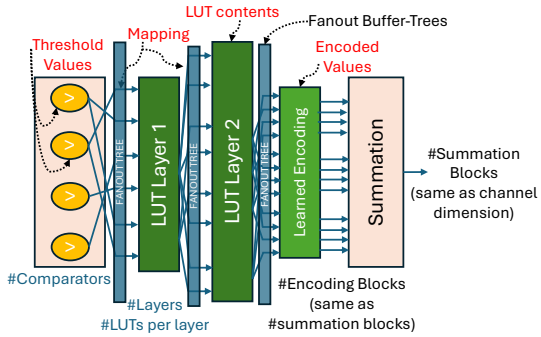

Fig. 6. Learnable parameters in the LUT block. The parameters highlighted in red are learned during training, while the ones in blue are configurable hyperparameters. The fanout buffer tree is inserted during the hardware implementation phase.

bits using learnable thresholds. For feature d and threshold index r, the binary activation is:

$$b_{d,r} = \mathbb{1}\left[h_d > \tau_{d,r}\right] = H(h_d - \tau_{d,r}), \tag{1}$$

where $\tau_{d,r}$ is a learned threshold and $H(\cdot)$ is the Heaviside step function. Since the thresholding operation is non-differentiable, we train it with a straight-through estimator (STE):

$$b_{d,r}^{\text{fwd}} = H(h_d - \tau_{d,r}), \qquad \frac{\partial b_{d,r}}{\partial h_d} \approx 1, \qquad \frac{\partial b_{d,r}}{\partial \tau_{d,r}} \approx -1. \tag{2}$$

Equivalently, the backward pass treats $H(h_d - \tau_{d,r})$ as the identity function applied to $(h_d - \tau_{d,r})$, allowing the thresholds to be optimized jointly with the rest of the model.

The resulting bit vector $\mathbf{b} \in \{0,1\}^B$ is mapped to the inputs of the first LUT layer through a learnable mapping. Following DWN [4], we associate a trainable score matrix $\mathbf{W} \in \mathbb{R}^{B \times Q}$ with the Q input connections of the next LUT layer. During the forward pass, each connection selects the input bit with maximum score:

$$I(\mathbf{W}, \mathbf{b})_q = b_{\arg\max_p W_{p,q}}, \qquad q = 1, \ldots, Q. \tag{3}$$

Thus, the learned mapping is discrete at inference time and introduces no additional computation beyond fixed bit routing. During training, gradients are propagated through the mapping scores using the DWN learnable mapping rule:

$$\frac{\partial I}{\partial \mathbf{W}} = (2\mathbf{b} - 1)^\top \mathbf{G}, \qquad \frac{\partial I}{\partial \mathbf{b}} = \mathbf{G}\,\text{softmax}_{\text{dim}=0}(\mathbf{W})^\top, \tag{4}$$

where $\mathbf{G}$ is the gradient backpropagated from the selected LUT inputs. This increases the scores of connections whose input bits agree with the gradient direction while still distributing gradient to candidate connections according to their softmax probabilities.

Each LUT contains trainable real-valued contents $\mathbf{u} \in \mathbb{R}^{2^n}$ for an n-input LUT. During the forward pass, the LUT contents are binarized with an STE and indexed by the integer address formed from the selected input bits $\mathbf{a} \in \{0,1\}^n$:

$$\hat{u}_i = H(u_i), \qquad y = A(\hat{\mathbf{u}}, \mathbf{a}) = \hat{u}_{\delta(\mathbf{a})}, \tag{5}$$

where $\delta(\mathbf{a})$ converts the binary address to an integer LUT index. The STE for the LUT contents uses

$$\hat{u}_i^{\text{fwd}} = H(u_i), \qquad \frac{\partial \hat{u}_i}{\partial u_i} \approx 1. \tag{6}$$

The LUT indexing operation is made differentiable using the EFD approximation from DWN [4]. For the LUT addressing function $A : \{0,1\}^{2^n} \times \{0,1\}^n \to \mathbb{R}$, the gradient with respect to the LUT contents is:

$$\frac{\partial A}{\partial u_i}(\mathbf{u}, \mathbf{a}) = \begin{cases} 1, & i = \delta(\mathbf{a}), \\ 0, & \text{otherwise}, \end{cases} \tag{7}$$

while the gradient with respect to address bit a_j is approximated as:

$$\frac{\partial A}{\partial a_j}(\mathbf{u}, \mathbf{a}) \approx \sum_{\mathbf{k} \in \{0,1\}^n} \frac{(-1)^{(1-k_j)} A(\mathbf{u}, \mathbf{k})}{H_{\setminus j}(\mathbf{k}, \mathbf{a}) + 1}. \tag{8}$$

Here $H_{\setminus j}(\mathbf{k}, \mathbf{a})$ is the Hamming distance between $\mathbf{k}$ and $\mathbf{a}$ excluding the jth bit. Unlike standard finite difference, which only considers addresses one bit away, EFD aggregates contributions from all LUT entries and weights them by distance from the currently addressed location.

After multiple LUT layers, let $\mathbf{z} \in \{0,1\}^{N_2}$ be the binary outputs of the final LUT layer, where N_2 is the number of LUTs in that layer. The final conditional summation layer maps these binary outputs back to the transformer hidden dimension using learned encoding values $\mathbf{E} \in \mathbb{R}^{N_2 \times D}$:

$$\mathbf{o} = \sum_{i=1}^{N_2} z_i \mathbf{E}_{i,:} = \mathbf{z}^\top \mathbf{E}, \qquad \mathbf{o} \in \mathbb{R}^D. \tag{9}$$

During training, this operation is implemented as a matrix multiplication. During inference, it becomes a conditional summation: the encoding vector $\mathbf{E}_{i,:}$ is accumulated only when the corresponding LUT output z_i is 1. The encoding values are initialized using He initialization [14]; they are trained in fp32 for smoother optimization and quantized after training to match the activation precision of the rest of the transformer.

Configurability. Note that the number of bits of the initial encoding, the number of LUTs in each layer, and the number of layers of LUTs are configurable hyperparameters. This is analogous to a feedforward layer, where the intermediate size in the MLP block is a configurable parameter. In our experiments, we perform a grid search and identify suitable values for these hyperparameters, and summarize them in Section 4.

Model Definition and Training. All layers in ET, including the recall block are defined in PyTorch as regular layers. As the gradients and mappings are well defined for all the layers, the entire model is trained and evaluated end-to-end. Note that this model is compatible with huggingface [53] and any optimized training library, and can be trained and fine-tuned on any dataset similar to a typical language model.

Architectural Details. Table 2 summarizes the architectural details of the default LUT block in ET. In the default configuration, we use two layers of LUTs, with each LUT in the layers being 6:1 LUTs—though this can be configured differently. We choose this to be the default configuration, as it aligns with the two feed-forward layer structure seen in typical MLPs present in transformer blocks. The LUTs inherently capture the non-linear representations, and therefore do not require additional intermediate non-linear activation functions. In our experiments, we observed that with learnable mappings enabled, adding more LUT layers does not yield any significant model performance improvements. Further, we note that we pick 6:1 sized LUTs in our default configuration considering the underlying logic fabric of FPGAs (6:1 LUTs), while also being consistent with prior works [4]. Based on the learned mapping for the outputs from Layer1 to the inputs of LUTs in Layer2, each output drives 12 inputs in the second layer on an average. This fanout is handled by using a buffer-tree based implementation, with a depth of 4. The second layer of LUTs connects to the summation layer (learned encodings), with each LUT connected to every summation block, with a buffer tree handling this fanout. The summation block performs the accumulation from the

Table 2. ET Architecture Details

Configuration Parameter	Default Value
Total # LUTs	768
# LUTs in Layer1	256
# LUTs in Layer2	512
Inputs to each LUT in all layers	6
Average LUT fanout in Layer1	12
Layer-1 Fan-out Buffer Tree Depth	4
# Summation Blocks	768
LUT fanout in Layer 2	768
Layer-2 Fan-out Buffer Tree Depth	10
# Comparators per input for binarization	8

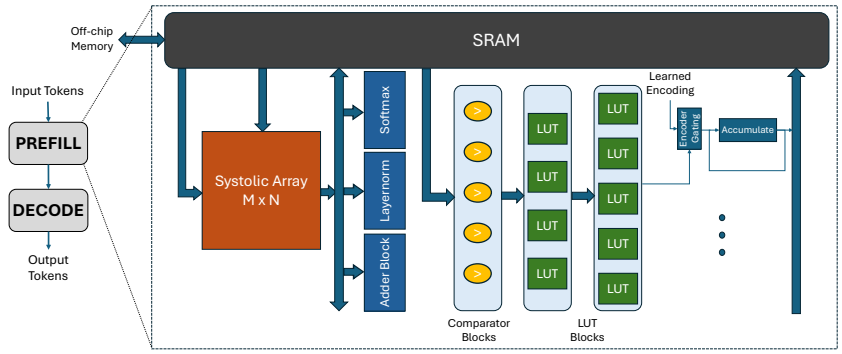

Fig. 7. Proposed hardware accelerator design for ET. The systolic array block caters to the computations involved in the MHA block, and the LUT blocks are implemented to realize the LUT-based feedforward block. For generative models (GPT), this block is replicated for the decode phase with 1D systolic arrays.

LUTs in previous layer serially. The binarization layer uses hardware comparators to compare the input values against thresholds valued learned during training—if the input is greater than the threshold, the corresponding output bit is set to 1 (and 0 otherwise). In the default case we consider 8 comparators, encoding each input value into a string of 8-bits.

3.2 Hardware Accelerator Design

We propose a custom inference-only hardware accelerator design for ET, targeting ASIC and FPGA devices. Figure 7 shows an overview of our design. Based on our characterization, we observe that for inference of generative models like GPT, while the prefill phase is compute bound, the decode phase is largely latency bound. Most operations in the prefill phase can be categorized as matrix operations, but in the decode phase, these are vector operations, as only one token is processed at a time. As the amount of computations involved in the two is highly disproportionate. Consequently, to ensure resources are optimally utilized, we design separate blocks to cater to the prefill and decode phases in a pipeline.

Prefill Block. Within the prefill block, the ET accelerator consists of a pipelined $M \times N$ systolic array block, with separate dedicated blocks to compute softmax, layernorm and skip connections (adder block). These special blocks are implemented based on the algorithm proposed in I-BERT [20]. We note that while newer quantized variants for these blocks exist, they do not have a high impact

on the overall efficiency, as the energy consumption from these blocks is only a small fraction of the total consumption per inference. For the LUT-based feed-forward block, the design consists of an initial comparator block, where the input activations are encoded into a sequence of binary values based on the learned thresholds. This operation is done in parallel along the feature dimension, as the subsequent layers would require all the bits to index into LUTs—the mapping could be in any randomized order. The following LUT layers are also implemented to be operating in parallel, such that all the LUTs output a bit each cycle. Note that these LUTs get mapped to the logic slices (CLBs) on FPGAs, or as gates on ASICs, and do not incur any memory overheads. For the final summation layer, a 2:1 MUX selects between the learned encoding and 0 based on the LUT output. These values are accumulated over all LUT outputs, with dedicated accumulators for processing each of the feature dimensions parallely. Contrary to the previous layers, the final summation layer is not fully parallel—the accumulation over all the LUT outputs is done serially. This is done to ensure that the area of the summation network does not scale up enormously. The above process is repeated for each row in the input activations sequentially, and the outputs are accumulated back.

Multiple Decoder Blocks. As the transformer architecture is regular with decoders stacked, we could ideally have an architecture with multiple compute blocks in a streaming fashion. However, we observe that the overall throughput is bottlenecked by the decode phase, where a token has to be processed through all the layers before the next token can be processed. Therefore, we opt to schedule multiple decoders on the same compute block, one after another.

Decode Phase. To cater to the decode phase, the accelerator implements a replica of the prefill compute block, except for a minor change—the configurations of the sytolic arrays are scaled down appropriately to match the reduced computational demands. In the decode phase, only a single token is processed at a time, and the matrix operations (such as QK^T, SV) end up being vector operations. Therefore, all systolic arrays in this block are designed to have a single row to ensure high hardware utilization.

Array Configurations. For the default implementation of ET, we optimize for the GPT2 model inference, and we consider 32×32 systolic arrays in the prefill block and 1×32 systolic arrays in the decode block. Similarly, we consider an array of 16 softmax operators. These parameters are configurable, and are decided by closely time matching the operations of various blocks described below.

Scheduling. In typical edge scenarios, single batched inferences are generally employed to support the incoming requests. As a result, we only adopt a very coarse grained pipeline—prompts are pipelined between the prefill block and decode block. While a prompt is catered to by the decode block with output tokens generated one after another, the prefill block caters to the next prompt to generate the first token. All the matrix (vector for decode phase) operations within the MHA block are scheduled serially on the systolic array. Note that for our custom ASIC, we consider all the model weights to be stationary on-chip, which is reasonable for the model sizes of interest—hence there is no data movement bottlenecks in our design.

3.3 Shared LUT Block

While the LUT-based feedforward block offers an energy-efficient design, we note that when implemented on hardware, these consume stationary on-chip resources (LUTs or chip area). In models with a large number of encoder/ decoder layers, the chip area could become a limiting factor for some devices. For mitigating this in the case of extreme edge devices, we adopt an optimization technique based on Mobillama [43] in the training phase, where the feedforward blocks are shared across the decoder layers (Figure 8). Pytorch autograd automatically handles gradient flows, by creating copies of it in the computational graph, and summing up partial gradients from each instance of the reused blocks in the backward pass. With this, we ensure that we only have to

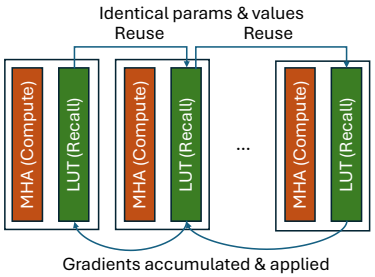

Fig. 8. LUT block sharing across the encoder/decoder layers.

implement a single LUT-block on hardware for the inference accelerator, which is reused by each layer. As our design was originally processing decoder layers serially (rolled/ unpipelined design), this does not add any latency overheads. Further, we note that in the decode phase of inference, only one token is generated at a time. Therefore, processing decoder layers serially does not impact the throughput of token generation.

3.4 ET on GPUs

In addition to a custom accelerator, ET models can also be accelerated on GPUs using optimized kernel implementations. Although EFD-based training might involve bit manipulation operations that do not map well onto GPUs, the operations performed at inference are fairly trivial. The mappings of bits to the inputs of the next layer are static, and the absence of computations to be performed in the LUT-based recall block would result in low-latency inference. To exploit the parallelism offered by GPUs, each thread in the kernel concurrently performs a LUT access, making the inference a highly parallel operation. The CUDA kernels for the recall block solely perform operations required for inference, as they no longer need to compute gradients and backpropagate them.

We also integrate our design with vLLM [22], a popular framework for inference serving systems. For the attention block, we utilize existing kernel implementations that support PagedAttention and other optimizations.

As we noted earlier, ET models are smaller in size compared to baselines, since the total parameter count in a conventional model is dominated by the feedforward layers. As shown in Figure 9, with ET inference serving, a reduced model size results in increased memory availability for KV cache and activations. This offers support for batching more requests concurrently, enabling improved throughput and reduced total execution time.

4 Evaluation

4.1 Experimental Setup

Model Training and Evaluation. We define ET models in PyTorch and train them using the Huggingface trainer library [53]. As we are particularly interested in edge transformers, we use small datasets specifically geared towards this for training. Particularly, we use the 10M token dataset from the BabyLM challenge [51] and the Wikitext-2 dataset [38] with 2M tokens for training, and report the train perplexity and downstream task accuracy. We adopt the following methodology for comparison—for the chosen baseline model, we retain the same attention block in ET, and construct the LUT-based feedforward layer with two layers of LUTs, with each layer having the same number of LUTs as the neurons in the baseline. The encoding values are trained with fp32 precision, and the entire network including the encoding is quantized post-training to 8-bit precision.

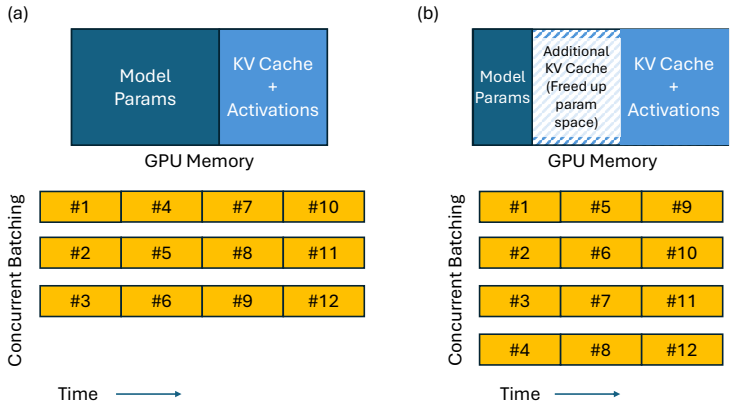

Fig. 9. (a) Inference serving multiple requests on a GPU. Based on the available memory for KV cache and activations, multiple requests are batched together to optimize for throughput. (b) With ET, a lower model size offers increased memory for KV caching, thereby offering a higher batch size that can be supported on the GPU. This results in an increased throughput of ET.

Training Setup. We train these models on a NVIDIA A100-40GB GPU on a system with an AMD EPYC 64-core processor. We adopt the same learning rate as the baseline model, and use the AdamW optimizer for training.

Hardware Accelerator Implementation. We implement the ET hardware accelerator in RTL, synthesize it and compare its performance against baselines. Parameterized RTL is written for the systolic array blocks. For the LUT-based block in ET, RTL code is generated from the saved PyTorch model using custom mako templating [6] as described next. For fair comparisons, we consider INT8 quantization of weights and activations (W8A8) for our hardware baselines as proposed in SmoothQuant [54].

RTL Generation. We write templated RTL codes for the LUT blocks in SystemVerilog using mako [6], with parameters for the learned values, including LUT contents and interconnects between LUTs. A python script is used to parse the model checkpoint, and dump the thermometer encoding thresholds, LUT entries, encoded values, and mapping information for each LUT block in the network. The scripts then parse through these dumped model files, and create SystemVerilog codes by instantiating LUTs with appropriate encoded values and interconnects using the pre-written RTL template codes.

ASIC Evaluation. To perform an area and power analysis of our design on a custom **ASIC**, we employ the Synopsys Design Compiler Q-2019.12-SP5-5 logic synthesis tool. The RTL is elaborated into an intermediate format that captures the design hierarchy. Post this, the tool performs optimization and synthesis, mapping the RTL to the standard cells in the specified technology library. For our experiments, we use the TSMC 7 nm library, with area and power optimization options, and synthesize at a target clock frequency of 1 GHz. Following synthesis, area and power optimizations are performed. We estimate the latencies for the operations using cycle-accurate simulation, and report the energy consumed based on the reported power. The on-chip SRAM is modeled by considering multiple 64 KB banks, each with a byte-wide single port operating at 1 GHz. This is implemented using the memory compiler from the same TSMC 7 nm library. We note that in the ASIC implementations of the accelerator, the LUT functions are physically realized using logic gates by the synthesis tool.

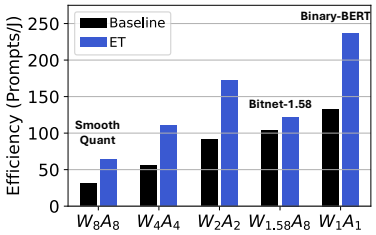

Fig. 10. Throughput per Joule of five variants of ET (blue bars) and corresponding five baseline 7 nm ASIC accelerators (black bars) with varying quantization schemes. The fourth pair is the popular bitnet 1.58.

FPGA Evaluation. On FPGA, we evaluate a few design choices for systolic arrays, and the LUT-block. For comparison with prior work, we also report energy and latency projections with a scaled down version of our design. We consider the Xilinx xcvu9p-flgb2104-2-i part and ZCU102 (16 nm technology node) as our target devices for the design choice study and comparison, respectively. We use the Xilinx Vivado 2022.1 tool and synthesize the RTL with a target clock frequency of 200 MHz. The synthesized design is placed and routed on the target FPGA without utilizing DSPs, and hardware resource utilization reports are generated. A vectorless power estimation for this design is performed, using a default switching activity factor of 12.5% to estimate workload-agnostic dynamic power in the reports. While vectorless power estimation has limitations, it ensures that a diverse range of toggle configurations are accounted for while being test-prompt agnostic, without the overheads of unrealistically high simulation times. BRAMs are used to realize the on-chip SRAM blocks, and the LUTs get directly mapped to the physical LUTs on the FPGA by the synthesis tool.
GPU Acceleration. We evaluate our design on a NVIDIA T4 GPU available on Google colab. With the vLLM framework, in order to simulate an edge-constrained environment, we restrict the GPU memory utilization to two configurations—2 GB and 1 GB, typical ranges in edge GPUs, using a vLLM flag. We perform evaluations on the CNN-Dailymail [36] and Alpaca [42] dataset, and report the achieved throughput and energy efficiency for processing the requests, measured using nvidia-smi. To minimize measurement variations, we repeat the experiments for 100 iterations, and report the average statistics.
Energy Efficiency and Latency. We consider energy efficiency (prompts/ Joule) and latency per prompt for our evaluations. We evaluate on a range of models and precisions, like quantized GPT2, BinaryBERT, and Bitnet. In our evaluations, we consider the configurations of a GPT2 125M model with a context length of 1024, and a decode phase that generates 10 tokens, unless specified otherwise.

4.2 Results

Hardware Acceleration. Figure 10 shows the energy efficiency of the ET accelerator on ASIC against multiple models with varying precisions (GPT-2 like backbone, i.e., model configurations), implemented on the same technology node. For each of these precisions, we consider baseline accelerators that use systolic arrays for the computations in the original MLPs—with the rest of the accelerator design remaining the same as ET. As could be seen, across the standard precisions considered by prior works, ET achieves about 1.2×–2× higher energy efficiency over the corresponding optimized quantized baseline accelerator, with similar results observed on FPGA evaluations. These energy savings are primarily achieved by the elimination of MAC operations corresponding to the original MLP layers. Further, for an ET designed with a Bitnet-1.58b [24] baseline, our accelerator achieves nearly 1.2× higher energy efficiency. Though Bitnet is efficient, it still contains many non-binary multiplications – while with LUT-based FF layers, ET eliminates all MACs in feedforward layers, and delivers increased throughput per Joule.

Table 3. Comparison of Our Work Against other Edge Optimized Accelerators on the ZCU102 Platform

Work	Model Variant	LUTs	DSPs	Freq. (MHz)	Compute Xput(GOPS)	Xput (Prompts/s)	Power (W)	Efficiency (Prompts/J)
BETA [16]	BinBERT, W1A1	191K	64	190	1,387.6	258.3	7.95	32.49
EFA-Trans [56]	Sp-BERT, W8A8	197K	1,532	150	279.8	52.1	5.48	9.51
BAT [15]	BinBERT, W1A2	146K	1,024	200	1,122.4	209.1	5.47	38.22
FQ-BERT [23]	BERT, W4A8	123K	1,751	214	22.7	4.22	9.80	0.43
ET (this work)	BERT, $W_{8/LUT}A8$	145K	16	200	513.4	242.7	4.73	51.31

For fair comparison, we consider the BERT model with a latent dimension of 384, and a sequence length of 128 for all these works.

Table 4. Overall Performance Evaluation for Prefill (Per Prompt) and Decode (Per Token) Phases, for a BERT like ET Decoder Model (Latent Dimension of 384), and Sequence Length of 128, on the ZCU102 FPGA

	Prefill (per prompt per decoder)		**Decode (per token per decoder)**	
Stage	Latency (cycles)	Energy (mJ)	Latency (cycles)	Energy (mJ)
Q, K, V	73,728 (each) (52.9%)	1.73 (each)	9,216 (each) (63.2%)	0.22 (each)
QK^T	24,576 (5.9%)	0.58	3,072 (7.0%)	0.07
Softmax	24,576 (5.9%)	0.58	384 (0.9%)	0.01
$S.V$	24,576 (5.9%)	0.58	3,072 (7.0%)	0.07
Multi-Head Concat	73,728 (17.6%)	1.73	9,216 (21.0%)	0.22
Feed-forward	49,152 (11.7%)	1.16	387 (0.9%)	0.01
Total (for all decoders)	5,013,504	117.84	525,348	12.48 (all decoders)

The other non-major contributors of energy are omitted for clarity.

Comparisons against prior FPGA implementations. Table 3 compares our work against prior works on highly quantized implementations of BERT models on FPGA. To fit within the constraints of the FPGA, we consider a smaller design with a 16×16 systolic array. As seen, even with our 8-bit precision variant, we achieve higher efficiency than all prior works, including the 1-bit models. This is because incase of ET, the 8-bit precision computations are only performed in the attention layers, and the lookup operations in the LUT block are performed much more efficiently. We note that ET models report a lower compute throughput (GOPS), as the absence of the MLP layers eliminates a major fraction of the required computations. We also note that our quantized systolic arrays are implemented using LUTs, rather than DSP slices, resulting in a low DSP utilization. We also note that the operating frequency of our design is largely limited by the summation layer in the LUT-based recall block.

Layerwise Breakdown. Tables 4 and 5 show a layerwise breakdown of the performance of the prefill and decode phases of a model served on the accelerator, on a FPGA (BERT-like backbone) and ASIC (GPT-2 like backbone), respectively. The LUT-based feedforward block shown offers about 22× energy reduction as opposed to a conventional MLP, thus resulting in significant overall savings compared to the quantized baselines.

Comparison against other edge accelerators. Table 6 compares our work against some other works that implement small transformer models. While this is not a one-to-one comparison as they are on different platforms, it provides a perspective of our work in the broader landscape.

GPU Acceleration. Figure 11 compares the performance of ET against baseline on a NVIDIA T4 GPU for a single-batched inference, and shows that ET achieves improved latency and 1.12× energy efficiency. We also simulate the performance of ET targeting a Jetson Orin configuration with AccelSim simulator [19], and report a 1.15× latency improvement.

Table 5. Overall Performance Evaluation for Prefill (Per Prompt) and Decode (Per Token) Phases, for a GPT-2 like ET Decoder Model with a Context Length of 1024, on a 7 nm ASIC Implementation

	Prefill (per prompt per decoder)		**Decode (per token per decoder)**	
Stage	Latency (cycles)	Energy (uJ)	Latency (cycles)	Energy (uJ)
Q, K, V	589,824 (each) (32.1%)	183.5 (each)	18,432 (each) (44.4%)	2.2 (each)
QK^T	786,432 (14.2%)	183.5	24,576 (19.7%)	2.2
Softmax	786,432 (14.2%)	0.2	768 (0.6%)	<0.01
$S.V$	786,432 (14.2%)	183.5	24,576 (19.7%)	2.2
Multi-Head Concat	589,824 (10.7%)	183.5	18,432 (14.8%)	2.2
Feed-forward	789,504 (14.3%)	53.1	771 (0.6%)	1.6
Total (for all decoders)	66,097,152	13,851.6	1,493,028	177.6 (all decoders)

The other non-major contributors of energy are omitted for clarity.

Table 6. Comparison of Our Work Against other Edge Optimized Accelerators on Similar Sized Models

Work	Model Variant	Platform	Efficiency (Prompts/J)
Matmulfree [58]	MatmulfreeLM-370M	D5005	1.70
Matmulfree [58]	MatmulfreeLM-370M	Loihi-2	0.58
Alireo [28]	Alireo-400M	GPU	0.17
LUT-GEMM [31]	GPT-350M	T4 GPU	0.38*
Bitnet-1.58b [24]	Bitnet-350M	7nm ASIC	51.97
ET (this work)	ET-Bitnet-350M	7nm ASIC	61.05

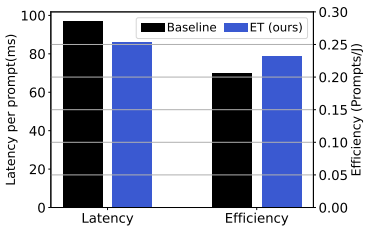

Fig. 11. Performance comparison in terms of latency per prompt and energy efficiency (#prompts per Joule) of ET vs. baseline model for single batched inference on a NVIDIA T4 GPU.

Figure 12 shows the performance of ET on memory constrained GPU systems compared against baseline GPT2 and GPT2-medium models with the vLLM inference framework. As seen, ET offers higher performance improvements in case of highly memory constrained scenario (1 GB), such as those seen in edge environments—with it enabling inference of a GPT2-medium backbone based model, while the baseline could not be served due to **out-of-memory error**. Therefore, with the reduction in weight storage possible by LUTs, language models that could not otherwise fit on edge GPUs can now be fit. For the system with 2 GB memory, the performance improvements are noticeable with the larger GPT2-medium model baseline, in terms of improved throughput and energy efficiency. This is made possible primarily by a $\sim$ 60% reduction in weight storage with the LUT layers, offering more memory for KV caches, and so on, enabling faster GPU inference. The energy efficiency achieved is offered by a $\sim$ 50% reduction in MAC computations, replaced by relatively cheaper bitwise operations in the LUTs. As noticed with the GPT2 variants on 2 GB and 4

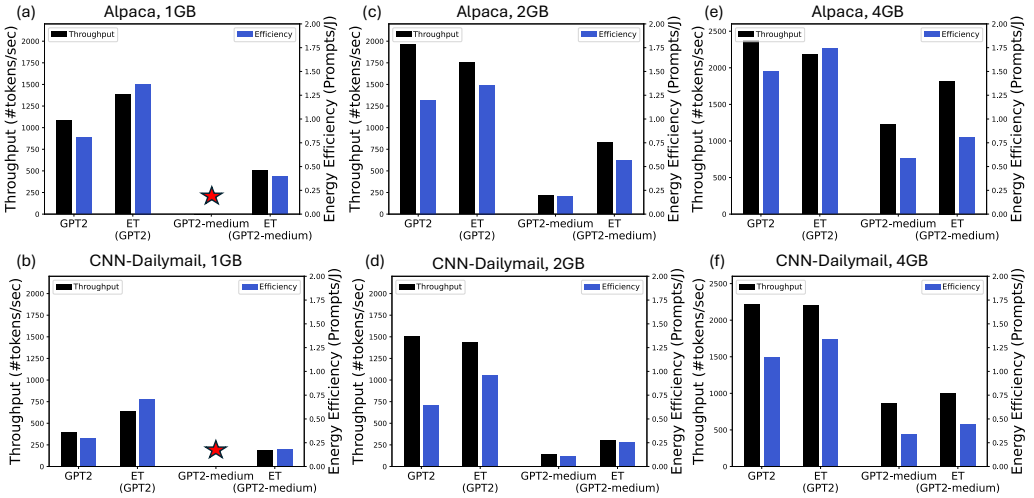

Fig. 12. Performance comparison in terms of throughput (#tokens/sec) and energy efficiency (prompts/J) of ET vs. baseline models for software acceleration on the CNN-Dailymail[36] and Alpaca datasets [42] for GPU memory configurations of 1 GB, 2 GB, and 4 GB. A red star denotes the model could not be run due to CUDA out-of-memory error. ET (GPT2) refers to the ET model designed with the GPT2 model backbone (configuration), and ET (GPT2-medium) refers to the ET model designed with the GPT2-medium model backbone (configuration).

GB constrained GPUs, the throughput of ET is worse than the baseline—as the additional memory savings offered by ET is not significant enough compared to the overall GPU memory. We note that Figure 12 particularly considers the scenario of inference serving using the vLLM framework on a GPU, with a large batch size as opposed to the cases of single batched inference considered in other experiments. In such a scenario it is observed that the optimized implementation for the conventional MLP in vLLM efficiently leverages the Tensor Cores in the GPUs to achieve a higher effective throughput, as opposed to the LUT-based feedforward block, which relies on the CUDA cores that are fewer in number—resulting in a lower throughput. However, in terms of the energy required to perform these operations, the lookup operations (in the LUTs) are much cheaper than the energy-intensive MAC operations in conventional MLPs. Therefore, we see gains in terms of tokens/ joule.

Cross-Platform Comparisons. As shown in Figure 13(a), for a GPT-2 backbone, in terms of end-to-end latency per prompt, ET achieves about 1.7 × improvement against a baseline ASIC accelerator. Further, as seen in Figure 13(b), ET achieves a 2.1× energy efficiency against baseline ASIC accelerator.

Impact on Accuracy. Table 7 shows the model perplexity (lower is better) of ET against the corresponding baselines on which they were built, during model training. As seen, for most of these, ET achieves comparable perplexity as the baselines—demonstrating that the LUT-based feedforward block does not introduce any significant accuracy degradation when trained from scratch. Table 8 shows the performance of ET against the corresponding Mobillama [43] baseline pretrained on BabyLM-10M corpus, on various downstream tasks, demonstrating similar accuracies.

5 Sensitivity Analysis

Scaling latent dimensions. Figure 14 shows the variations in area, power and latency of the feed-forward block with latent dimensions of the network. As illustrated in the figure, all these performance metrics scale roughly linearly for each of the constituent blocks. However, the results

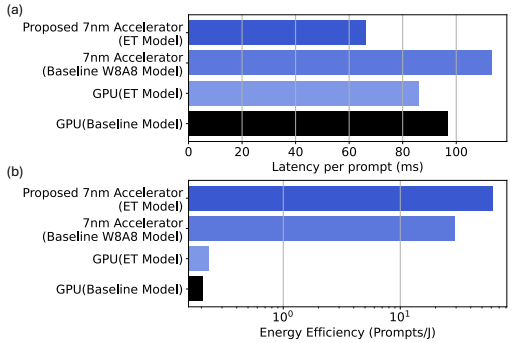

Fig. 13. Performance improvements of ET over baselines (a) latency per prompt, (b) energy efficiency.

Table 7. Model Train Perplexity (Lower is Better) of Baseline Transformer Models vs. ET trained on Different Datasets

Model Variant	#Params	Dataset	Baseline	**ET (this work)**
GPT-2 [35]	117M	Wikitext-2	3.52	3.82
MM-Free [58]	370M	Wikitext-2	5.05	5.53
Mobillama [43]	84M	Wikitext-2	6.04	6.68
Mobillama [43]	84M	Bookcorpus	21.28	23.17

Table 8. Downstream Task Accuracy of Baseline Transformer Model vs. ET pre-Trained on BabyLM-10M Data Corpus

Suite	Task	Mobillama [43]	**ET (this work)**
	qqp	61.5	63.2
	cb	41.1	41.1
	copa	63.0	63.0
SuperGLUE	boolq	41.3	37.8
	rte	52.7	52.7
	wic	52.5	50.0
	wsc	63.5	63.5
BLiMP	Agg.	53.7	55.8

indicate that for very large latent dimensions, the area requirement, particularly of the summation layer, would be significantly large—potentially exceeding practical area budgets. This suggests that for larger model sizes beyond the scope of this work, a redesign of the accelerator design would be warranted, in order to fit within edge area constraints. Potential methods to address this include instantiating fewer adder blocks (partially serializing the summation), implementing LUTs and interconnects using memory cells and crossbars (enabling reuse of LUTs within and across layers), among others.

Varying context length. As the feed-forward block operates on rows of the activation matrix sequentially, with increasing context lengths, the total energy consumption and latency would scale linearly. However, this does not affect the area or power of the implementation, which would remain constant.

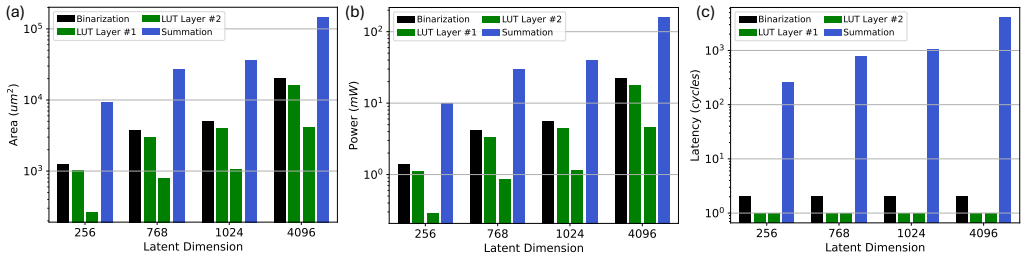

Fig. 14. Variation in (a) area (b) power, and (c) latency of the components of the feed-forward block in ET with model dimensions, on a 7 nm technology node.

Table 9. Performance Comparison of Various Pipelined Systolic Array Configurations for the GEMMs in a Feed-forward (MLP) Block in GPT2 Against the Corresponding Feedforward Block in the Equivalent ET Model

		FPGA				ASIC (7nm)	
Method	1/Xput (cycles)	LUTs	FFs	BRAMs	Energy/inf	Area (mm^2)	Energy/inf
8×8 Systolic	75,497,472	7,792	3,178	0	0.047 J	0.05	3838.8 uJ
16×16 Systolic	18,874,368	25,266	12,522	3	0.026 J	0.11	2054.7 uJ
32×32 Systolic	4,718,592	94,619	50,545	11	0.015 J	0.25	1170.2 uJ
64×64 Systolic	1,179,648	370,939	203,023	27	0.011 J	0.57	753.2 uJ
ET Feed Forward Block	789,504	413,248	124,547	0	0.004 J	0.04	53.1 uJ

The MLP block considered here is a GPT-2 model with a context length of 1024, involving two GEMMs of dimensions $(1024 \times 768)^*(768 \times 3072)$ and $(1024 \times 3072)^*(3072 \times 768)$.

Varying Systolic Array Configurations. Table 9 compares the performance of various pipelined-systolic array configurations typically used in the literature [17, 55]. As seen, the implementation achieved with ET is more energy efficient than 32×32 block computing the MLP layers, by about 22×. Note that the larger systolic arrays require BRAMs to be instantiated to stage the operands for the matrix computations.

Varying number of layers. As the number of decoder layers in the model increases, the energy consumption of the model increases linearly. With the Mobillama [43] approach of shared feed-forward blocks, the associated area of the ASIC implementation remains unchanged.

LUT Block Hyperparameters. We can infer the performance impacts of the configurable hyper-parameters in the LUT-based recall block from Figure 14. While the summation block dominates the overall area, power and latency, it is important to note that this (number of summation units) is affected by the latent dimension, D, which is determined by the overall network. As the number of LUTs in each layer increase, the area and power consumption of that layer increase linearly, while the corresponding latency remains constant owing to parallel execution. However, as noted, since the LUT layers only contribute a small fraction to the total, the overall increase is significantly low. Similarly, the number of comparators in the binarization layer also linearly impact the area and power for that layer, though the overall increase is minimal. In this case too, the latency remains constant.

6 Discussion

LUTs do not trade off memory for compute. The LUT-based implementation of the feedforward layer does not introduce any additional memory access overhead. Furthermore, on FPGAs/ASICs, the LUTs are implemented in logic slices, and does not incur the typical memory access latencies.

ASIC Design Adaptability. We note that the LUT contents and the routing between the LUTs are learned parameters. Therefore, though these values and the model configurations can be altered during training, once the accelerator is designed, these learned parameters get implemented as logic functions or gates in hardware—essentially, "engrained" in hardware. This goes beyond classic neural network implementation dataflow, which keeps weights in nearby SRAM but still fetches them at runtime; by contrast, ET's recall block eliminates runtime weight storage and movement by compiling parameters into logic. This is unlike conventional model implementations where there is an external dependency in the form of model weights.

GPU inference kernel savings shown are conservative. The performance improvements shown for software acceleration are conservative, as the LUT entries are stored in higher precision due to memory access limitations with CUDA/ C++, explaining the reason for the modest improvement of only 1.12× on GPUs. On a platform with efficient bit-level operations, ET is expected to show higher speedup.

GPU training Limitations. Due to the limitations with existing GPU technologies, the bitwise operations in training LUTs are less efficient when performed on them—resulting in large training times compared to the baselines. However, since this work is primarily designed as an inference-efficient solution, the one-time training cost, which is amortized over numerous inferences after deployment, is not a major concern.

Model Sizes and Datasets for evaluation. Owing to resource constraints, the evaluations presented in the paper are restricted to smaller models like GPT2 and BERT, and datasets that are relevant in the edge context. We can estimate the area and energy benefits for larger models and they are similar to what is observed for the smaller models (since approximately half the hardware is in the MLP layers in them and would be eliminated). But we present the smaller models since resource limitations prevent us from training and demonstrating iso-accuracy in the case of larger models, which take thousands to millions of GPU hours to train [48].

GPU Architecture Limitations. Current GPU architectures lack inherent support for low-bit precisions, owing to which the latency savings offered by ET are not the most optimal. However, even with just the software optimizations, we still achieve about 1.12× improvement in energy efficiency. Architectural optimizations to support low-precision datawidths have been added to tensor and CUDA cores in GPUs and more low-precision datawidth support is expected in the future. Support for LUT based models on GPUs could be explored as a future direction.

FPGA Reconfiguration: When deploying ET designs on FPGAs, the resource constraints of the target FPGA determine the largest model that can be fit on it. Reconfiguring the FPGA is typically a slow process (ranging up to a few seconds), with a serial JTAG port hindering real-time reconfiguration. As a consequence, commercial FPGAs can only support ET designs if all LUT layers can be fit within the resource constraints. The actual reconfiguration time for the FPGA, is a few seconds using a conventional serial JTAG port. However, reconfiguration time optimizations such as FPGA architectures utilizing multiple scan chains [1, 7] (100× reduced time by 100 scan chains) or optimized frame-based configuration methods [25, 50] could potentially make it feasible to implement larger ET models, beyond the resource constraints of the FPGA.

7 Related Work

In recent years, numerous studies have explored energy-efficient transformer architectures and accelerators. SwifTron [26] introduces a specialized integer-arithmetic-based accelerator. BinaryBERT [5] and Bitnet 1.58b [24] employ extreme quantization, and traditional MAC operations are replaced with ternary sum operations.

Recently, a transformer architecture was proposed that approximates floating-point multiplications with piece-wise affine functions, achieving comparable performance while reducing

computational demands primarily for energy savings [21]. LUT-GEMM [31] introduces an approximate matmul method using LUTs by substituting multiplications with lookup operations to achieve post-training speedup. In contrast, our work focuses on employing LUTs as the core computational units or "neurons" of the model, learning functions directly during training. As a result, we not only construct models with fewer model weights, but also minimize the consequent number of memory accesses as compared to these multiplication-free models. Further, we show that our technique is complimentary to works like Bitnet [24], and a quantized attention block could be integrated with the feedforward block in ET for an optimized model, that could be served on the proposed accelerator.

This work builds on our recent work, LL-ViT [29], where we employ LUT based neurons to improve the efficiency of vision transformers deployed on edge FPGAs. While LL-ViTs were restricted to vision transformers and FPGAs, this work targets a wide range of language models, and accelerates their inference across FPGAs, ASICs and GPUs. Furthermore, we incorporate algorithm-hardware enhancements to support the deployment of relatively larger language models on ASICs without incurring unreasonably high area overheads, with the inclusion of a Mobillama [43] based approach.

8 Conclusion

The deployment of transformer models at the edge is a growing trend, driven by the need for real-time processing, and enhanced privacy. In this work, we introduce ET, a novel algorithm hardware codesign technique for efficient language model inference at the edge. We draw analogies with the human brain, and identify an opportunity to increase the energy-efficiency of language model inferences by utilizing LUT-based neurons for information retrieval. By eliminating the feedforward layers in conventional transformer-based models and introducing LUT-based layer, ET achieves roughly 60% reduction in computations and model weights across typical transformer models. Based on our model and performance evaluations, we report significant energy savings when deployed on a constrained GPU and a custom ASIC. Compared to vanilla GPU implementations, ET achieves a 1.12× energy efficiency with software acceleration. Further, our results demonstrate how this technique could be leveraged to deploy models on edge GPUs, that otherwise could not be fit due to memory constraints. These results illustrate that ET is a promising lightweight alternative to traditional transformers—paving the way for energy-efficient tiny language models.

References

[1] Mohamed Abdelfattah. 2011. Evaluation of Advanced Techniques for Structural FPGA Self-Test, MSC Thesis. Retrieved from https://www.mohsaied.com/assets/pdf/msc_thesis.pdf

[2] Marta Andronic and George A. Constantinides. 2023. PolyLUT: Learning piecewise polynomials for ultra-low latency fpga lut-based inference. In *Proceedings of the 2023 International Conference on Field Programmable Technology (ICFPT)*. IEEE. DOI: https://doi.org/10.1109/icfpt59805.2023.00012

[3] Marta Andronic and George A. Constantinides. 2024. NeuraLUT: Hiding neural network density in boolean synthesizable functions. In *Proceedings of the 2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 140–148. DOI: 10.1109/FPL64840.2024.00028

[4] Alan T. L. Bacellar, Zachary Susskind, Mauricio Breternitz Jr., Eugene John, Lizy K. John, Priscila M. V. Lima, and Felipe M. G. França. 2024. Differentiable weightless neural networks. In *Proceedings of the 41st International Conference on Machine Learning (ICML'24)*. JMLR.org, Vienna, Austria.

[5] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jin Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. 2021. BinaryBERT: Pushing the limit of BERT quantization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Association for Computational Linguistics, Online, 4334–4348. DOI: https://doi.org/10.18653/v1/2021.acl-long.334

[6] Michael Bayer. 2021. Mako Templates for Python. Retrieved from https://www.makotemplates.org/

[7] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. 1999. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, USA.

[8] Severin Bochem, Victor J. B. Jung, Arpan Suravi Prasad, Francesco Conti, and Luca Benini. 2025. Distributed inference with minimal off-chip traffic for transformers on low-power MCUs. In *2025 Design, Automation & Test in Europe Conference (DATE)*. 1–7. DOI: https://doi.org/10.23919/DATE64628.2025.10992712

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS'20)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Curran Associates Inc., Vancouver, BC, Canada.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. DOI: https://doi.org/10.18653/v1/N19-1423

[11] Mor Geva, Avi Caciularu, Kevin Wang, and Yoav Goldberg. 2022. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 30–45. DOI: https://doi.org/10.18653/v1/2022.emnlp-main.3

[12] Sukhpal Singh Gill, Muhammed Golec, Jianmin Hu, Minxian Xu, Junhui Du, Huaming Wu, Guneet Kaur Walia, Subramaniam Subramanian Murugesan, Babar Ali, Mohit Kumar, Kejiang Ye, Prabal Verma, Surendra Kumar, Felix Cuadrado, and Steve Uhlig. 2024. Edge AI: A taxonomy, systematic review and future directions. *Cluster Computing* 28, 1 (October 2024). DOI: https://doi.org/10.1007/s10586-024-04686-y

[13] The SHD Group. 2025. Edge-AI Market Analysis: Applications, Processors Ecosystem Guide. Retrieved from https://theshdgroup.com/wp-content/uploads/2025/04/Edge-AI-2025-Report-Final.pdf

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*. 1026–1034.

[15] Yuhao Ji, Chao Fang, Shaobo Ma, Haikuo Shao, and Zhongfeng Wang. 2025. Co-Designing binarized transformer and hardware accelerator for efficient end-to-end edge deployment. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design* (Newark Liberty International Airport Marriott, New York, NY, USA) *(ICCAD '24)*. Association for Computing Machinery, New York, NY, USA, Article 180, 9 pages. DOI: https://doi.org/10.1145/3676536.3676733

[16] Yuhao Ji, Chao Fang, and Zhongfeng Wang. 2024. BETA: Binarized energy-efficient transformer accelerator at the edge. In *Proceedings of the 2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. DOI: https://doi.org/10.1109/ISCAS58744.2024.10558636

[17] Liancheng Jia, Liqiang Lu, Xuechao Wei, and Yun Liang. 2020. Generating systolic array accelerators with reusable blocks. *IEEE Micro* 40, 4 (2020), 85–92. DOI: https://doi.org/10.1109/MM.2020.2997611

[18] Christoforos Kachris. 2025. A survey on hardware accelerators for large language models. *Applied Sciences* 15, 2 (Jan. 2025), 586. DOI: https://doi.org/10.3390/app15020586

[19] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020. Accel-sim: An extensible simulation framework for validated GPU modeling. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture* (Virtual Event) *(ISCA '20)*. IEEE Press, 473–486. DOI: https://doi.org/10.1109/ISCA45697.2020.00047

[20] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. I-BERT: Integer-only BERT quantization. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Marina Meila and Tong Zhang (Eds.). Vol. 139, PMLR, 5506–5518. Retrieved from https://proceedings.mlr.press/v139/kim21d.html

[21] Atli Kosson and Martin Jaggi. 2024. Multiplication-free transformer training via piecewise affine operations. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 360, 16 pages.

[22] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

[23] Zejian Liu, Gang Li, and Jian Cheng. 2021. Hardware acceleration of fully quantized bert for efficient natural language processing. In *Proceedings of the 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 513–516. DOI: https://doi.org/10.23919/DATE51398.2021.9474043

[24] Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. 2024. The era of 1-bit LLMs: All large language models are in 1.58 bits. arXiv:2402.17764. Retrieved from https://arxiv.org/abs/2402.17764 (2024).

[25] U. Malik and O. Diessef. 2005. A configuration memory architecture for fast run-time reconfiguration of FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications, 2005.* 636–639. DOI: https://doi.org/10.1109/FPL.2005.1515802

[26] A. Marchisio, D. Dura, M. Capra, M. Martina, G. Masera, and M. Shafique. 2023. SwiftTron: An efficient hardware accelerator for quantized transformers. In *Proceedings of the 2023 International Joint Conference on Neural Networks (IJCNN).*

[27] Fikadu Y. Mekonnen, Mohammad F. Al Bataineh, Dima Abu Abdoun, Ahmed Serag, Kumela T. Tamiru, Wondwosen Abula, and Samuel Darota. 2025. Development of a fully autonomous offline assistive system for visually impaired individuals: A privacy-first approach. *Sensors (Basel)* 25, 19 (Sept. 2025), 6006. DOI: https://doi.org/10.3390/s25196006

[28] Michele Montebovi. 2024. Alireo-400M: A Lightweight Italian Language Model. Retrieved from https://huggingface.co/DeepMount00/Alireo-400m-instruct-v0.1

[29] Shashank Nag, Alan T.L. Bacellar, Zachary Susskind, Anshul Jha, Logan Liberty, Aishwarya Sivakumar, Eugene B. John, Krishnan Kailas, Priscila M.V. Lima, Neeraja J. Yadwadkar et al. 2025. LL-ViT: Edge deployable vision transformers with look up table neurons. In *Proceedings of the 2025 International Conference on Field Programmable Technology (ICFPT).* 19–29. DOI: https://doi.org/10.1109/ICFPT67023.2025.00013

[30] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, et al. 2024. GPT-4 Technical Report. arXiv:2303.08774. Retrieved from https://arxiv.org/abs/2303.08774 (2024).

[31] Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, {Se Jung} Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. 2024. LUT-GEMM: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models. Publisher Copyright: © 2024 12th International Conference on Learning Representations, ICLR 2024. All rights reserved.; 12th International Conference on Learning Representations, ICLR 2024 ; Conference date: 07-05-2024 Through 11-05-2024.

[32] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) *(UIST '23).* Association for Computing Machinery, New York, NY, USA, Article 2, 22 pages. DOI: https://doi.org/10.1145/3586183.3606763

[33] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient generative llm inference using phase splitting. In *Proceedings of the 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA).* 118–132. DOI: https://doi.org/10.1109/ISCA59077.2024.00019

[34] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems*, D. Song, M. Carbin, and T. Chen (Eds.). Curan, 606–624. Retrieved from https://proceedings.mlsys.org/paper_files/paper/2023/file/c4be71ab8d24cdfb45e3d06dbfca2780-Paper-mlsys2023.pdf

[35] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. Retrieved from https://api.semanticscholar.org/CorpusID:160025533

[36] Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* Association for Computational Linguistics, Vancouver, Canada, 1073–1083. DOI: https://doi.org/10.18653/v1/P17-1099

[37] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training multi-billion parameter language models using model parallelism. arXiv:1909.08053. Retrieved from https://arxiv.org/abs/1909.08053 (2020).

[38] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Wikitext-2. arXiv:1609.07843. Retrieved from (2016). https://arxiv.org/abs/1609.07843

[39] Zachary Susskind, Aman Arora, Igor D.S. Miranda, Luis A.Q. Villon, Rafael F. Katopodis, Leandro S. De Araújo, Diego L.C. Dutra, Priscila M.V. Lima, Felipe M.G. França, Mauricio Breternitz et al. 2022. Weightless neural networks for efficient edge inference. In *Proceedings of the 31st International Conference on Parallel Architectures and Compilation Techniques (PACT).* DOI: https://doi.org/10.1145/3559009.3569680

[40] Zachary Susskind, Aman Arora, Igor D. S. Miranda, Alan T. L. Bacellar, Luis A. Q. Villon, Rafael F. Katopodis, Leandro S. de Araújo, Diego L. C. Dutra, Priscila M. V. Lima, Felipe M. G. França et al. 2023. ULEEN: A novel architecture for ultra-low-energy edge neural networks. *ACM Transactions on Architecture and Code Optimization* 20, 4, Article 61 (dec 2023), 24 pages. DOI: https://doi.org/10.1145/3629522

[41] Xiaohu Tang, Yang Wang, Ting Cao, Li Lyna Zhang, Qi Chen, Deng Cai, Yunxin Liu, and Mao Yang. 2023. LUT-NN: Empower efficient neural network inference with centroid learning and table lookup. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking* (Madrid, Spain) *(ACM MobiCom '23)*. Association for Computing Machinery, New York, NY, USA, Article 70, 15 pages. DOI: https://doi.org/10.1145/3570361.3613285

[42] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. Retrieved from https://github.com/tatsu-lab/stanford_alpaca.

[43] Omkar Thawakar, Ashmal Vayani, Salman Khan, Hisham Cholakkal, Rao Muhammad Anwer, Michael Felsberg, Timothy Baldwin, Eric P. Xing, and Fahad Shahbaz Khan. 2024. MobiLlama: Towards accurate and lightweight fully transparent GPT. arXiv:2402.16840. Retrieved from https://arxiv.org/abs/2402.16840 (2024).

[44] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar et al. 2023. LLaMA: Open and efficient foundation language models. arXiv:2302.13971. Retrieved from https://arxiv.org/abs/2302.13971 (2023).

[45] Yaman Umuroglu, Yash Akhauri, Nicholas James Fraser, and Michaela Blott. 2020. LogicNets: Co-designed neural networks and circuits for extreme-throughput applications. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. 291–297. DOI: https://doi.org/10.1109/FPL50879.2020.00055

[46] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17)*. ACM. DOI: https://doi.org/10.1145/3020078.3021744

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[48] Stephen M. Walker. 2022. *Everything We Know About GPT-4*. Retrieved October 20, 2025 from https://klu.ai/blog/gpt-4-llm

[49] Fali Wang, Minhua Lin, Yao Ma, Hui Liu, Qi He, Xianfeng Tang, Jiliang Tang, Jian Pei, and Suhang Wang. 2025. A survey on small language models in the era of large language models: Architecture, capabilities, and trustworthiness. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2* (Toronto ON, Canada) *(KDD '25)*. Association for Computing Machinery, New York, NY, USA, 6173–6183. DOI: https://doi.org/10.1145/3711896.3736563

[50] Yabin Wang, Yuan Wang, and Jinmei Lai. 2007. An FPGA configuration circuit used for fast and partial configuration. In *Proceedings of the 2007 7th International Conference on ASIC*. 157–160. DOI: https://doi.org/10.1109/ICASIC.2007.4415591

[51] Alex Warstadt, Aaron Mueller, Leshem Choshen, Ethan Wilcox, Chengxu Zhuang, Juan Ciro, Rafael Mosquera, Bhargavi Paranjabe, Adina Williams, Tal Linzen et al. 2023. In *Proceedings of the BabyLM Challenge at the 27th Conference on Computational Natural Language Learning*. Association for Computational Linguistics, Singapore.

[52] Olivia Weng, Marta Andronic, Danial Zuberi, Jiaqing Chen, Caleb Geniesse, George A. Constantinides, Nhan Tran, Nicholas J. Fraser, Javier Mauricio Duarte, and Ryan Kastner. 2025. Greater than the sum of its luts: Scaling Up lut-based neural networks with amigolut. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (Monterey, CA, USA) *(FPGA '25)*. Association for Computing Machinery, New York, NY, USA, 25–35. DOI: https://doi.org/10.1145/3706628.3708874

[53] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Qun Liu and David Schlangen (Eds.). Association for Computational Linguistics, Online, 38–45. DOI: https://doi.org/10.18653/v1/2020.emnlp-demos.6

[54] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 38087–38099. Retrieved from https://proceedings.mlr.press/v202/xiao23c.html

[55] Rui Xu, Sheng Ma, Yang Guo, and Dongsheng Li. 2023. A survey of design and optimization for systolic array-based dnn accelerators. *ACM Computing Surveys* 56, 1, Article 20 (aug 2023), 37 pages. DOI: https://doi.org/10.1145/3604802

[56] Xin Yang and Tao Su. 2022. EFA-trans: An efficient and flexible acceleration architecture for transformers. *Electronics* 11, 21 (2022), 3550. DOI: https://doi.org/10.3390/electronics11213550

[57] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8bit bert. In *Proceedings of the 2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*. 36–39. DOI: https://doi.org/10.1109/EMC2-NIPS53020.2019.00016

[58] Rui-Jie Zhu, Yu Zhang, Ethan Sifferman, Tyler Sheaves, Yiqiao Wang, Dustin Richmond, Peng Zhou, and Jason K. Eshraghian. 2024. Scalable matmul-free language modeling. arXiv:2406.02528. Retrieved from https://arxiv.org/abs/2406.02528 (2024).

Received 9 December 2025; revised 24 April 2026; accepted 3 June 2026