Lecture 12:
Transformer & DNN Training Accelerators

Notes

- Final Presentation
  - 5/8/2025 (9am-8pm, 370 Jay Street, Rm 1013)
  - Preferfer in-person, but can be online presentation if necessary.
  - Signup spreadsheet (will be sent tonight)
  - Presentation time:
    - 20mins + 5mins QA, a timer will be used.
    - The duration may be shorter (~20 mins) for projects involving a single student.
    - The presentation will include the following parts: Introduction, background, methodology, evaluation, conclusion.

Notes

- Due on **May 15 11:59pm**
- NeurIPS format: https://www.overleaf.com/latex/templates/neurips-2024/tpsbbrdqcmsh
- Four-seven pages
  - Introduction
  - Individual contribution (if more than one student)
  - Problem Description
  - Related work
  - Method
  - Experiment results
  - Conclusion

NYU SAI LAB

Guest Talk Next Week

- Title: Compression in the Age of Foundation Models

- Hao Wang is a Senior Principal Research Scientist with the Red Hat AI Innovation Team and the MIT-IBM Watson AI Lab. He earned his Ph.D. in Applied Mathematics from the School of Engineering and Applied Sciences (SEAS) at Harvard University. Prior to joining Harvard, he received his B.S. in Mathematics and Applied Mathematics from the University of Science and Technology of China (USTC) in 2016. He was awarded the 35th Guo Moruo Scholarship, the highest honor at USTC.

Recap

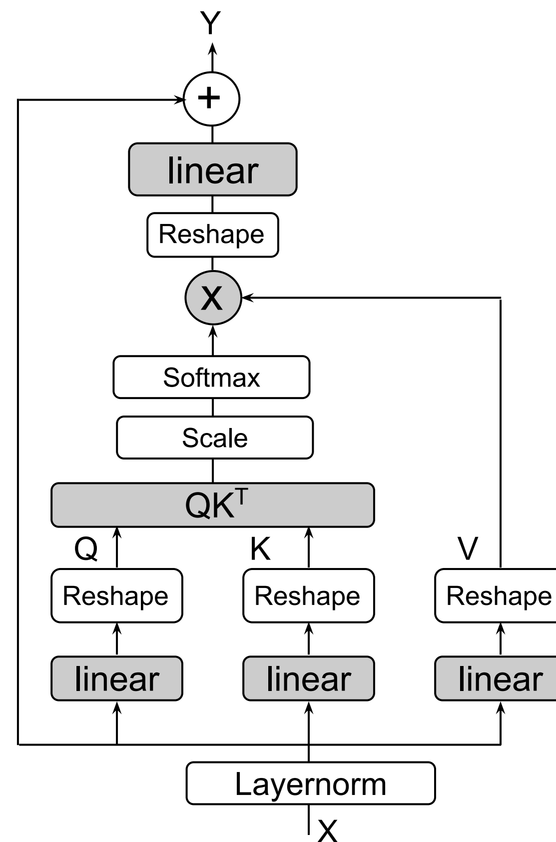
- Systolic Array
- Popular CNN Accelerator Design
- Transformer Accelerator

Topics

- Hardware design for Operations other than Matrix Multiplication
- Hardware architecture for backward propagation design.
- Training Accelerator Design

Self-Attention Block

- Given input x, the first step in calculating self-attention is to create three vectors from each of the input x’, denoted as: Query (Q), Key (K), Value (V).
    - $(B,L,E) \times (E \times E) \rightarrow (B \times L \times E)$
- The second step in calculating self-attention. This will compute the attention score between each pair of input tokens.
    - $QK^T \rightarrow (B, L \times E) \times (B,E \times L) \rightarrow (B, L \times L)$
- Scale and normalize the score using softmax.
    - $\text{Softmax}(QK^T) \rightarrow (B, L \times L)$
- Multiply each value vector by the softmax score.
    - $\text{Softmax}(QK^T) \times V$
    - $(B, L \times L) \times (B, L \times E) \rightarrow (B, L \times E)$
- Pass the result to the linear layer, sum with the input.



NYU SAI LAB

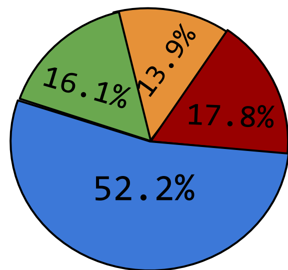
Operations Other than Multiplications

- Transposition
- Nonlinear operations
  - Softmax
  - LayerNorm
  - GeLU

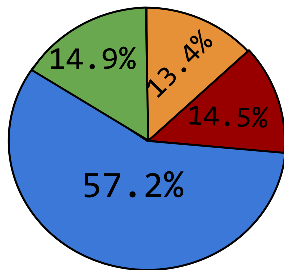
Breakdown on Computational Cost

Latency Breakdown

Matmul Normalization Softmax Others



GPT2

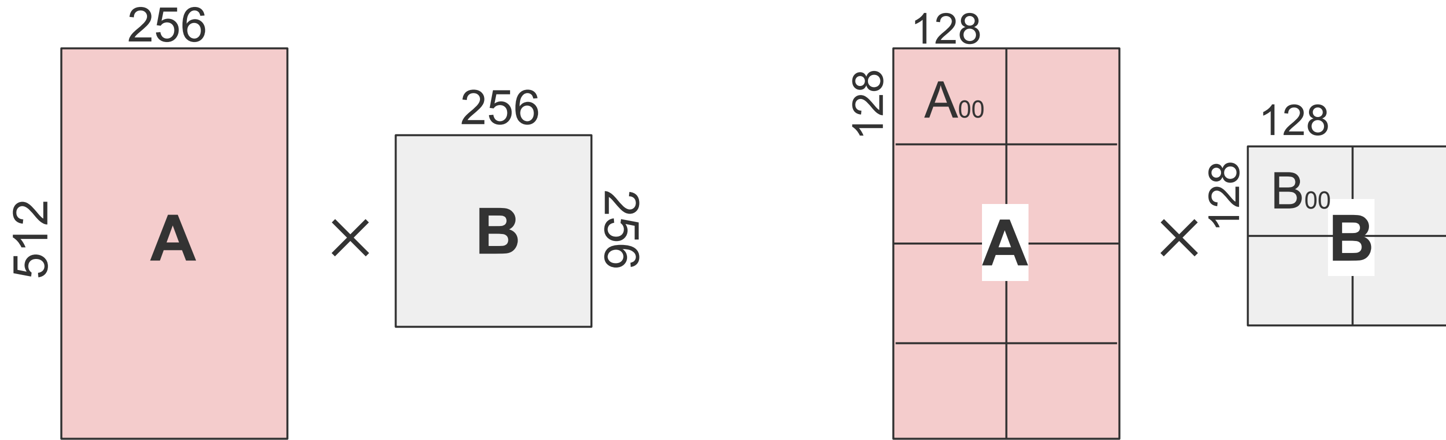


OPT

- Matmul still contributes to majority of the overall latency.
- Nonlinear operations are not negligible.
- Also other operations (e.g., transposition) also contributes to a great portion of the overall latency.

NYU SAI LAB

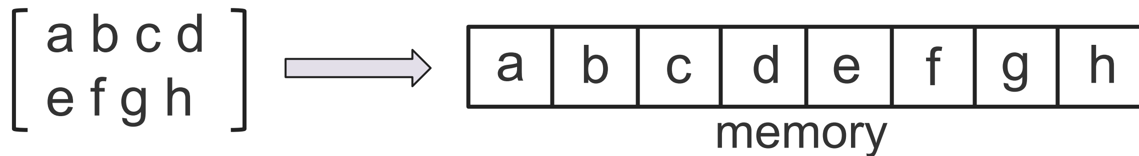
Matrix Multiplication

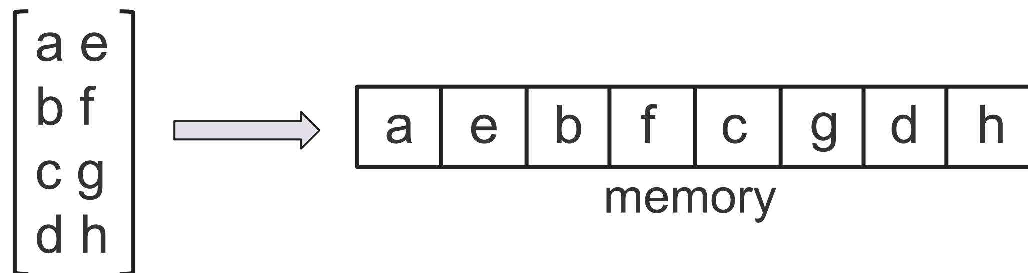


- The large matrix operands are first partitioned into tiles that can fit the size of the compute core.

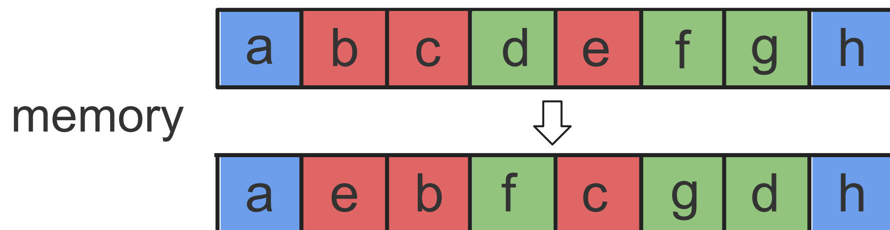
In-Place Matrix Transposition

- In-place matrix transposition refers to the process of transposing a matrix directly within its existing memory space, requiring only a minimal amount of extra storage.

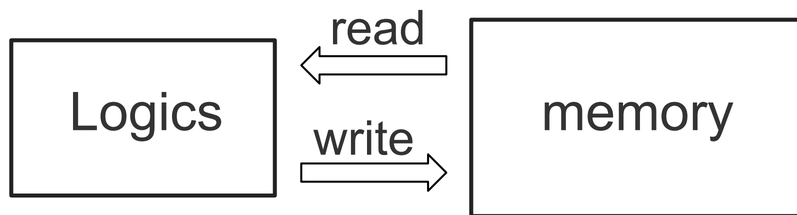




In-Place Matrix Transposition



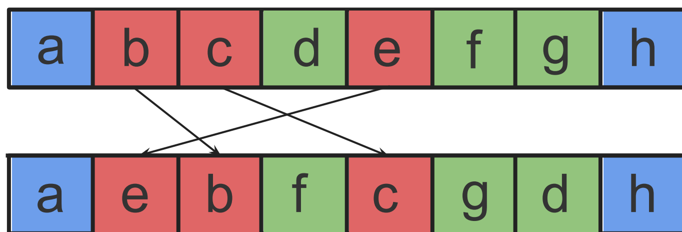
(b,c,e) → (e,b,c)
(d,f,g) → (f,g,d)



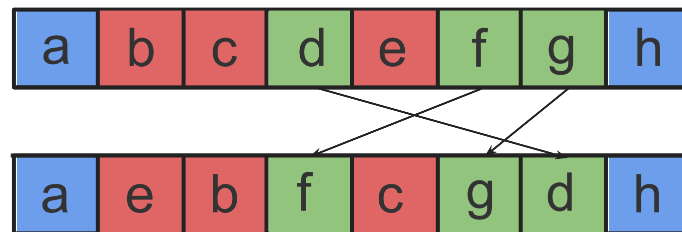
- Need to read multiple entries from the memory, permute them and write them back.
- This operation should be performed efficiently with minimal memory access cost.

NYU SAI LAB

In-Place Matrix Transposition



Step 1



Step 2

- The search for optimal swapping patterns that minimize permutations is a well-established problem in mathematics.

NYU SAI LAB

Norman Brenner, "Algorithm 467: matrix transposition in place," ACM Transactions on Mathematical Software 16 (11), p. 692-694 (1973). doi:10.1145/355611.362542

Implementation of Nonlinear Operations: Softmax

- Softmax operations are heavily adopted in the transformer.

$$s_i = \frac{e^{z_i}}{\Sigma_{j=0}^{j=N-1} e^{z_j}} \quad For\ i = 1, 2, \cdots, N$$

- For positive z with INT representation, we can approximate the values of e^z using the following derivations:

$$e^z = 2^{z\ log_2 e} = 2^{u+v} \qquad log_2 e \approx 1.0111_2$$

$$z\ log_2 e \approx z + (z >> 2) + (z >> 3) + (z >> 4)$$

- To compute 2^{u+v}, we can perform shift and multiplication:

$$e^{z'} = 2^{u+v} \approx 2^u (1 + v/2)$$

u and v are the integer and fractional part of the exponent, v/2 is the mantissa, u is the exponent

NYU SAI LAB

Taylor Approximation

- A Taylor series is a series expansion of a function about a point. A one-dimensional Taylor series is an expansion of a real function f(x) about a point x=a is given by:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \ldots .$$

- For small v, e^v can be approximated as:

$$e^v \approx 1 + \frac{v}{2} \qquad 2^v \approx 1 + \frac{v}{2} \qquad log(1+x) \approx x$$

NYU SAI LAB

Xia, Tianhua, and Sai Qian Zhang. "Softmax Acceleration with Adaptive Numeric Format for both Training and Inference." *arXiv preprint arXiv:2311.13290* (2023).

Taylor Approximation

- A Taylor series is a series expansion of a function about a point. A one-dimensional Taylor series is an expansion of a real function f(x) about a point x=a is given by:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \ldots.$$

- For small v, e^v can be approximated as:

$$e^v \approx 1 + \frac{v}{2} \qquad 2^v \approx 1 + \frac{v}{2} \qquad log(1+x) \approx x$$

NYU SAI LAB

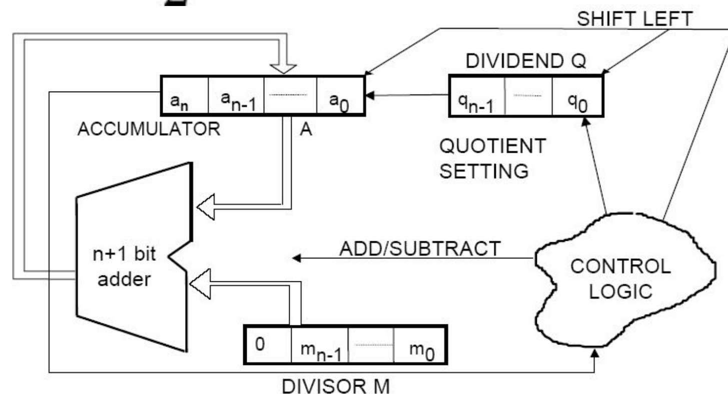
Xia, Tianhua, and Sai Qian Zhang. "Softmax Acceleration with Adaptive Numeric Format for both Training and Inference." *arXiv preprint arXiv:2311.13290* (2023).

Division

- To implement division operation with FP format, we can always apply the following derivations:

$$\frac{a}{b} = 2^{e_a}(1+m_a)/2^{e_b}(1+m_b) = 2^{e_a - e_b + log_2(1+m_a) - log_2(1+m_b)}$$
$$\approx 2^{e_a - e_b + m_a - m_b} \approx 2^{e_a - e_b}(1 + \frac{m_a + m_b}{2})$$

- For INT division, we can also implement the hardware divisor.



NYU SAI LAB

Implementation of Nonlinear Operations: LayerNorm

- For the input vector z, the normalization operation requires to computes its mean and variance, then the intermediate results are scaled with some predefined values.

$$s = \alpha \frac{z - \mu_z}{\sigma_z} + \beta \qquad \mu_z = \frac{\sum_i z_i}{N} \qquad \sigma_z = \sqrt{\frac{\sum_i (z_i - \mu_z)^2}{N}}$$

- Most of the operations are supported, the inverse of square root can be computed as follows:

$$y = \frac{1}{\sqrt{x}} \qquad \log_2(y) = -\frac{1}{2}\log_2(x)$$

NYU SAI LAB

Implementation of Nonlinear Operations: LayerNorm

- Most of the operations are supported, the inverse of square root can be computed as follows:

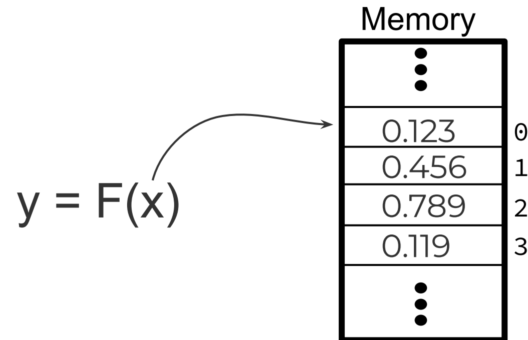
$$y = \frac{1}{\sqrt{x}} \qquad \log_2(y) = -\frac{1}{2}\log_2(x)$$

$$x = 2^{E_x - Q}(1 + M_x/2^L) \qquad \begin{aligned} \log_2 x &= E_x - Q + \log_2(1 + M_x/2^L) \\ &\approx E_x - Q + M_x/2^L + \sigma_x \end{aligned}$$

- Q is the bias, Ex and Mx are the binary representations of the exponent and mantissa, respectively.

Table Lookup

- For other complicated nonlinear functions, we can always precompute the results and store them in the buffer.



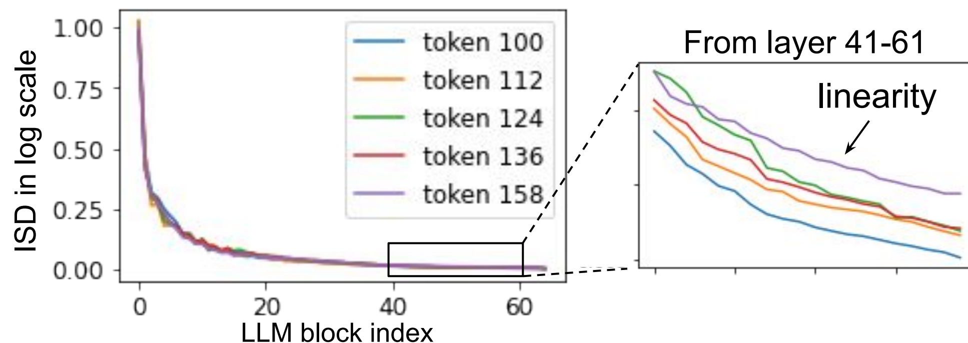
- However, this will inevitably lead to additional memory access cost and footprint.

HAAN: LayerNorm Accelerator

Layer Normalization:

$$s = \alpha \frac{z - \mu_z}{\sigma_z} + \beta$$

Computing the inverse of standard deviation of costly



- Exploit correlation in input statistics across layers.
- Skip redundant computations and estimate normalization statistics.

NYU SAI LAB

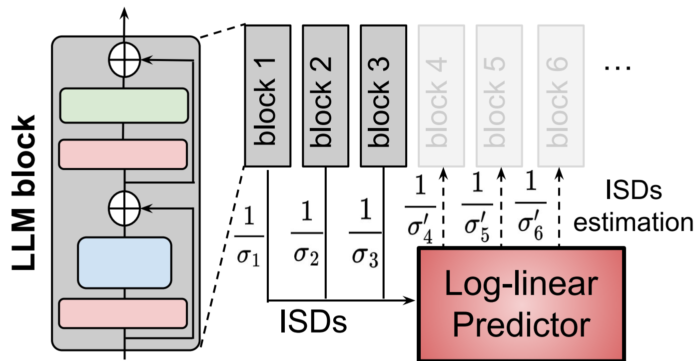
Peng, Tianfan, et al. "HAAN: A Holistic Approach for Accelerating Normalization Operations in Large Language Models." *arXiv preprint arXiv:2502.11832* (2025).

HAAN: LayerNorm Accelerator

Layer Normalization:

$$s = \alpha \frac{z - \mu_z}{\sigma_z} + \beta$$

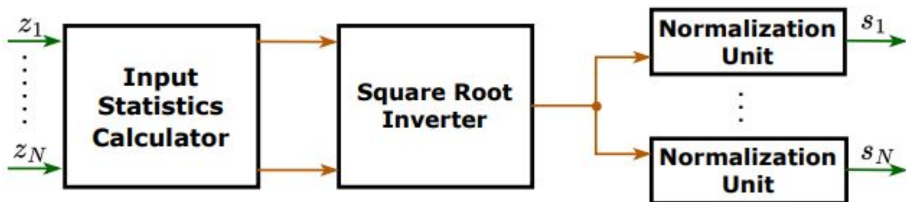
Computing the inverse of standard deviation of costly

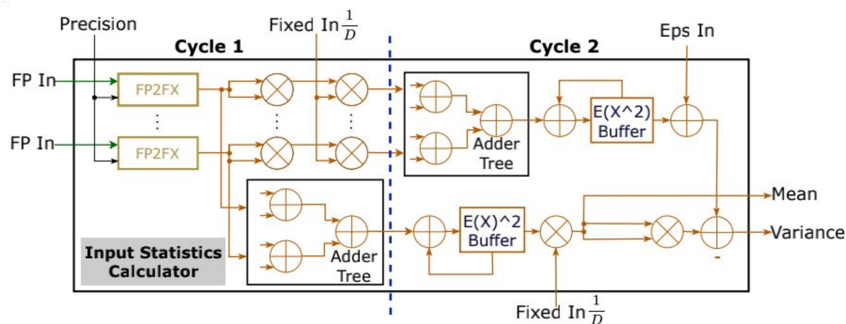


- Exploit correlation in input statistics across layers.
- Skip redundant computations and estimate normalization statistics.

NYU SAI LAB

HAAN: LayerNorm Accelerator

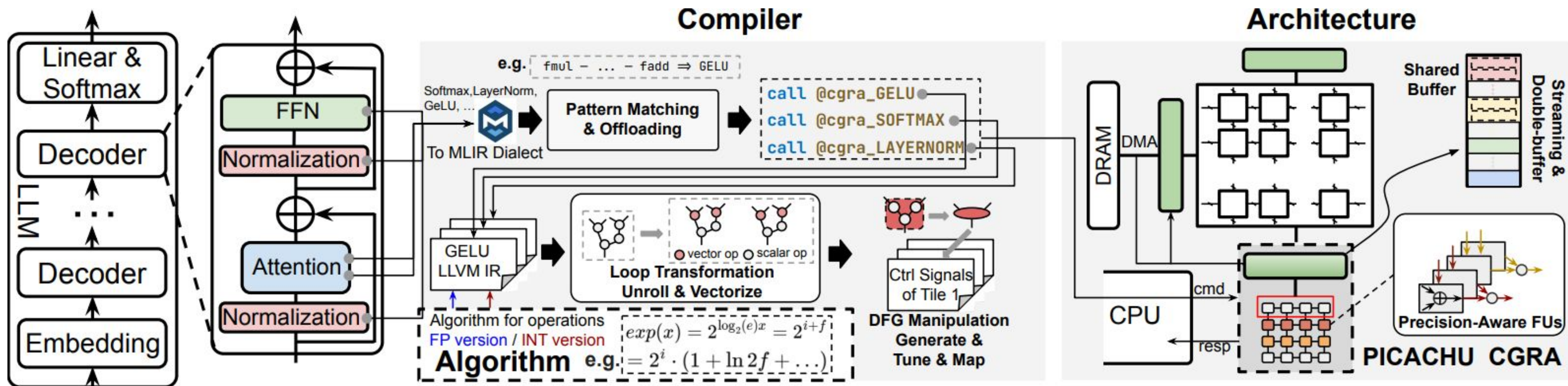




- **Overall Architecture**
  - Input Statistics Calculator.
  - Square Root Inverter.
  - Normalization Unit.
- **Input Statistics Calculator**
  - Compute mean and variance.
  - Parallel processing to reduce latency.
- **Square Root Inverter**
  - Approximate inverse square root using Newton's method.
  - Support for layer skipping.

NYU SAI LAB

PICACHU



- PICACHU is a plug-in coarse-grained reconfigurable accelerator tailored to efficiently handle nonlinear operations by using custom algorithms and a dedicated compiler toolchain.

Qin, Jiajun, et al. "PICACHU: Plug-In CGRA Handling Upcoming Nonlinear Operations in LLMs." Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 2025.

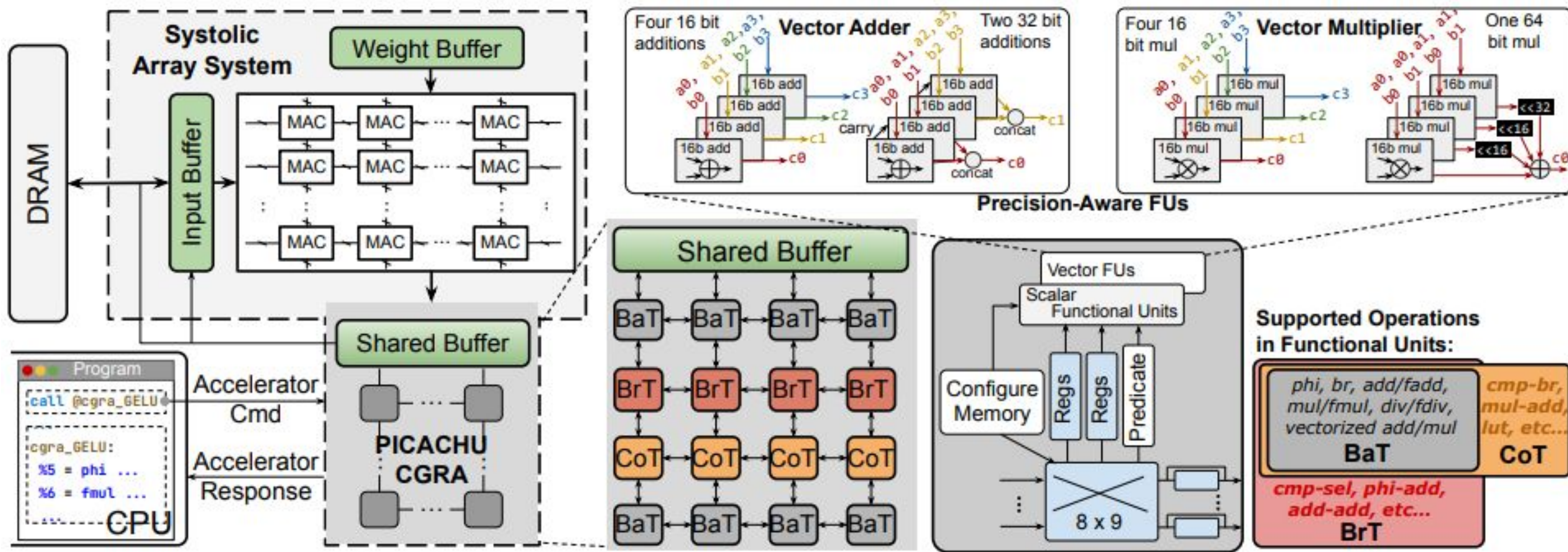
PICACHU

Categories	Nonlinear Operations	Mathematical Operator	Representative LLMs
Activation Function	Softmax$(x_i) := \frac{\exp(x_i)}{\sum_{j=1}^{k}\exp(x_j)} = \frac{\exp(x_i - u)}{\sum_{j=1}^{k}\exp(x_j - u)}$; $u = \max_{j=1} x_j$	Division, Exponential	All
	$\mathrm{ReLU}(x) := \max(0, x)$	Maximum	OPT [145], T5 [90]
	$\mathrm{GeLU}(x) := 0.5x\left(1 + \mathrm{Tanh}(\sqrt{2/\pi}(x + 0.044715x^3))\right)$; $\mathrm{Tanh}(x) = (\exp(x) + \exp(-x))/(\exp(x) - \exp(-x))$	Division, Exponential	GPT [14, 84, 87, 88], BLOOM [57], Falcon [83], PanGu-α [144], Jurassic-1 [64], Gopher [89]
	$\mathrm{GeGLU}(x) := \mathrm{GeLU}(xW + b) \oplus (xV + c)$	Division, Exponential	LaMDA [110], GLM-130B [143]
	$\mathrm{SwiGLU}(x) := \mathrm{SiLU}(xW + b) \oplus (xV + c)$; $\mathrm{SiLU}(x) = x \cdot \mathrm{sigmoid}(x) = x \cdot \frac{1}{1+\exp(-x)}$	Division, Exponential	PaLM [17], LLaMA [113, 114], Qwen [7], DeepSeek [11], InternLM [15], Yi [135]
Normalization Function	LayerNorm$(x_i) := \frac{x_i - \mu}{\sigma}$; $\mu = \frac{1}{C}\sum_{i=1}^{C} x_i$, $\sigma = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(x_i - \mu)^2 + \epsilon}$	Inverted Square Root	GPT [14, 84, 87, 88], BLOOM [57], BERT [20], OPT [145], PanGu-α [144], Jurassic-1 [64]
	RMSNorm$(x_i) := \frac{x_i}{\sigma}$; $\sigma = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(x_i)^2 + \epsilon}$	Inverted Square Root	LLaMA [113, 114], T5 [90], Mistral [43], Qwen [7], DeepSeek [11], Gopher [89]
Positional Embedding	RoPE $\begin{pmatrix} x_{2i-1} \\ x_{2i} \end{pmatrix} = \begin{pmatrix} x_{2i-1}\cos(m\theta_i) - x_{2i}\sin(m\theta_i) \\ x_{2i-1}\sin(m\theta_i) + x_{2i}\cos(m\theta_i) \end{pmatrix}$; $\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \ldots, d/2]$	Sine, Cosine	GPTNeo-20B [13], LLaMA [113, 114], PaLM [17], GLM-130B [143], Qwen [7], DeepSeek [11]

- All nonlinear operations within LLM can be broken down into various mathematical operators.

Qin, Jiajun, et al. "PICACHU: Plug-In CGRA Handling Upcoming Nonlinear Operations in LLMs." Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 2025.

PICACHU



- These tiles are classified into three types: Branch-optimized Tile (BrT), Basic Tile (BaT) and Compute Tile (CoT), each with distinct FUs, creating a heterogeneous CGRA.

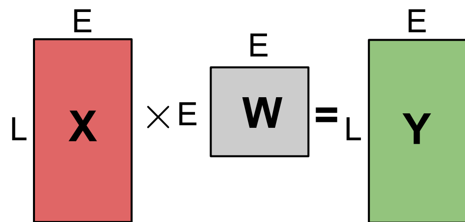
Qin, Jiajun, et al. "PICACHU: Plug-In CGRA Handling Upcoming Nonlinear Operations in LLMs." Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. 2025.

Topics

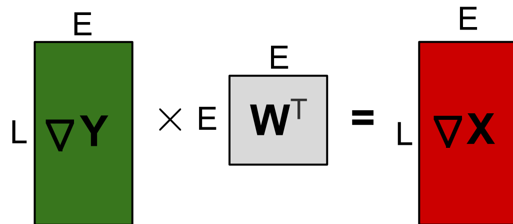
- Hardware design for Operations other than Matrix Multiplication
- Hardware architecture for backward propagation design
- Training Accelerator Design

NYU SAI LAB

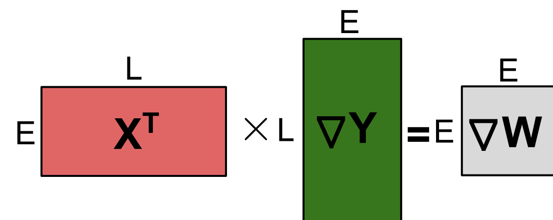
In-place Transposed Matrix Multiplication



Forward propagation



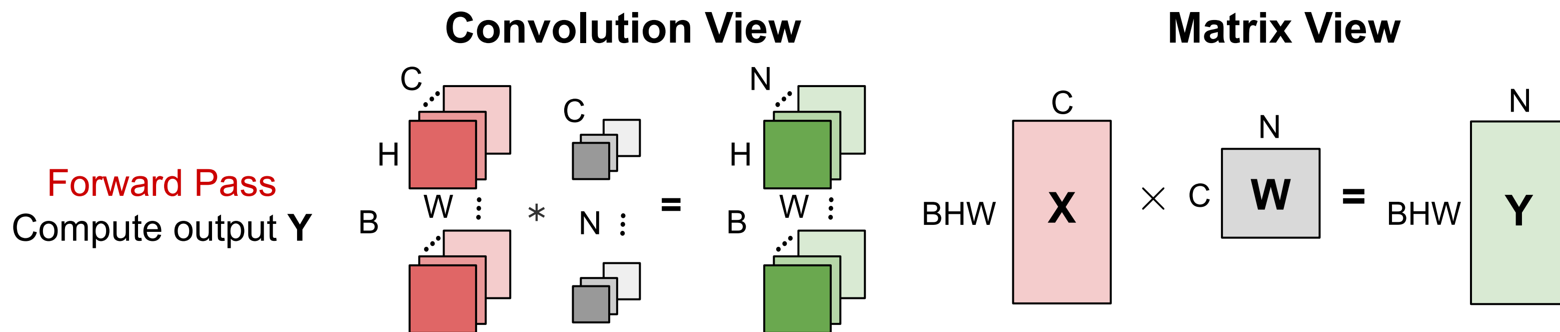
Backward propagation: weight gradient computation



Backward propagation: input gradient computation

NYU SAI LAB

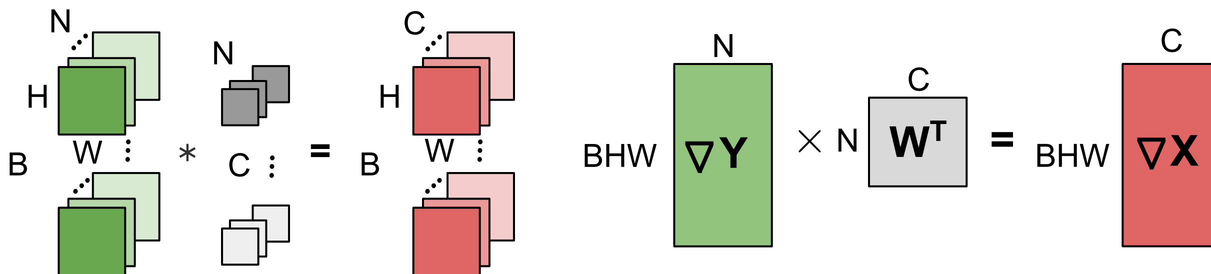
Forward Pass for Convolutional Layer



- Assume a weight kernel size of 1✖1.

Backward Pass for Convolutional Layer

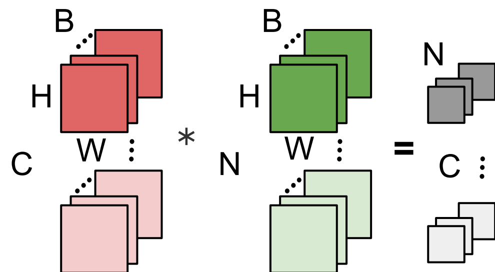
Backward Pass
Compute Activation gradients **∇X**



NYU SAI LAB

Backward Pass for Convolutional Layer

Backward Pass
Compute weight gradients $\nabla \mathbf{W}$



BHW

N

N

$C \; \mathbf{X}^T \times BHW \; \nabla \mathbf{Y} = C \; \nabla \mathbf{W}$

In-place Transposed Matrix Multiplication

- In the training of neural networks, we need to perform transposed matrix multiplication
- Instead of using a separate hardware for matrix transposition, transposed matrix multiplication can be performed using a systolic array.

C, BHW, X × N, C, W = N, BHW, Y

N, BHW, ∇Y × C, N, W^T = C, BHW, ∇X

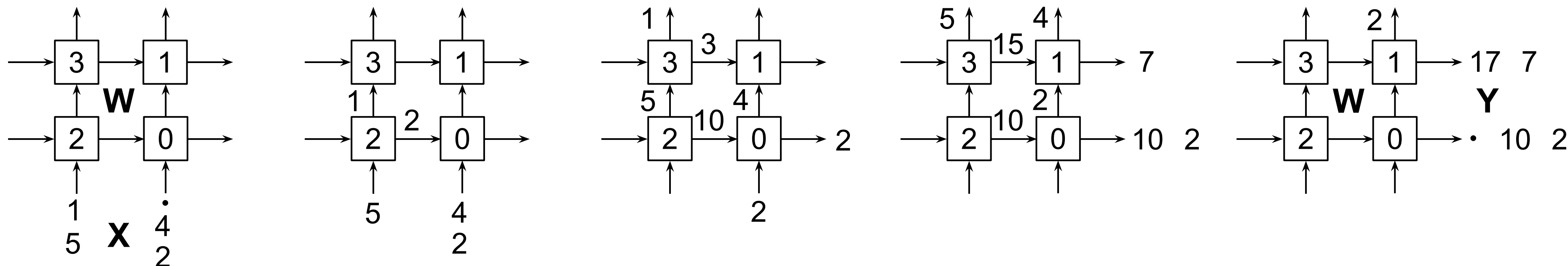
BHW, C, X^T × N, BHW, ∇Y = N, C, ∇W

NYU SAI LAB

In-place Transposed Matrix Multiplication

$$\underbrace{\begin{bmatrix} 1 & 4 \\ 5 & 2 \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix}}_{\mathbf{W}} = \underbrace{\begin{bmatrix} 2 & 7 \\ 10 & 17 \end{bmatrix}}_{\mathbf{Y}}$$

- Weight stationary, input from bottom, accumulation from left to right



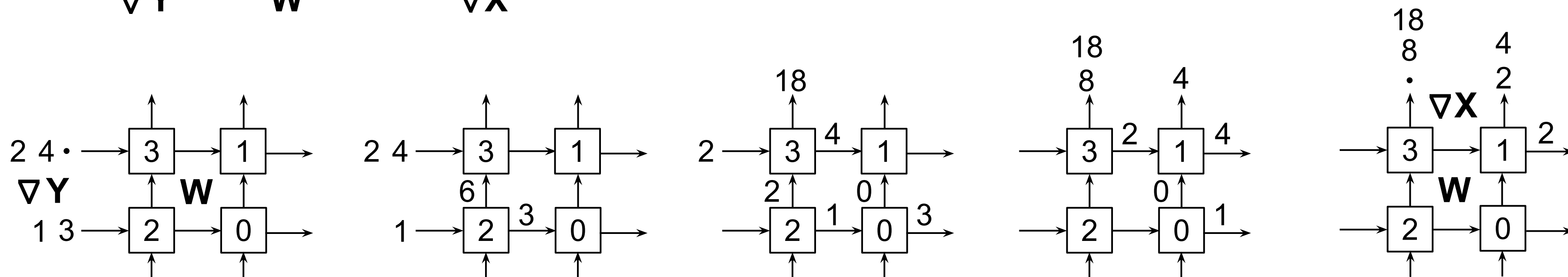
- The weights are preloaded into the systolic array, while the input matrix is streamed into the array from bottom to top.
- The output is produced at the right.

NYU SAI LAB

In-place Transposed Matrix Multiplication

$$\underset{\nabla Y}{\begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix}} \underset{W^T}{\begin{bmatrix} 2 & 0 \\ 3 & 1 \end{bmatrix}} = \underset{\nabla X}{\begin{bmatrix} 18 & 4 \\ 8 & 2 \end{bmatrix}}$$

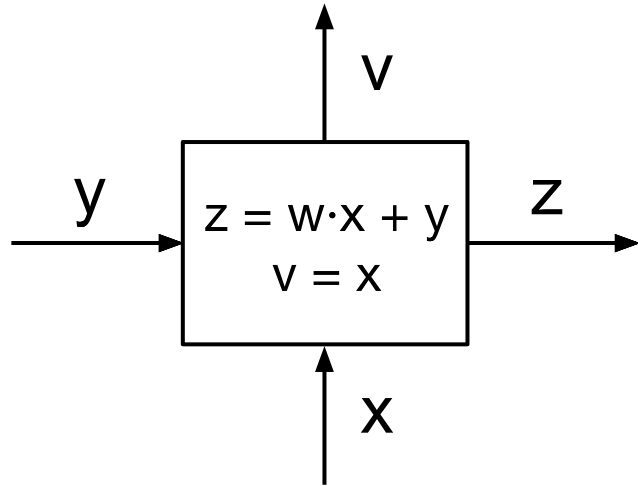
- Weight stationary, input from left, accumulation upwards



- To compute the input gradient, the data gradient is fed into the systolic array from the left, and the output is produced at the top.

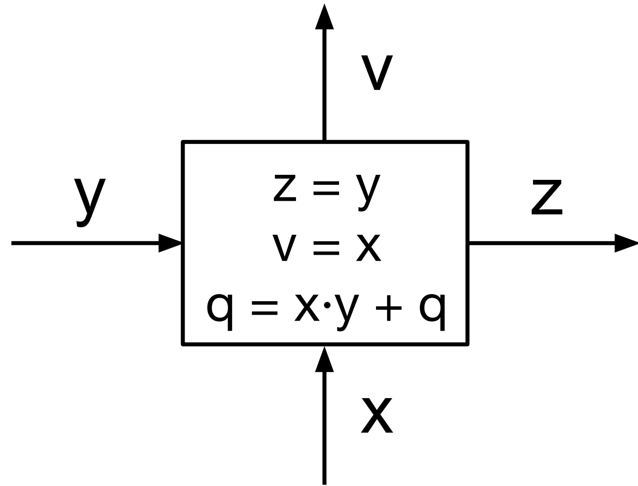
NYU SAI LAB

Systolic Array: Weight-Stationary Version



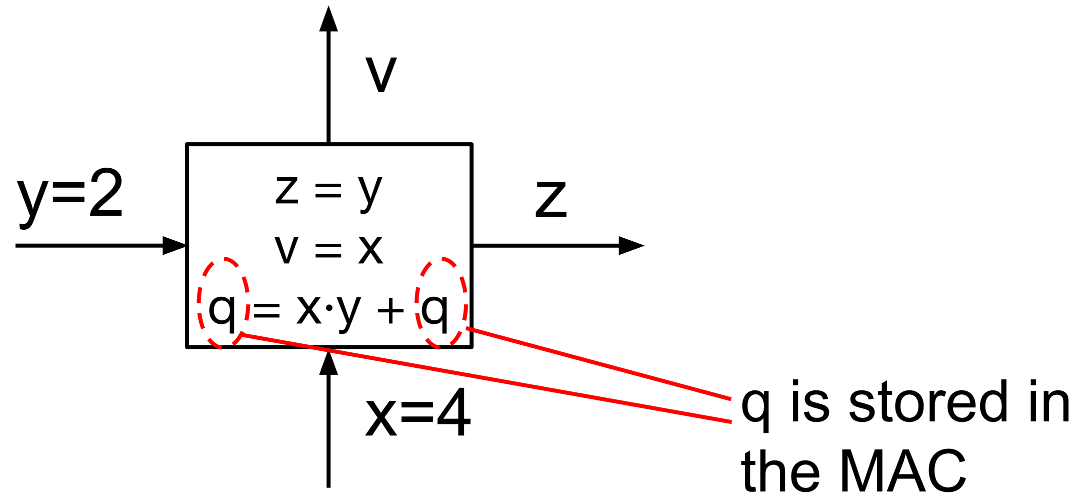
- Takes data (x and y) as input
- w stays in the systolic cell
- Performs a multiply-accumulate operation

Systolic Array: Accumulation-Stationary Version



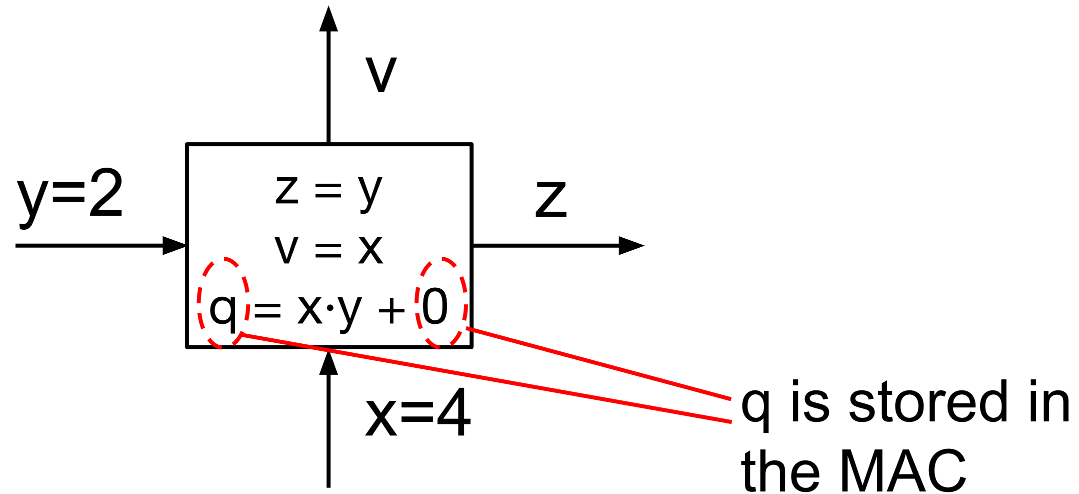
- Takes data (x and y) as input
- Accumulated result q stays in the systolic cell
- Performs a multiply-accumulate operation

Systolic Cell

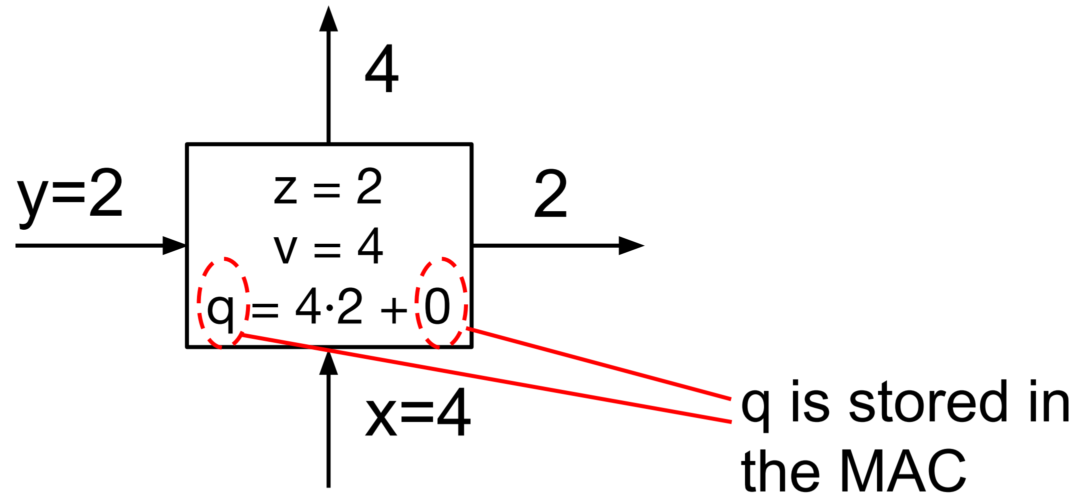


NYU SAI LAB

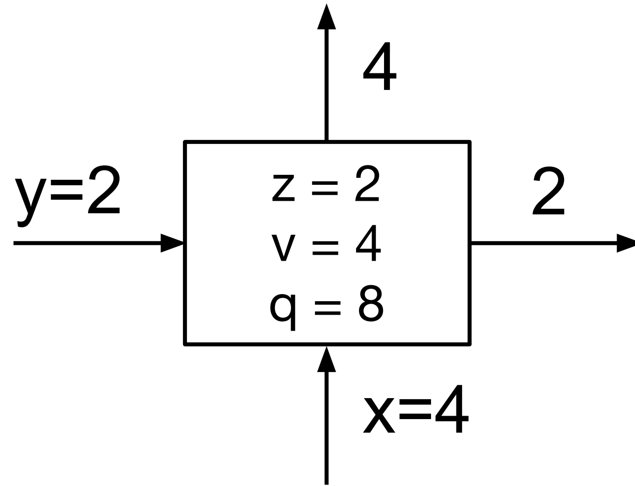
Systolic Cell



Systolic Cell



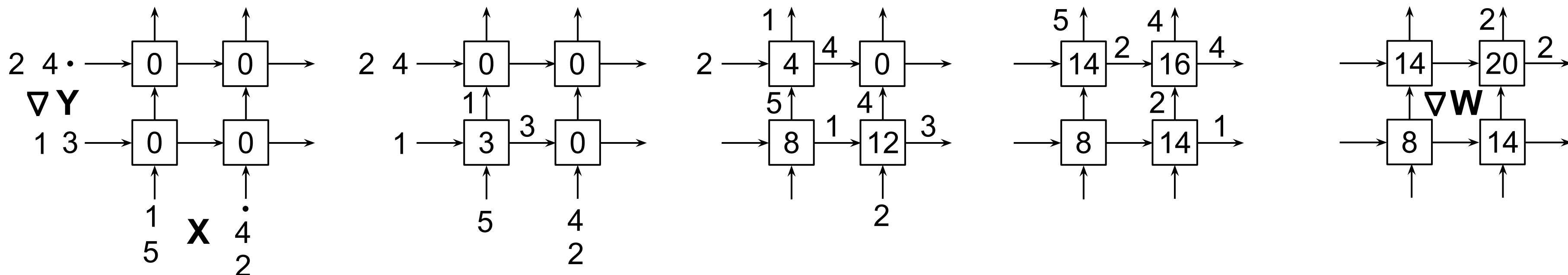
Systolic Cell



In-place Transposed Matrix Multiplication

$$\underset{X^T}{\begin{bmatrix} 1 & 5 \\ 4 & 2 \end{bmatrix}} \underset{\nabla Y}{\begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix}} = \underset{\nabla W}{\begin{bmatrix} 8 & 14 \\ 14 & 20 \end{bmatrix}}$$

- Input from left and bottom, accumulation stationary.

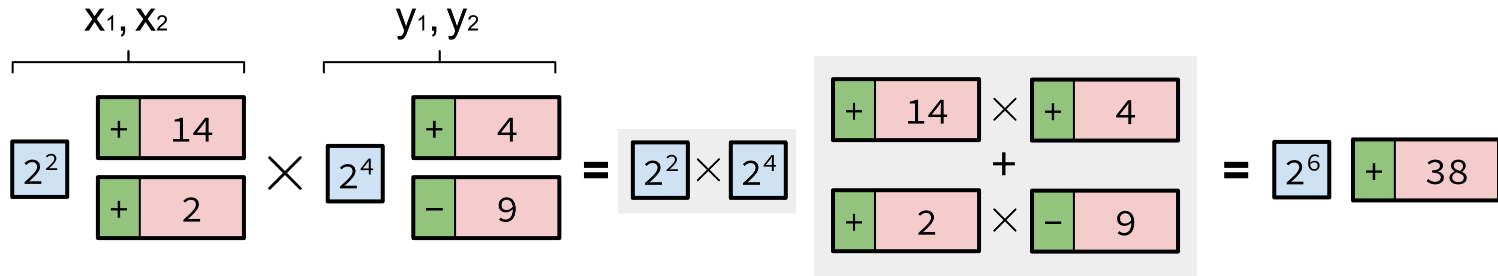


- To compute the weight gradient, the data gradient is input from the left side of the systolic array, while the input activations are fed from the bottom. The resulting weight gradients are accumulated and stored within the systolic cells.

Topics

- Computation during backward propagation
- Hardware architecture for backward propagation design
- Training Accelerator Design

FAST



- The Dot product is performed using BFP format.
- Compare with floating point (FP) format, block floating point (BFP) perform exponent additions and mantissa alignments only at the group level, rather than at the individual elements level.

Fast First, Accurate Second Training

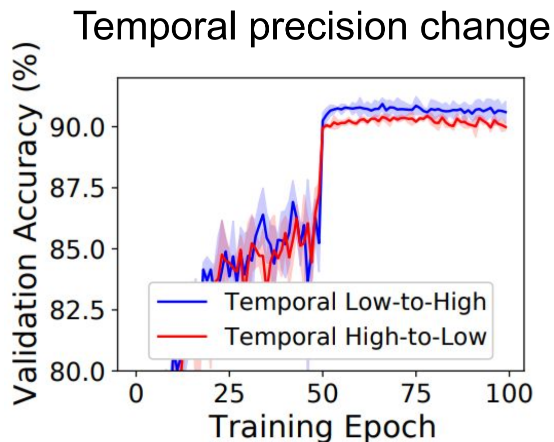
- Previous literature has demonstrated that adding zero-mean Gaussian noise to the weight gradient ∇W can reduce overfitting and improve the convergence of training.

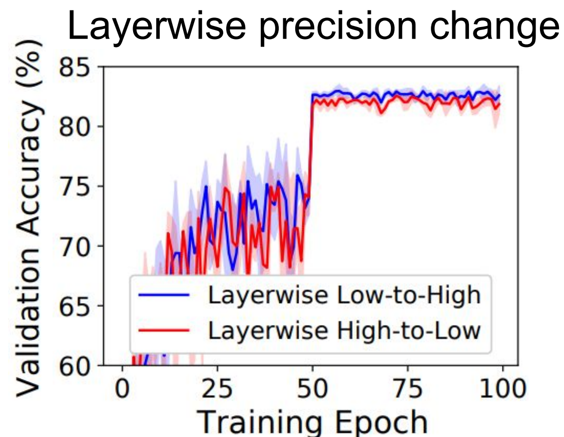
- Decreasing the variance of the noise over iterations achieves better performance than using fixed Gaussian noise throughout training.

- We hypothesize that a similar effect can be achieved by adjusting the BFP precision of weights, activations, and gradients from low to high precisions over training.

Zhang, Sai Qian, Bradley McDanel, and H. T. Kung. "Fast: Dnn training under variable precision block floating point with stochastic rounding." *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022.

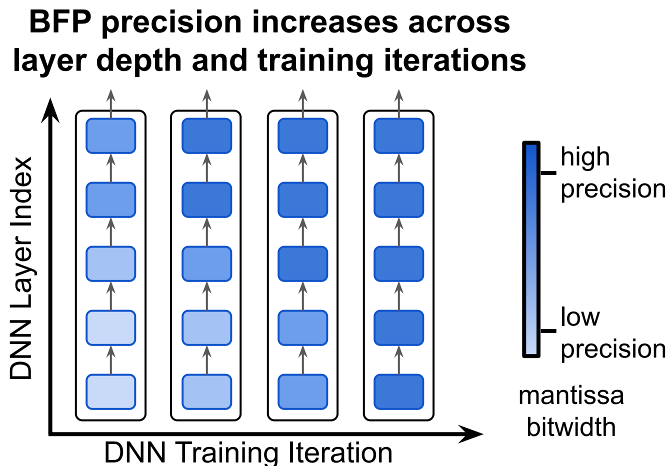
Variable Precision Training





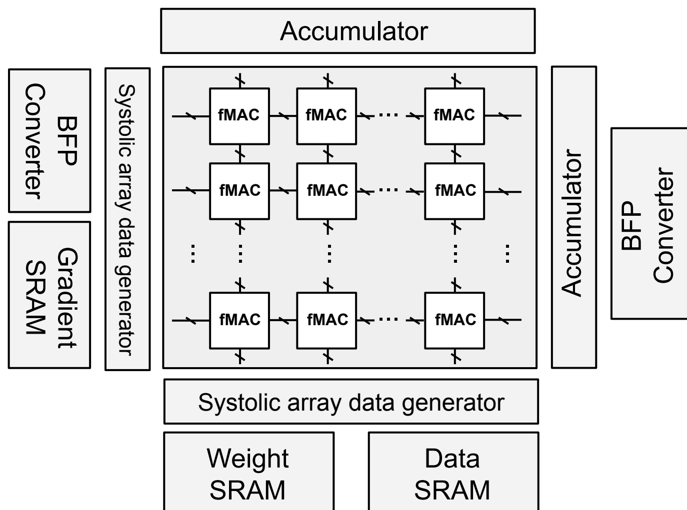
- Under the Temporal High-to-Low scheme, we use FP32 for weights, activations, and gradients for the first part of training, and lower-precision BFP for the second part of training
- Under the Layerwise High-to-Low scheme, we use FP32 precision for the first ten layers, and lower-precision BFP for later layers

Variable Precision Training



- We progressively increase the BFP precision of weights, activations, and gradients along both layer depth and training iterations
- We name this approach **FAST** (Fast First, Accurate Second Training)

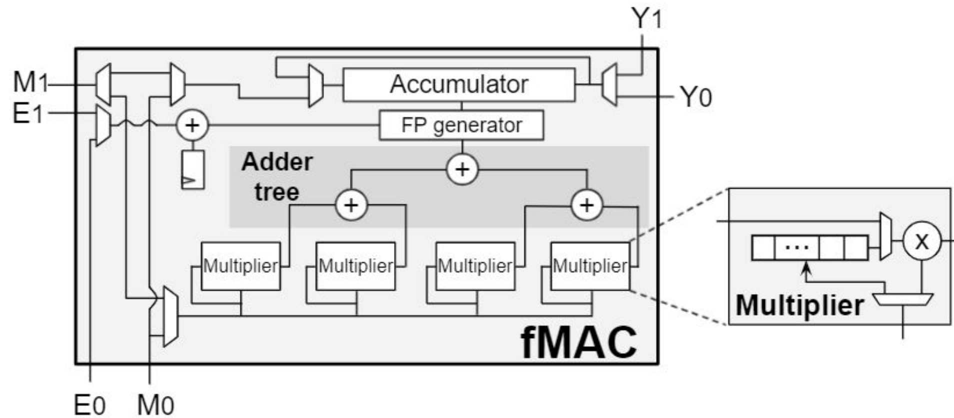
FAST System Design



- Major components of FAST system:
  - Systolic array with FAST multiplier and accumulator (fMAC)
  - BFP converter
  - Accumulator and systolic array input generator
  - Memory subsystem

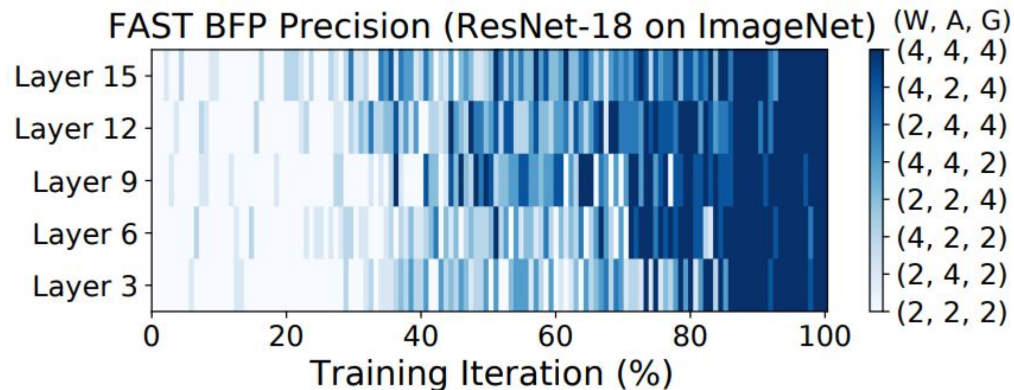
NYU SAI LAB

FAST System Design



- fMAC operates on chunks of BFP mantissas (e.g., 2-bit chunks) to support variable-width mantissas in 2-bit increments

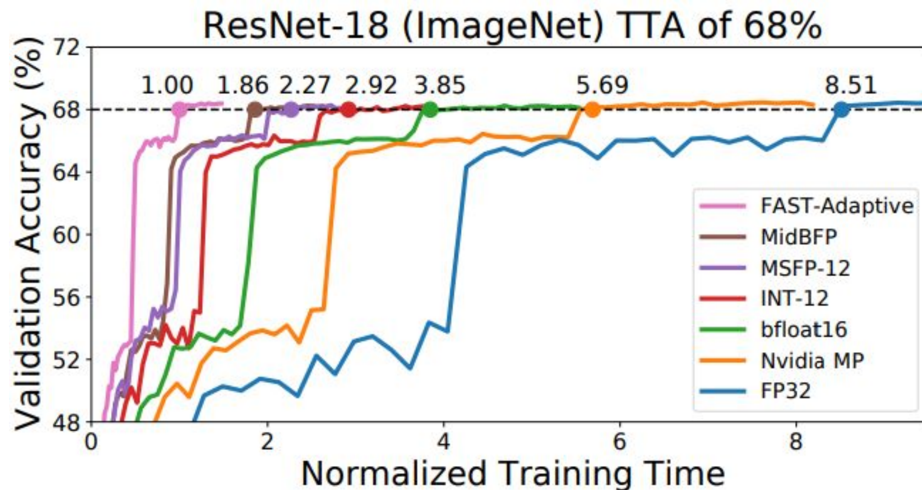
Evaluation



- FAST progressively increases the BFP precision across both layer depth and iterations during the training process

NYU SAI LAB

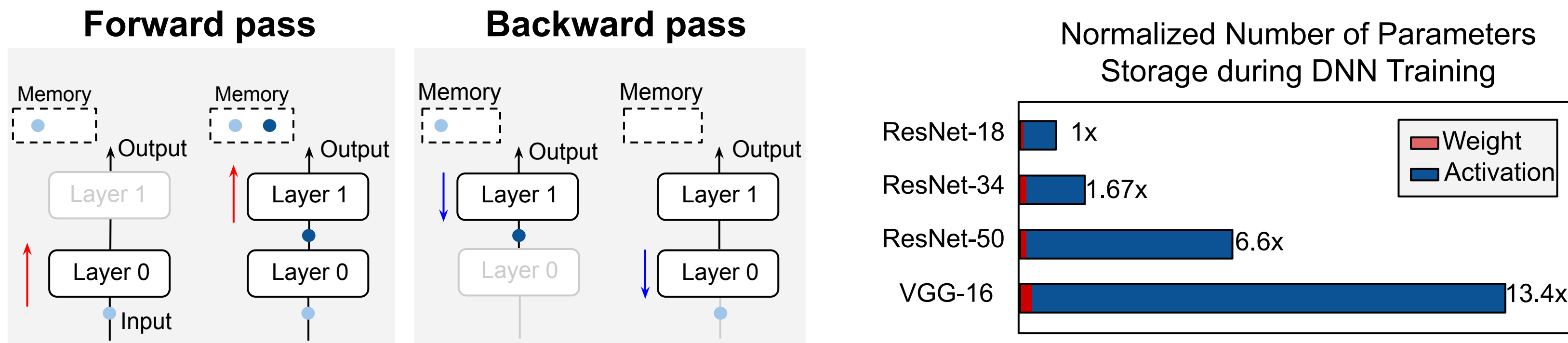
Evaluation



- We use Time-to-Accuracy (TTA) as the evaluation metric to compare different approaches
- Our FAST approach achieves the lowest TTA across all the numeric formats

NYU SAI LAB

Memory Efficient Neural Network Training



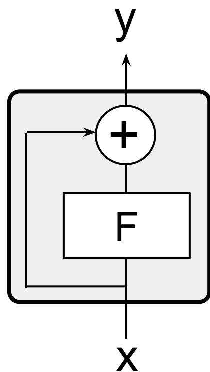
- The memory footprint grows proportional with the layer depth.
- On top of this, small edge devices typically have limited on-chip storage, leading to frequent and costly accesses to off-chip memories.

NYU SAI LAB

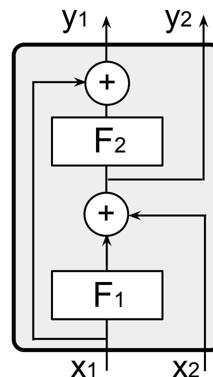
Zhang, Sai Qian, et al. "CAMEL: Co-Designing AI Models and eDRAMs for Efficient On-Device Learning." *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024.

Memory Efficient Neural Network Training

Residual Architecture



Reversible Architecture



Forward pass:

$y_2 = F_1(x_1) + x_2$

$y_1 = F_2(y_2) + x_1$

Backward Pass:

$x_1 = y_1 - F_2(y_2)$

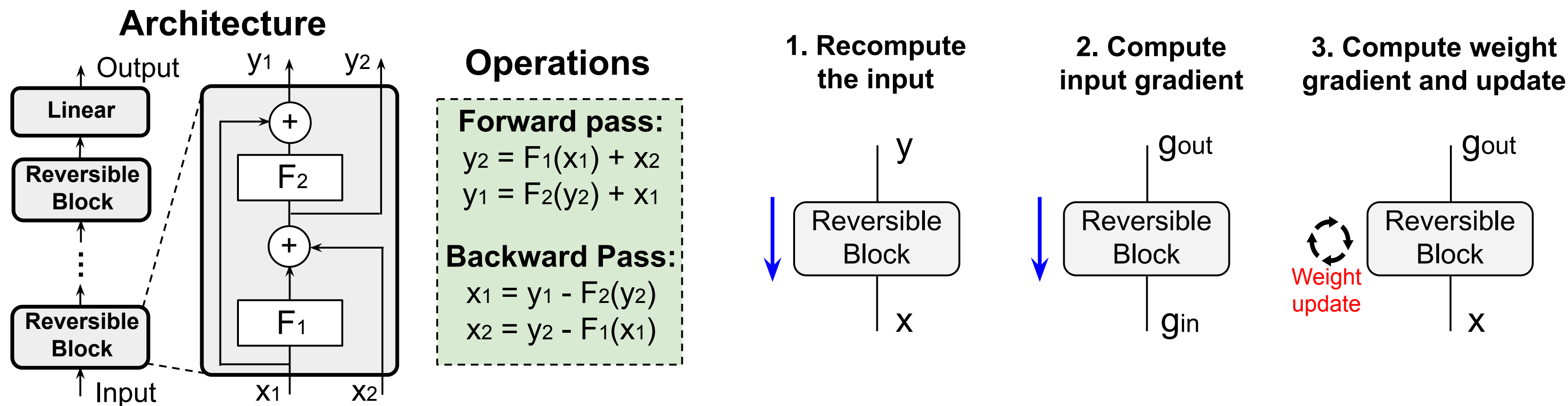
$x_2 = y_2 - F_1(x_1)$

- A reversible residual network (RevNet) is a variant of the canonical residual neural network (ResNet).

NYU SAI LAB

Zhang, Sai Qian, et al. "CAMEL: Co-Designing AI Models and eDRAMs for Efficient On-Device Learning." *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024.

Memory Efficient Neural Network Training

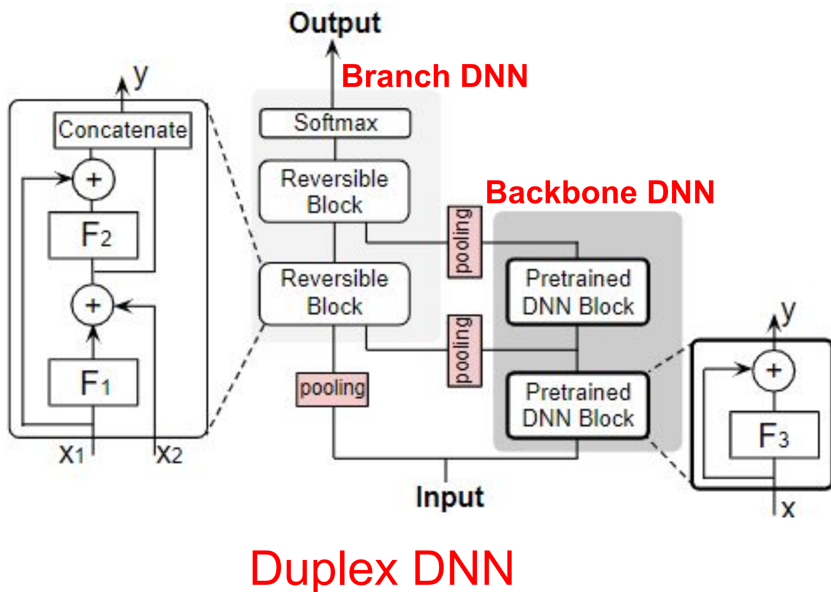


- The reversible architecture enables the backward pass computations to be performed without the need to store the input activations.
- Given the output y, the input activations are first recomputed. Afterwards, the input and weight gradients are computed with standard backward pass operations.

NYU SAI LAB

Zhang, Sai Qian, et al. "CAMEL: Co-Designing AI Models and eDRAMs for Efficient On-Device Learning." *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024.

Memory Efficient Neural Network Training



- This approach in turn imposes higher compute demands.
- We propose to judiciously train a subset of the model parameters to minimize training.
- The backbone DNN is frozen during the backward pass of the DNN.
- The normalization layers are removed from the branch DNN to facilitate the training process.

NYU SAI LAB

Zhang, Sai Qian, et al. "CAMEL: Co-Designing AI Models and eDRAMs for Efficient On-Device Learning." *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024.

Presentation

- Tender: Accelerating Large Language Models via Tensor Decomposition and Runtime Requantization
  - Alex
- DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine Learning
  - Hivansh, Vincent, Ali