8-bit Transformer Inference and Fine-tuning for Edge Accelerators

Weiming Hu

2024.05.09

Outline

Paper info

- 8-bit Transformer Inference and Fine-tuning for Edge Accelerators, ASPLOS’24
- Priyanka Raina's Research Group at Stanford University
  - VLSI, CGRA
  - Amber VLSI’22, Simba MICRO’19

Background

- Introduction of numerical type used in inference and training

8-bit inference and 8-bit training

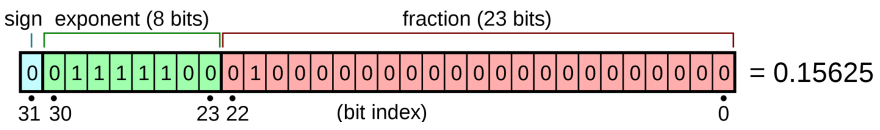
- Specific technical point this paper propose

Arch & Evaluation

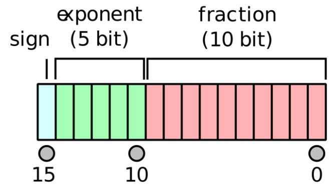
Discussion

Background

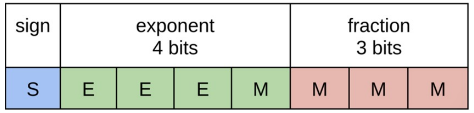
FP32



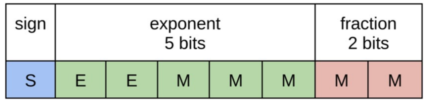
FP16



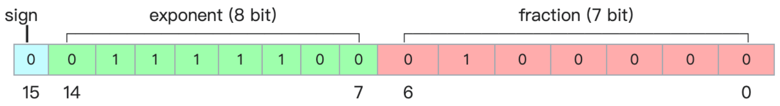
FP8, E4M3



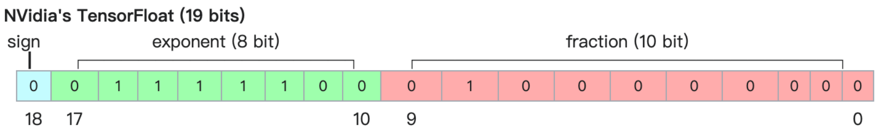
FP8, E5M2



BF16, a truncated IEEE 754 FP32

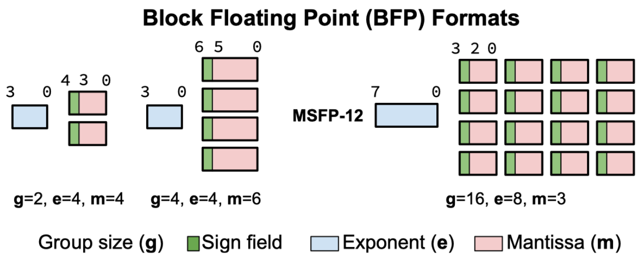


Tensor Float 32, TF32



Others

- Block Floating Point
- Posit
- Flint



2017	2018	2020		2022		2024
posit	BF16, TPUv2 [Google Brain]	TF32, A100 [NVIDIA]	INT4, A100 [NVIDIA]	FP8, H100 [NVIDIA]	Flint [MICRO’22]	FP6, FP4, B200 [NVIDIA]

https://moocaholic.medium.com/fp64-fp32-fp16-bfloat16-tf32-and-other-members-of-the-zoo-a1ca7897d407

Background - Posit

Posit field, posit 8 for example

- Sign bit (1)
- Regime (Variable Length)
- Exponent (Variable Length)
- Mantissa (Fraction, Remaining Bits)

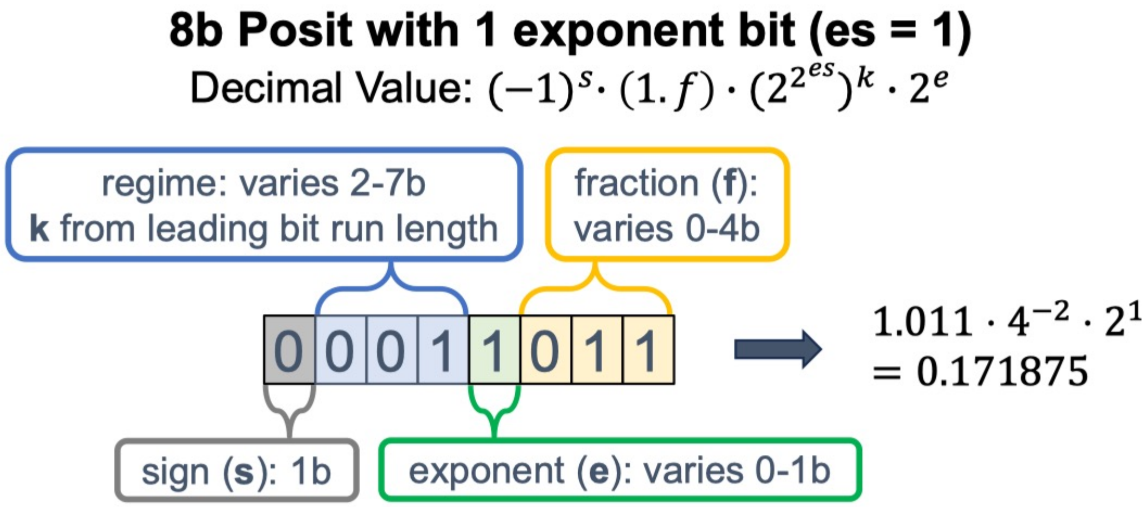
Regime field

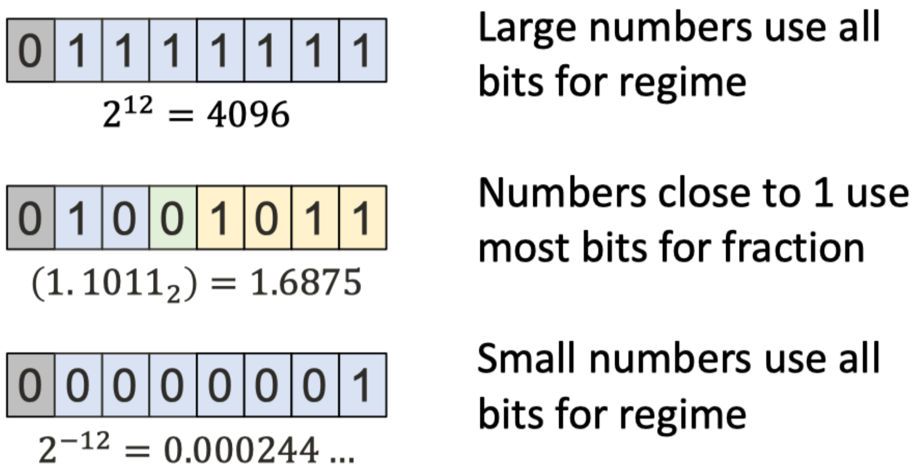
- Start from 0, stop at the first 1. 0001, 001, 01
- Start from 1, stop at the first 0. 1110, 110, 10

k, depend on the run length

- Start from 0, k = -num(0). 001 -> k=-2
- Start from 1, k = num(1)-1. 110 -> k=1

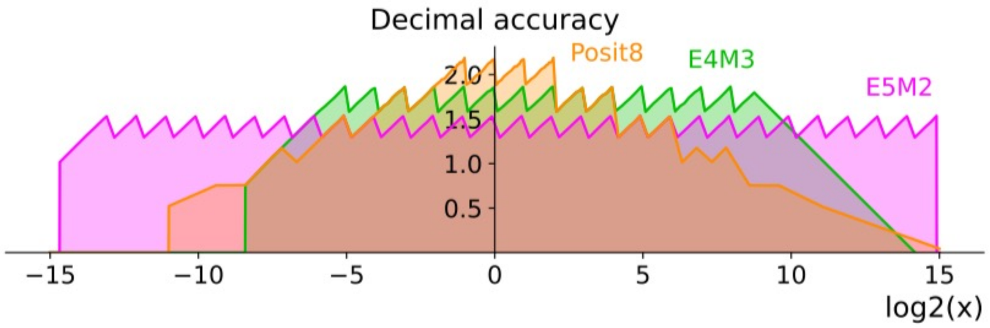
es, number of exponent bit

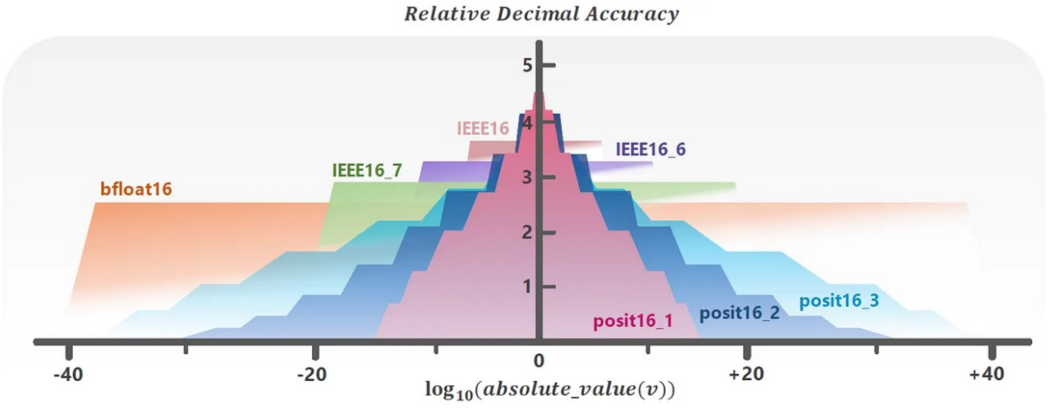




Background - Posit

Posit provides higher decimal accuracy than FP8 around the 0 point.





Range of posit

As an example, fig. 4 shows a build up from a 3-bit to a 5-bit posit with $es = 2$, so $useed = 16$:

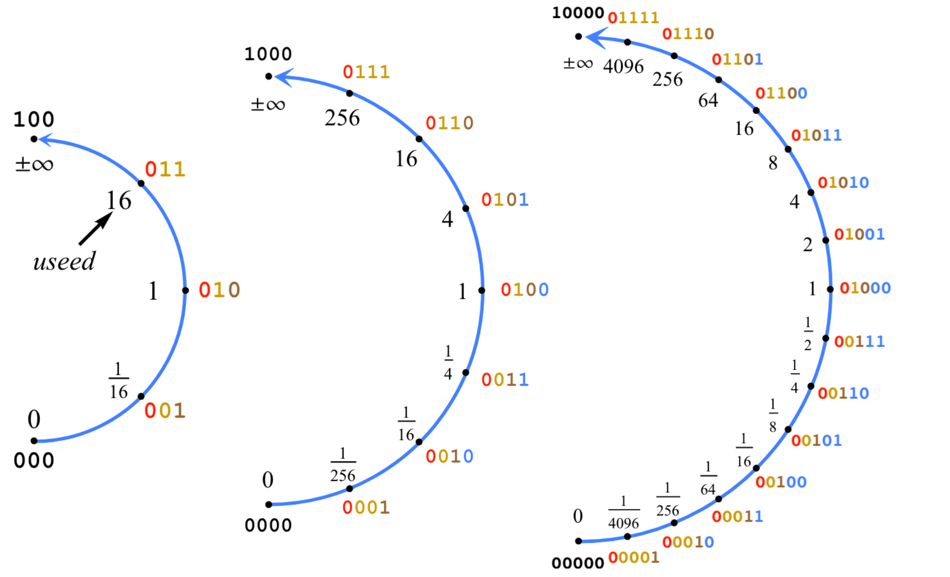
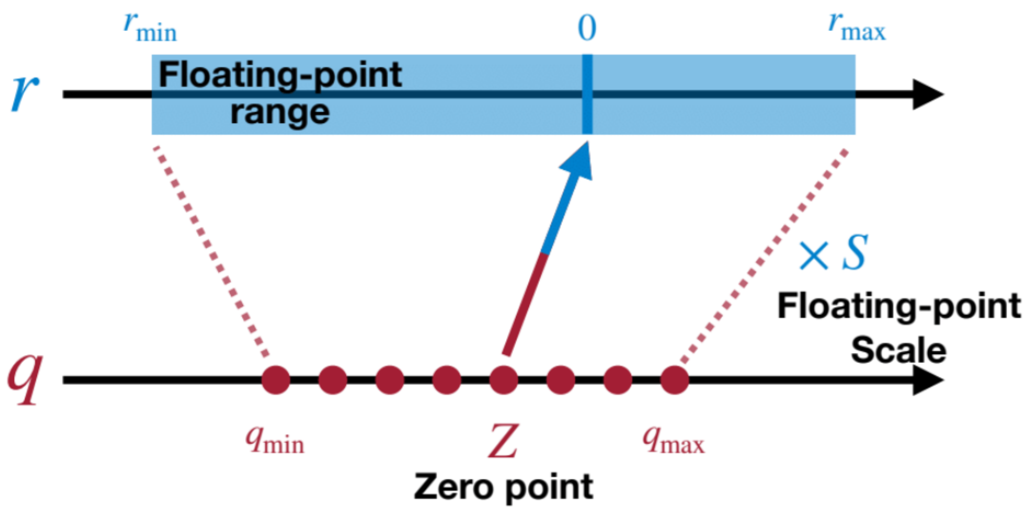


Figure 4. Posit construction with two exponent bits, $es = 2$, $useed = 2^{2^{es}} = 16$

Background - Quantization

Linear Quantization is an affine mapping of integers to real numbers

High-precision format -> low-precision format



$$r_{\max} = S\left(q_{\max} - Z\right)$$

$$r_{\min} = S\left(q_{\min} - Z\right)$$

$$\downarrow$$

$$r_{\max} - r_{\min} = S\left(q_{\max} - q_{\min}\right)$$

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$$

Contribution

Employ FP8 and Posit8 in LLM inference and training

Operation fusion to mitigate PTQ accuracy loss

8-bit fine-tuned with improved LoRA

An area- and power-efficient posit-based softmax

Motivation

Some people employ INT8 quantization in inference, but int8 lacks the precision and range required for training

INT8 is inefficient in LLM inference and training

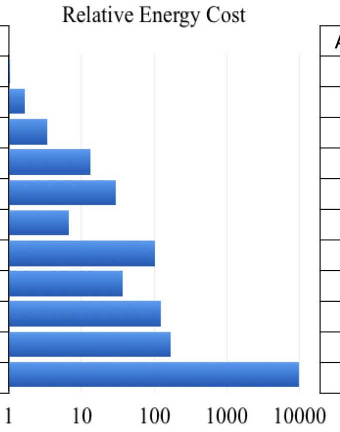
- per-channel or per-vector scaling incurs an additional set of scaling factors
- special handling of outliers

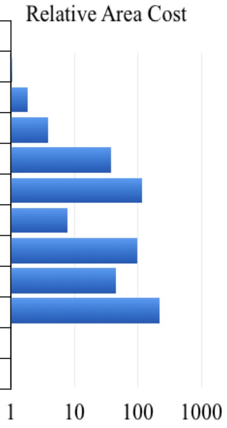
Prior work only use FP8 quantization in GEMM operation

Less bit width -> less energy

Need to Understand Cost of Operations And Communication

Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A





Energy numbers are from Mark Horowitz “Computing’s Energy Problem (and what we can do about it)”, ISSCC 2014
Area numbers are from synthesized result using Design Compiler under TSMC 45nm tech node. FP units used DesignWare Library.

Motivation

Prior Work

Name	Data Type	Inference	Training	Operations Quantized	Techniques
LLM.int8()	int8	✅	❌	GEMM	A high precision (int16) matrix to store outliers
SmoothQuant	int8	✅	❌	GEMM	Per-token + per-channel scaling
QLoRA	NF4	✅	✅	None	Per-vector scaling on pre-trained weights
NVIDIA FP8	E4M3, E5M2	✅	✅	GEMM	Per-tensor scal[illegible]ll GEMM
This work	Posit8, FP8	✅	✅	All operations	Only per-tenso[illegible], during backward pass

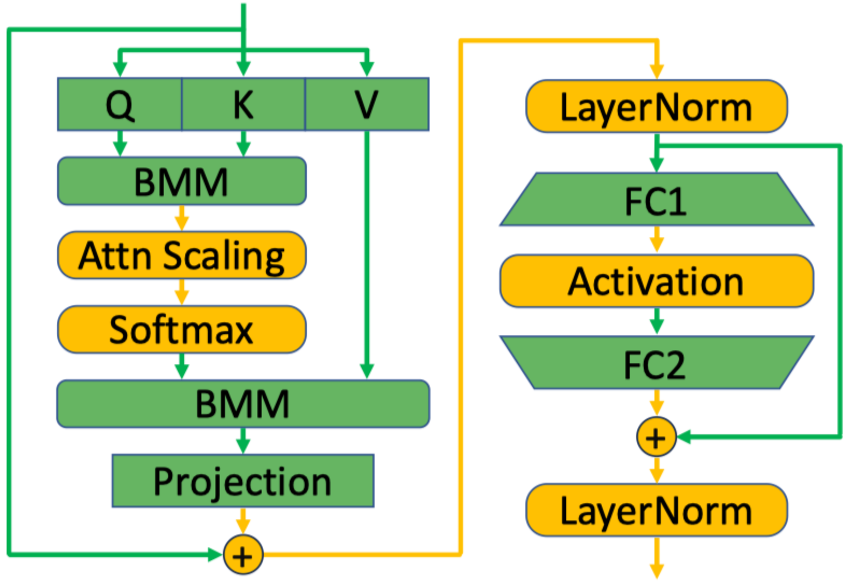
8-bit Inference-Operation Fusion

Goal

- mitigate the accuracy loss from quantization
- fuse element-wise operations with the preceding GEMM operation

No fuse means quantize different ops to Posit8 and FP8 independently

a gradual improvement in accuracy with higher levels of fusion



Model	Size	BF16	No Fusion		GEMM + Attn Scaling Fusion		+ Activation Fusion		+ LayerNorm Fusion		+ Residual Fusion	
			Posit8	E4M3	Posit8	E4M3	Posit8	E4M3	Posit8	E4M3	Posit8	E4M3
$\text{MobileBERT}_{\text{tiny}}$	16M	88.8	86.3	87.0	87.4	87.1	87.7	87.5	**87.9**	87.8	88.4	**88.1**
MobileBERT	25M	89.9	65.1	82.7	85.0	84.9	88.3	86.7	**89.0**	87.9	89.4	88.6
$\text{DistilBERT}_{\text{base}}$	66M	86.9	**86.2**	**86.1**	86.4	86.1	86.7	86.4	86.7	86.5	86.7	86.5
$\text{BERT}_{\text{base}}$	109M	88.2	87.1	**87.7**	**88.1**	88.0	88.1	88.0	88.1	88.0	88.1	88.0
$\text{BERT}_{\text{large}}$	334M	93.2	92.3	**93.0**	**92.8**	93.1	93.0	93.1	93.0	93.2	93.1	93.1

Table 2. Transformers' F1 scores on SQuAD v1.1 using Posit8 and FP8 with varying levels of operation fusion. Figures in bold indicate the minimum fusion level needed to achieve within 1% accuracy drop. For MobileBERT models, we need to fuse all operations to achieve within 1% drop. For BERT models, we can easily achieve the same goal even without any fusion.

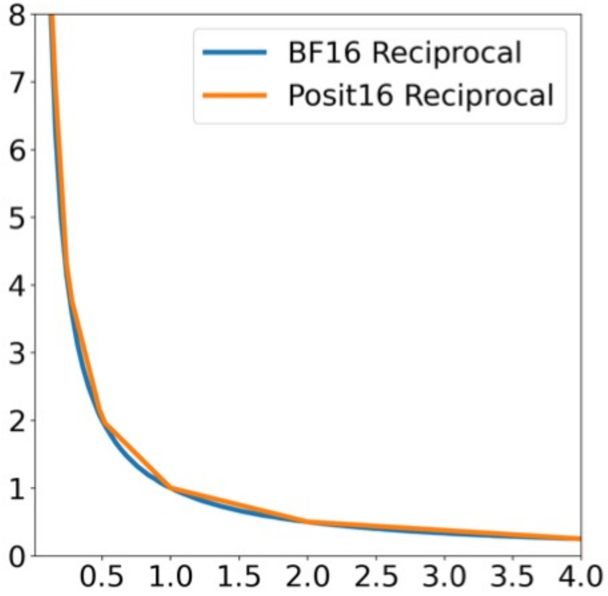
8-bit Inference-Approximate Operations

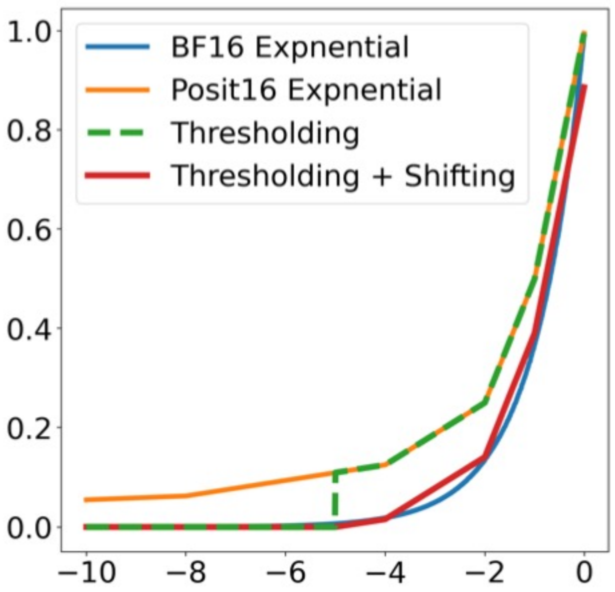
Goal

- Solve the incomputability of posit8
- Avoid expensive operation in hardware

Approximate Operations Using Posits

- Reciprocal Approximation and Exponential Approximation
- Method: Fast approximations of activation functions in deep neural networks when using posit arithmetic





S(x) -> Sigmod

$$S(x) = \frac{1}{1+e^{-x}} \Rightarrow e^{x} = \frac{1}{S(-x)} - 1$$

$$f(x) = \begin{cases} \frac{1}{S(-x)} - \epsilon & \text{if } x \geq \theta \\ 0 & \text{if } x < \theta \end{cases}$$

8-bit Inference-Approximate Operations

Softmax

- Exponential
- Division

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K.$$

Replacing the division in softmax with this approximate reciprocal results in only 0.8% accuracy loss on MobileBERT models and 0.1% on BERT models during inference

	e^x	$1/x$	MobileBERT	$\text{BERT}_{\text{base}}$
BF16	-	-	89.9	88.2
Posit8	-	-	89.4	88.0
Posit8	✓	-	88.9	88.1
Posit8	-	✓	88.8	88.0
Posit8	✓	✓	88.6	87.9

Table 4. F1 scores of MobileBERT and BERT on SQuAD v1.1 with softmax built using approximate posit exponential (e^x) and posit reciprocal ($1/x$).

8-bit Inference-LLM

Posit (8, 2) has a unique advantage than Posit (8, 1) and FP8 in large models due to its wider range.

Model	BF16	Data Type	No Fusion	Fuse GEMM + Attn Scaling	+ Activation Fusion	+ LayerNorm Fusion	+ Residual Fusion
GPT-2 Large (762M)	16.38	Posit (8, 1)	18.00	17.75	17.50	17.50	16.63
		Posit (8, 2)	17.50	17.50	17.50	17.50	16.63
		E4M3	17.13	17.13	17.13	17.13	16.63
GPT-2 XL (1.5B)	14.69	Posit (8, 1)	18.00	17.75	17.75	17.50	14.94
		Posit (8, 2)	17.75	17.75	17.75	17.75	14.94
		E4M3	15.63	15.63	15.63	15.63	14.94
LLaMA 2 (7B)	5.19	Posit (8, 1)	5.56	5.53	5.53	5.52	5.30
		Posit (8, 2)	5.44	5.40	5.38	5.37	5.29
		E4M3	5.80	5.80	5.77	5.75	5.36
LLaMA 2 (13B)	4.63	Posit (8, 1)	4.85	4.78	4.78	4.77	4.72
		Posit (8, 2)	4.86	4.82	4.81	4.80	4.72
		E4M3	5.10	5.09	5.07	5.06	4.73

Table 6. Perplexity of LLMs on WikiText-103 using Posit (8, 1), Posit (8, 2), and FP8 with incremental levels of operator fusion.

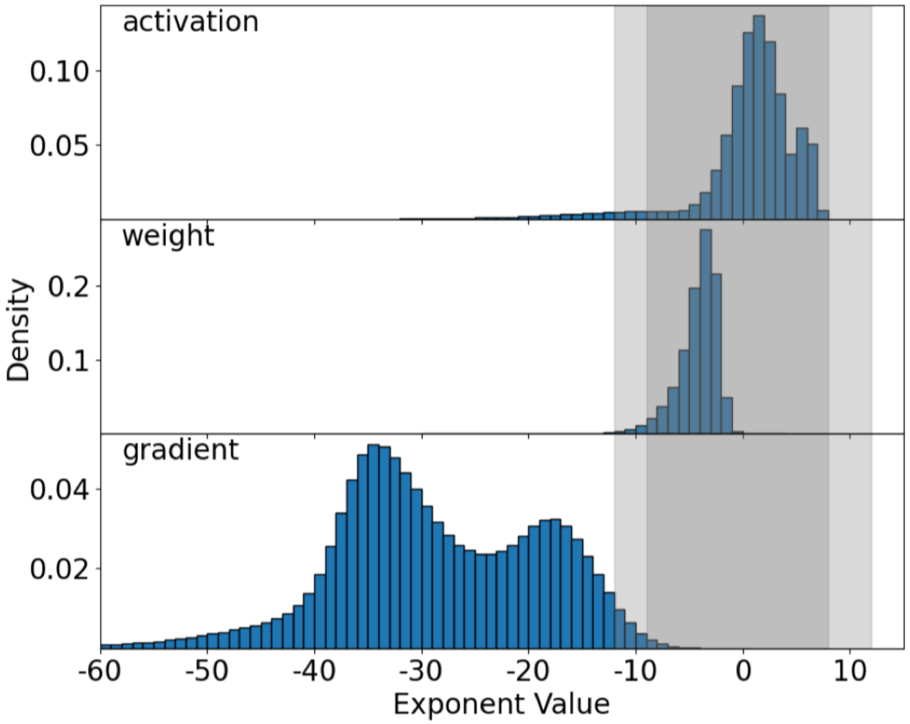
8-bit fine-tuning

Problem

- activation gradient are small magnitude values, beyond the range of Posit8 and FP8

They use a per-tensor scaling, allowing each tensor to have its own exponent bias

use historical gradient statistics to predict the amax for this iteration and compute the scaling factor based on the prediction



8-bit fine-tuning

Problem

- Approximate Softmax derivation
- The exponential function can be directly applied without any modification for the backward pass

Revised softmax backward operation

$$\sigma(\vec{z})_j = e^{z_j} \cdot f(\sum_{k=1}^{K} e^{z_k})$$

We can apply the product rule of derivative:

$$\frac{\partial \sigma(\vec{z})_j}{\partial z_i} = \frac{\partial}{\partial z_i} e^{z_j} \cdot f(\sum_{k=1}^{K} e^{z_k}) + e^{z_j} \cdot f'(\sum_{k=1}^{K} e^{z_k}) \cdot \frac{\partial}{\partial z_i} \sum_{k=1}^{K} e^{z_k}$$

$$\frac{\partial \sigma(\vec{z})_j}{\partial z_i} = \begin{cases} \sigma(\vec{z})_j + e^{z_j} \cdot f' \cdot e^{z_i} & \text{if } i = j \\ e^{z_j} \cdot f' \cdot e^{z_i} & \text{if } i \neq j \end{cases} \quad (4)$$

where f' is a piece-wise linear function that models the derivative of posit reciprocal (Figure 7):

$$f' = -2^{-\lfloor \log_2 (\sum_{k=1}^{K} e^{z_k}) \rfloor \cdot 2 - 1} \quad (5)$$

8-bit fine-tuning-LoRA

Problem of previous LoRA

- upscale quantized pre-trained weights to a high-precision format and merge them with trainable low-rank matrices before linear operations.
- This prevents the use of smaller, more efficient MACs with 8-bit arithmetic.

Their mehod: $h = W_0 x + \Delta W x = W_0 x + \alpha \cdot BAx$

W_0, pre-trained weight matrix

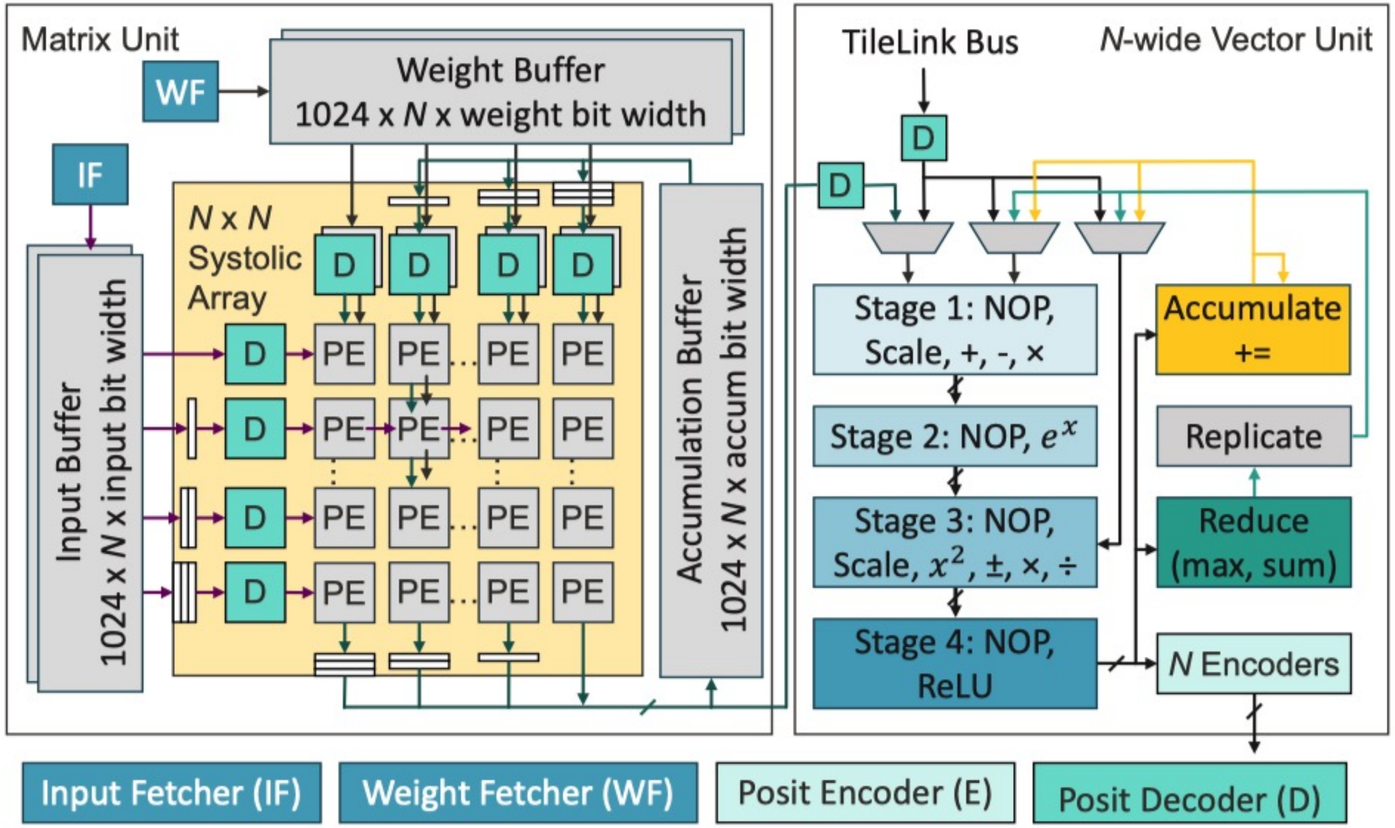
ΔW is the weight update to W_0

B and A are the low-rank decomposition of ΔW with a scaling factor of α

B and A in 16-bit floating-point

$$h = \mathrm{quant}(W_0^8 + \alpha \cdot \mathrm{quant}(B^{16})\mathrm{quant}(A^{16}))x$$

Architecture



A Posit8 MAC -> E5M4 MAC

Evaluation

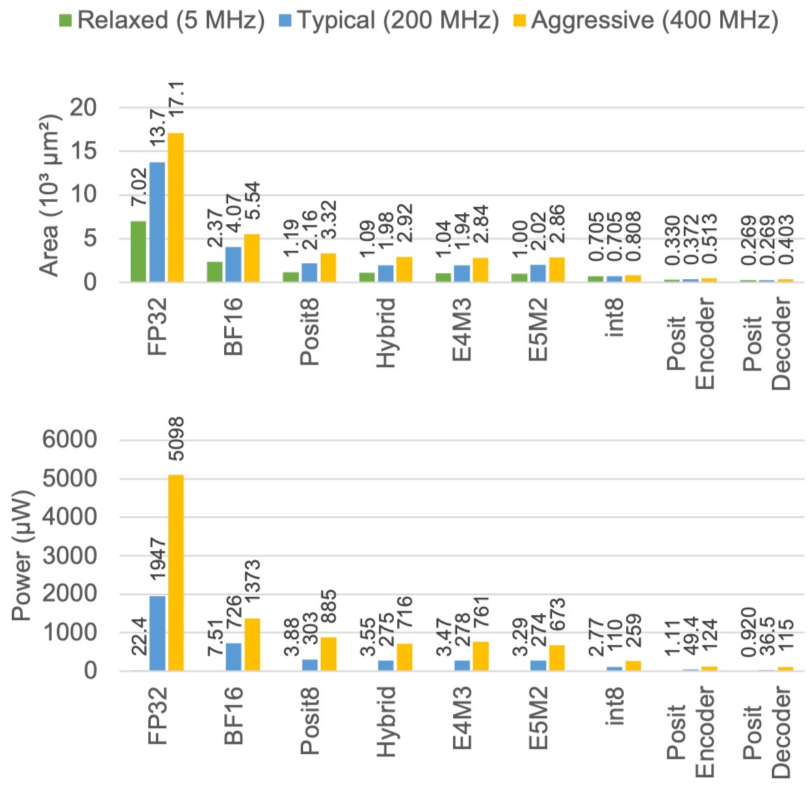
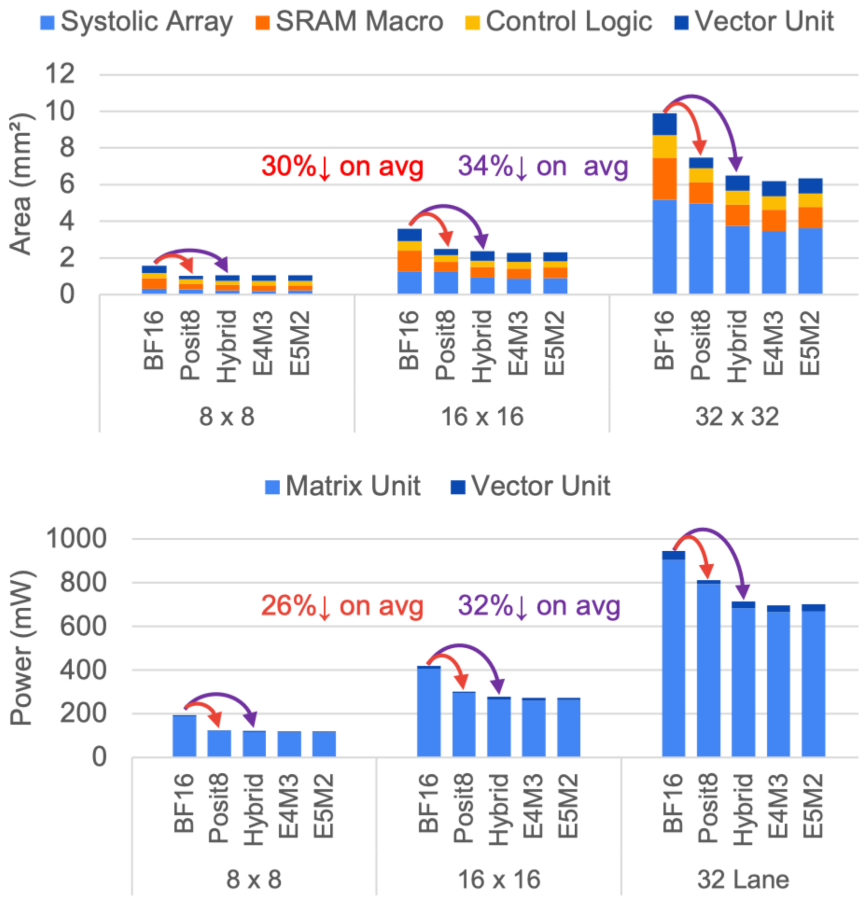


Figure 12. MAC area and power without encoding and decoding logic and Posit8 encoder and decoder area and power.



Posit exploits the approximate operation, thus it does not have vector unit overhead

Discussion

What the feature in hardware is needed for LLM inference and training edge device?

- Memory and compute efficiency
- Low area and power overhead
- High memory bandwidth? Efficient attention computation? Dataflow?

Memory bound, HBM process is limited, how to design LLM hardware

- Compression technology

Unified inference and training architecture

- Data type
    - Inference: FP4, INT4
    - Fine-tuning: FP8

Algorithm and application requirement motivates the architecture design