ELLME: An FPGA-based LLM Accelerator for End-to-end Qwen-2.5 Inference

Yujie Ma, Linqi Li, Yuman Zeng, Zhirui Huang, Chixiao Chen, Qi Liu, and Haozhe Zhu
State Key Lab of Integrated Chips and Systems, Fudan University, Shanghai, China
Email: zhuhz@fudan.edu.cn

***Abstract*—The rapid advancement in Large Language Models (LLM) research has significantly enhanced natural language processing capabilities across diverse applications. However, the massive model parameters and substantial computational demands impose strong requirements for high-throughput hardware to enable efficient edge-side inference. Due to the severe memory imbalance of prefilling and decoding phases, and the trade-off between accuracy and resource cost, it is thirsty for architecture for optimization on memory and computation. The significant memory imbalance between the prefilling and decoding phases, combined with the accuracy-resource trade-off, creates critical demand for architectural optimizations in both memory and computation. We propose an FPGA-based accelerator, ELLME, employing a novel architecture with memory-friendly dataflow scheduling, W4A8/W8A8 low-bit-width matrix computation, and approximate nonlinear computation to address these challenges. We implement the Qwen-2.5 model on our acceleration system and develop a quantization algorithm to restore inference accuracy loss, achieving 99% accuracy on the WNLI dataset. ELLME achieves 307 GOPS at 300 MHz operation frequency, delivering 280 tokens/s in prefilling and 1.9 tokens/s in decoding phases under real task workloads.**

***Index Terms*—FPGA, large language model, Qwen, inference, network quantization.**

I. Introduction

Large Language Models (LLMs) have demonstrated exceptional performance across diverse natural language processing tasks, including text comprehension, generation, sentiment analysis, machine translation, and interactive question answering. [1] [2] [3] The exponential growth in computational demands and model parameters creates urgent demand for edge-device deployment, driving the need for innovative high-throughput hardware architectures. Research into FPGA-based architectures for LLM acceleration has seen growing demand, with numerous studies demonstrating significant advancements in the field.

Many recent distinguished works focus on FPGA-based LLM acceleration. Several studies leverage sparsity to address resource constraints while improving throughput. For instance, EdgeLLM [4] proposed a mixed-precision systolic architecture that efficiently processes FP16×FP16 attention and FP16×INT4 projection layers, which supports weight sparsity. FlightLLM [5] proposed a configurable sparse DSP chain for multiple sparsity patterns, coupled with an on-chip encoding scheme to boost memory access bandwidth. STA [6] proposes a flexible hardware architecture supporting N:M semi-structured sparsity, capable of both sparse-dense and dense-dense multiplication. Similar to our work, EFA-TRANS [7] optimizes nonlinear computation circuits and adopts fine-grained scheduling for complete Transformer inference. VIA [8] implements reusable processing engines to support diverse internal stream dataflow, such as overlapping and pipeline. Beta [9] introduces a configurable QMM engine adaptable to various activation precisions in binary Transformers.

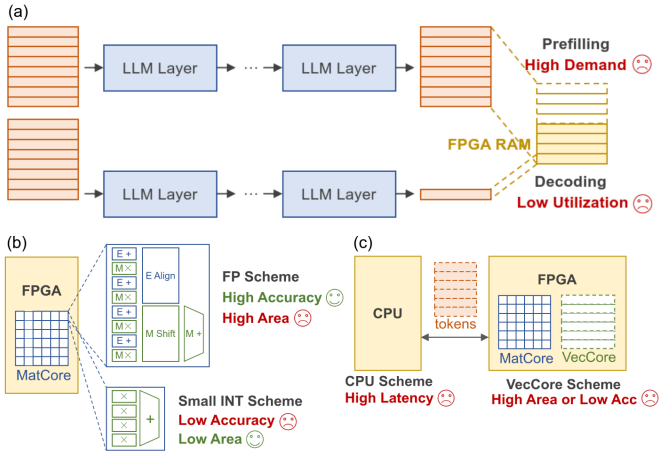

Fig. 1. The challenges of hardware development for LLMs. (a) Unbalanced RAM utilization between prefilling and decoding. (b) Trade-off between low-bit-width and full-precision computation (c) On-chip non-linear operation implementations.

Hardware development for LLMs is still facing significant challenges. The prefilling process typically requires on-chip memory resources that far exceed available BRAM capacity, while the per-token decoding phase leaves most memory resources underutilized, as Fig. 1(a). Furthermore, full-precision (e.g., FP32/FP16) computation incurs prohibitive area overhead, whereas low-bit-width (e.g., INT4/INT8) degrades inference accuracy, as Fig. 1(b). Additionally, traditional implementations offload non-linear operations to the CPU, incurring performance latency from both data transfer overhead and low-parallelism execution latency, as Fig. 1(c).

To address these challenges, we propose **ELLME**, an FPGA-based LLM accelerator optimized for resource-constrained edge deployment while achieving high inference throughput. We implement the ELLME acceleration system on an AMD Xilinx KV260 evaluation board. The main contributions of this paper include:

1) **Memory-friendly dataflow of prefilling and decoding**

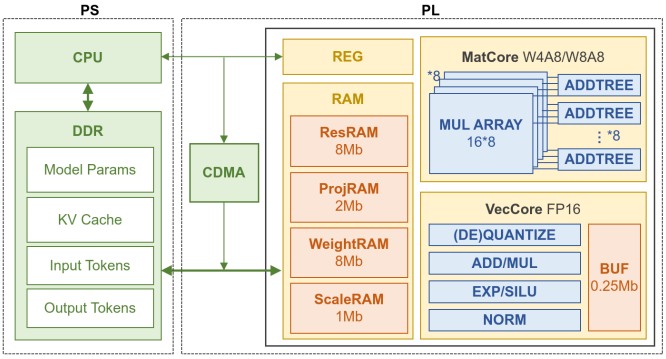

Fig. 2. LLM acceleration system overall architecture. ELLME is implemented on the PL side, which comprises three core components: on-chip RAM modules, a dedicated matrix computation unit and a vector processing.

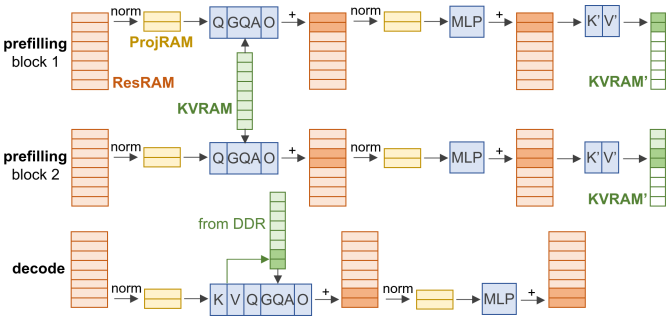

Fig. 3. Memory-Friendly dataflow in prefilling and decoding phases. In the prefilling phase, tokens are partitioned to blocks going through fused layers. In the decoding phase, most recent tokens proceed full KV recomposition.

phases. ELLME addresses the high memory demand during prefilling through layer fusion while simultaneously improving resource utilization in decoding and reducing KV-Cache transfer latency.

2) **Precision-reconfigurable DSP systolic array for W4A8 projection and W8A8 attention.** We developed a matrix computation core supporting both W4A8 and W8A8 precision modes, complemented by an algorithmic framework to enable low-bit-width and restore accuracy loss.
3) **Low-area approximate non-linear computation units.** We implement approximate computation circuits using single-stage multiply-add operations to process exponent, SiLU, and normalization calculations, achieving high accuracy while reducing resource costs.
4) To validate the proposed accelerator, we deploy a Qwen-2.5 model [10] on the ELLME system. A quantization algorithm is adopted to restore inference accuracy loss, achieving 99% accuracy on the WNLI dataset [11]. ELLME achieves 307 GOPS at 300 MHz operation frequency, delivering 280 tokens/s in prefilling and 1.9 tokens/s in decoding phases under real task workloads.

In the content below, we focus on hardware design details, presenting our innovations in Section II. Then, we present evaluation experiments in Section III, followed by conclusions in Section IV.

II. ELLME Hardware Design

A. Overall Architecture

The overall architecture of the proposed ELLME accelerator is depicted in Fig. 2. The system pre-stores model parameters and input tokens in DDR memory, and data is transferred between the accelerator and DDR via a central DMA (CDMA) controller. As shown in Fig. 2, ELLME comprises three core components: (1) on-chip RAM modules for data buffering, (2) a dedicated matrix computation unit for dense operations, and (3) a vector processing unit for element-wise operations.

The on-chip RAM modules consist of four specialized RAM modules: ResRAM, ProjRAM, WeightRAM, and ScaleRAM. ResRAM is used to store full-precision tokens for attention-input and MLP-input residual addition operations post. ProjRAM buffers a subset of quantized tokens involved in projection computations, and Q, S during GQA computation. WeightRAM stores both the quantized weights of projection layers and runtime-generated KV-Cache. ScaleRAM maintains quantization scaling factors along with other model hyperparameters.

The matrix computation process is based on the outer product of matrix multiplication, integrating eight parallel processing arrays, each sized 16×8. Each array performs per-cycle outer products between a 16-element vector and an 8-element vector, with inter-cycle accumulation. These arrays operate on different parts along the hidden dimension, generating partial results, which are subsequently aggregated through an adder tree. To eliminate routing congestion and enhance programmability, each multiplication array adopts a systolic architecture to minimize fan-out of weight and activation interconnects. ”The matrix multiplication results are then pipelined as per-cycle vectors out of the arrays and streamed to the vector computation core.

The vector computation core processes all operations in the LLM layers, excluding matrix multiplication, integrating quantization/dequantization units, element-wise arithmetic units (addition and multiplication), exponential units, and normalization units, along with a local buffer for temporary operand storage. The quantization/dequantization unit performs static per-token quantization and dequantization operations on activations. The arithmetic unit executes residual additions, gated multiplications, and RoPE. The buffer synchronizes different parts of the token vectors for normalization operations, and caches gate-projection results in MLP layers. To minimize on-chip resource utilization, approximate computing methods are involved in both the exponential and normalization units.

B. Memory-friendly Dataflow Scheme

A specialized dataflow is implemented to reduce RAM requirements during prefilling and improve RAM utilization in decoding, meanwhile optimizing KV-Cache transfer latency, as Fig. 3.

During the prefilling phase, the primary challenge is mainly caused by the limited capacity of ProjRAM, which cannot accommodate all tokens involved in the computation. We fuse all non-K, V-projection operations in the current layer with K, V projections in the subsequent layer. The computation follows a block-wise execution pattern where each token block completes all processing stages—from current-layer Q projection to next-layer K, V projections—before proceeding to the next block. This design requires explicit data synchronization only for the KV-Cache before attention, for which a spare memory region in WeightRAM is reserved. When on-chip memory cannot accommodate all weights, as a common scenario in FPGA implementations, this strategy significantly increases weight transferring. To prevent loading weight from becoming the bottleneck, the token block size must exceed a minimum threshold determined by the memory-compute ratio.

During the decoding phase, conventional single-token decoding exhibits severe latency imbalance, where the computational delay is dominated by KV cache access overhead, meanwhile, single-token processing leaves memory and compute resources with poor utilization. Consequently, we implement a selective KV-Cache dataflow strategy: The most recent tokens proceed full KV recomputation, while the remaining KV-Cache is stored off-chip and transferred as needed. This strategy achieves a 10% reduction in KV-Cache transfer latency while balancing memory-computation latencies and enhancing on-chip resource utilization.

C. Dynamic Precision Computing Units

To balance on-chip resource utilization and computational accuracy, we employ low-bit-width and approximate computing methods in our circuit design. Moreover, we implement nearly all computations on DSP units to ensure numerical precision, maintain timing closure, and minimize LUT consumption.

In the multiplication array, each DSP unit performs two parallel W4A8 or W8A8 multiply-accumulate operations per cycle, requiring both operations to share the same activation input. Sign-bit compensation is applied to the LSB of weight elements to handle negative values. After computation, the results are then staged through the final pipeline registers in each DSP unit.

The exponential computation unit utilizes a direct linear approximation of the floating-point literal value. The Floating-Point representation inherently simplifies power calculations through its power-of-two exponent scaling. There has been solid analysis demonstrating that the bit-level value of e^x can be approximately represented as $(k_{\text{int16}} x_{\text{fp16}})_{\text{int16}} + b_{\text{int16}}$, where $k = 0\text{x}5c5$ and $b = 0\text{x}3bc5$. To further improve accuracy, we incorporate an LUT that dynamically selects optimal k and b pairs based on exponent ranges, reducing approximation error to less than 1%—well within Softmax computation requirements. This single-stage multiply-add approach also extends to SiLU and reciprocal operations, delivering significant area savings while maintaining numerical fidelity.

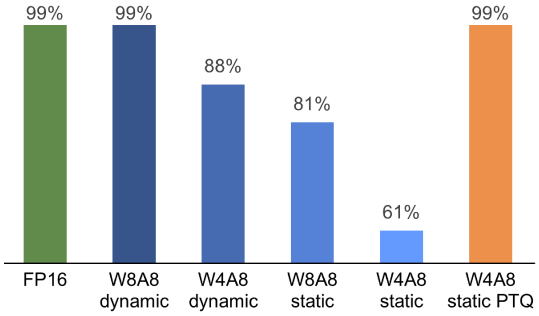

Fig. 4. Accuracy evaluation results of various quantization methods. The W4A8 static PTQ method adopted achieves 99% accuracy.

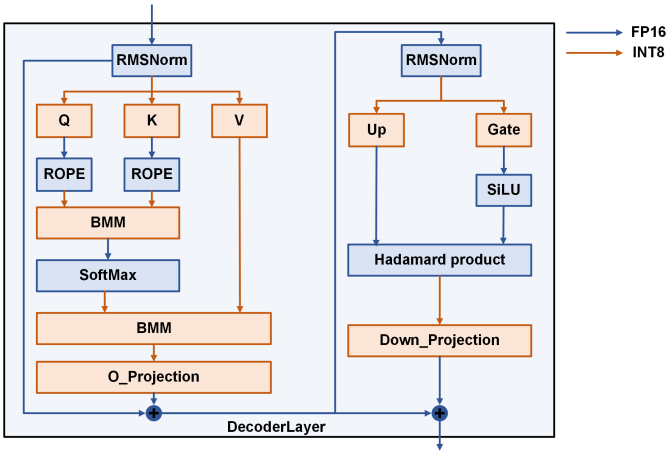

Fig. 5. Precision mapping for a Transformer block. All compute-intensive operators like linear layers and batched matmul (BMMs) use INT8 arithmetic.

III. Evaluation

A. Model Finetuning and Quantization

Due to competition requirements set by the AICAS organizing committee, our validation and testing dataset selection was restricted to the WNLI. For model optimization, we performed full-parameter fine-tuning using SmoothQuant [12] with extended support for W4A4/W8A8 precision, enabling flexible quantization schemes.

The conventional Post-Training Quantization (PTQ) [13] method results in significant model accuracy degradation, particularly under low-bit quantization settings. This effect becomes more pronounced as the bit width decreases. To mitigate this issue, we employ Quantization-Aware Training (QAT) [14], which inserts fake quantization nodes into the network during training. These nodes simulate quantization noise, ensuring that weight updates account for the model's eventual quantization. Since our approach uses static quantization, both the weights and activation scales are optimized jointly during training.

During training, we observed that the backward gradients in fake quantization nodes struggled to converge effectively due to the truncation effects induced by quantization. To address this, we adopted the straight-through estimator (STE) method, which enables proper gradient flow during backpropagation.

TABLE I
COMPARISON WITH CPU BASELINE AND SOTA FPGA-BASED LLM ACCELERATORS

	CPU-Baseline	STA [6]	BETA [9]	EFA-Trans [7]	ViA [8]	*Ours*
Platform	Cortex-A53(4 Cores)	ZC702	ZCU102	ZCU102	Alveo U50	KV260
Technology	16nm	16nm	16nm	16nm	16nm	16nm
Frequency (Hz)	2G	150M	190M	N/A	300M	300M
Test Network	Qwen2.5-0.5B	N:M Sparse	BiT	Sparse	Swin	Qwen2.5-0.5B
W/A Precision	W16A16	W16A16	W1A1	W8A8	W16A16	W4A8
Decoder Support	✓	×	×	×	×	✓
Throughput (GOPS)	0.64	109.45	1240.98	279.80	309.60	307.2
Prefill Performance(Token/s)	1.67	N/A	N/A	N/A	N/A	280
Decode Performance(Token/s)	2.78	N/A	N/A	N/A	N/A	1.9
Power (W)	N/A	2.71	7.18	5.48	38.99	3.58
Energy Efficiency (GOPS/W)	N/A	40.39	172.41	51.06	7.94	85.71

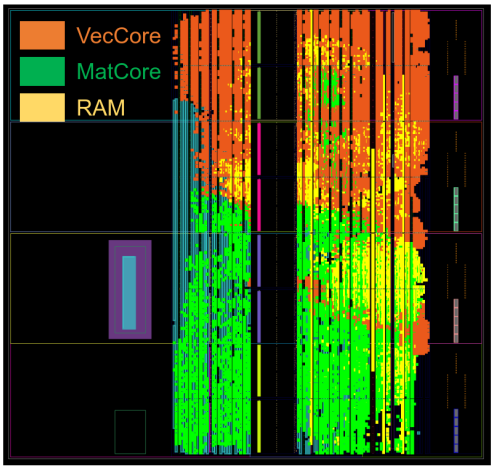

Fig. 6. Overall layout of ELLME after implementation. Blocks in yellow are BRAM and URAM utilized, and units of MatCore are in green, VecCore in orange.

We systematically evaluated various quantization schemes for both activation values and weights. The comparative results of these different quantization approaches are presented in Fig. 4, providing valuable insights into the trade-offs between model accuracy and computational efficiency. After careful consideration of the accuracy results and resource constraints on the FPGA platform, we ultimately selected the W4A8 static quantization method as the most suitable solution.

As shown in Fig. 5, in this implementation, all compute-intensive operations, including linear layers and batched matrix multiplications (BMMs), are executed using INT8 arithmetic. For static weights in linear layers, we have implemented a more aggressive INT4 quantization scheme, achieving an optimal balance between computational efficiency and model accuracy within the FPGA's resource limitations. Other non-linear operations are remaining FP16 in reference.

B. FPGA Deployment

The layout of the acceleration system implemented on board is shown as Fig. 6. Utilization of on-chip programmable resources are reported as shown in Table: II

TABLE II
UTILIZATION OF ON-CHIP PROGRAMMABLE RESOURCES

Resource	Utilization	Utilization Rate
LUT	55519	47.40%
FF	100330	42.83%
DSP	704	56.41%
BRAM	129	89.58%
URAM	64	100%

We employ the W4A8 QAT method to balance between hardware deployment strategies and algorithm accuracy. We define a set of instruction sets for ELLME, which includes DMA transfer instructions and computational instructions. During each computation, the required weights are transferred from the DDR to specified locations within the custom-designed IP via a series of DMA instructions. Additionally, we formulate computational task instructions containing the addresses of weights and hyperparameters to manage the computation process during inference. At the same time, DMA instructions can also transfer data from the specified addresses within ELLME to the DDR for extracting the final computation results and storage of the large KV cache.

In hardware deployment, we first use Python to load the binary file containing the model weights into the FPGA's DDR and generate an index dictionary for each weight. Then, based on the model's computation sequence, we generate a C-language file containing DMA and computational instructions to manage data transfer and computation execution for each layer. Next, we use Python to process the model's embedding layers and retrieve the token corresponding to the embedding data from the vocabulary using the index. The executable file, compiled from the C-language file, coordinates the computation and DMA register writes of the IP according to the predefined sequence. We use Python to process the final results, thereby enabling the output determination for the specific task.

C. Comparison with SOTA and Baseline

Due to the hardware-software co-design of ELLME, we deploy the quantized Qwen2.5-0.5B model on KV260. At a frequency of 300 MHz, ELLME achieves 307.2 GOPS and

85.71 GOPS/W energy efficiency for W4A8 computation. Compared to SOTA [6]–[8], ELLME achieves higher energy efficiency at similar computing levels. Beta [9], operating at 190 MHz and using a W1A1 model, reaches 1240.98 GOPS, but its computational power for W4A8 is theoretically lower than that of ELLME. Overall, ELLME outperforms other SOTA LLM accelerators in terms of both computational power and energy efficiency.

At the same time, we use the 4-core Cortex-A53 CPU of the FPGA's PS part operating at 2 GHz to deploy the W16A16-quantized Qwen2.5-0.5B as the baseline, comparing throughput in the prefilling and decoding phases. While executing the prefilling phase, we strategically schedule the transfer of parameters using DMA and computing operations on the FPGA, ensuring that the transfer time can almost overlap the computation time. As a result, the model runs in a computation-dominated manner. During prefilling, ELLME achieves a throughput of 280 tokens/s, which represents a 167.7x improvement over the baseline.

Since the computing workload of decoding is relatively small, the model runs in a memory-dominated manner. The computation time cannot overlap the time required for DMA data transfer, and the execution efficiency is mainly influenced by the DMA transfer efficiency. The CPU baseline, operating at a much higher frequency of 2 GHz, achieves a throughput of 2.78 tokens/s, which is similar to our result operating at 300 MHz. If the operating frequency of ELLME is scaled, it would also achieve significantly higher throughput during the decoding phase.

IV. Conclusion

In this paper, we purposed an FPGA-based accelerator, ELLME, which features on memory-friendly dataflow scheduling scheme, a W4A8/W8A8 low-bit-width matrix computation systolic array, and approximate nonlinear computation circuits. Qwen-2.5 model is deployed on our acceleration system for further evaluation and ELLME achieves 307 GOPS at 300 MHz operation frequency, performing 280 tokens/s in prefilling and 1.9 tokens/s in decoding phases under real task workloads. ELLME provides an efficient solution for memory unbalance in prefilling and decoding phases, and trade-off between accuracy and resource utilization of computation units.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[3] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[4] D. Xu, W. Yin, H. Zhang, X. Jin, Y. Zhang, S. Wei, M. Xu, and X. Liu, "Edgellm: Fast on-device llm inference with speculative decoding," *IEEE Transactions on Mobile Computing*, vol. 24, no. 4, pp. 3256–3273, 2025.

[5] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang, Y. Dai, J. Li, Z. Wang, R. Zhang, K. Wen, X. Ning, and Y. Wang, "Flightllm: Efficient large language model inference with a complete mapping flow on fpgas," 2024. [Online]. Available: https://arxiv.org/abs/2401.03868

[6] C. Fang, A. Zhou, and Z. Wang, "An algorithm–hardware co-optimized framework for accelerating n: M sparse transformers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 11, pp. 1573–1586, 2022.

[7] X. Yang and T. Su, "Efa-trans: An efficient and flexible acceleration architecture for transformers," *Electronics*, vol. 11, no. 21, p. 3550, 2022.

[8] T. Wang, L. Gong, C. Wang, Y. Yang, Y. Gao, X. Zhou, and H. Chen, "Via: A novel vision-transformer accelerator based on fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4088–4099, 2022.

[9] Y. Ji, C. Fang, and Z. Wang, "Beta: Binarized energy-efficient transformer accelerator at the edge," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2024, pp. 1–5.

[10] Qwen, :, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu, "Qwen2.5 technical report," 2025. [Online]. Available: https://arxiv.org/abs/2412.15115

[11] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1804.07461

[12] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," 2024. [Online]. Available: https://arxiv.org/abs/2211.10438

[13] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," 2023. [Online]. Available: https://arxiv.org/abs/2210.17323

[14] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, and V. Chandra, "Llm-qat: Data-free quantization aware training for large language models," 2023. [Online]. Available: https://arxiv.org/abs/2305.17888