A Low-Latency and Scalable Vector Engine with Operation Fusion for Transformers

Mincheol Cha[1], Keehyuk Lee[2], Xuan Truong Nguyen[3], and Hyuk-Jae Lee[4]
Inter-University Semiconductor Research Center (ISRC)
Department of Electrical and Computer Engineering, Seoul National University
Seoul, Korea
Email: {chamj61047, lkh099, truongnx, hyukjae}@capp.snu.ac.kr

***Abstract*—Recently, transformer models have been widely deployed for AI services at data centers. However, one of the noticeable deployment challenges is the intensive usage of vector operations such as layer normalization (LayerNorm) and Softmax that generally show sub-optimal performance on general-purpose CPU and GPU due to their low arithmetic intensities and long data dependency. To address the problem, this study presents a low-latency and scalable FPGA-based engine for accelerating vector operations. Specifically, we built a dedicated circuit to effectively execute both element-wise operations and compound fused operations. More importantly, our engine can calculate input mean and variance in parallel, which significantly reduces the instruction count in computing LayerNorm and Softmax. Experimental results show that our design achieves a latency reduction of 50% and 40% for Softmax and LayerNorm, respectively, compared with the SOTA design, while only consuming an additional 20% DSPs, 27% BRAMs, 18% FFs, and 39% LUTs.**

***Index Terms*—Vector Processor Unit, VPU, Layer Normalization, Softmax, FPGA.**

I. Introduction

Transformer-based models have achieved remarkable performance and are extensively utilized across various domains such as natural language processing [11], [13], image recognition [4], and multi-camera object association [12]. The success of transformers largely stems from their adoption of a self-attention mechanism, which excels at capturing long-range dependencies while preserving contextual and positional information [8]. Alongside general matrix multiplications, essential vector operations like Layer Normalization (LayerNorm) [2] and Softmax activation function play pivotal roles in transformers.

For instance, LayerNorm aids in normalizing the distributions of intermediate layers, thereby facilitating smoother gradients, faster training, and improved generalization accuracy [15]. Softmax, in combination with dot-product attention, offers an efficient computational approach to modeling interactions between queries and keys within the self-attention mechanism. However, these vector operations, such as LayerNorm and Softmax, present challenges in computation for CPUs and GPUs due to their low arithmetic intensities and wide-ranging data dependencies.

To address these challenges, recent advancements have seen widespread adoption of techniques such as kernel fusion and recomputation. These techniques aim to mitigate massive data transfers between processing cores and memory [1], [3], [14], offering more efficient and effective solutions for model training as opposed to inference.

Custom hardware, such as Field-Programmable Gate Arrays (FPGAs), is a promising alternative for accelerating Transformers, particularly in addressing challenging vector operations [6], [9]. Unlike CPUs and GPUs, FPGAs offer highly tailored and configurable computing architectures suited for domain-specific tasks [10]. For example, FPGA-based Vector Processing Units (VPUs) optimize datapaths to minimize data dependency overhead [5], [7]. However, previous works have overlooked discussions on loop datapath optimization, accuracy, and bandwidth scalability. To address these shortcomings, this study proposes a low-latency and scalable engine for vector operations, with three main contributions.

- First, we observe an alternative method to compute LayerNorm by calculating mean and variance in parallel. Consequently, we design a dedicated datapath that reduces the loop count by 50% for LayerNorm and Softmax.
- Second, we propose an instruction set architecture to encode both element-wise operations and fusion operations.
- Lastly, we show a consistent performance gain for various bandwidths and intensive accuracy analysis for different accumulation precisions.

To demonstrate our design methodology, the proposed engine is implemented with Vitis HLS and verified on a Xilinx ZCU106 FPGA board. The experimental results show that our design achieves latency reduction by 40% with a resource utilization increase of 21%, compared with the baseline.

II. Background and Motivation

A. Vector Operations

Among the vector operations performed in self-attention, Softmax and LayerNorm have comparatively long latencies. It is because they have long-range dependency across the whole vector for input and intermediate results.

$$y(x_i) = \frac{e^{x_i - x_{\max}}}{\sum_j e^{x_j - x_{\max}}} \tag{1}$$

$$y(x_i) = \gamma_i \left(\frac{x_i - \mu}{\sigma} \right) + \beta_i \tag{2}$$

Taking LayerNorm for example, it occupies 0.1% amidst the total number of operations but takes up 9.9% of total latency for GPT-2. To optimize these vector operations, dedicated hardware should be employed.

B. *Operation Flow for Softmax and LayerNorm*

We choose the VPU architecture of DFX as our baseline [5]. Fig.1 shows the order of micro-instructions [7].

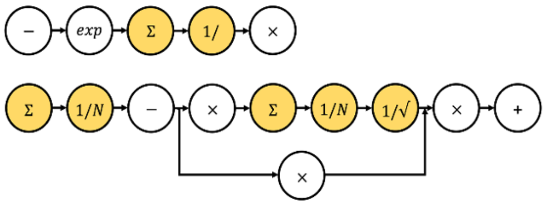


Fig. 1: Micro-instruction graph of Softmax and LayerNorm [7].

In Figure 1, yellow circles in succession denote accumulation operations, meaning that they sum up an entire input vector. White circles denote elementwise operations, which share computing patterns for each vector element. The former is usually implemented using an adder tree, while the latter is implemented using SIMD-style hardware [7]. All of the operations in Figure 1 have data dependencies with operations connected with arrows. Especially, branching arrows or arrows connecting differently colored circles cannot be pipelined or executed in parallel. For those connections, there should be an alternate calculation pattern and a separate data path for pipeline or parallel execution. Since elementwise and accumulation operations often appear in alternating order, a data path including them in series can be proposed [7].

C. *Instruction Flow for Softmax and LayerNorm*

The datapath for LayerNorm is partitioned into 6 individual loops, each loop comprising of at most one elementwise operation and one adder tree accumulation. Softmax operations require two loops, one for computing the exponent of inputs and summing them, and the other for dividing the exponent results by their sum [7]. This instruction flow for LayerNorm imposes a data dependency where σ must be calculated after μ. Consequently, at least 4 loops are required to calculate the normalized vector. The remaining operations which are rescaling and bias require 2 more loops, resulting in a total of six loops, as shown in Algorithm 1. In Algorithm 1, Σ is calculated by an adder tree, while dedicated calculators perform reciprocal square root and mult by 1/N.

III. Proposed Architecture

A. *Data Execution Order*

To reduce the data hazard appearing in the original execution order, we change the way of calculation for LayerNorm to Algorithm 2. Specifically, to remove data dependency between μ and σ, we can get σ and mean(X_i) with one instruction. This is done using two adder trees each calculating the mean of inputs and the mean of squared inputs. This removes the two-loop gap between mean and variance calculation. To further

Algorithm 1: Baseline LayerNorm Algorithm (DFX [5])

```
Data: $\boldsymbol{X}$, $\gamma$, $\boldsymbol{\beta}$
Result: LayerNorm($\boldsymbol{X}$)
1 (loop 1) $\mu = \Sigma(\boldsymbol{X}) * \frac{1}{N}$;
2 (loop 2) $\boldsymbol{Y} \leftarrow$ ADD($\boldsymbol{X}$, $-\mu$);
3 (loop 3) $\frac{1}{\sigma}$ = recipSqrt( ($\Sigma$ MUL($\boldsymbol{Y}, \boldsymbol{Y}$)) $* \frac{1}{N}$);
4 (loop 4) $\boldsymbol{Z} \leftarrow \boldsymbol{Y} * \frac{1}{\sigma}$
5 (loop 5) $\boldsymbol{Z} \leftarrow$ MUL($\boldsymbol{Z}$,$\gamma$);
6 (loop 6) LayerNorm($\boldsymbol{X}$) $\leftarrow$ ADD($\boldsymbol{Z}$, $\boldsymbol{\beta}$);
```

reduce the number of loops, a piece of dedicated hardware is appended to the first loop to apply the following operation without loop count increase. The normalized X_i is subsequently streamed directly to BRAM, which is then forwarded to undergo elementwise multiplication and addition within the subsequent two loops. This layout has the apparent advantage of reducing the number of loops executed for LayerNorm from 6 to 3. One loop is necessary to concurrently calculate μ and σ for input normalization, while two loops are needed for rescaling by β and γ.

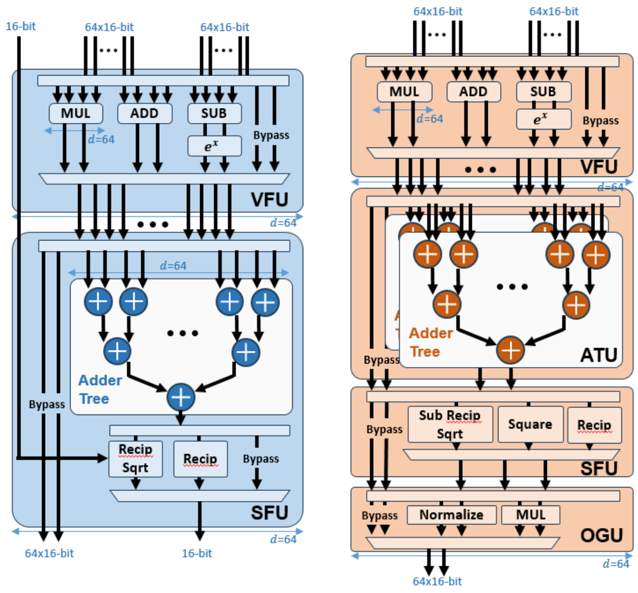
Algorithm 2: Proposed LayerNorm Algorithm

```
Data: $\boldsymbol{X}$, $\gamma$, $\boldsymbol{\beta}$
Result: LayerNorm($\boldsymbol{X}$)
1 (loop 1)
2     $\mu = \Sigma(\boldsymbol{X}) * \frac{1}{N}$ , E($\boldsymbol{X}^2$) = ($\Sigma$ MUL($\boldsymbol{X}, \boldsymbol{X}$)) $* \frac{1}{N}$;
3     $\frac{1}{\sigma}$ = recipSqrt(($E(\boldsymbol{X}^2) - \mu^2$));
4     $\boldsymbol{Y} \leftarrow$ MUL( SUB($\boldsymbol{X}$, $\mu$), $\frac{1}{\sigma}$);
5 (loop 2) $\boldsymbol{Y} \leftarrow$ MUL($\boldsymbol{Y}$,$\gamma$);
6 (loop 3) LayerNorm($\boldsymbol{X}$) $\leftarrow$ ADD($\boldsymbol{Y}$, $\boldsymbol{\beta}$);
```

B. *Micro-Architecture*

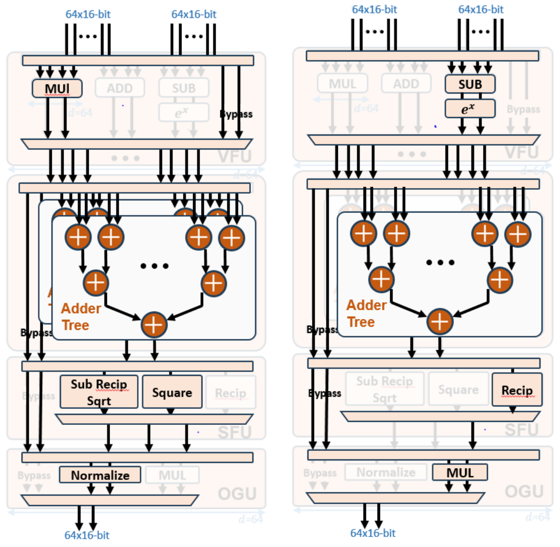
One loop consists of elementwise, accumulation, and rescaling/biasing operations, including at most one of each. In accordance, the datapath is a pipeline of the following modules. Packets, containing up to 64 FP16 values each, are the unit they operate on per cycle.

- **Vector Funciton Unit(VFU).** Two vector operations among 4, addition or multiplication, and sub-exponential or bypassing, are selected and performed on the input vector streams.
- **Adder Tree Unit(ATU).** Two adder trees calculate accumulations over input streams, and pass the outputs along with the bypassing vector stream.
- **Special Function Unit(SFU).** The output from the adder trees undergo specific operations, such as division by input size N, squaring, and subtraction followed by reciprocal square root for LayerNorm. For Softmax, the reciprocal of the exponent sum is calculated.
- **Output Generator Unit(OGU).** The vector is normalized with SFU results. For LayerNorm and Softmax, the values $(1/\sigma, \mu)$ and $(1/\text{sum}, 0)$ are selected as



(a) DFX VPU [5] (b) Proposed VPU

Fig. 2: Micro-architecture of the proposed design.



(a) LayerNorm First Loop Datapath (b) Softmax Datapath

Fig. 3: Proposed Datapath for LayerNorm and Softmax.

(weight, bias). The resulting vector can be either written back to BRAM or to the output memory address.

For each vector element processed per cycle by the VPU, an additional adder and multiplier are needed by the OGU, posing a bad tradeoff for resource utilization. An alternate option would be to remove the OGU by enabling multiple elementwise operations per loop. We moved X-μ to loop 2 before multiplying γ, and multiplication of $\frac{1}{\sigma}$ to loop 3 before adding β. This requires sequential elementwise add-mult, and mult-add respectively. Add-mult is achieved by enabling their sequential execution in the VFU. Mult-add performs multiplication in the VFU, and uses the leaf nodes of the ATU as elementwise addition datapath. This is done by splitting packets from $\frac{(X-\mu)}{\sigma} \cdot \gamma$ and β into subpackets with half bandwidth. Next, subpackets sharing the same index range are concatenated and put into each adder tree, passing through one layer of adders.

C. Latency

For LayerNorm, the baseline model passes the VFU 6 times, and SFU (including adder tree) 2 times. On the other hand, our VPU passes the VFU 3 times, the ATU, SFU, and OGU once each. Also, the latencies of VFU, ATU, and OGU are inversely proportional to the bandwidth of vector elements per cycle. Setting the length of the vector as d_l, latency as L, and bandwidth as BW, the latency of the baseline architecture and ours are calculated as below.

$$L_{baseline} = 6\frac{L_{VFU}d_l}{BW} + 2\frac{L_{AdderTree}d_l}{BW} + 2L_{SFU} \tag{3}$$

$$L_{proposed} = 3\frac{L_{VFU}d_l}{BW} + \frac{(L_{ATU} + L_{OGU})d_l}{BW} + L_{SFU} \tag{4}$$

Expecting $L_{\text{AdderTree}}$, L_{VFU}, L_{ATU}, and L_{OGU} to have similar latency and d_l to be large enough, the expected speedup of the new design is x1.6.

D. High-level Instruction Controller

The controller indexes wide-bit instructions into micro-instructions that are used as size, address, or select bit per hardware block. One high-level instruction corresponds to one loop. Compared to the RISC-V ISA, our instructions are simple, due to not using a complicated datapath. The vector source is determined between BRAM and DDR, discriminated by their base address. In the same way, the destination is determined between BRAM and DDR. The structure of our instructions is shown in TABLE I.

TABLE I: Instruction Set Bit Mapping

Baseline Model *(Instruction Mapping 64-bit)*		**Proposed Model** *(Instruction Mapping 64-bit)*	
Field name	Bit range	Field name	Bit range
VFU$_{opcode}$	1-0(2bit)	VFU$_{opcode}$	1-0(2bit)
SFU$_{opcode}$	3-2(2bit)	ATU$_{opcode}$	3-2(2bit)
vd(vector dest)	15-4(12bit)	SFU$_{opcode}$	5-4(2bit)
vs1(vector src1)	27-16(12bit)	OGU$_{opcode}$	7-6(2bit)
vs2(vector src2)	39-28(12bit)	OGU$_{opcode}$	9-8(2bit)
size(vector size)	55-40(16bit)	vd(vector dest)	21-10(12bit)
funct8	63-56(8bit)	vs1(vector src1)	33-22(12bit)
		vs2(vector src2)	45-34(12bit)
		size(vector size)	61-46(16bit)
		funct2	63-62(2bit)

IV. Evaluation

A. Evaluation Methodology and Experimental Settings

Our design is implemented using the Vitis High-Level Synthesis (HLS). VFU, ATU, SFU, and OGU blocks are implemented with the HLS stream interface. The VPU block

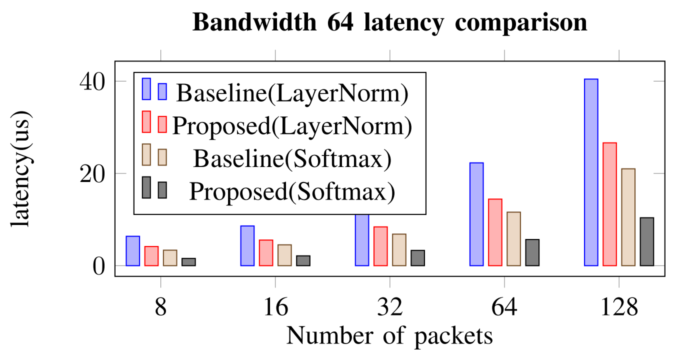
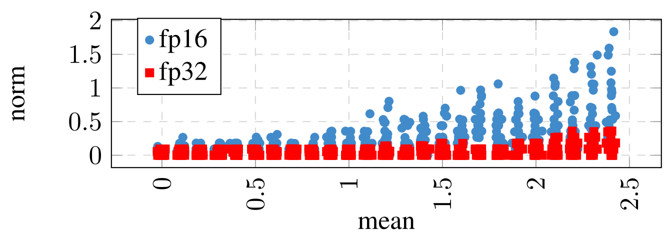
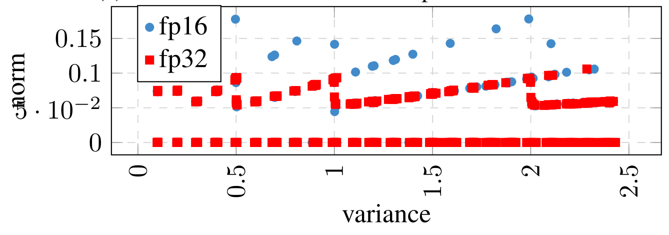


Fig. 4: Latency Evaluation with LayerNorm and Softmax



(a) Normalized L2 Error w.r.t. Input Data Mean



(b) Normalized L2 Error w.r.t Input Data Variance

Fig. 5: Accuracy Evaluation with fp16 and fp32 Accumulators.

is then wrapped and put into a Vivado block design. We use the Xilinx HLS Co-simulation tool for evaluating resource utilization and latency, and Xilinx ZCU106 (Ultrascale+) board for experimenting with multiple configurations. The configuration parameters we experiment with are input vector size and bandwidth.

B. Experimental Results

Performance. We evaluate the latency of LayerNorm compared to the baseline. The datatype used is FP16, and the bandwidth for each packet is chosen between 4 to 64 elements. The input packet number varies between 8, 16, 32, 64, 128. Comparing the latency of computing Softmax and LayerNorm on the baseline model and our VPU, the VPU shows a 40% latency decrease in LayerNorm and a 50% latency decrease in Softmax. This performance increase becomes clearer as the length of input vector increase, influenced by the fact that our model takes fewer loops to operate compared to the baseline.

Hardware Resource. For a bandwidth of 64, our model uses 2130 BRAM, 1033 DSPs, 117365 FFs, and 129772 LUTs, which is 61% more DSPs compared to the baseline. On the other hand, the aforementioned alternate option uses 2415 BRAM, 774 DSPs, 101696 FFs, and 146227 LUTs, reducing DSP by **25%** which is **20.4%** more compared to the baseline. The latency worsens by 1%, still at **50.48%** improvement. In terms of computing units, it uses 193 adders, 65 multipliers, and 64 exponent calculators.

TABLE II: FPGA resource utilization
(Above : Baseline, Below : Proposed model)

BW	number of units			HLS utilization			
	Add	*Mult*	*SubExp*	*BRAM*	*DSP*	*FF*	*LUT*
64	128	64	64	1897	642	86172	105093
	257	129	64	2130	1033	117365	129772
32	64	32	32	965	322	48210	59236
	129	65	32	1086	521	65440	73560
16	32	16	16	499	162	29288	35128
	65	33	16	564	265	39915	44085
8	16	8	8	266	82	19820	23172
	33	17	8	303	137	27092	29012
4	8	4	4	206	42	15549	17464
	17	9	4	255	73	21489	23102

(Vitis HLS) Add: 2DSP, Mult: 2DSP, SubExp: 4DSP

Accuracy. The computation pattern for Softmax remains unchanged from the baseline, but not for LayerNorm. Our measure of accuracy for LayerNorm was L2 norm between our model's output and the baseline, according to change in mean and variance of input data. As in Fig.5(a), the proposed model drops in accuracy as the mean of input increases, attributed to increased sparsity of FP16 numbers far from zero. The proposed architecture encounters this issue when calculating the square sum of inputs, while the baseline does not. Changing the datatype of accumulation from FP16 to FP32 can significantly suppress the error between our model and the baseline's output. After the change to FP32, the range of error decreases irrelevant to the square sum of inputs, due to a more precise datatype. Also, this method limits outliers of L2 distance. As shown in Fig.5(b), L2 distance according to input variance is organized into several lines, whose slope is $\Delta(\frac{1}{\sigma})$. Accumulating by FP32 and casting back, the error becomes at most the FP16 unit error for the current range. This improvement's cost is minimal since only the accumulator following the adder tree is replaced, while the adders constituting the adder tree are not.

Conclusion

We proposed a VPU design to effectively address long-latency operations such as LayerNorm and Softmax. Specifically, we propose an alternative execution order and its dedicated hardware to significantly decrease loop counts in LayerNorm. We also present a method to mitigate an error caused by the alternative execution order. In summary, our proposed design can be utilized to accelerate vector operations in transformer-based networks such as GPT-2.

Acknowledgment

This work was supported in part by BOS Semiconductors Co., Ltd. and in part by the Technology Innovation Program (or Industrial Strategic Technology Development Program –

No. 20014490, Development of Technology for Commercializing Lv.4 Self-driving Computing Platform Based on Centralized Architecture) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea). We thank Donghyun Lee, Vladimir Tyan, and Wook Choii from BOS Semiconductors Co., Ltd for their invaluable comments on the design and the manuscript.

References

[1] Asif M Adnan, Sridhar Radhakrishnan, and Suleyman Karabuk. Efficient kernel fusion techniques for massive video data analysis on gpgpus. *arXiv preprint arXiv:1509.04394*, 2015.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[5] Seongmin Hong, Seungjae Moon, Junsoo Kim, Sungjae Lee, Minsub Kim, Dongsoo Lee, and Joo-Young Kim. Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 616–630. IEEE, 2022.

[6] Suyeon Hur, Seongmin Na, Dongup Kwon, Joonsung Kim, Andrew Boutros, Eriko Nurvitadhi, and Jangwoo Kim. A fast and flexible fpga-based accelerator for natural language processing neural networks. *ACM Transactions on Architecture and Code Optimization*, 20(1):1–24, 2023.

[7] Seongho Jeong, Minseok Seo, Xuan Truong Nguyen, and Hyuk-Jae Lee. A low-latency and lightweight fpga-based engine for softmax and layer normalization acceleration. In *2023 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pages 1–3. IEEE, 2023.

[8] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

[9] Bingbing Li, Santosh Pandey, Haowen Fang, Yanjun Lyv, Ji Li, Jieyang Chen, Mimi Xie, Lipeng Wan, Hang Liu, and Caiwen Ding. Ftrans: energy-efficient acceleration of transformers using fpga. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 175–180, 2020.

[10] Murad Qasaimeh, Kristof Denolf, Jack Lo, Kees Vissers, Joseph Zambreno, and Phillip H Jones. Comparing energy efficiency of cpu, gpu and fpga implementations for vision kernels. In *2019 IEEE international conference on embedded software and systems (ICESS)*, pages 1–8. IEEE, 2019.

[11] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[12] Minseok Seo, Hyuk-Jae Lee, and Xuan Truong Nguyen. Vit-p3de: vision transformer based multi-camera instance association with pseudo 3d position embeddings. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 1340–1350, 2023.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[14] Guibin Wang, YiSong Lin, and Wei Yi. Kernel fusion: An effective method for better power efficiency on multithreaded gpu. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 344–350. IEEE, 2010.

[15] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.