HW/SW Co-Design for Efficient GPT-2 Inference on FPGA via High-Level Synthesis[*]

Shao-Tang Sung[*], Yi-Wen Tang[*], Fen-Yu Hsieh[*], Rong-Yi Lin[*], Fang-Yu Hsu[*], Chih-Tsun Huang
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 300

Abstract—This paper presents an efficient hardware accelerator for GPT-2 inference on an AMD Alveo U280 FPGA using High-Level Synthesis (HLS). With row-wise GEMM scheduling, we optimize GPT-2 components via data packing, loop unrolling, and kernel fusion. Our design achieves a 2.16× speedup over CPU while consuming only 23% of the power, demonstrating a scalable and sustainable solution for deploying Transformer-based models in resource-constrained environments.

I. Introduction

Natural language processing (NLP) is a discipline centered on the computational analysis and representation of human languages [1]. It encompasses a wide range of applications, including text generation, classification, summarization, and translation. Earlier NLP models, such as recurrent neural networks (RNNs) and long short-term memory (LSTM), struggled with capturing long-range dependencies. This limitation was mitigated by the introduction of the attention mechanism in Transformer architecture [2].

Transformer-based models have since transformed NLP [3], enabling the emergence of large language models (LLMs) such as BERT (Bidirectional Encoder Representations from Transformers) [4] and GPT (Generative Pre-trained Transformers) [5]. These models utilize self-attention mechanisms to effectively capture intricate linguistic patterns.

While newer LLMs such as LLaMA [6] offer architectural optimizations, they continue to adhere to the foundational Transformer structure. GPT-2 retains this foundational structure, including multi-head attention and feedforward layers, making it both representative of contemporary LLMs and suitable for hardware exploration. Its adherence to standard Transformer configurations ensures that accelerator designs based on GPT-2 remain compatible with future model variants. GPT-2 also delivers strong zero-shot performance [7, 8]. Additionally, its open-source availability further justifies its adoption for research studies.

Despite their effectiveness, Transformer-based LLMs demand substantial computational and energy resources. While Graphics Processing Units (GPUs) offer high performance, they incur significant energy costs. Conversely, Central Processing Units (CPUs) are more energy-efficient but often lack the throughput required for real-time inference [9, 10].

To address both issues, we propose an efficient GPT-2 inference accelerator implemented on the AMD Alveo U280 FPGA platform, as FPGAs offer reconfigurability, parallelism, and the ability to efficiently execute the repetitive computation patterns of Transformer-based networks [11]. To improve productivity, our design leverages High-Level Synthesis (HLS), which enables hardware implementation using high-level languages such as C/C++. HLS significantly reduces development complexity and accelerates design space exploration, facilitating rapid implementation and optimization of deep learning workloads on reconfigurable hardware. Our key contributions are as follows:

- We design a comprehensive GPT-2 inference accelerator that integrates software-level model optimization with hardware-level acceleration.
- Our accelerator incorporates row-wise GEMM scheduling, memory layout transformation, HLS pragmas, URAM-aligned pack-and-unpack techniques, and kernel-level tiling to maximize computational throughput and memory efficiency on the target FPGA architecture.
- Our design achieves a perplexity reduction on the WikiText-2 dataset from 1.463 to 1.239, a 2.16× speedup over CPU-based inference, and consumes only 23% of the power compared to GPU-based execution. These outcomes highlight the effectiveness of our approach in improving both model quality and system-level efficiency for Transformer inference acceleration.

II. Background and Related Works

FPGAs have become increasingly attractive for deep learning inference due to their performance and energy

[*]This work was supported in part by National Science and Technology Council, Taiwan ROC, under contract NSTC 113-2218-E-007-020, NSTC 113-2640-E-A49-004, and NSTC 113-2640-E-007-007.

[*] These authors contributed equally to this work.

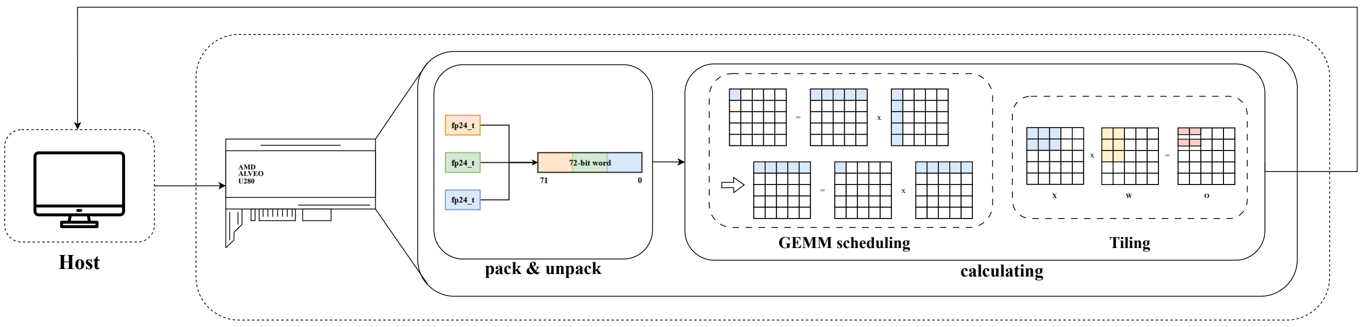

Fig. 1. Key hardware-level optimization strategies implemented in our FPGA-based GPT-2 accelerator.

efficiency. High-Level Synthesis (HLS) simplifies FPGA development and has enabled rapid deployment of various neural network models. Initial efforts in HLS-based accelerator design predominantly focused on convolutional and recurrent architectures:

- CNN accelerators: Prior works like [12, 13] demonstrate modular, system-on-chip (SoC)-based CNN (convolutional neural network) accelerators that achieved notable speedups through HLS-based design space exploration and pipeline optimization.
- RNN/LSTM accelerators: Works such as [14, 15] demonstrated that HLS can simplify the design of recurrent neural network (RNN) and long short-term memory (LSTM) accelerators, achieving reduced latency and lower development overhead.

Compared to CNNs and RNNs, Transformer-based models, particularly GPT-2, remain relatively underexplored in the HLS/FPGA design landscape. Prior work such as [16] presented an FPGA design of a scaled down GPT-2 variant, but the design was limited to a single decoder layer and a reduced hidden size ($N = 128$), constraining its applicability to realistic use cases.

In contrast, our work targets the *full GPT-2* model of 12 layers, a hidden size of $N = 768$, and multi-head attention, deployed on an AMD Alveo U280 FPGA using the Vitis HLS toolchain. To address the architectural and performance challenges posed by this full-scale deployment, we introduce several FPGA-specific optimizations, including data packing, operation tiling, kernel fusion, and pragma-guided parallelism. Additionally, to enable a fair evaluation, we develop a baseline GPU implementation of GPT-2 using a pure C++ framework, allowing direct comparison of performance and energy consumption across platforms.

III. Proposed Design Methodology

A. Model and Platform Overview

While GPT-2 supports a range of NLP tasks, we focus specifically on autoregressive text generation, fine-tuning

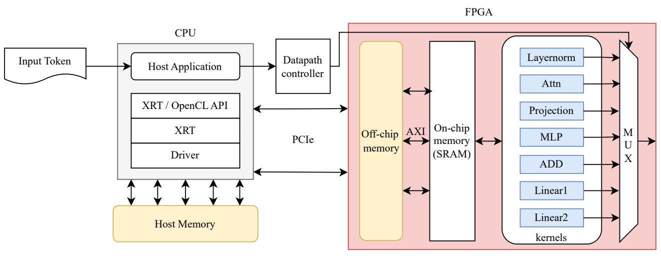

Fig. 2. Overall architecture of our design.

the model on the WikiText-2 dataset [17] for this purpose.

The entire inference is partitioned into two functional components: the host and the kernel, illustrated in Fig. 2. The host application, running on the CPU, manages the pre-trained GPT-2 weights in host memory and coordinates data transfers via PCIe interface. Based on execution flow, it loads the required weights into on-chip SRAM buffers on the FPGA. As highlighted in Fig. 3, orange-colored modules denote operations executed on the kernel side, whereas the remaining components are handled by the host. To further elaborate, Fig. 4 presents the internal structure of the Transformer layer, which corresponds to a key component within the overall architecture in Fig. 3.

B. Host-Side Processing Pipeline

B.1. Text Tokenization

Tokenization converts raw text into a sequence of subword tokens. GPT-2 utilizes Byte Pair Encoding (BPE), a subword-level tokenization scheme that offers a balance between vocabulary compactness and representational granularity. BPE enables efficient handling of rare words and morphological variants by encoding at the subword and character levels.

B.2. Token and Positional Embeddings

Each input token x_i is represented by the sum of a learned token embedding and a positional embedding [16].

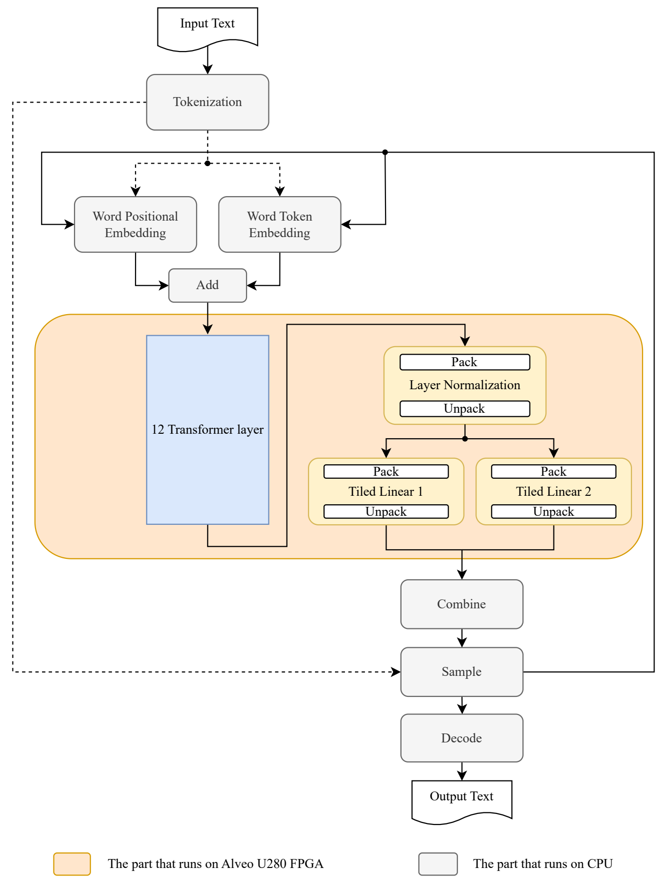

Fig. 3. Overall model structure of GPT-2 small, with dotted paths executed only for the prefill stage.

The token embedding matrix $E_w \in \mathbb{R}^{V \times d}$ maps each token ID to a d-dimensional vector, while the positional embedding matrix $E_p \in \mathbb{R}^{T \times d}$ encodes its sequence position. The combined embedding for token x_i at position i is:

$$e_i = e_w(x_i) + e_p(i), \tag{1}$$

where $e_w(x_i) = E_w[\mathrm{ID}(x_i)]$ and $e_p(i) = E_p[i]$.

B.3. *Token Sampling Strategy*

To generate the next token, we adopt top-k sampling with $k = 40$ from the full vocabulary of 50,257 tokens. The logits for the top-k tokens are normalized using the softmax function:

$$P(y_i) = \frac{\exp(s_i)}{\sum_{j \in \mathcal{T}} \exp(s_j)}, \tag{2}$$

where $\mathcal{T}$ denotes the top-k token set and s_i is the score for token i. A new token y is sampled from this probability distribution:

$$y \sim P(y_i). \tag{3}$$

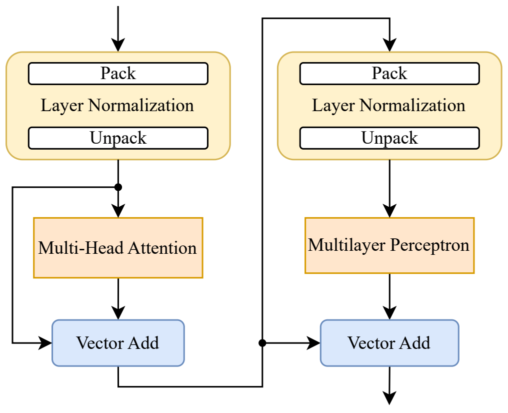

Fig. 4. The model architecture of Transformer layer.

B.4. *Text Decoding and Output Reconstruction*

Upon generation completion, the output tokens are passed to a decoding module that performs byte-to-Unicode mapping to reconstruct readable text. The token sequence, comprising both input and generated tokens, is then concatenated to yield the final output string.

C. *FPGA Kernel Design and Optimization*

C.1. *GEMM Architecture and Memory Optimization*

General Matrix Multiplication (GEMM), which computes $XW + b$ for input matrix X, weight matrix W, and bias b, constitutes a core operation in GPT-2 and represents a primary computational bottleneck. To accelerate inference, we restructure computation scheduling to favor linear memory access patterns on the FPGA. Specifically, we adopt a row-wise update strategy using loop interchange, which enhances memory access efficiency.

Given the limited capacity of BRAM and URAM, non-contiguous access patterns often degrade memory utilization. To address this, we flatten multi-dimensional arrays into one-dimensional representations, improving memory alignment and access consistency.

Parallelism is further exploited using HLS pipeline pragmas to achieve an initiation interval (II) of 1. Additionally, we employ `bind_storage` pragmas to map data buffers to dual-port BRAMs, enabling concurrent read/write operations. These optimizations are consistently applied across all GEMM modules to improve throughput and resource efficiency on the FPGA.

C.2. *Layer Normalization: Fixed-Point Optimization and Packing*

In GPT-2, Layer Normalization (LN) is applied to each token, with the normalized output scaled and shifted by

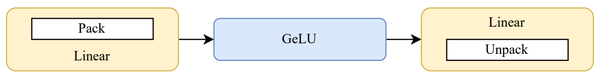

Fig. 5. The MLP architecture with additional packing and unpacking stages.

learned parameters. However, computing mean and variance introduces data dependencies that impact performance. Moreover, processing the entire token sequence simultaneously imposes high memory demands. The following sections describe strategies to optimize memory utilization and computational efficiency.

Resource Utilization Efficiency To minimize memory usage, we adopt a token-wise processing strategy. While BRAM on the Alveo U280 FPGA offers low-latency access, its limited capacity necessitates the use of URAM, where each block of which provides 4k×72 bits. To maximize memory utilization, we convert floating-point values to fixed-point format and store them as unsigned integers. Specifically, three `fp24_t` values are cast to `ap_uint<24>` and packed into a single `ap_uint<72>` word, which is unpacked during computation. This efficient representation aligns with the URAM word size, achieving a 77% reduction in URAM usage.

Reducing Data Dependency Overhead To alleviate performance bottlenecks from data dependencies, we employ loop unrolling at each computational stage to maximize parallelism. Using the `#pragma HLS unroll` directive, loops are parallelized effectively in the HLS toolchain. Combined with the aforementioned data packing technique, this enables concurrent retrieval and processing of multiple values from a single memory location. These optimizations collectively yield a 6× speedup over the baseline implementation.

C.3. *MLP Acceleration and Memory Reuse Techniques*

As illustrated in Fig. 5, the MLP (multilayer perception) module comprises two linear layers and a GELU activation. Given the constrained URAM resources, we implement two key strategies to reduce memory usage: (1) memory reuse, which provides a 40% savings, and (2) data packing, which contributes an additional 50% reduction, resulting in a cumulative 70% decrease.

To mitigate dataflow disruption from data-weight sharing, we apply pipelining and loop unrolling, enhancing parallelism but with an initial drop in clock frequency. This reduction is subsequently mitigated by incorporating tiling and kernel-level parallelism, restoring the original clock rate. While marginally slower than the initial FPGA design, the optimized MLP kernel achieves at least a 7× speedup over CPU-based inference.

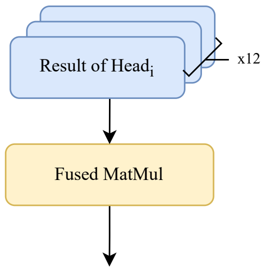

Fig. 6. Concatenation of the results from 12 attention heads followed by fused linear projection.

C.4. *Multi-Head Attention Scheduling and Fusion Strategies*

GPT-2 employs the Multi-Head Attention (MHA) mechanism consisting of 12 attention heads, typically executed in parallel. However, the limited on-chip memory of the Alveo U280 FPGA constrains the concurrent storage of all required weights. To address this, we exploit the independence among attention heads by processing them sequentially and aggregating the outputs into a single matrix before the final fused matrix multiplication, as illustrated in Fig. 6.

We also fuse consecutive operations, such as reshaping and transposing, which merely reorganize data layout without altering values. Instead of explicitly performing these operations, we refine index mappings in subsequent computations, as illustrated in Fig. 7. This approach maintains correctness while reducing memory overhead and preserving linear memory access patterns. As a result, we enhance MHA efficiency by avoiding unnecessary data reordering and optimizing computation scheduling.

Further performance gains are achieved by unrolling inner loops using the `#pragma HLS unroll` directive, enabling fine-grained parallelism. This pragma duplicates loop operations, allowing multiple iterations to execute concurrently on the hardware.

KV-cache Utilization During the decoding stage of generative language models, each new token depends on previously generated ones, necessitating access to all prior Key and Value (KV) pairs. To avoid redundant computation, we cache these pairs in external DDR memory. Although the KV-cache grows with the number of generated tokens, up to 72MB, we mitigate this by updating the cache incrementally after computing each attention head for efficient memory usage.

IV. Experimental Results and Evaluation

A. *Language Modeling Accuracy (Perplexity)*

Our fine-tuned model achieves a significant enhancement in language modeling accuracy on the WikiText-2

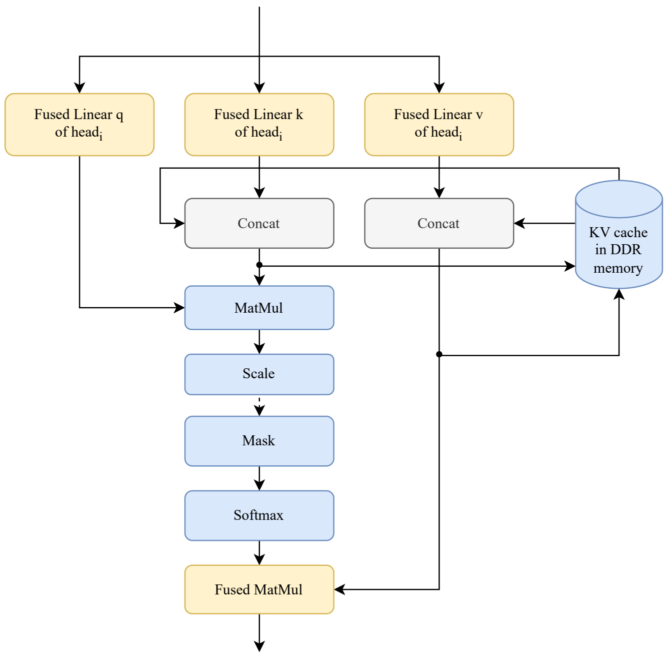
Fused Linear q of head$_i$
Fused Linear k of head$_i$
Fused Linear v of head$_i$
Concat
Concat
KV cache in DDR memory
MatMul
Scale
Mask
Softmax
Fused MatMul

Fig. 7. attention mechanism for a single head. Fused blocks indicate that operations such as reshape and transpose have been integrated into the subsequent matrix multiplication.

TABLE I
Language Modeling Perplexity on WikiText-2

Model	Perplexity (PPL) ↓
Original GPT-2	1.463
Our Fine-tuned GPT-2	1.239

test set, reducing perplexity from 1.463 to 1.239. This indicates increased confidence in token predictions and superior performance compared to the original GPT-2. Detailed results are provided in Table I.

B. *Inference Latency and Speedup*

The proposed FPGA-based accelerator achieves a 2.16× speedup over the CPU while consuming only 50% of the GPU's power, highlighting its efficiency for hardware-accelerated inference. Inference time comparisons are summarized in Table II.

TABLE II
Inference Latency Comparison across Different Devices

Device	Time (s)
CPU (Intel Core i9-14900K)	659.51 (baseline)
GPU (NVIDIA RTX 3060 Laptop)	16.46 (40×)
FPGA (Alveo U280)	305.42 (2.16×)

TABLE III
Power Consumption Comparison across Different Devices

Device	Power (W)
CPU (Intel Core i9-14900K)	53.02 (100%)
GPU (NVIDIA RTX 3060 Laptop)	24.54 (46%)
FPGA (Alveo U280)	12.32 (23%)

TABLE IV
Normalized Performance per DSP for attention and MLP Layers

Component	Ops Type	Ops Count	DSPs Used	Perf (Ops/s)	Perf per DSP
Attention (Prior Work)	IOPs	4.19×10^6	960	55.35	**0.058**
MLP (Prior Work)	IOPs	1.05×10^6	128	27.58	**0.215**
Attention (Proposed)	FLOPs	2.49×10^8	78	1,725.4	**22.12**
MLP (Proposed)	FLOPs	5.99×10^8	944	14,805	**15.68**

C. *Power Efficiency Analysis*

We present a fully self-contained GPT-2 inference accelerator developed in C++ and synthesized for the Alveo U280 FPGA using HLS. This end-to-end implementation integrates all model components and optimizations.

The accelerator incorporates sophisticated hardware optimizations, including data packing, tiling, operation fusion, and pragma-guided parallelism, to enhance inference performance and reduce latency. We evaluate its power and runtime efficiency on the FPGA implementation against CPU and GPU baselines. Comparative results are summarized in Table III.

D. *Layer-Level Performance Analysis*

To compare our design with prior work [16], we evaluate data complexity, latency, and throughput normalized by DSP utilization for both the attention and MLP components. While the baseline targets a reduced GPT-2 configuration with a single decoder layer, single-head attention, and hidden size $N = 128$, our implementation supports 12 decoder layers, multi-head attention, and a hidden size of $N = 768$. As a result, our design handles a substantially larger computational workload while delivering superior efficiency.

As summarized in Table IV, the throughput is normalized by the number of DSP blocks to account for hardware resource discrepancies. Our attention layer processes 2.49×10^8 FLOPs in 144,600 ms using only 78 DSPs, resulting in 22.12 FLOPs/s/DSP, approximately 381× higher than the 0.058 IOPs/s/DSP achieved by the baseline. Similarly, the MLP layer executes 5.99×10^8 FLOPs in 40,477 ms using 944 DSPs, achieving 15.68 FLOPs/s/DSP, a 73× improvement over the prior design's 0.215 IOPs/s/DSP. These results highlight not only enhanced raw throughput but also significantly improved resource efficiency.

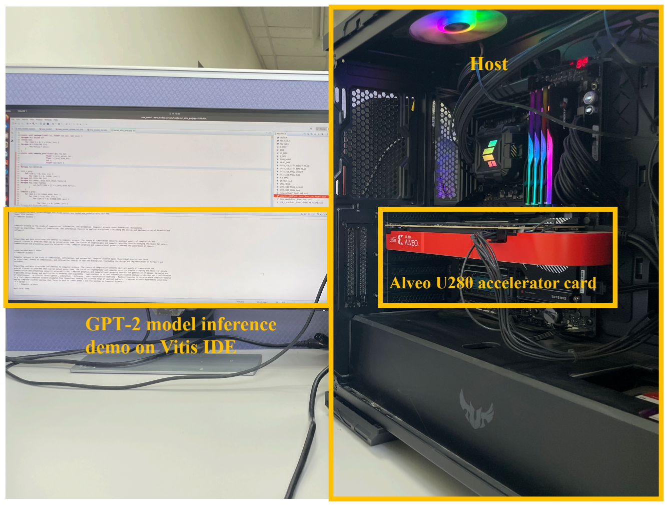
Host
Alveo U280 accelerator card
GPT-2 model inference demo on Vitis IDE

Fig. 8. System-level demonstration of FPGA-accelerated GPT-2 inference, showing interaction between text input, host, and FPGA.

E. System Demonstration and Real-Time Inference

To demonstrate the practical viability of our design, we developed a comprehensive end-to-end demo system. As illustrated in Fig. 8, the system integrates the FPGA-accelerated GPT-2 model with a host-side application capable of processing text input and displaying inference results in real time.

V. Conclusion

This paper presents an end-to-end GPT-2 inference accelerator using HLS on the AMD Alveo U280 FPGA. Our design addresses the dual challenges of high energy consumption associated with GPUs and limited computational throughput of CPUs. By leveraging the parallelism and configurability of FPGAs, our solution demonstrates the viability of efficient hardware acceleration for large language model inference. Experimental results validate the effectiveness of our approach. The proposed design achieves a 2.17× speedup over CPU inference and consumes only 50% of the power required by GPU-based execution, offering a favorable trade-off between performance and energy efficiency. Furthermore, the model improves inference quality, reducing perplexity on the WikiText-2 dataset from 1.463 to 1.239. These results highlight the effectiveness of our approach in delivering scalable, energy-efficient solutions for Transformer-based models.

References

[1] K. R. Chowdhary, *Natural Language Processing.* New Delhi: Springer India, 2020, pp. 603–649.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[3] D. Luitse and W. Denkena, "The great Transformer: Examining the role of large language models in the political economy of AI," *Big Data & Society*, vol. 8, no. 2, p. 20539517211047734, 2021.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "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)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186.

[5] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018.

[6] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and Efficient Foundation Language Models," 2023.

[7] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[8] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "DFX: A Low-latency Multi-FPGA Appliance for Accelerating Transformer-based Text Generation," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, 2022, pp. 1–17.

[9] E. BUBER and B. DIRI, "Performance Analysis and CPU vs GPU Comparison for Deep Learning," in *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, 2018, pp. 1–6.

[10] D. Zhao, S. Samsi, J. McDonald, B. Li, D. Bestor, M. Jones, D. Tiwari, and V. Gadepally, "Sustainable Supercomputing for AI: GPU Power Capping at HPC Scale," in *Proceedings of the 2023 ACM Symposium on Cloud Computing*, ser. SoCC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 588–596.

[11] G. Tzanos, C. Kachris, and D. Soudris, "Hardware Acceleration of Transformer Networks using FPGAs," in *2022 Panhellenic Conference on Electronics & Telecommunications (PACET)*, 2022, pp. 1–5.

[12] T. Yang, C. Li, Y. Zhang, K. Luo, Z. Dong, and J. Li, "A Convolutional Neural Network Accelerator with High-level Synthesis," in *2024 IEEE 4th International Conference on Electronic Technology, Communication and Information (ICETCI)*, 2024, pp. 38–43.

[13] M. Sarg, A. H. Khalil, and H. Mostafa, "Efficient HLS Implementation for Convolutional Neural Networks Accelerator on an SoC," in *2021 International Conference on Microelectronics (ICM)*, 2021, pp. 1–4.

[14] Z. Li and S. Fu, "Accelerating RNN on FPGA with Efficient Conversion of High-Level Designs to RTL," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 3355–3364.

[15] M. L. González, R. Lozada, J. Ruiz, E. Skibinsky-Gitlin, M. García-Vico, J. Sedano, and J. R. Villar, "Exploring the implementation of LSTM inference on FPGA," in *2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, 2023, pp. 1–5.

[16] A. Yemme and S. S. Garani, "A Scalable GPT-2 Inference Hardware Architecture on FPGA," in *2023 International Joint Conference on Neural Networks (IJCNN)*, 2023, pp. 1–8.

[17] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer Sentinel Mixture Models," in *International Conference on Learning Representations*, 2017.