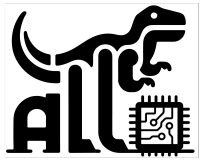
ACT,TAIDL
[MICRO'25]

MINISA
[ISPASS'26]

FEATHER
[ISCA'24]



Allo
[PLDI'24]

Reconfigurable AI Computing (RAIC)

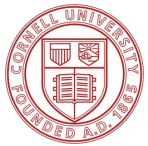
A generation flow for Compiler->Architecture->Deployment

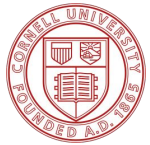
Jianming Tong*, Niansong Zhang*, Devansh Jain*, Charith Mendis, Zhiru Zhang, Tushar Krishna






Georgia Tech



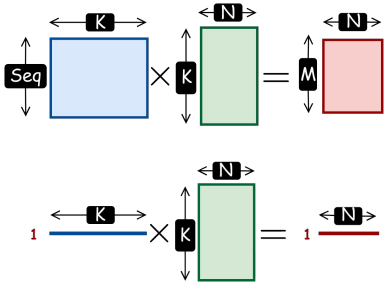



Georgia Tech

Contact: reconfigureai@googlegroups.com

Outline

Workload



Diverse Workload

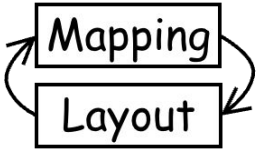
Architecture

SetIVNLayout
SetWVNLayout
SetOVNLayout
ExecuteMapping
ExecuteStreaming

(Mapping, Layout) Co-Switch

FEATHER [ISCA'24]

Compilation



Compiler Generation

ACT, TAIDL [MICRO'25]

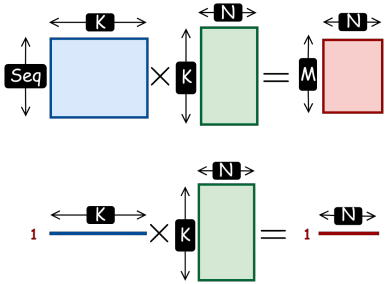
Deployment

Hardware Generation

Allo [PLDI'24]

Outline

Workload



Diverse Workload

Architecture

SetIVNLayout
SetWVNLayout
SetOVNLayout
ExecuteMapping
ExecuteStreaming

(Mapping, Layout) Co-Switch

FEATHER [ISCA'24]

Compilation

Compiler Generation

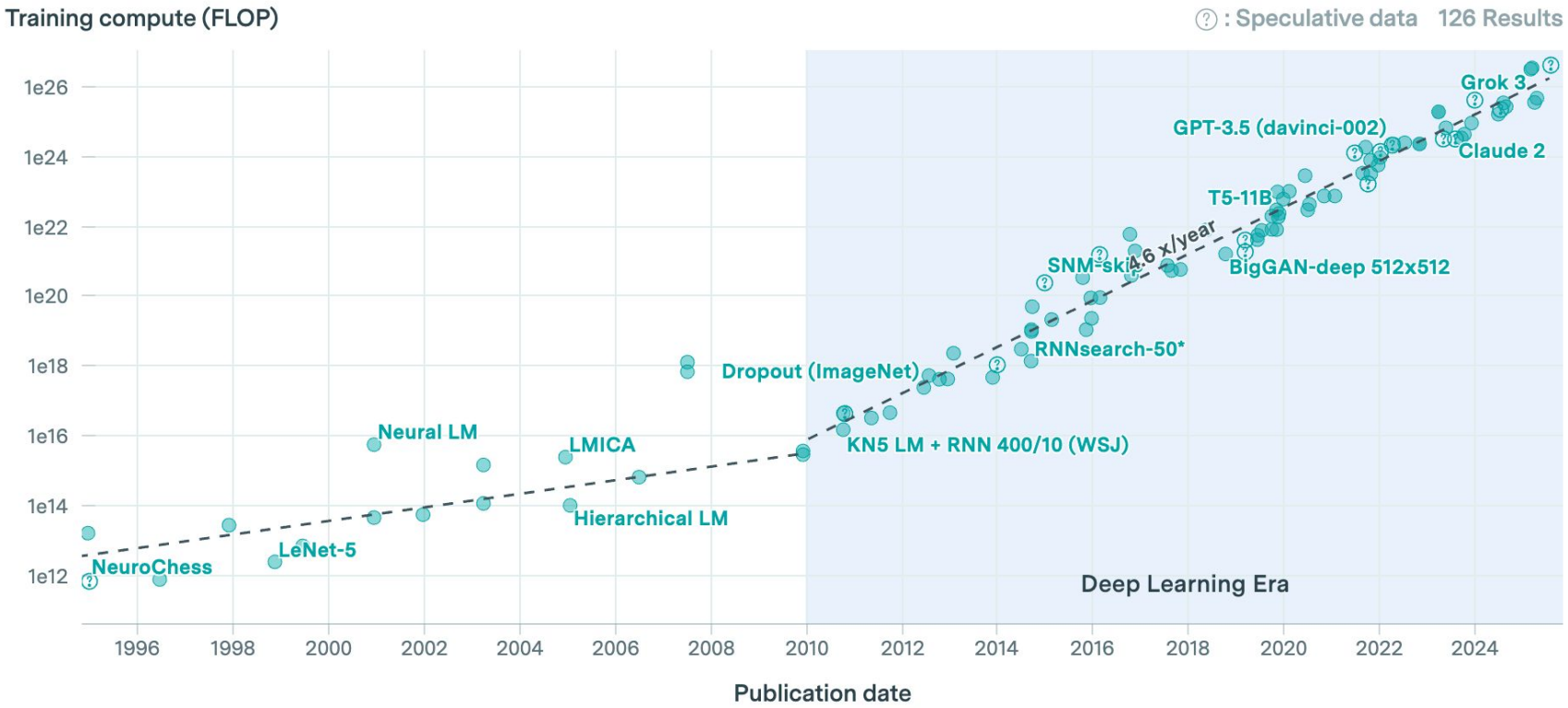
ACT, TAIDL [MICRO'25]

Deployment

Hardware Generation

Allo [PLDI'24]

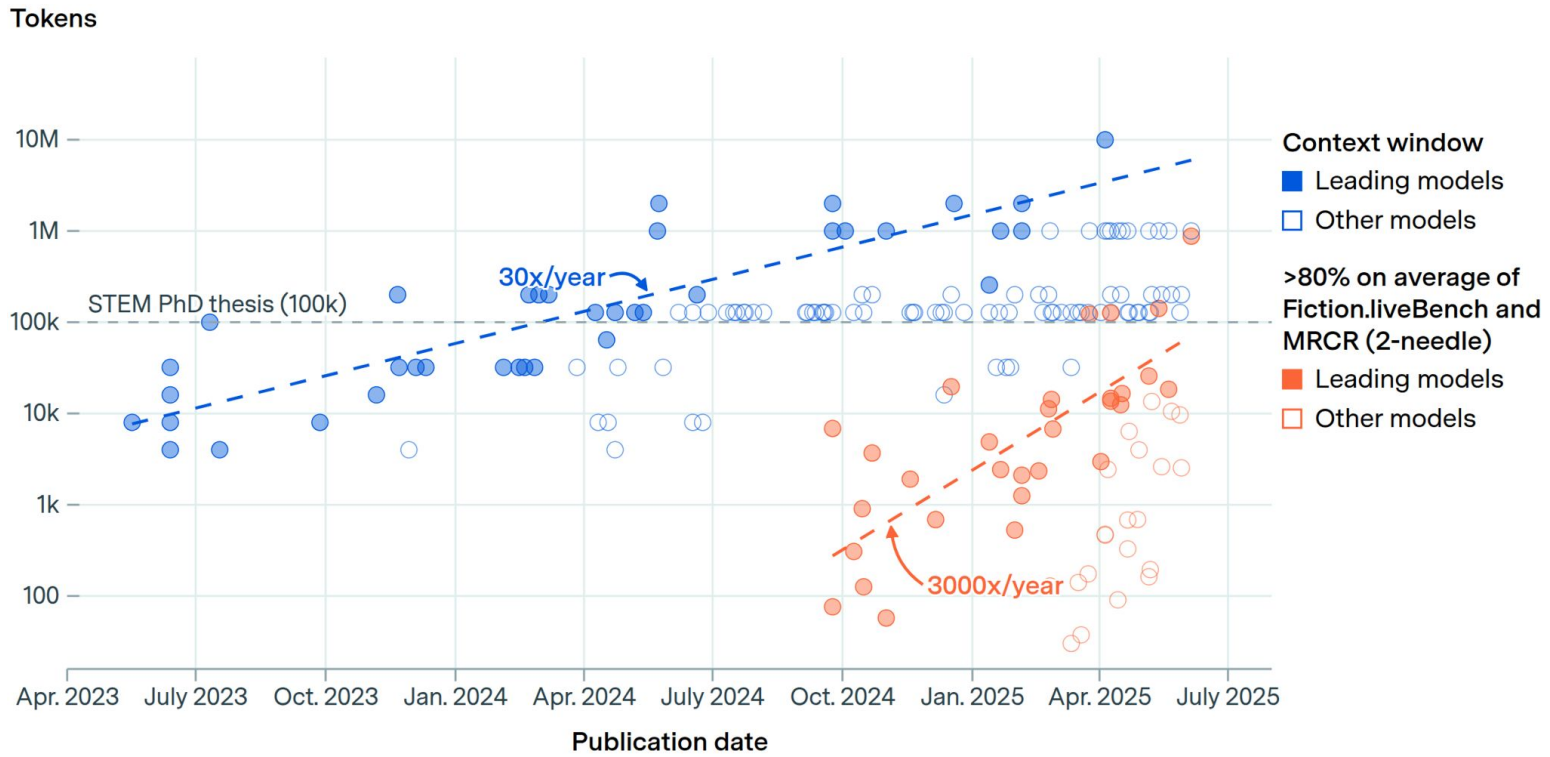
Trend: Training Cost for Frontier Model Grows



To support AI Growth, we need 5x Computing Power Per Year!

Data from Epoch.AI

Trend: Inference Cost Growth



30x Longer Context per year -> Inference Cost!

Data from Epoch.AI

Background: Demand and Supply Analysis

What We Need

Training compute

Confident

5 x/year

Training compute for frontier language models has been growing by 5x since 2020.

90% confidence interval: 4x to 6x.

LLM context windows

Confident

30 x/year

LLM context windows have grown by 30x since 2023.

90% confidence interval: 10x to 50x.

What We Have

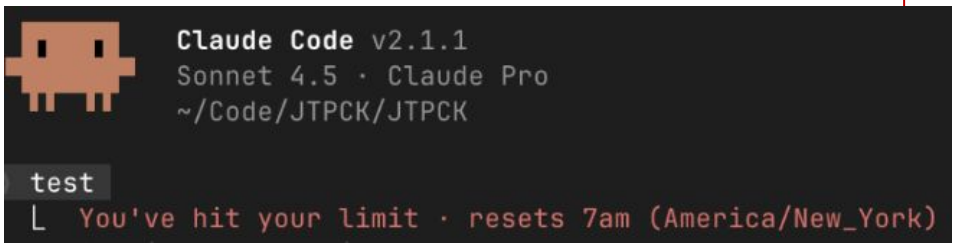
Compute stock growth

Likely

2.3 x/year

The total computing power of the stock of NVIDIA chips is growing by 2.3x.

90% confidence interval: 2.2x to 2.5x.

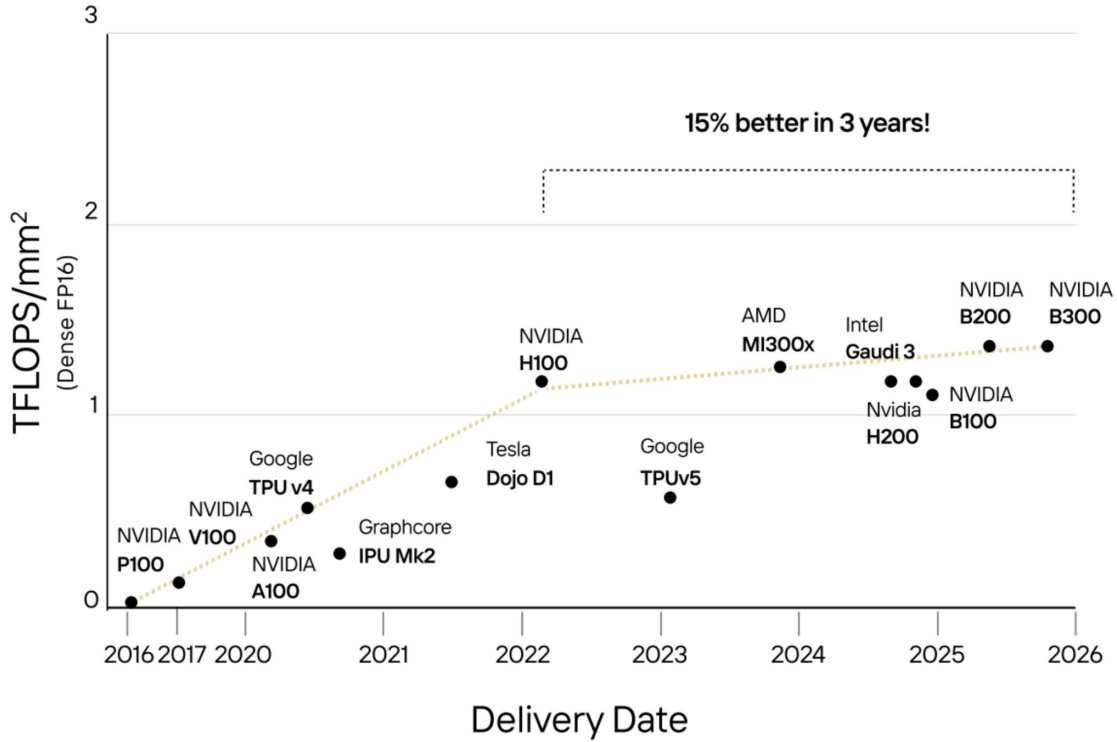


Quota Limits everywhere!

We are short of computing power

Data from Epoch.AI

Background: Throughput/Area Growth Slows Down

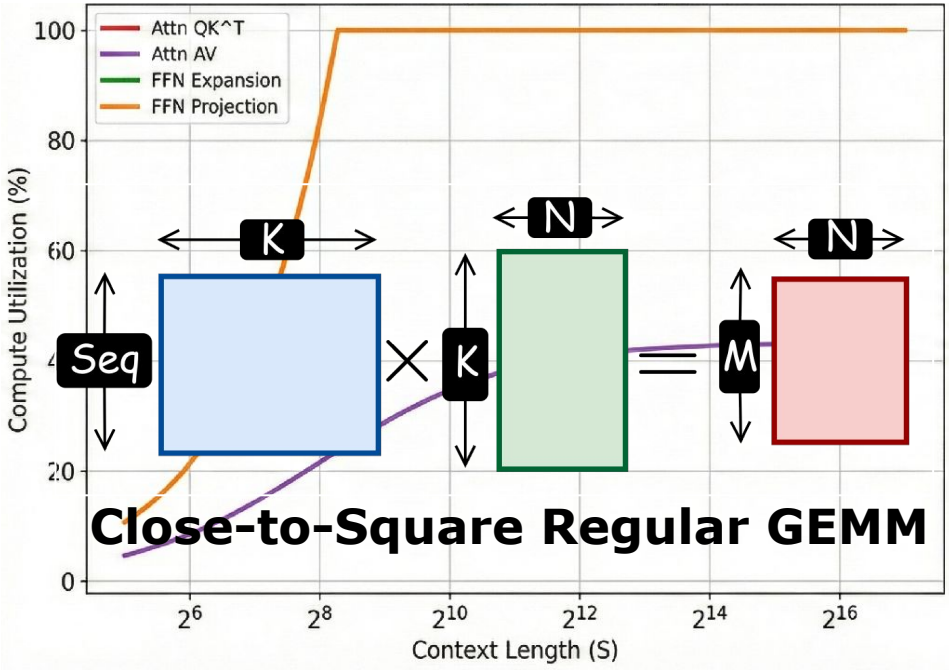


Throughput density Grows Slow -- Need Clever Architecture for Use them

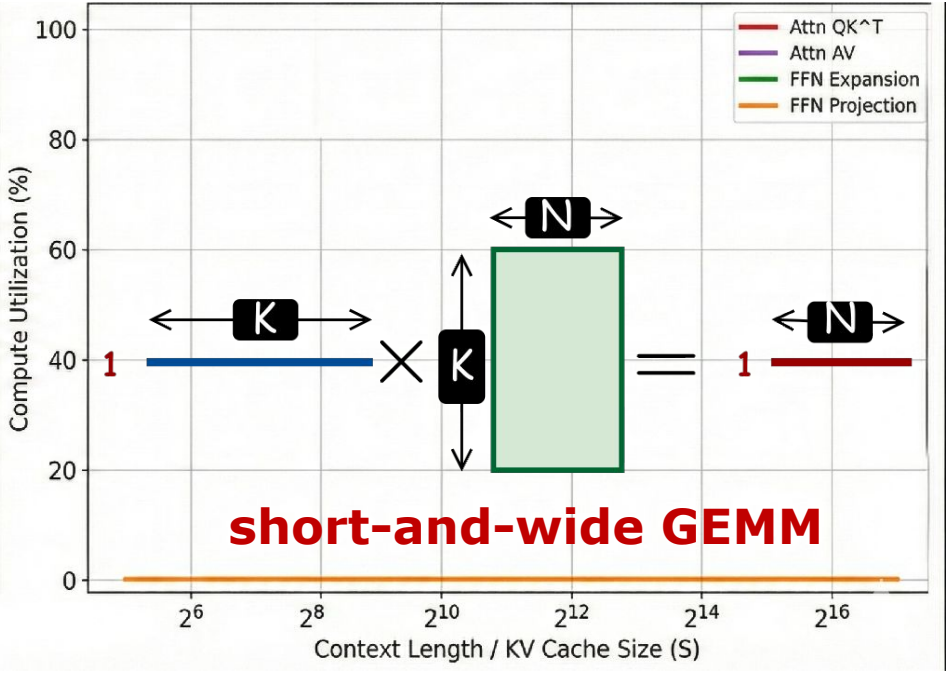
Data from Epoch.AI

Challenge: Under-Utilization in Single-batch LLM Inference

Prefilling (H100) **Large Context**



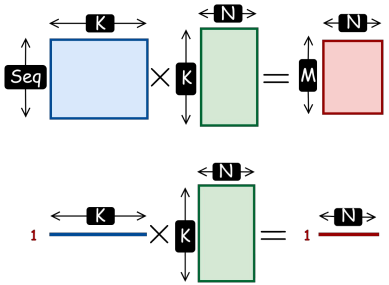
Decoding (H100) **~1%**



Reason of low utilization:
Diverse GEMM NOT divide Fixed-shape MatMul Engine

Outline

Workload



Diverse Workload

Architecture

SetIVNLayout
SetWVNLayout
SetOVNLayout
ExecuteMapping
ExecuteStreaming

(Mapping, Layout) Co-Switch

FEATHER [ISCA'24]

Compilation

Compiler Generation

ACT, TAIDL [MICRO'25]

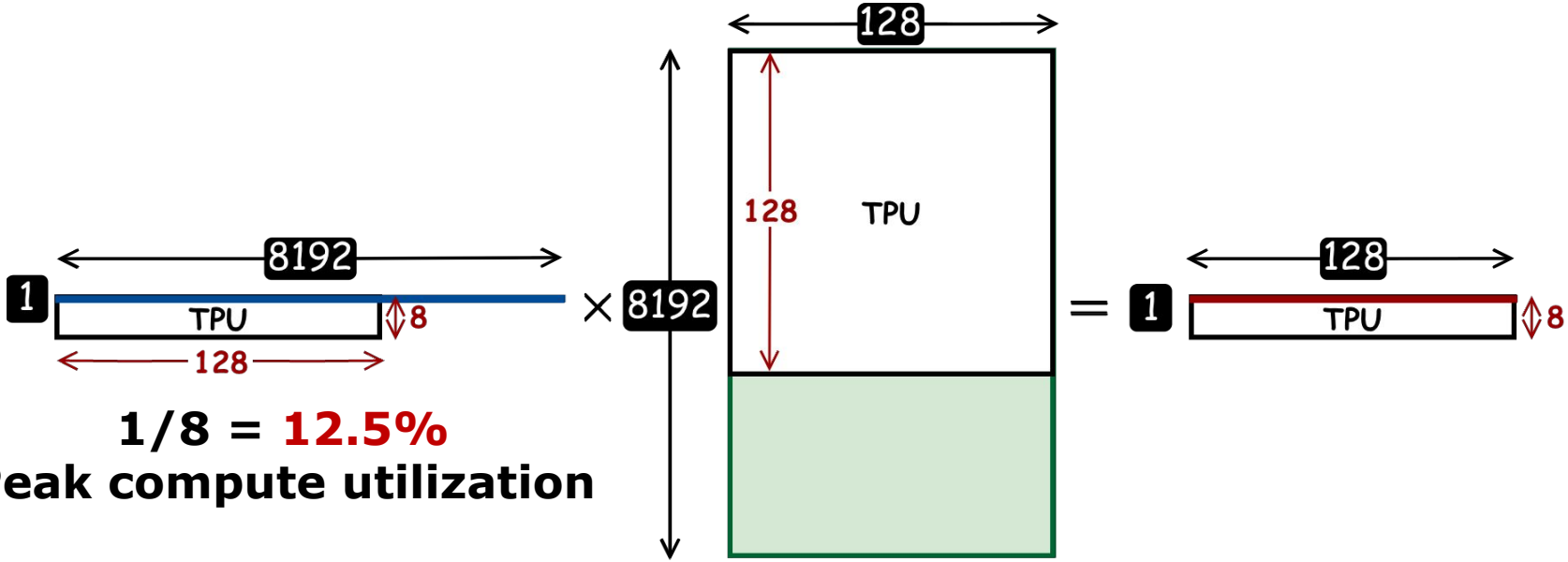
Deployment

Hardware Generation

Allo [PLDI'24]

Example: Fixed-Functional MatMul Engine Limits Utilization

Workload: Single-batch Decoding – 1x8192x128

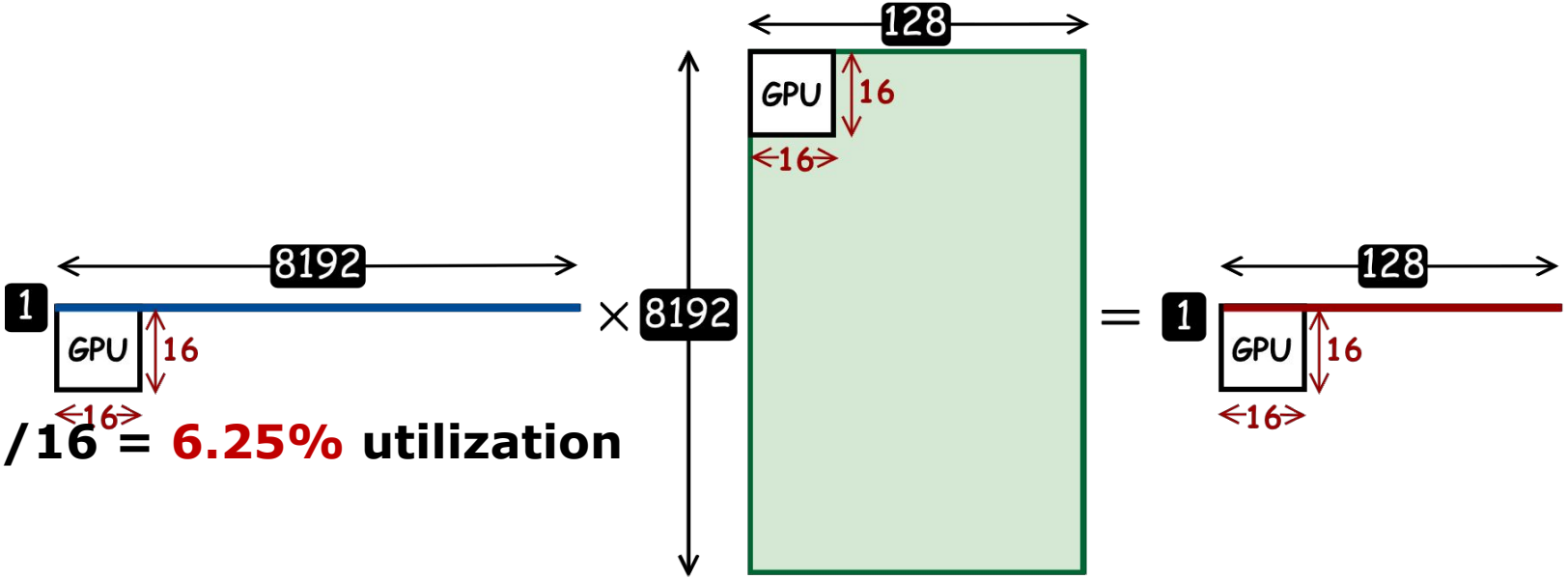


1/8 = 12.5%
Peak compute utilization

TPU: 8x128x128 (BF16) – 12.5% peak compute utilization

Example: Fixed-Functional MatMul Engine Limits Utilization

Workload: Single-batch Attention Score in Decoding – 1x8192x128



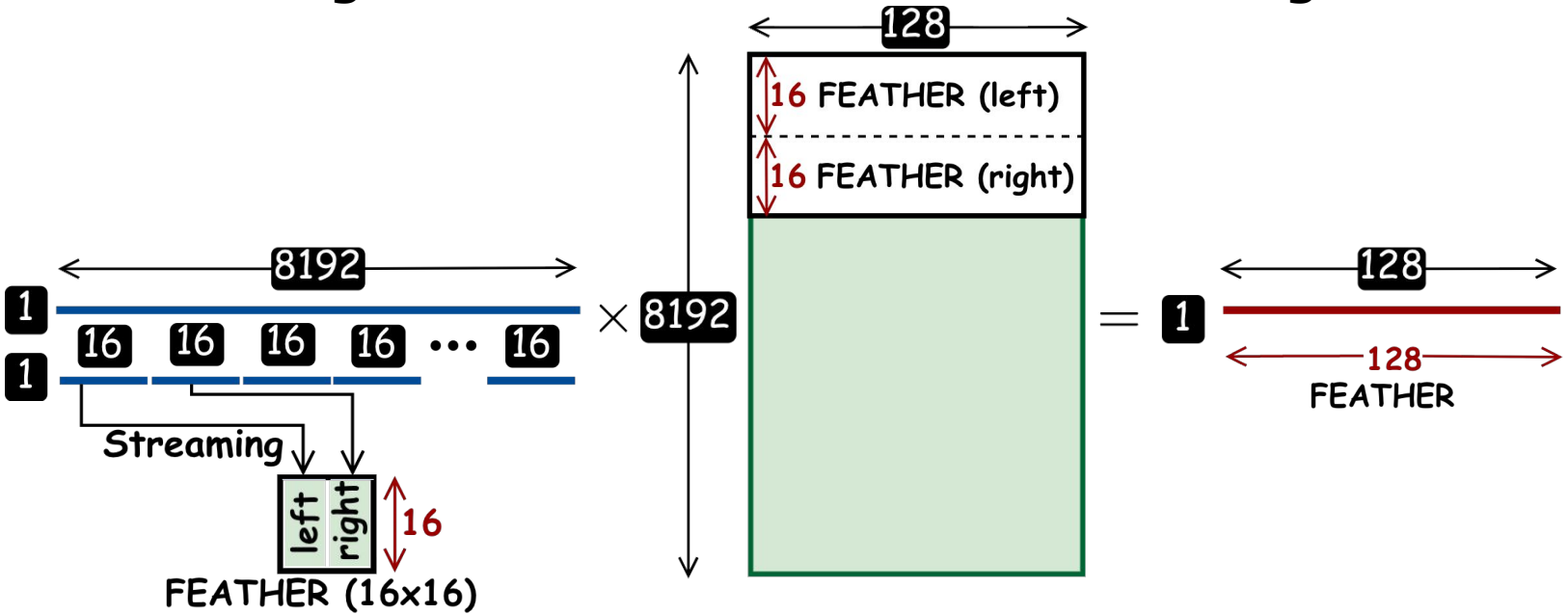
1/16 = 6.25% utilization

TPU: 8x128x128 (BF16) – 12.5% peak compute utilization

GPU: 16x16x16 (BF16) – 6.25% peak compute utilization

Solution: Compute + Memory Reconfiguration

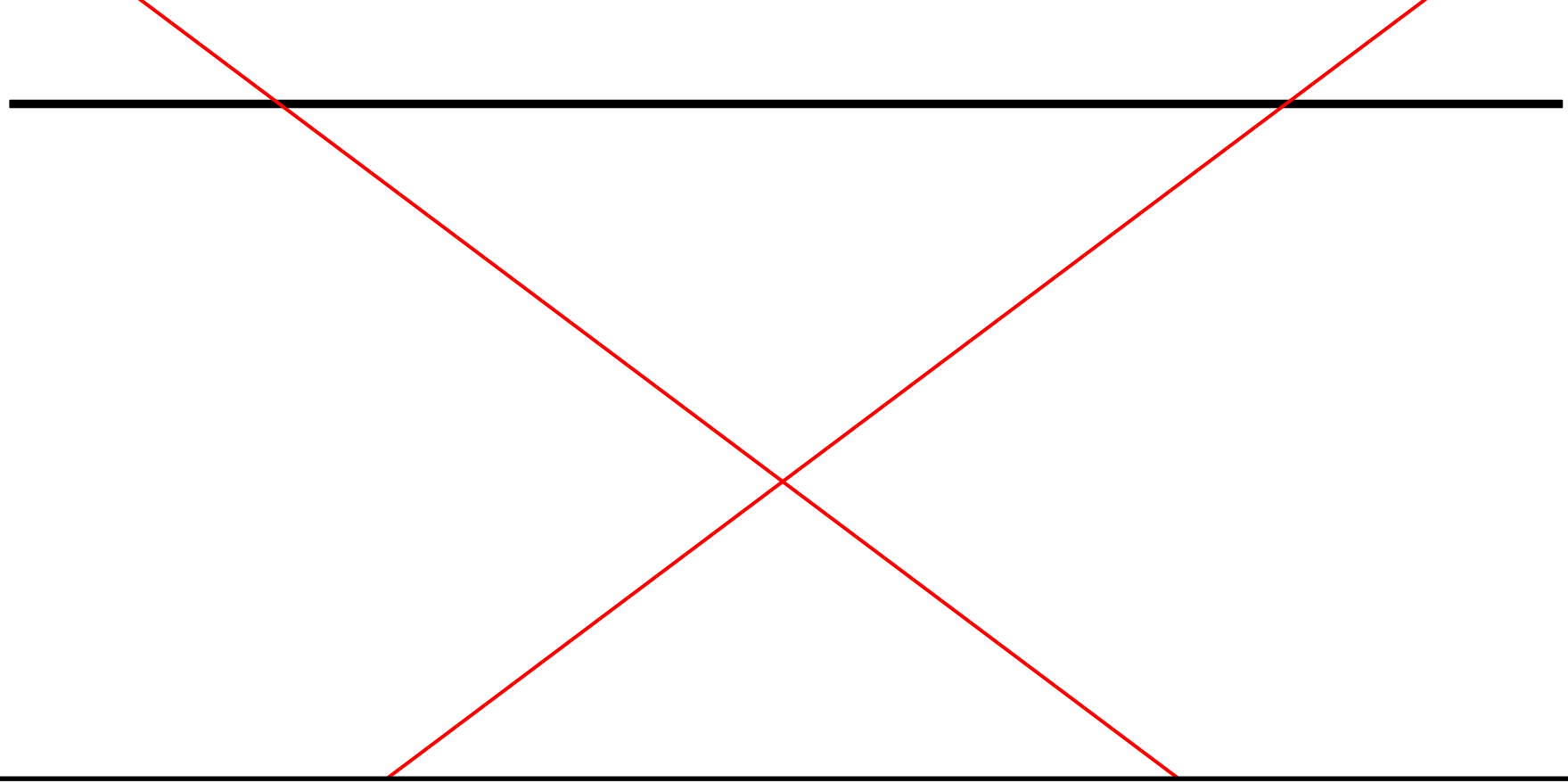
Workload: Single-batch Attention Score in Decoding – 1x8192x128



TPU: 8x128x128 (BF16) – 12.5% peak compute utilization

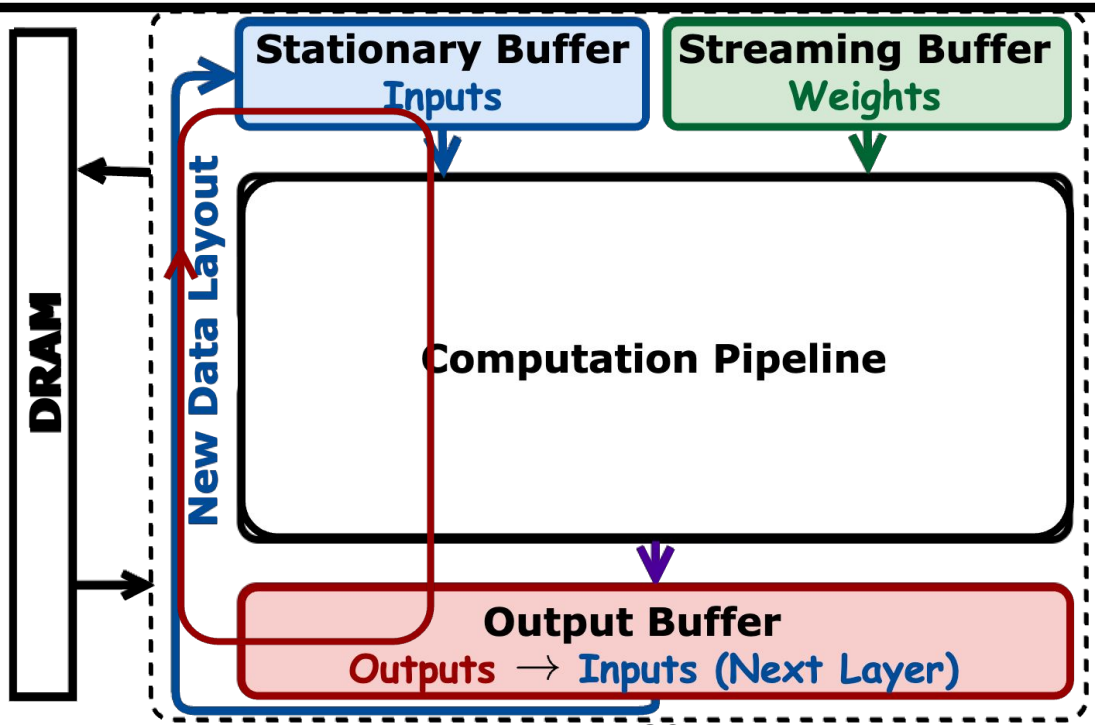
GPU: 16x16x16 (BF16) – 6.25% peak compute utilization

FEATHER (16x16 PEs): 1x32x128, 8x16x32 … (Flexible)



Reconfiguration Improves Compute Utilization
Higher Performance and Lower Latency!

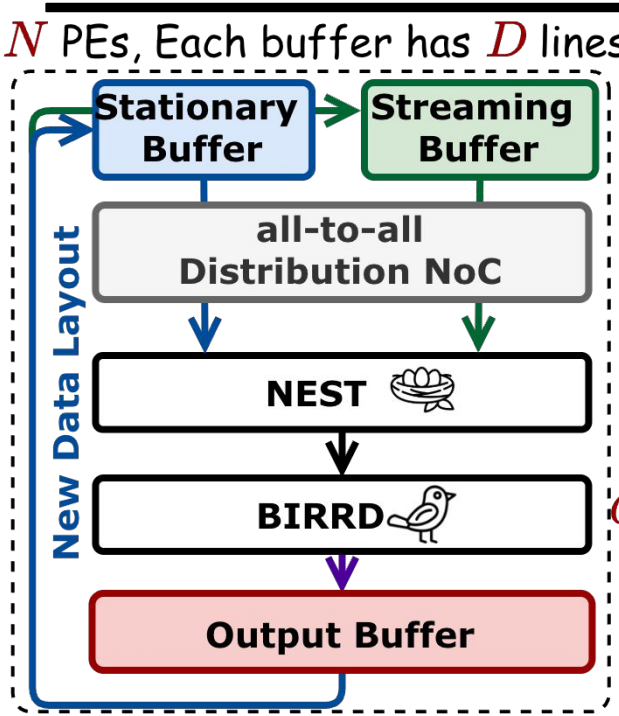
FEATHER Overview - Co-Switch (Mapping, Layout)



[Tong et al, ISCA'24]

- Read from StaB→Change Layout in Compute→Write to StaB (**New Layout**)
- **Co-Switch (Mapping, Layout)** per iteration of compute pipelining
- NEST – Flexible Mapping; BIRRD – Flexible Memory Layout

ISA: Minimal while Expressive Instruction Set Arch.



Reconfiguration Choices	ISA
$O(N \cdot D)$ Layout Choices	SetIVNLayout, SetWVNLayout
$O\left(\sqrt{N}^2\right)$ Reordering Choices	
$O\left(N^{1.5}\right)$ Mapping Choices	ExecuteMapping, ExecuteStreaming
$O\left(\sqrt{N}\log\sqrt{N}\right)$ Reorder-In-Reduction Choices	
$O(N \cdot D)$ Layout Choices	SetOVNLayout

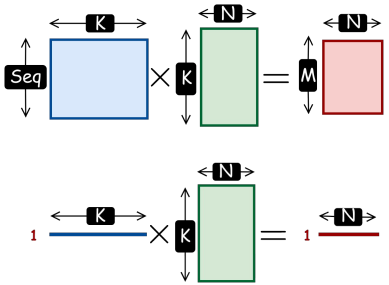
Program Abstraction: Virtual Neuron (VN)

Encapsulate Mapping and Layout choices into 5 ISAs

FEATHER ISA: necessary as needed and as minimal as possible

Outline

Workload



Diverse Workload

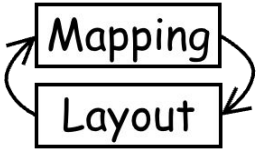
Architecture

SetIVNLayout
SetWVNLayout
SetOVNLayout
ExecuteMapping
ExecuteStreaming

(Mapping, Layout) Co-Switch

FEATHER [ISCA'24]

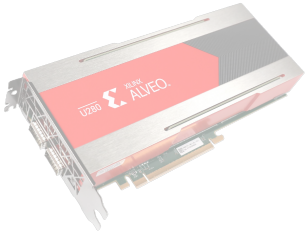
Compilation



Compiler Generation

ACT, TAIDL [MICRO'25]

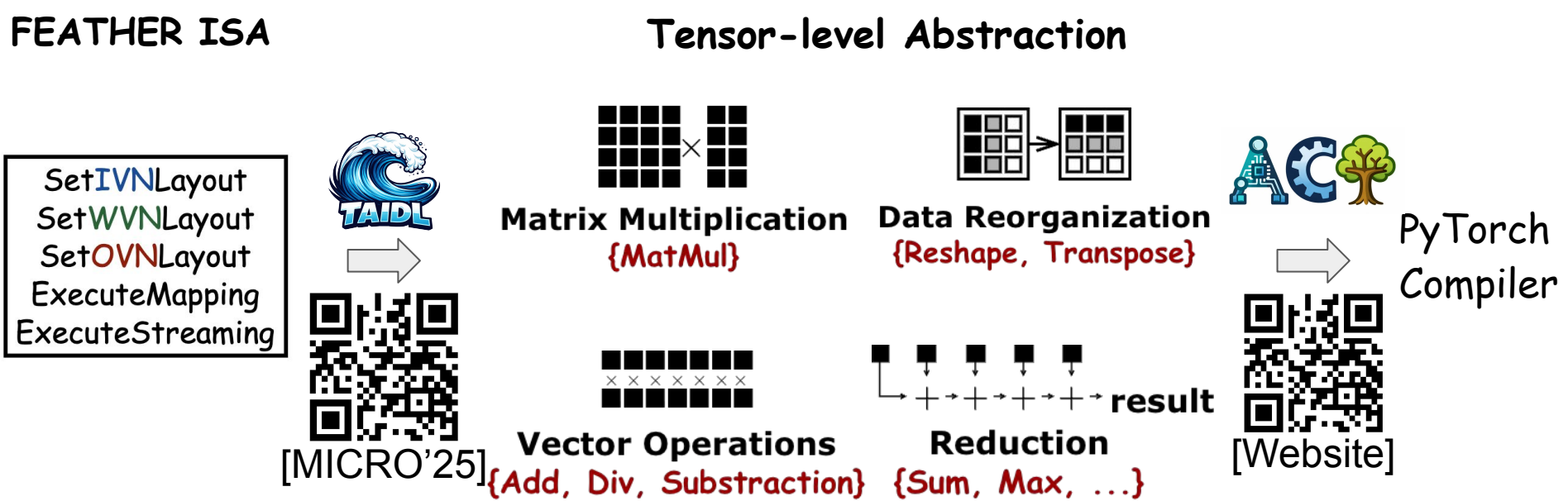
Deployment



Hardware Generation

Allo [PLDI'24]

Compiler Generation: Modeling Reconfigurable ISA in TAIDL



Principle: Model FEATHER ISA using tensor-level abstraction

Compilation Flow: ACT generated PyTorch Compiler for FEATHER

PyTorch Model

PyTorch Compiler

Mapping

Layout

(Mapping, Layout) Co-Search

Instruction Set Architecture

SetIVNLayout
SetWVNLayout
SetOVNLayout
ExecuteMapping
ExecuteStreaming

Configure Layout

Configure Mapping

Reconfigurable Arch.

Change (Mapping, Layout) Per Layer

PyTorch Model

Generated PyTorch Compiler

FEATHER's ISA
[MINISA @ ISPASS'26]

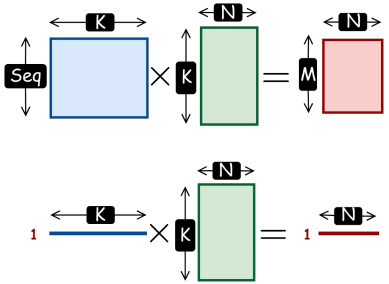
FEATHER
[ISCA'24]

Generated by Allo
[PLDI'24]

Prune Layout Configuration Space using E-Graph and TAIDL Specification

Outline

Workload → Architecture → Compilation → Deployment



SetIVNLayout
SetWVNLayout
SetOVNLayout
ExecuteMapping
ExecuteStreaming

Mapping
Layout

Diverse Workload

(Mapping, Layout) Co-Switch

FEATHER [ISCA'24]

Compiler Generation

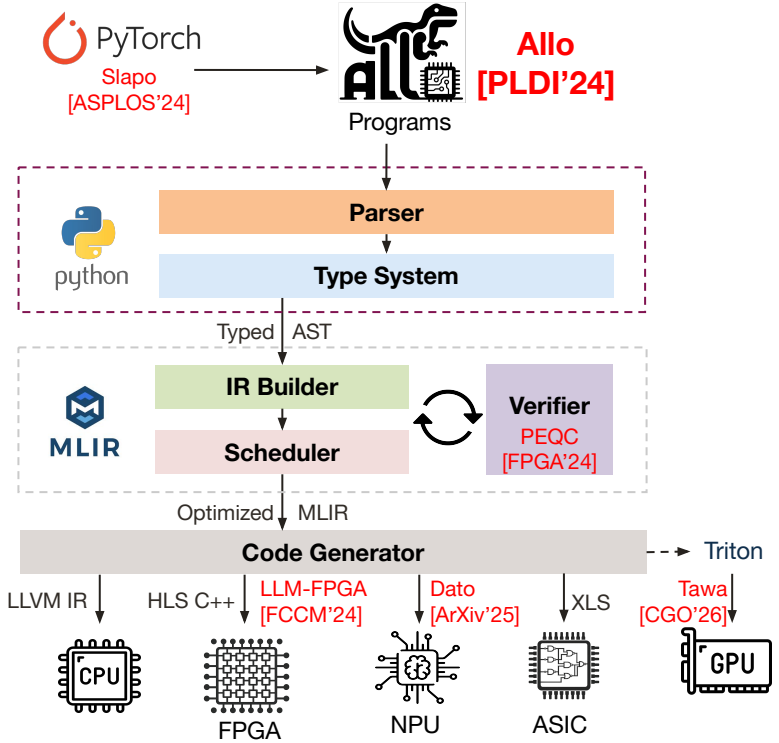
ACT, TAIDL [MICRO'25]

Hardware Generation

Allo [PLDI'24]

Allo: A Unifying Framework

Latest Allo introduces **SPMW**, **a grid-based abstraction** that explicitly captures regularity, while unifying accelerator design and programming



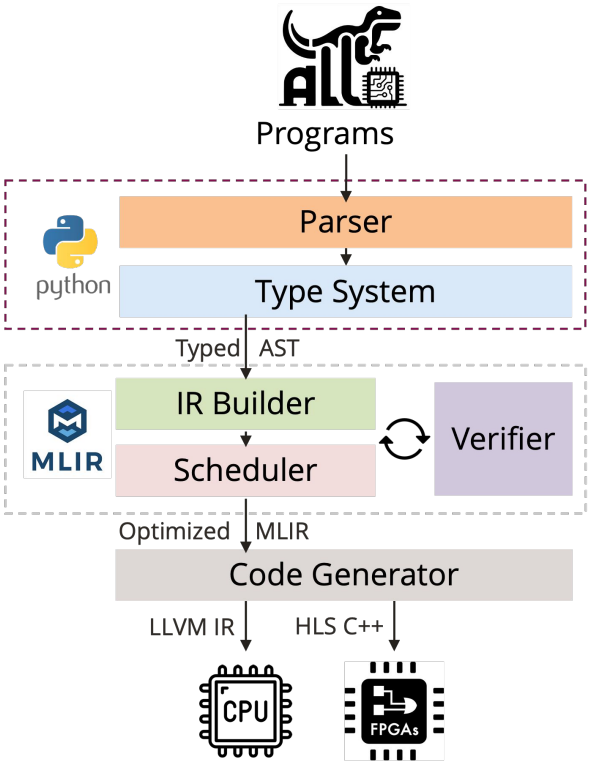
cornell-zhang/allo

Allo Accelerator Design and Programming Framework (PLDI'24)

https://github.com/cornell-zhang/allo

38 Contributors · 60 Issues · 29 Discussions · 388 Stars · 70 Forks

Allo Accelerator Design Language (ADL) and Compiler



- Python-embedded Accelerator Design Language

Algorithm Specification

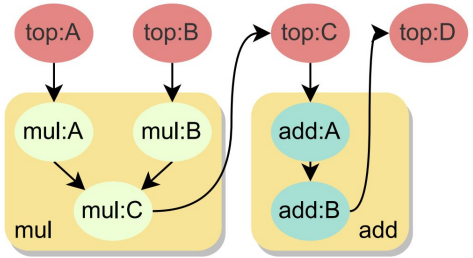
```
def matmul(A: float32[M, K],
           B: float32[K, N])
           -> float32[M, N]:
  C: float32[M, N] = 0.0
  for i, j, k in allo.grid(M, N, K):
    C[i, j] += A[i, k] * B[k, j]
  return C
```

- **Verifiable** decoupled customizations

Compute Cust.
Memory Cust.
Comm. Cust.

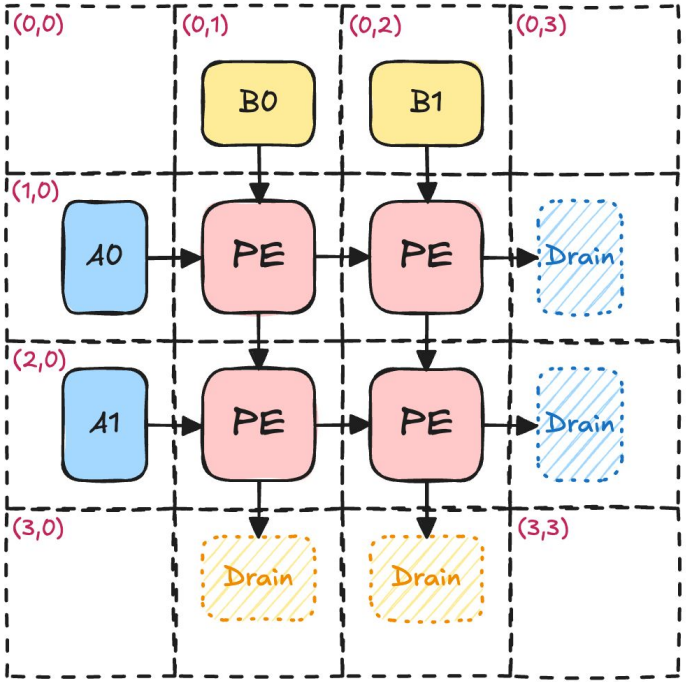
```
s = allo.customize(matmul)
s.reorder("k", "j")
s.buffer_at(s.C, axis="i")
s.pipeline("j")
```

- Type-safe kernel composition

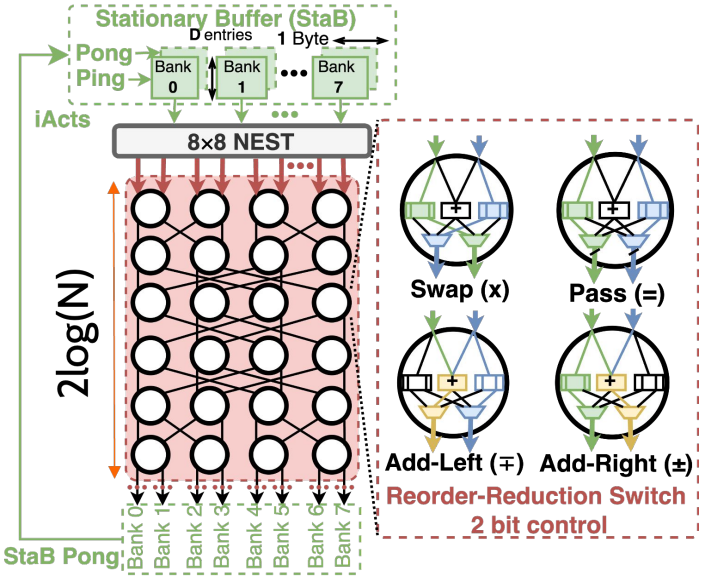


Allo SPMW: Fast, Flexible, and Modular DSA Design Abstraction

- Allo introduces SPMW: **S**ingle **P**rogram **M**ultiple (Connected) **W**ork Units
- **Key insight:** Accelerators are typically regular and spatially scalable by design, fitting naturally into a grid
- **Explicit, flexible connectivity and modularity**: Simpler typing, verification, and physical design



Implementing FEATHER with Allo



FEATHER in RTL: 2491 lines of Verilog (16x16 array)

```
@allo.kernel(grid=[P0, P1])
def BIRRD():
  i, j = df.get_pid()
  inst = inst_input[i, j].get()
  mix = lambda L, R: [(L, R), (L, L+R), (L+R, R), (R, L)][inst]
  for _ in range(AH):
    if (i == 0):
      L, R = connection[0, 2*j].get(), connection[0, 2*j+1].get()
      a, b = mix(L, R)
      connection[i+1, reverse_bits(2*j, 2)].put(a)
      connection[i+1, reverse_bits(2*j+1, 2)].put(b)
    elif (i == P0-1):
      L, R = connection[P0-1, 2*j].get(), connection[P0-1, 2*j+1].get()
      a, b = mix(L, R)
      connection[P0, 2*j].put(a)
      connection[P0, 2*j+1].put(b)
    else:
      L, R = connection[i, 2*j].get(), connection[i, 2*j+1].get()
      a, b = mix(L, R)
      w = min(LOG2_AW, 2+i, 2*LOG2_AW-i)
      connection[i+1, reverse_bits(2*j, w)].put(a)
      connection[i+1, reverse_bits(2*j+1, w)].put(b)
```

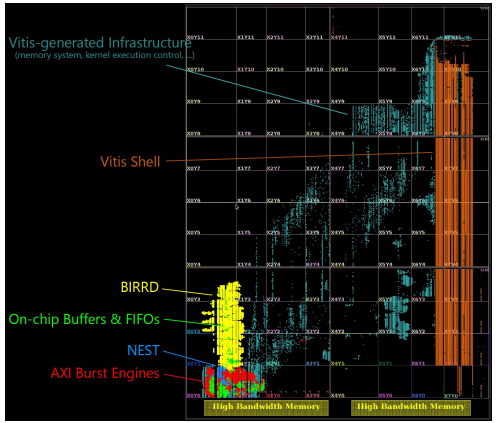
Single Program, Multiple connected Work Units (**SPMW**), over a 2D grid

Type-safe specification of interconnections among work units

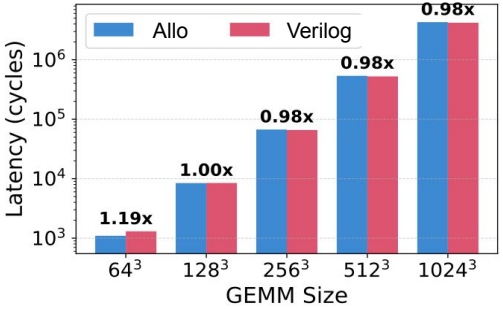
FEATHER in Allo: 220 lines of Python (same performance, lower area)

Implementing FEATHER with Allo

- Allo provides **SPMW, a grid-based abstraction** that explicitly captures regularity
- Line-of-code comparison
  - Verilog RTL: 2941 lines
  - Allo: **220** lines
- Comparable performance
  - 16x16 PE array
  - FEATHER RTL: 3024 cycles
  - Allo-FEATHER: **2970** cycles



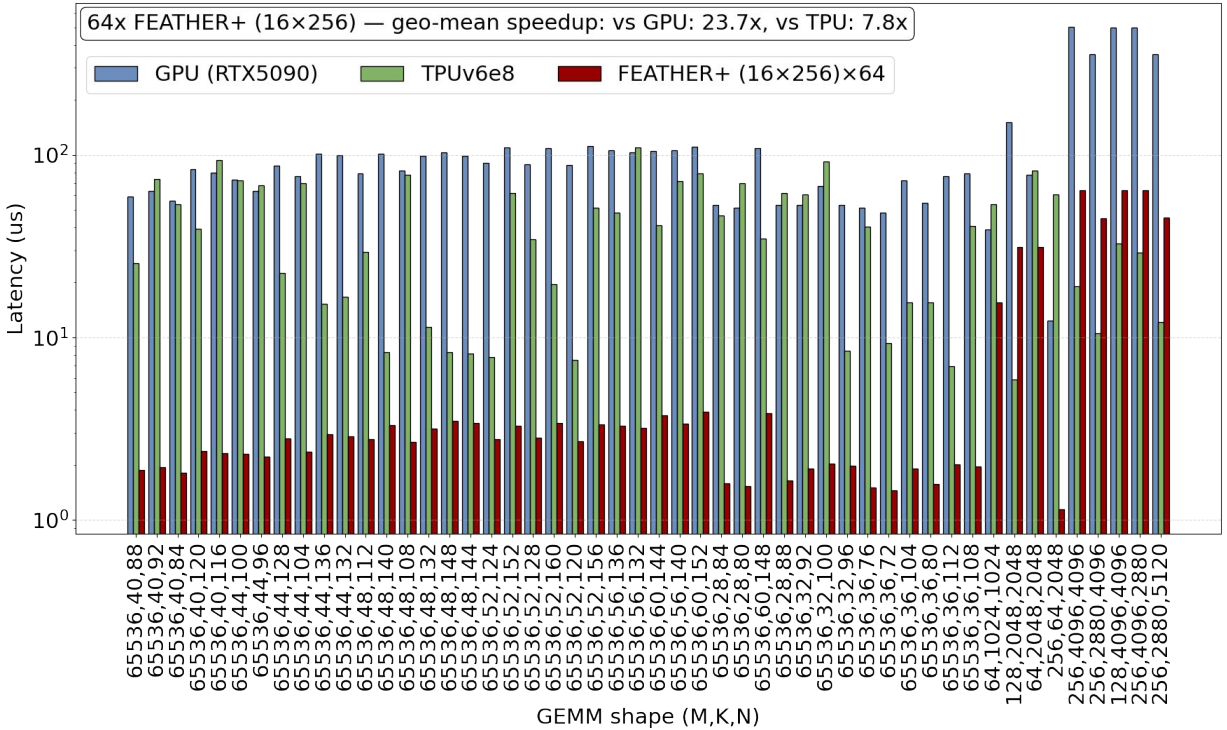
(a) Allo-Generated FEATHER on U280

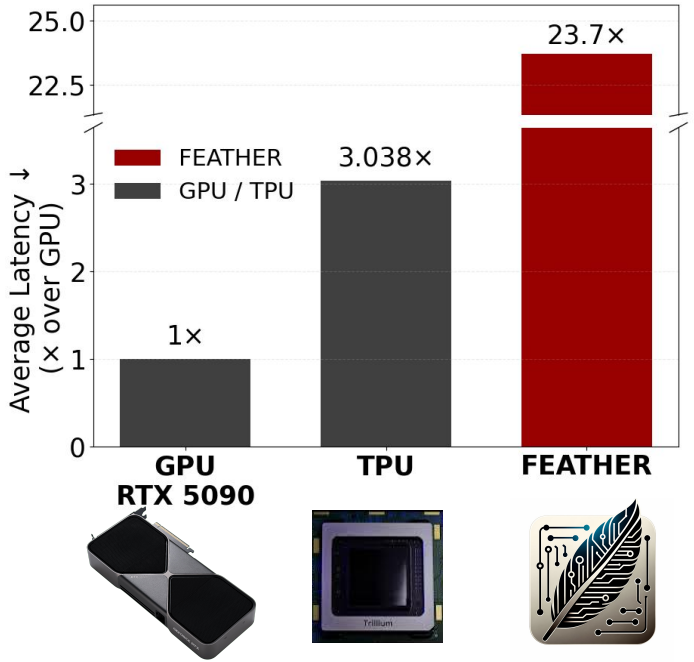


(b) Customizable Template

	LUT	FF	BRAM
Allo-FEATHER	**29181**	**35808**	**24**
FEATHER RTL	50434	82815	30

Evaluation: Comparison Against GPU and TPU





8 chips

64 Cores

Faster in Single-batch short-and-wide MatMul, Slower in large square MatMul

Impact: Award @ ISPASS and Tutorial @ ASPLOS

ISPASS

IEEE International Symposium on Performance Analysis of Systems and Software

ISPASS 2026

Seoul, South Korea

DISTINGUISHED ARTIFACT HONORABLE MENTION

MINISA: Minimal Instruction Set Architecture for Next-gen Reconfigurable Inference Accelerator

Presented to

Jianming Tong[1], Yujie Li[1], Devansh Jain[2], Charith Mendis[2], Tushar Krishna[1]

Georgia Institute of Technology[1], University of Illinois Urbana-Champaign[2]

John Kim
ISPASS 2026 General Chair

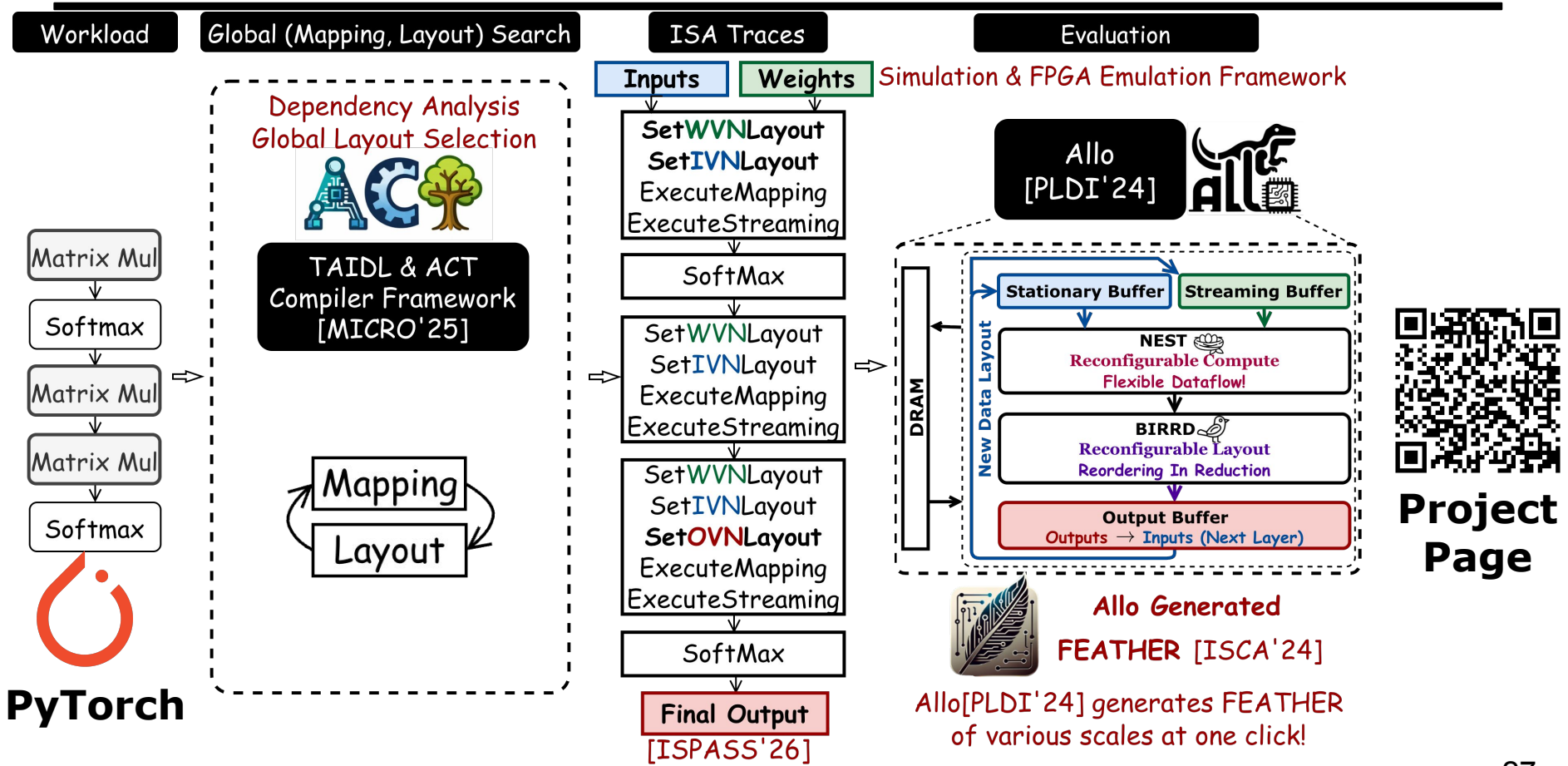
Brandon Reagen
ISPASS 2026 Program Chair

IEEE TCuARCH IEEE computer society IEEE COMPUTER SOCIETY mobilint panmnesia HYPER ACCEL NSF

Distinguish Artifacts Honorable Mention
ISPASS'26

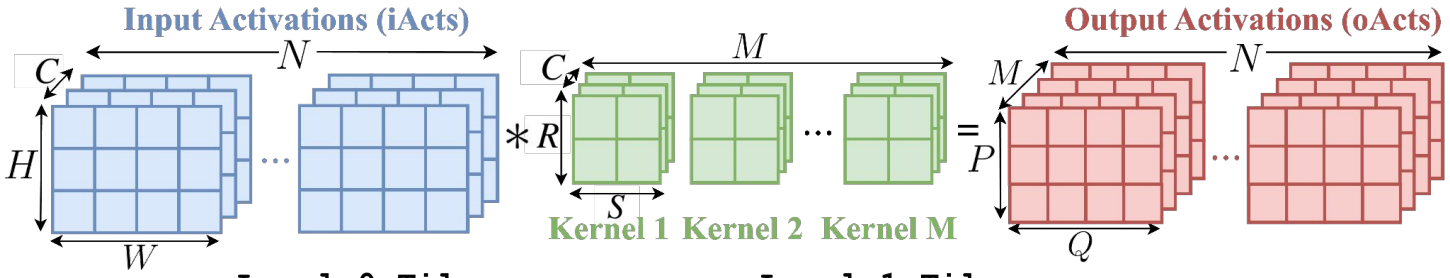
RAIC Tutorial
ASPLOS'26

Summary: Open End-to-end Reconfigurable AI Computing Flow



Mapping: Compute and Memory Access Scheduling

Dataflow: fine-grained compute and memory scheduling over space and time [5]




```
Level-0 Tile                    Level-1 Tile
for nt in [0,N,1):              for n in [0,1,1):
 for mt in [0,M,16):             for h in [0,4,1):
  for ct in [0,C,64):             for w in [0,4,1):
   for ht in [0,R,4):              for r in [0,2,1):
    for wt in [0,Q,4):              for s in [0,2,1):
     for rt in [0,R,2):              for m in [0,16,4):
      for st in [0,Q,2):              for c in [0,64,4):
```

Tile — Order — Order — Parallelism

Tiling + Dataflow = Mapping

(1,16,64,4,4,2,2) | NMCHWRS, MC

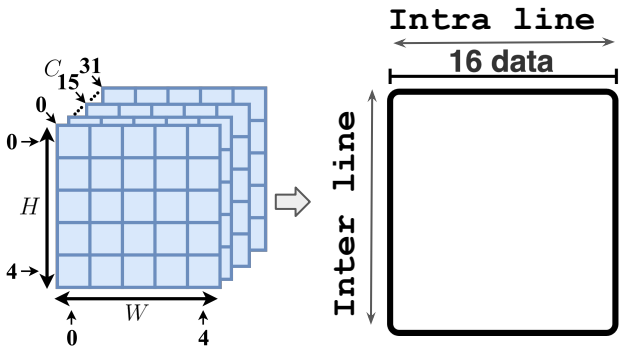
T — O — P

Reuse over time ← → Reuse over space

10^{26} choices!

[5] Kao et al , SIGMETRICS 2022

Layout: Fine-grain Organization of Data in on-chip Buffer

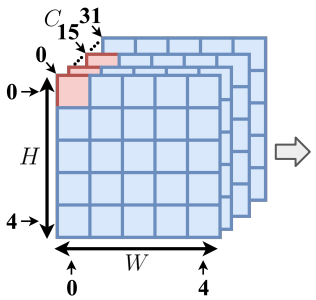


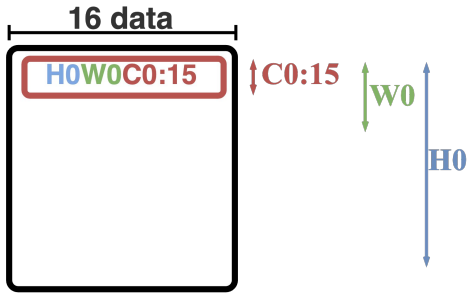

```
# Inter-line dimension order

          [Start, End, Step]
H ------> for ht in [0,4,1]:
 W -----> for wt in [0,4,1]:
  C ----> for ct in [0,31,16]:
```

```
# Intra-line dimension order
```

Layout: Fine-grain Organization of Data in on-chip Buffer



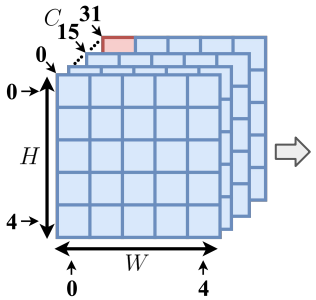


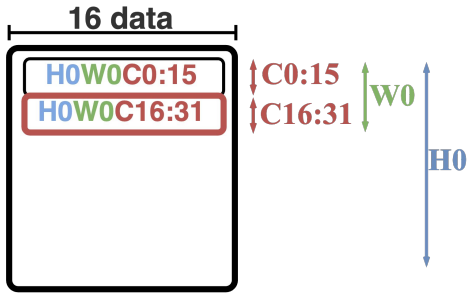

```
# Inter-line dimension order

                [Start, End, Step]
0 H  ------>for ht in [0,4,1]:
  0 W ------>for wt in [0,4,1]:
0:15 C ----->for ct in [0,32,16]:
```

```
# Intra-line dimension order
```

Layout: Fine-grain Organization of Data in on-chip Buffer



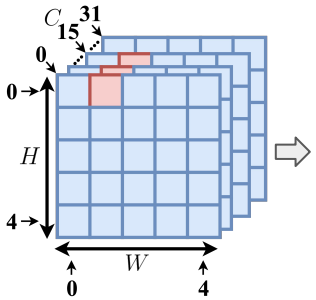


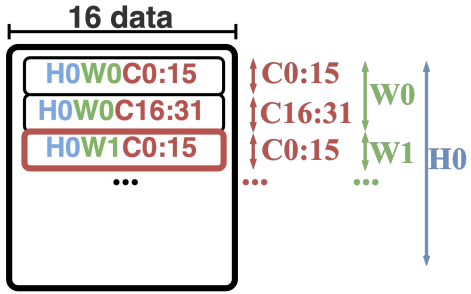

```
# Inter-line dimension order

                [Start, End, Step]
0  H  ------>for ht in [0,4,1]:
 0 W  ------>for wt in [0,4,1]:
16:31 C ---->for ct in [0,32,16]:
```

```
# Intra-line dimension order
```

Layout: Fine-grain Organization of Data in on-chip Buffer



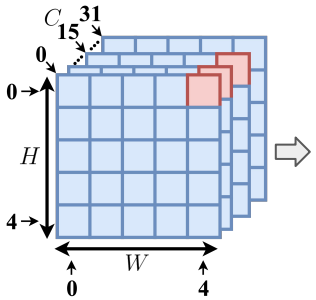


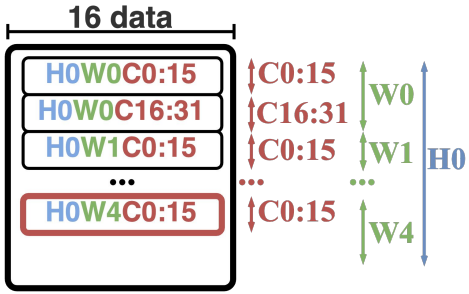

```
# Inter-line dimension order

                [Start, End, Step]
0  H  ------>for ht in [0,4,1]:
 1  W  ------>for wt in [0,4,1]:
0:15  C  ----->for ct in [0,32,16]:
```

```
# Intra-line dimension order
```

Layout: Fine-grain Organization of Data in on-chip Buffer



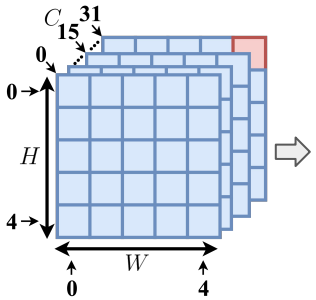


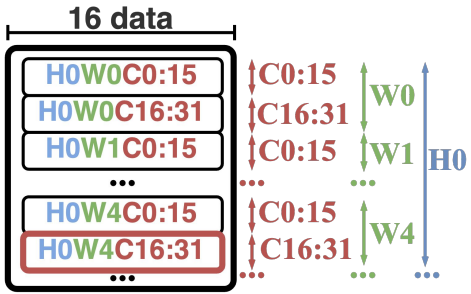

```
# Inter-line dimension order

                [Start, End, Step]
0 H  ------>for ht in [0,4,1]:
 4 W ------>for wt in [0,4,1]:
0:15 C ------>for ct in [0,32,16]:
```

```
# Intra-line dimension order
```

Layout: Fine-grain Organization of Data in on-chip Buffer





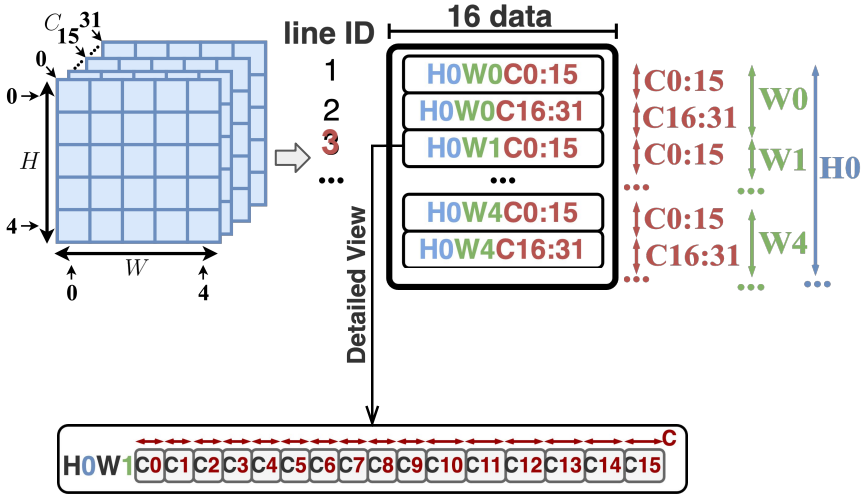

```
# Inter-line dimension order

                [Start, End, Step]
0  H  ------>for ht in [0,4,1]:
 4  W  ------>for wt in [0,4,1]:
16:31 C  ---->for ct in [0,32,16]:

-------------------------------------------------------------

# Intra-line dimension order
```

Layout: Fine-grain Organization of Data in on-chip Buffer



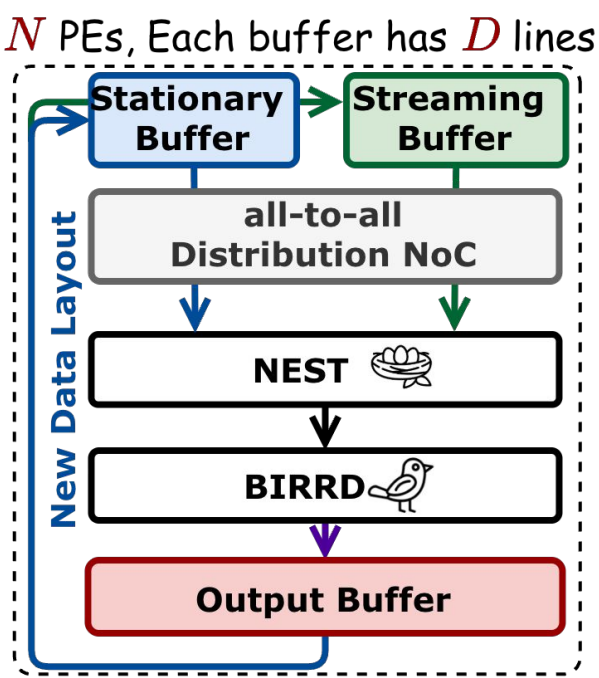

```
# Inter-line dimension order
          [Start, End, Step]
H  ------>for ht in [0,4,1]:
 W ------>for wt in [0,4,1]:
  C ------>for ct in [0,32,16]:
```

```
# Intra-line dimension order
WX1------>for w in [wt,wt+1,1]:
 HX1------>for h in [ht,ht+1,1]:
  CX16 ---->for c in [ct,ct+16,1]:
```

Layout Terminology: <inter_line_loop>_<intra_line_loop>

HWC_WX1HX1CX16 →HWC_CX16 (Omit dimensions w/o parallelism)

Challenge: Reconfiguration -> Significant Control Costs



Reconfiguration Choices

$O(N \cdot D)$ Layout Choices

$O(N^{1.5})$ Mapping Choices

$O(\sqrt{N}^2)$ Reordering Choices

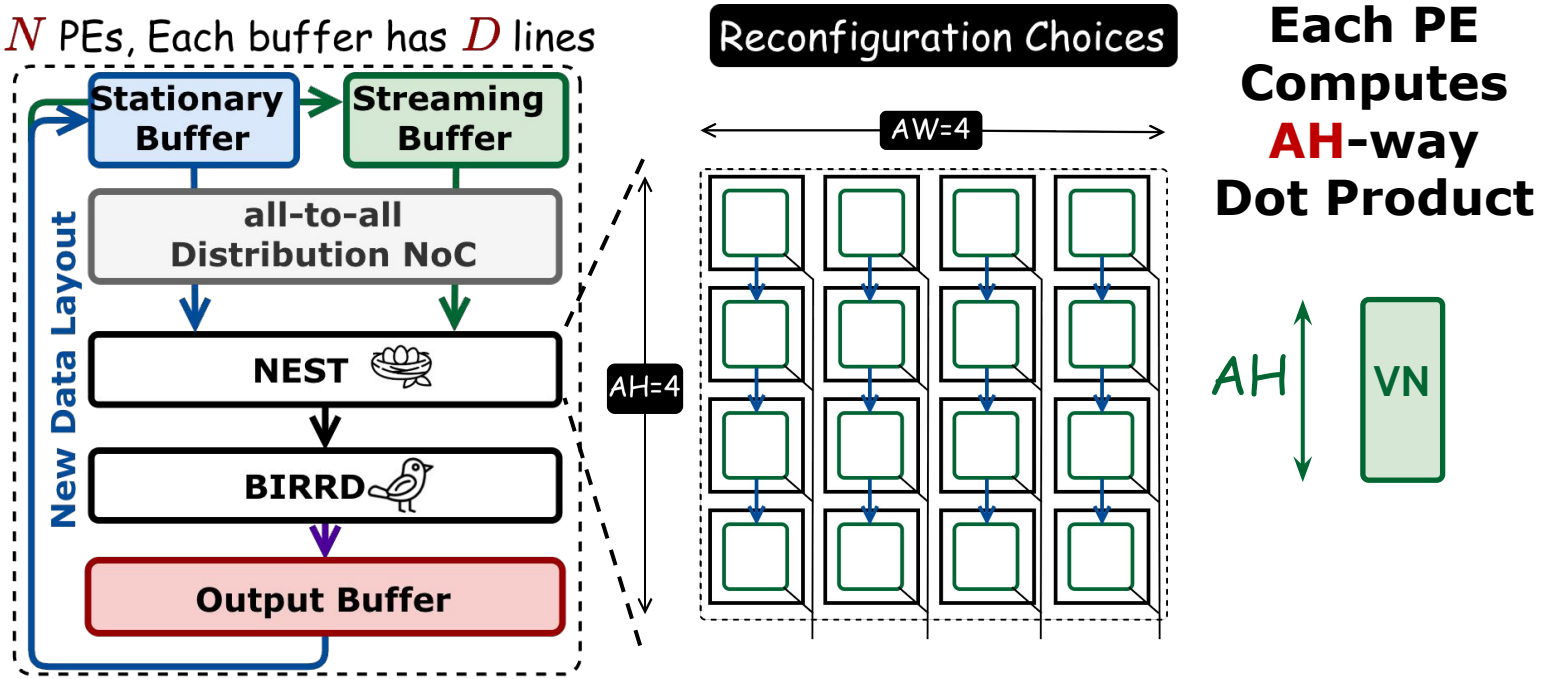
$O(\sqrt{N} \log \sqrt{N})$ Reorder-In-Reduction Choices

$O(N \cdot D)$ Layout Choices

FEATHER	4×4	8×8	4×64	16×16	8×128	16×256
stall	0	0	75.3%	65.2%	90.4%	96.9%

Explicit Control Stalls up-to 96.9% Execution Time!

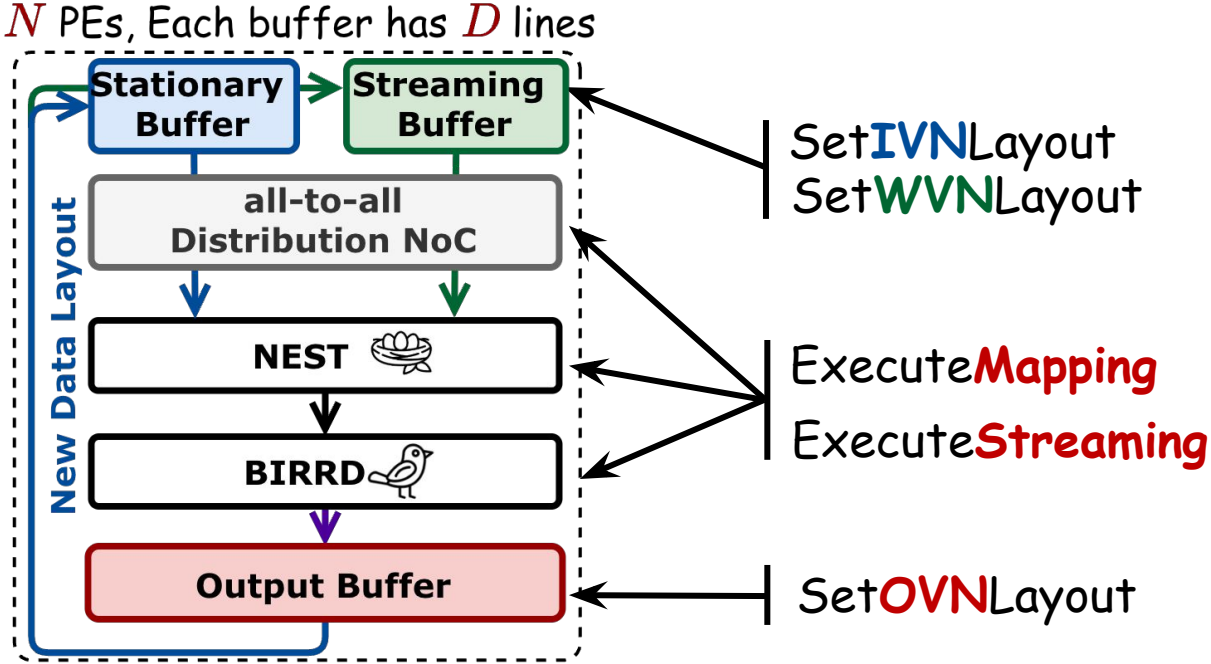
Solution: Virtual Neuron Abstraction



One dot-product performed by one PE

VN is Exact abstraction that maintain reconfiguration w/o redundancy

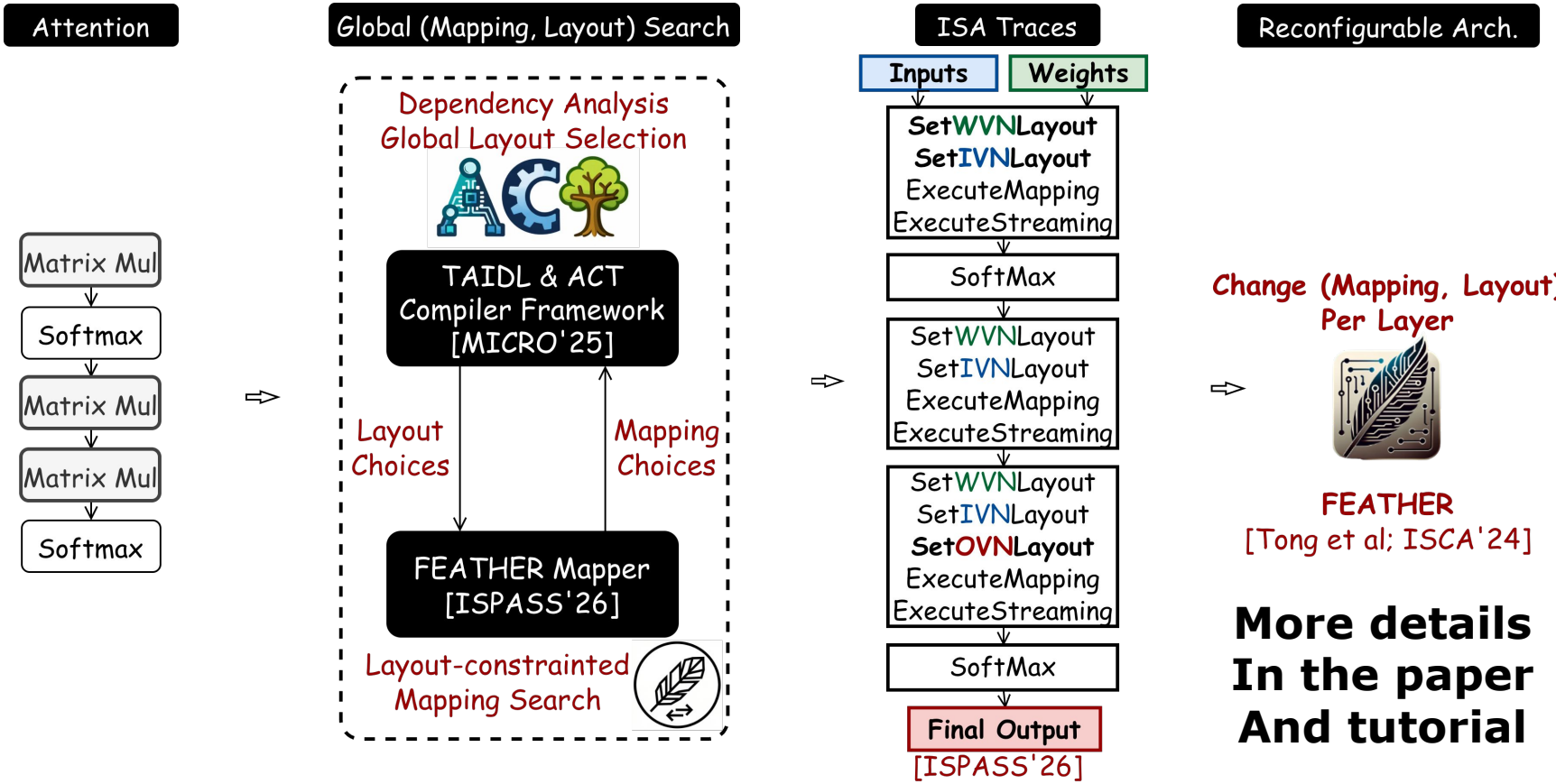
MINISA: A minimal set of instruction set on VN granularity



Redesign Mapping and Layout choice in VN Granularity – 5 ISAs

Resolve the Prohibitive Control Overheads

Compilation Flow for Reconfigurable Accelerator



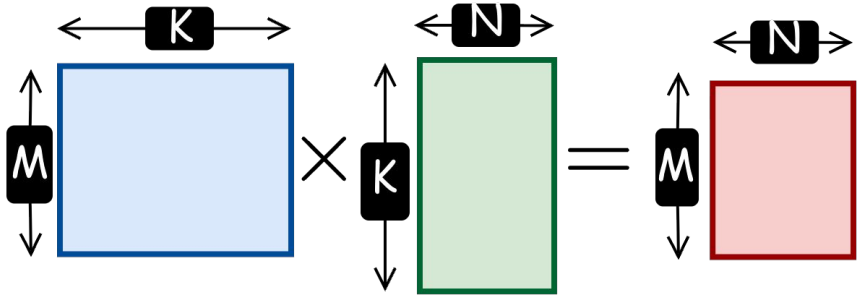
Outlines

- Background and Motivation
- MINISA
- **Evaluation**
- Summary

Experimental Setup -- Workload

Workloads

Single batch Matrix Multiplication of Various Shapes



Workload	Matrix Multiplication of Shape $O^{P=M,Q=N}_{m,n} = I^{M=M,J=K}_{m,k} \cdot W^{K=K,N=N}_{k,n}$
FHE: BConv	$(65536 \times K) \cdot (K \times N)$, $K \in [28, 60]$, $N \in [72, 160]$ (41 shapes)
FHE: NTT	$J = K = N \in \{1024, 2048, 4096\}$, $M \in \{64, 128, 256\}$, $M \leq K/16$
ZKP: NTT	$J = K = N \in \{8192, 16384, 32768\}$, $M \in \{K/32, K/16\}$
GPT-oss	$M = 2048$, $(J = K, N) \in \{(64, 2048), (2880, 4096/5120/201088), (4096, 2880)\}$

Hardware

Roughly Same TDP

RTX 5090

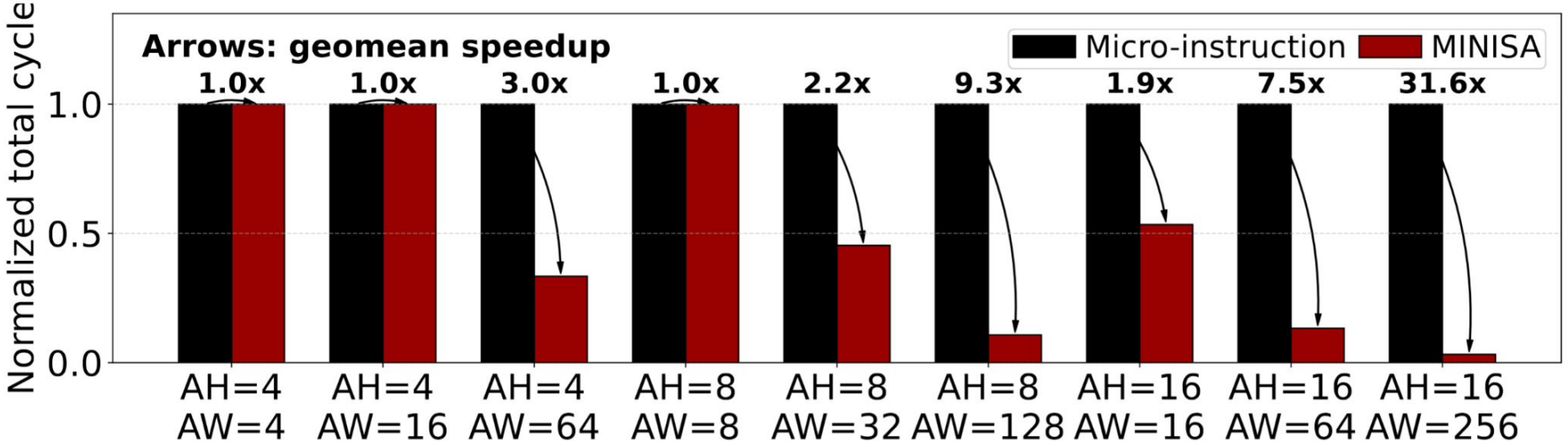
TPUv6e

Eight Chips

FEATHER

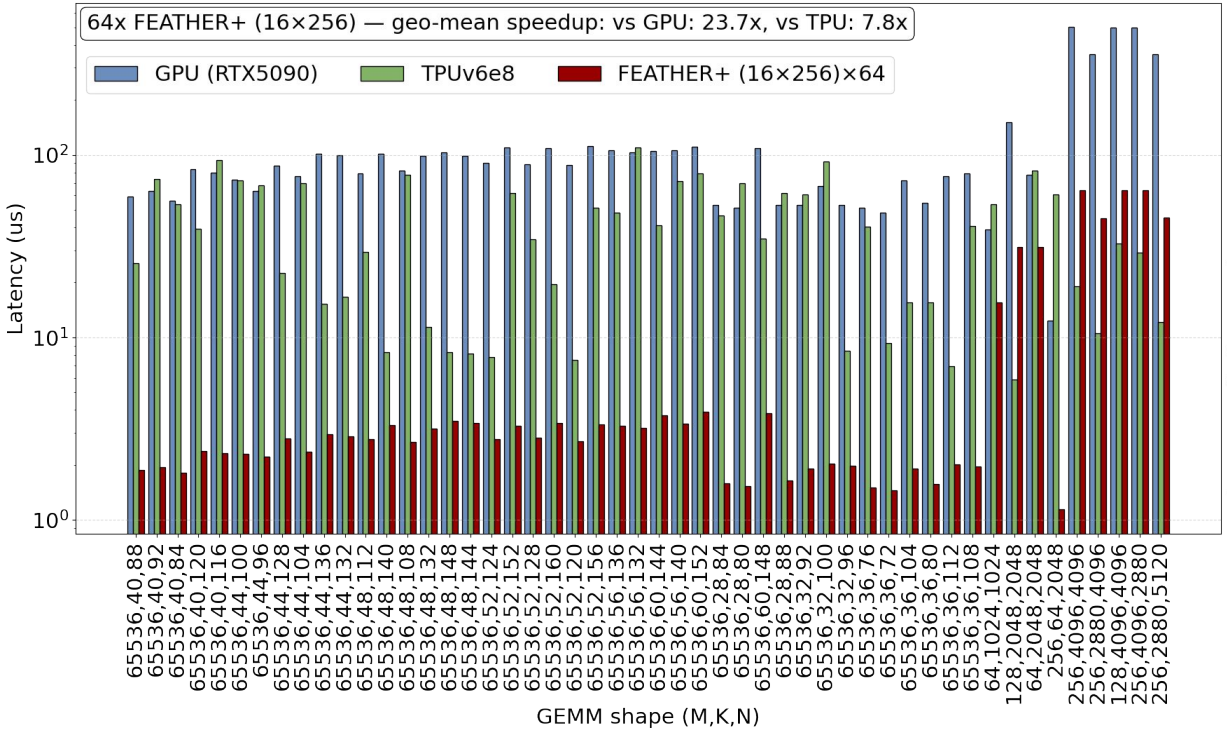
64 cores

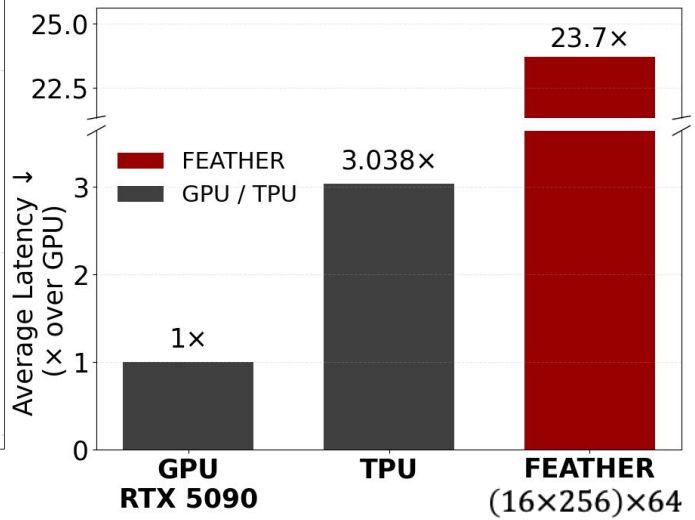
End-to-End Latency Evaluation Under Different Scale



31.6x Speedup: MINISA Eliminates the Instruction Fetch Stall

Comparison Against GPU and TPU



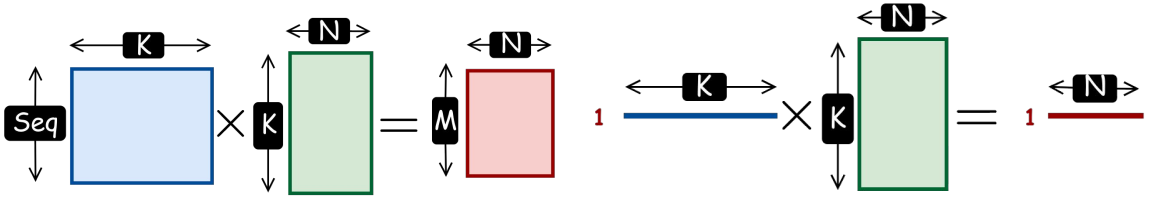


Faster in Skewed Matrix, Slower in Dense Matrix Multiplication.

Outlines

- Background
- Motivation
- MINISA:
- Evaluation
- **Summary**

Summary: Future Reconfigurable AI System



Paper

Code

Virtual Neuron Representation

Mapping

Layout

MINISA [ISPASS'26]
Compilation and ISA
(Mapping, Layout)
Co-search per workload

ASPLOS Tutorial

FEATHER [ISCA'24]

Collaborated with IBM

Future Reconfigurable AI System

Thank you!

Contact Us