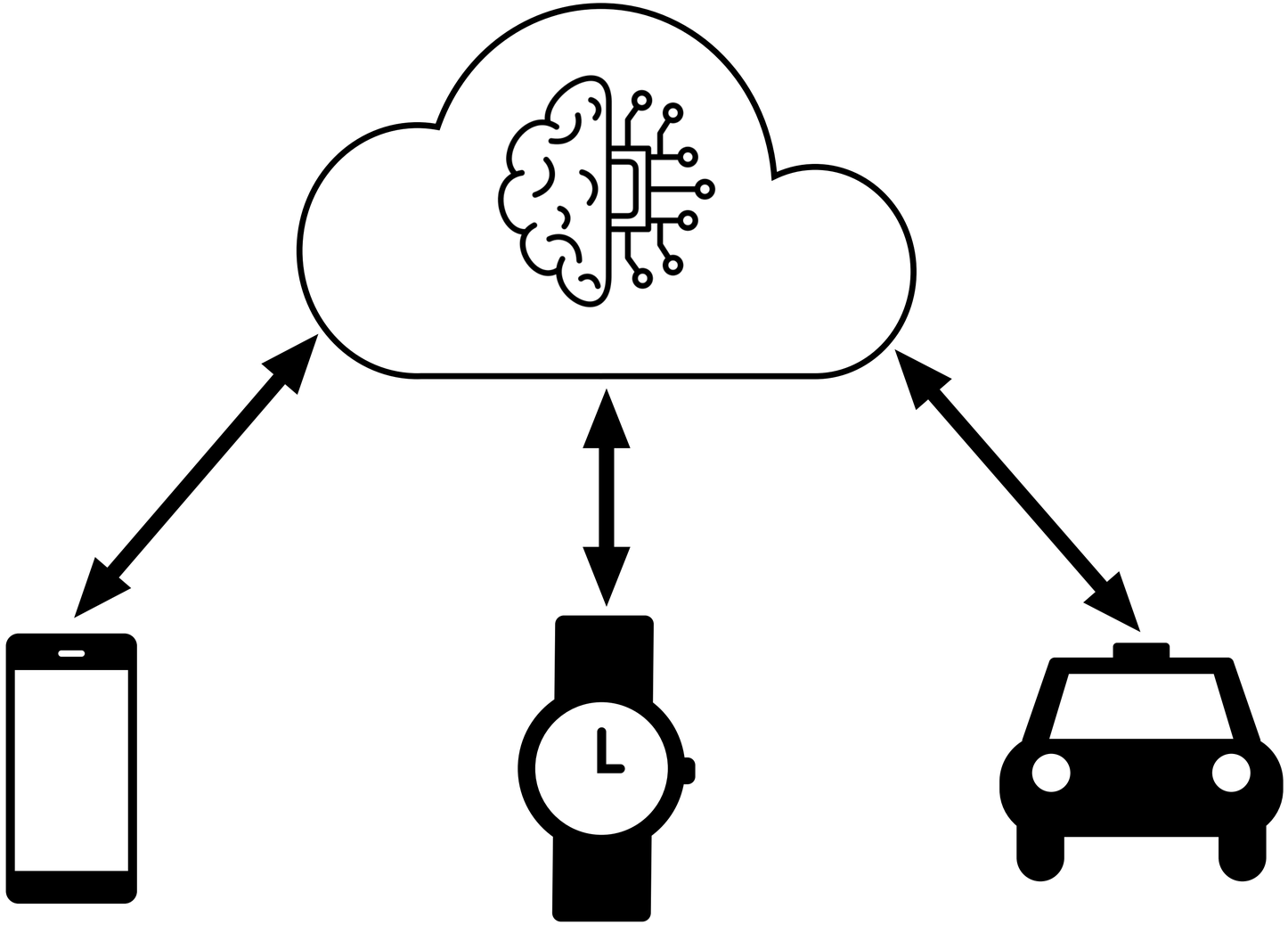
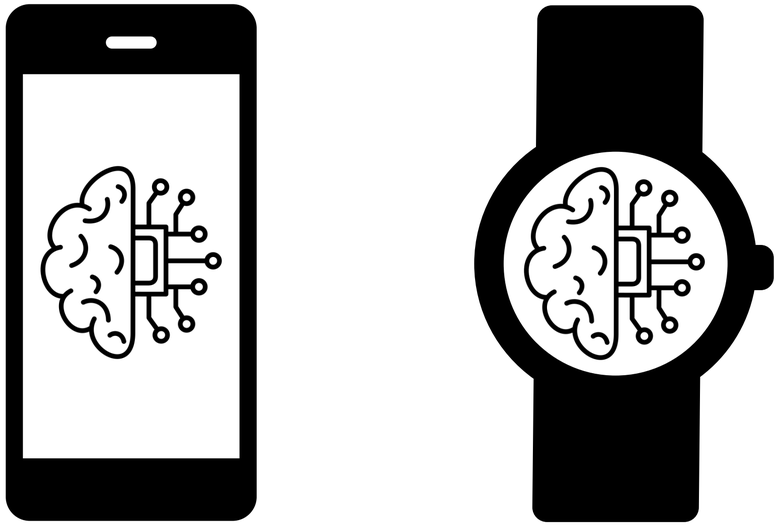
From Art to Science

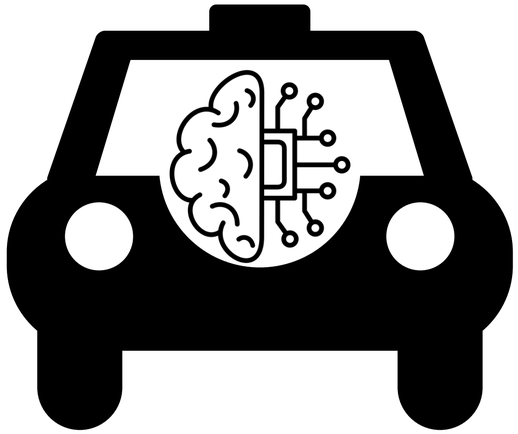
Frameworks for the Systematic Design of Tensor Accelerators

Brandon Schühlein, 30.07.2025



AI is moving to Embedded Devices





Reduced Latency

Offline

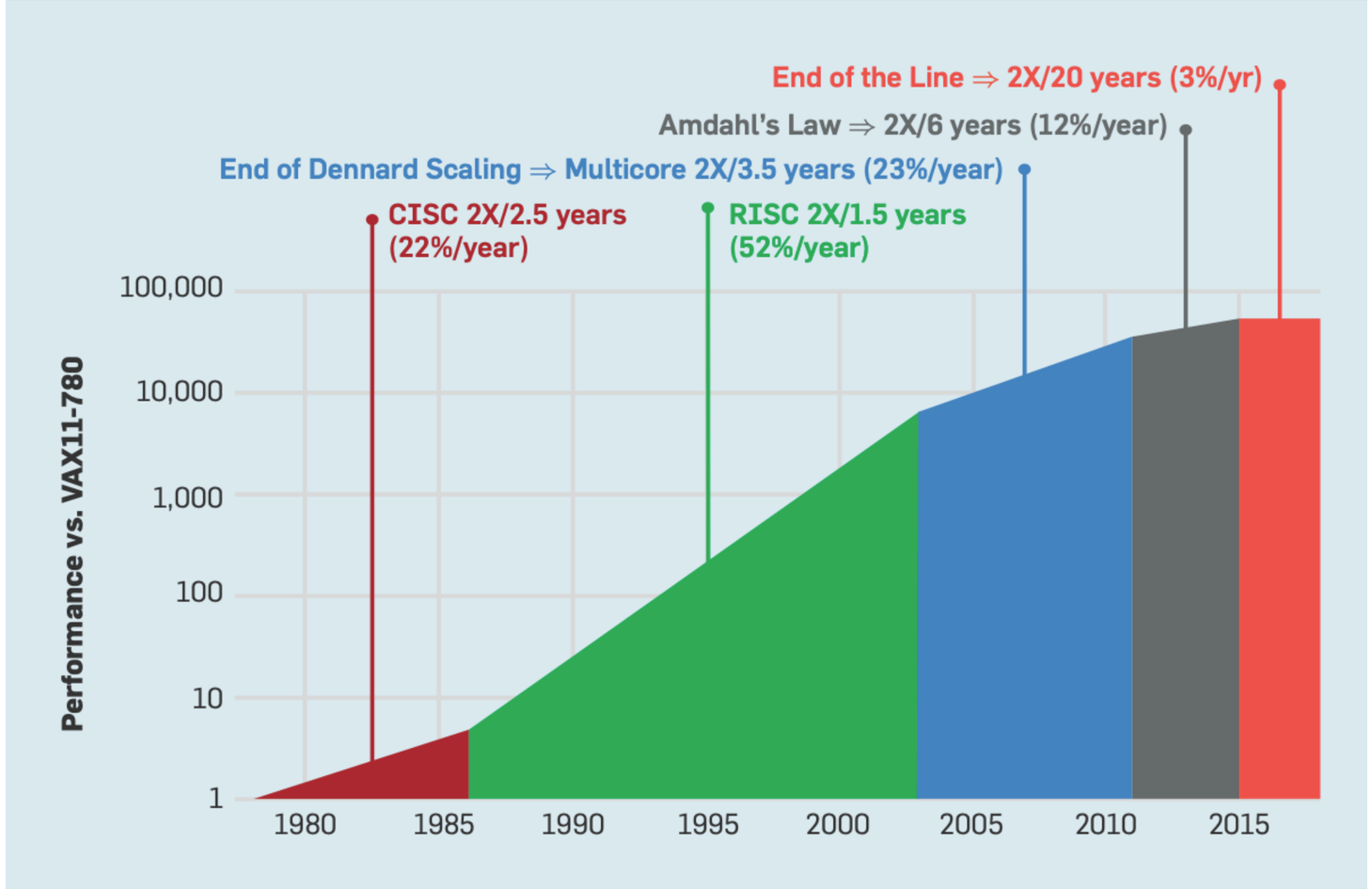
Privacy

Energy

Area

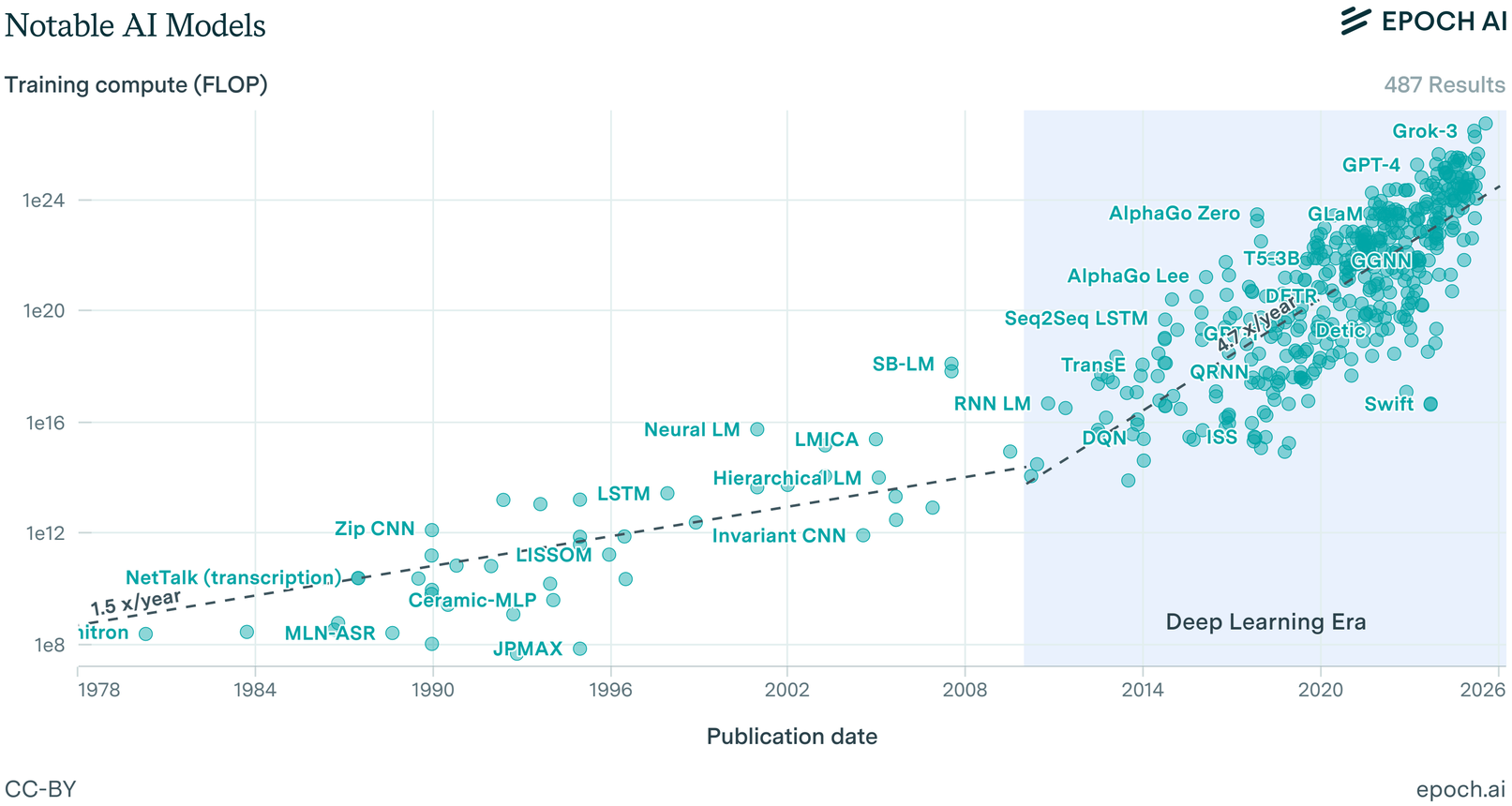
Why do we need Tensor Accelerators?

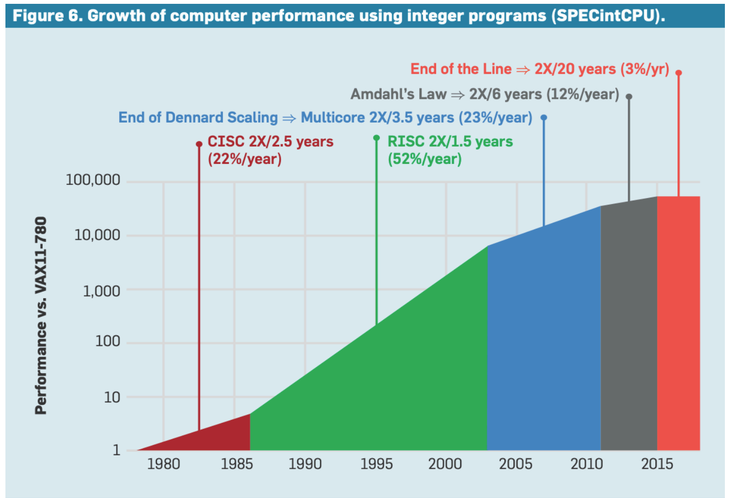
Figure 6. Growth of computer performance using integer programs (SPECintCPU).



[Hennesy19]

Why do we need Tensor Accelerators?





[Hennesy19]

Specialization, not magic

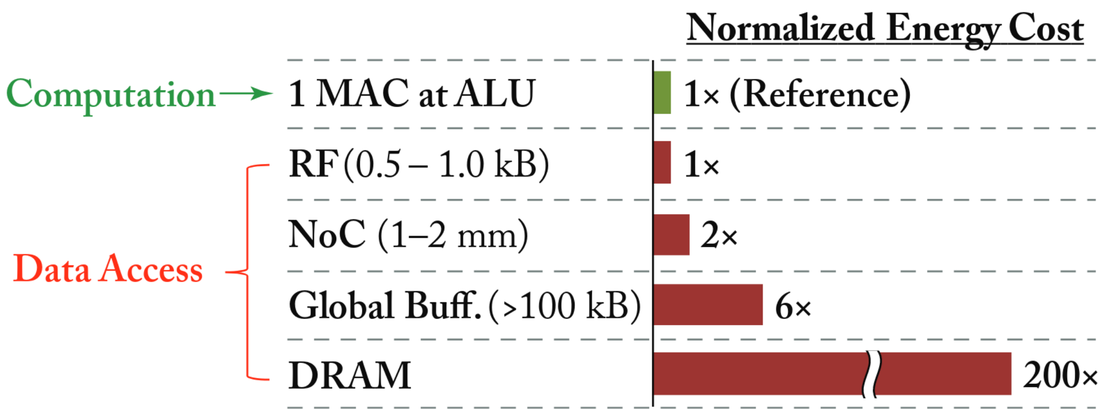
Data Movement Bottleneck

Data movement is expensive

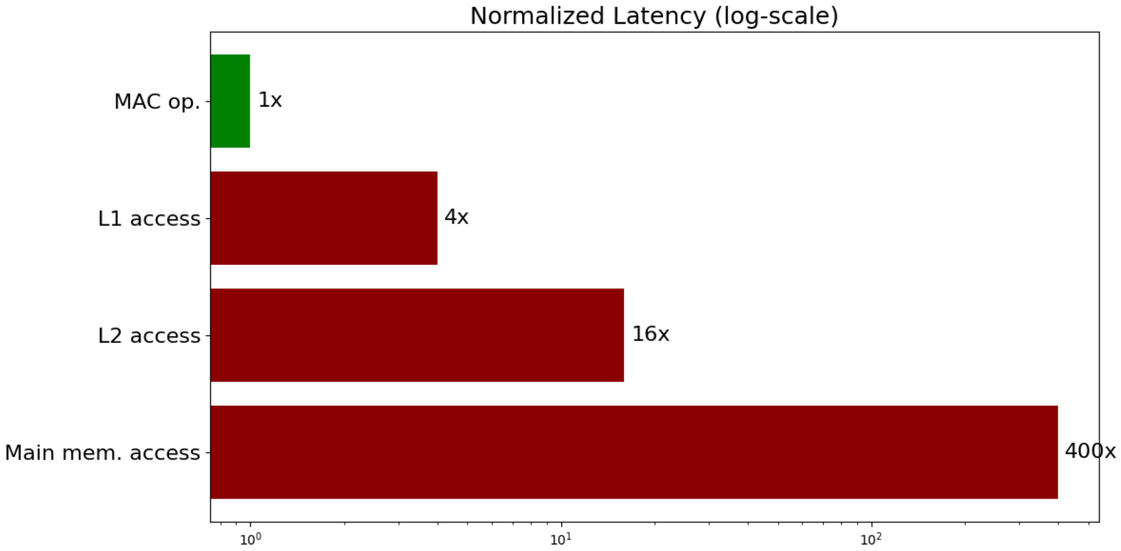
- High energy cost
- High latency

Solution: Specialized Accelerators

- Maximize data reuse
- Exploit massive DNN parallelism



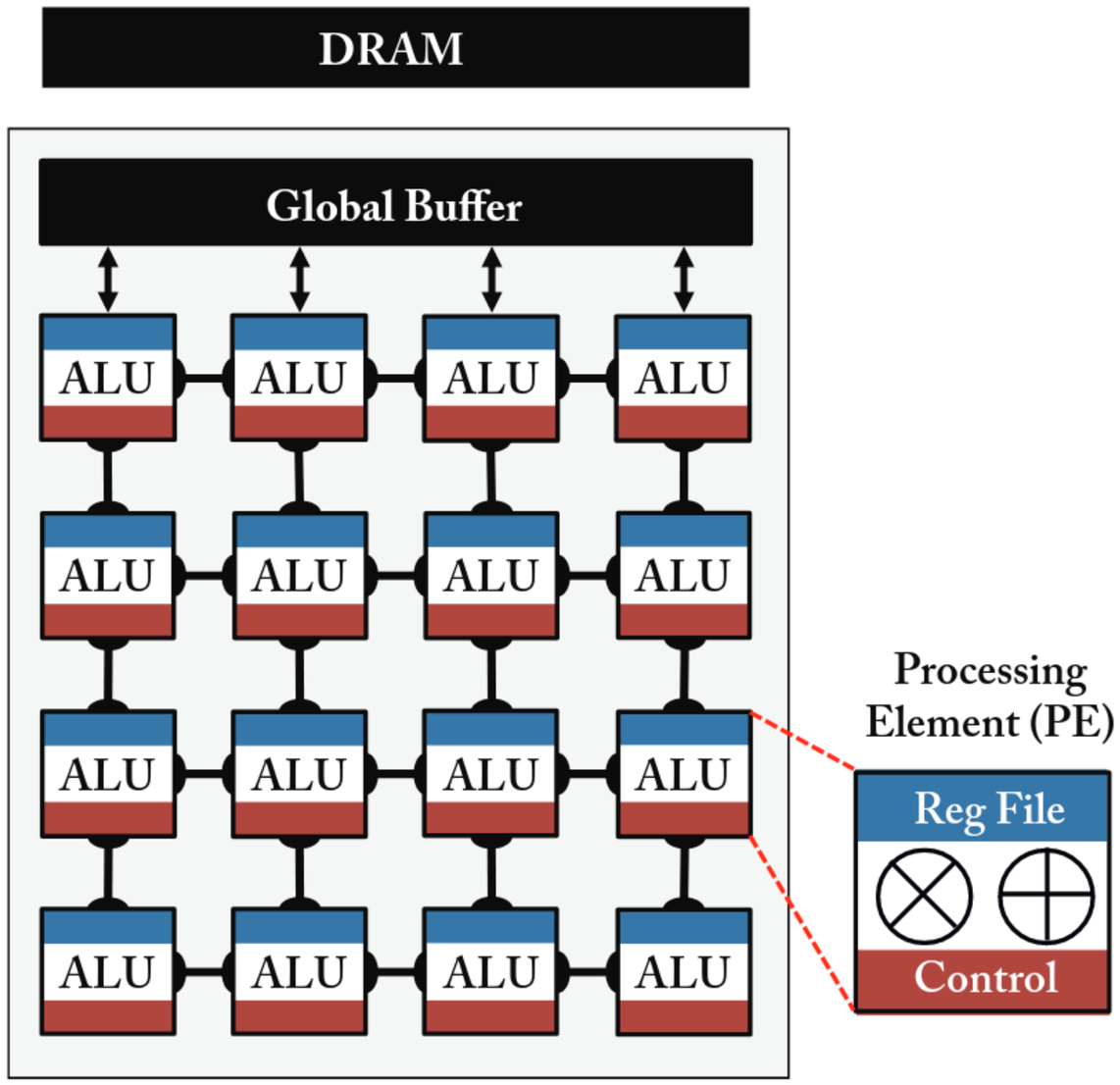
[Sze20]



Tensor Accelerator

Relies on 3 key components:

1. Processing Elements (PEs)
2. Memory Hierarchy
3. Network-on-Chip (NoC)



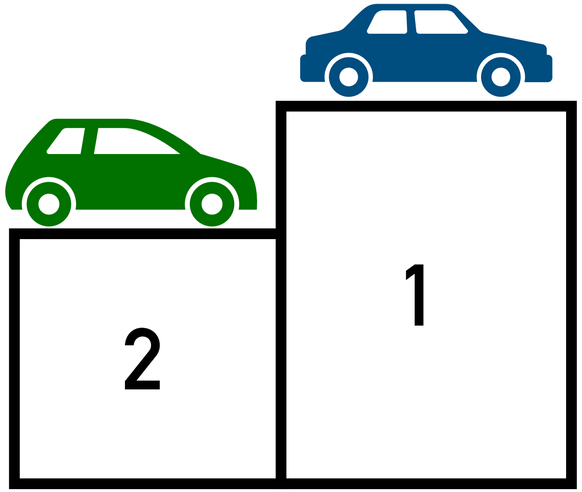
[Sze20]

Comparing Architectures

Race Car Analogy

Architecture = 

Workload/
Algorithm = 



Which car is faster?

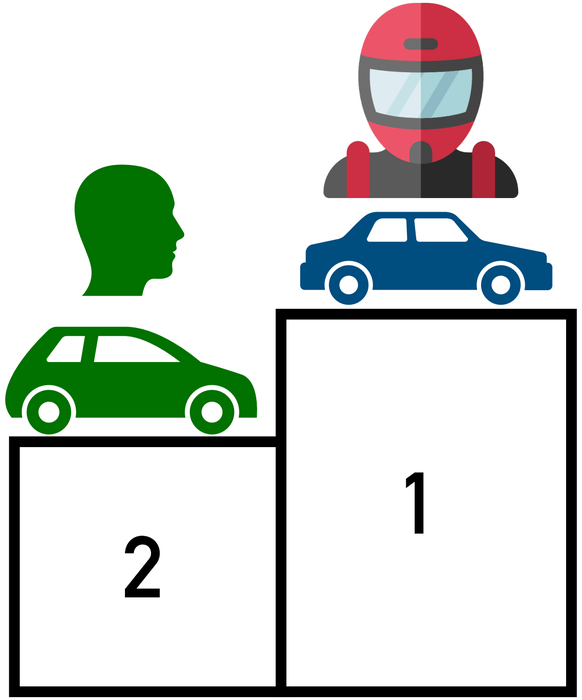
Comparing Architectures

Race Car Analogy

Mapping = Driver

Architecture = 

Workload/
Algorithm = 



Which car is faster?

We don’t know

Workload & Mapping

Convolution

$$\text{Output}[q] = \sum_{s=0}^{s=S-1} \text{Input}[q+s] \times \text{Filter}[s]$$

```
for q in range(Q):
    for s in range(S):
        o[q] += i[q+s] * f[s]
```

Output stationary (OS)

```
for s in range(S):
    for q in range(Q):
        o[q] += i[q+s] * f[s]
```

Weight stationary (WS)

Workload & Mapping

Convolution

$$\text{Output}[q] = \sum_{s=0}^{s=S-1} \text{Input}[q+s] \times \text{Filter}[s]$$

```
for q1 in range(Q1):
    for s in range(S):
        for q0 in range(Q0):
            q = q1 * Q0 + q0
            o[q] += i[q+s] * f[s]
```

OS + Tiling

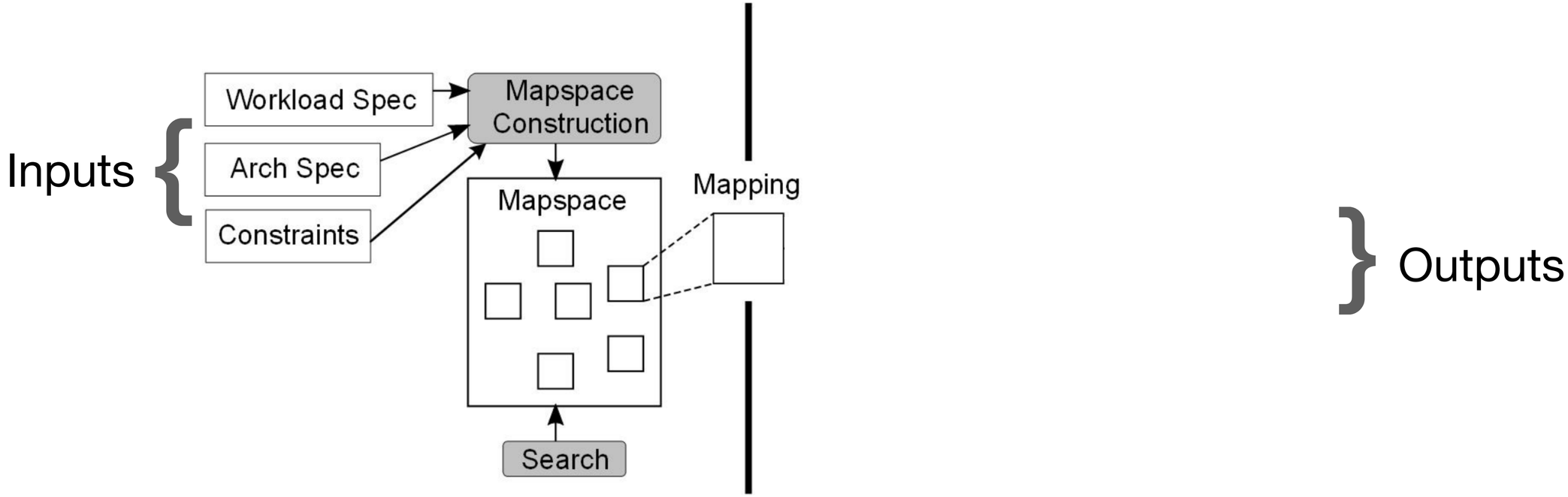
```
for q1 in range(Q1):
    for s in range(S):
        #parallel
        for q0 in range(Q0):
            q = q1 * Q0 + q0
            o[q] += i[q+s] * f[s]
```

OS + Tiling + Parallelism

- ✅ Why do we need Tensor Accelerators?
- ✅ How can they achieve better performance on DNN workloads than CPUs?
- ✅ What is a mapping?
- ☐ How can we fairly compare architectures?

Timeloop

Searching the best “driver” for every “car”



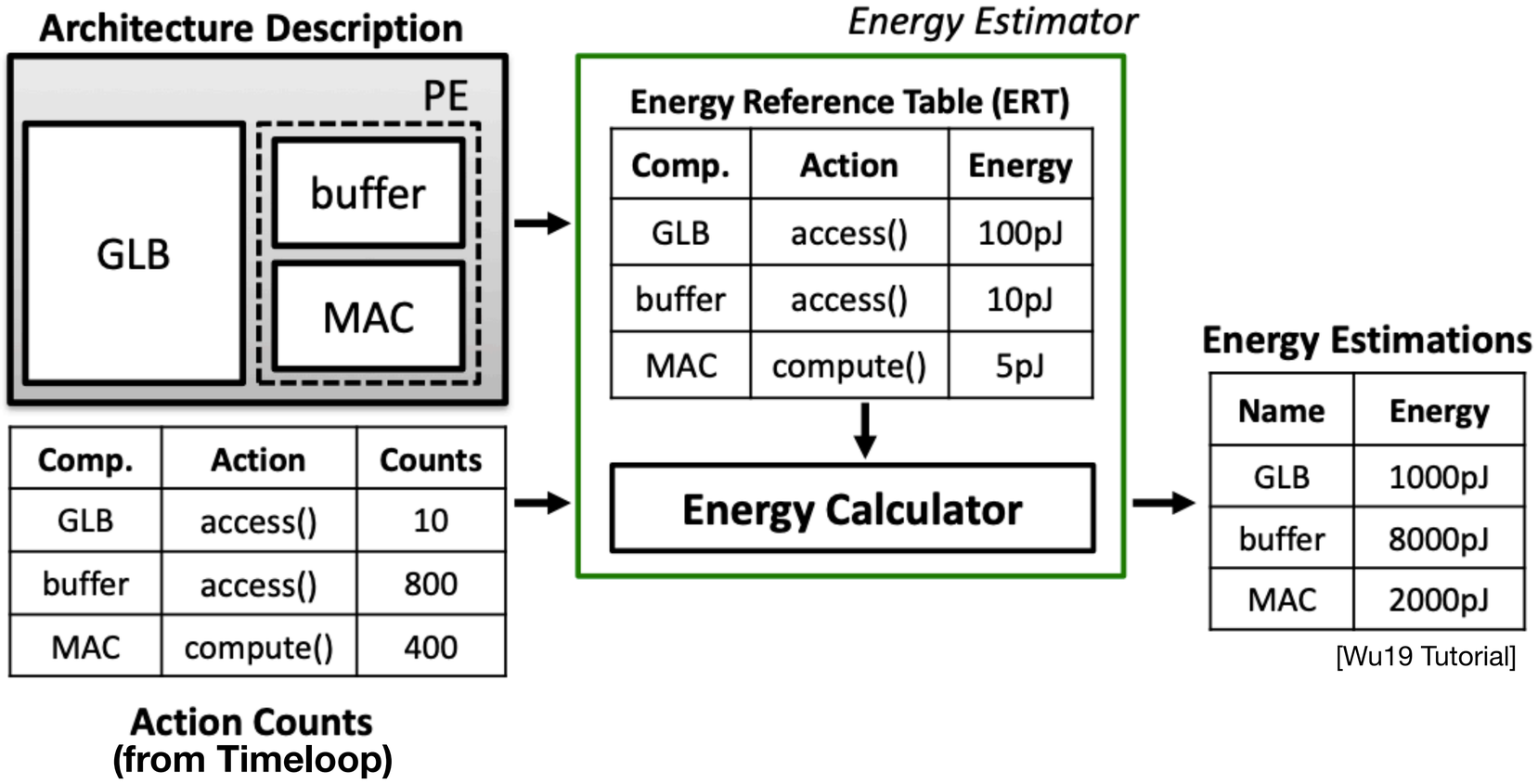
Construct & Search Mapspace

Analytically Model Performance

[Parashar19]

Accelergy

Calculating the “fuel” consumption



[Wu19 Tutorial]

All models are wrong, but some are useful

Assumptions of Timeloop & Accelergy:

	Addressed by	
• Perfect factorization	→	Ruby
• Single-layer scope	→	LoopTree
• Fixed data layout	→	Layoutloop
• Dense computations	→	Sparseloop
• Value independent energy consumption	→	CiMLoop

Ruby

Increase PE utilization via Imperfect Factorization

10 Processing Elements, 1D tensor with length 28

```
for i in range(28):
    ...
```

Perfect Factorization

```
# 28 % 4 == 0
# 28 % 7 == 0
for i in range(4):
    #parallel
    for j in range(7):
        ...
```

Imperfect Factorization

Iter. 0-19

```
for i in range(2):
    #parallel
    for j in range(10):
        ...
```

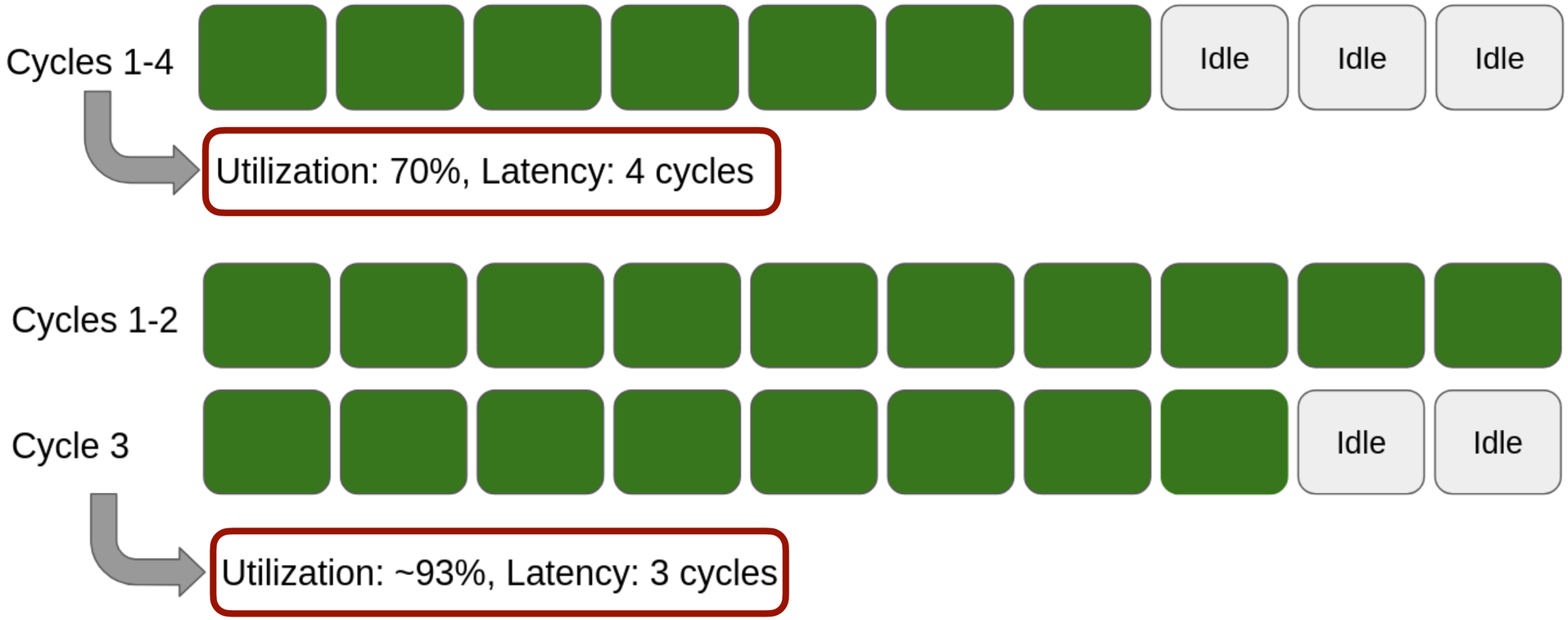
Iter. 20-27

```
# parallel
for i in range(2*10, 28):
    ...
```

Ruby

Increase PE utilization via Imperfect Factorization

Perfect vs. Imperfect Factorization



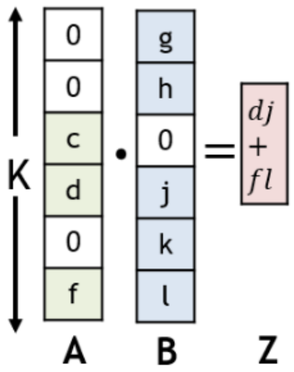
Sparseloop

Save time & energy by avoiding ineffectual computations

Problem: Time & Energy wasted on ineffectual operations (e.g. 0 + 1)

Hardware Solution:

- Gating: go idle for ineffectual ops. (Energy Save)
- Skipping: bypass them entirely (Energy & Latency Save)



processing steps →

SAFs	1	2	3	4	5	6
None	read(0) read(g) compute(0,g)	read(0) read(h) compute(0,h)	read(c) read(0) compute(c,0)	read(d) read(j) compute(d,j)	read(0) read(k) compute(0,k)	read(f) read(l) compute(f,l)
Gating on B storage accesses based on A *(Gate B ← A)*	read(0) ~~read(g)~~ ~~compute(0,g)~~	read(0) ~~read(h)~~ ~~compute(0,h)~~	read(c) read(0) compute(c,0)	read(d) read(j) compute(d,j)	read(0) ~~read(k)~~ ~~compute(0,k)~~	read(f) read(l) compute(f,l)
Skipping on B storage accesses based on A *(Skip B ← A)*	read(c) read(0) compute(c,0)	read(d) read(j) compute(d,j)	read(f) read(l) compute(f,l)			

[Wu22]

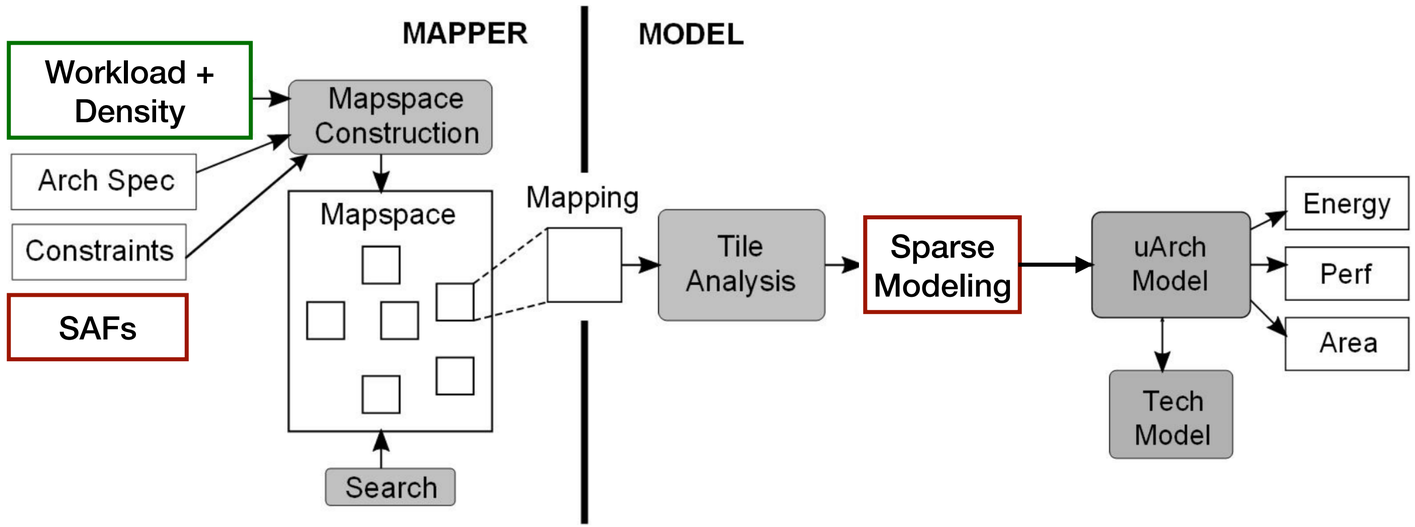
Sparseloop

Save time & energy by avoiding ineffectual computations

Input:

- NEW: Supported Sparsity-Aware-Features (gating, skipping) per memory level
- Adjusted: Workload requires statistical density

Process: adjust dense action counts based of SAFs and sparsity



How accurate are Timeloop's energy & latency estimates?

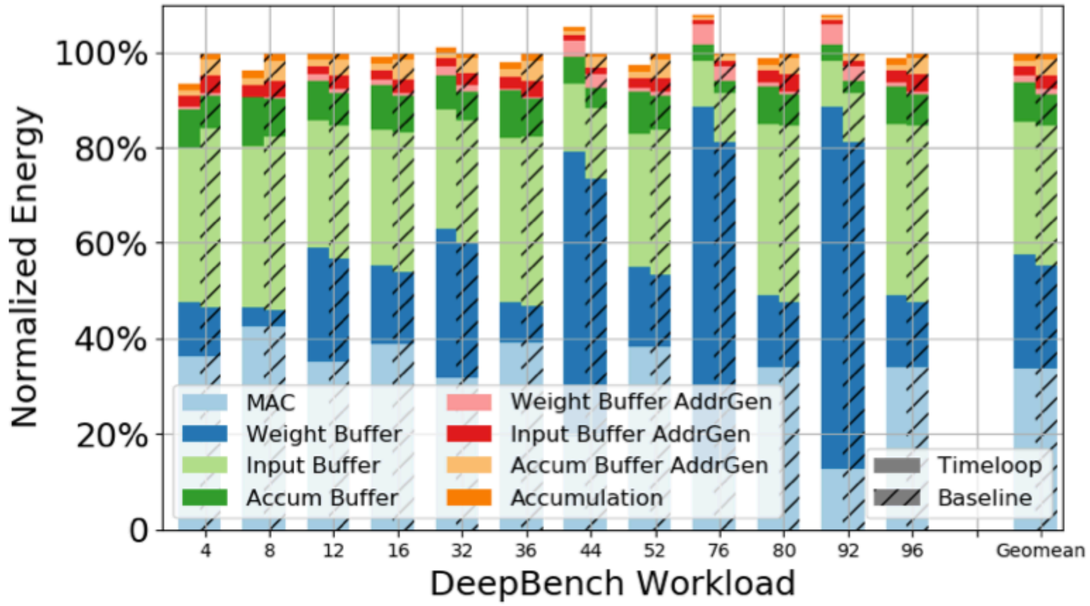


Fig. 8. Energy validation results for NVDLA-derived architecture.

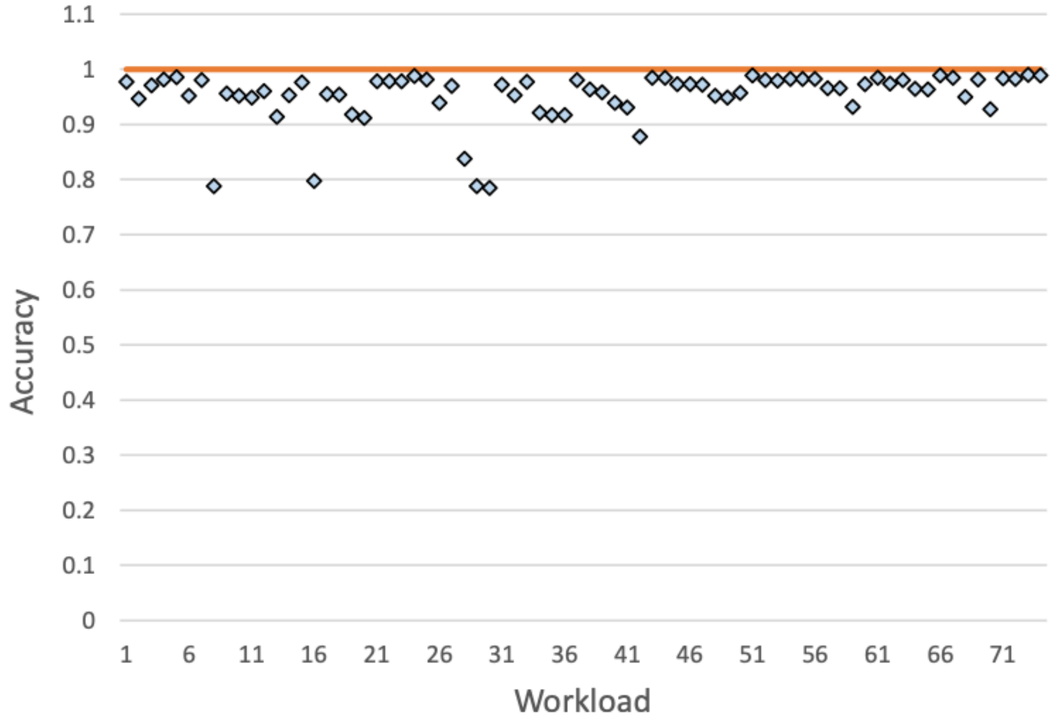


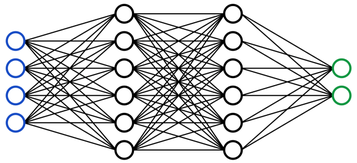
Fig. 9. Performance validation results for NVDLA-derived architecture.

[Parashar19]

Future Directions

Automated Architecture Search

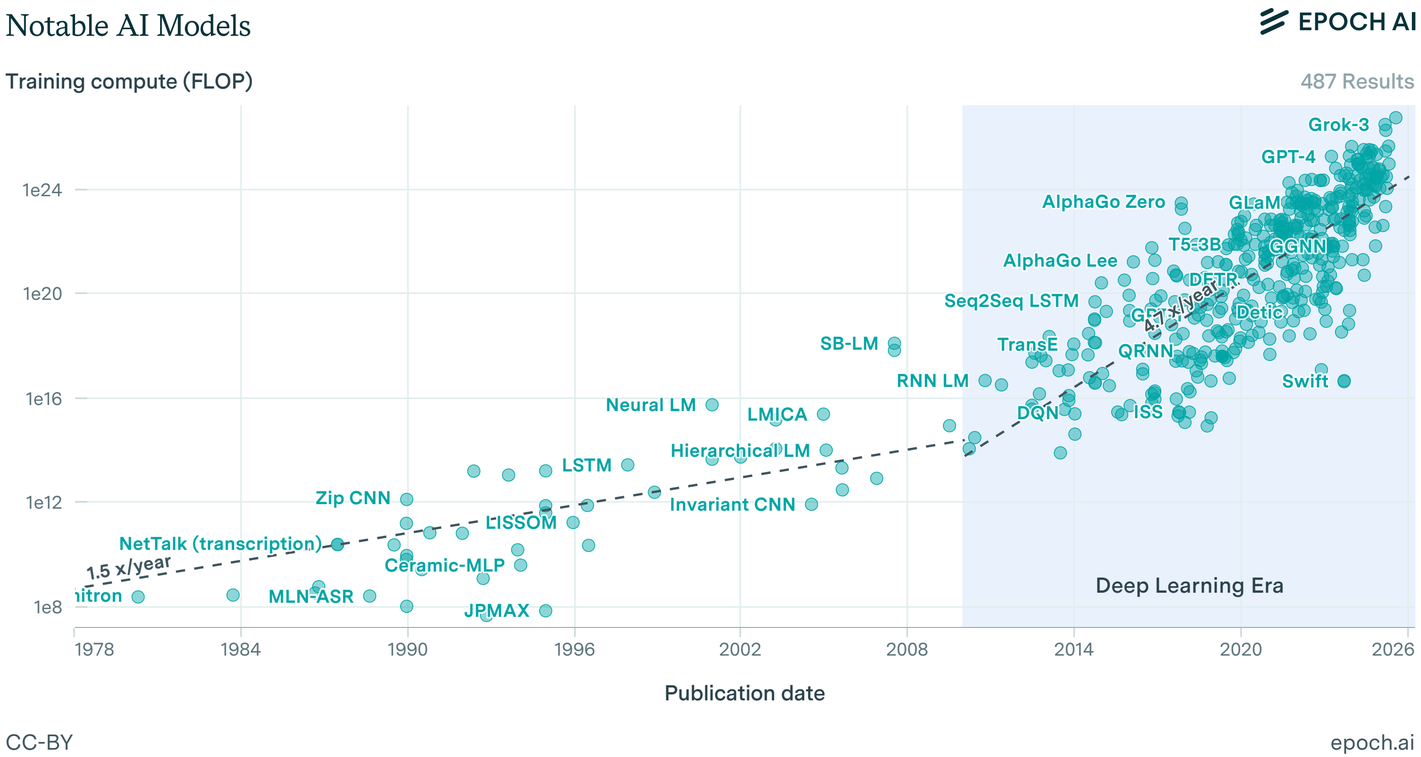
Workload Uncertainty Awareness

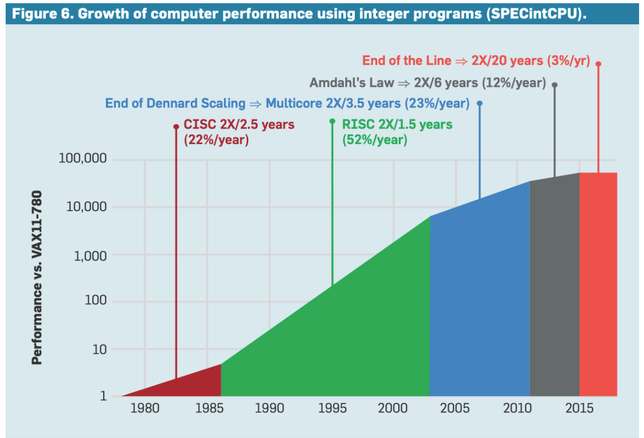


Full Network Optimization

Conclusion

Tensor Accelerators are essential to meet compute demands



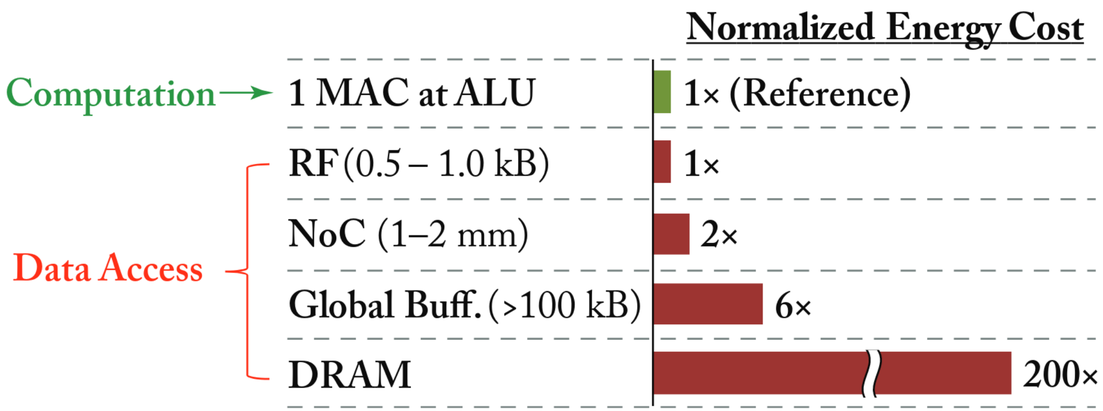


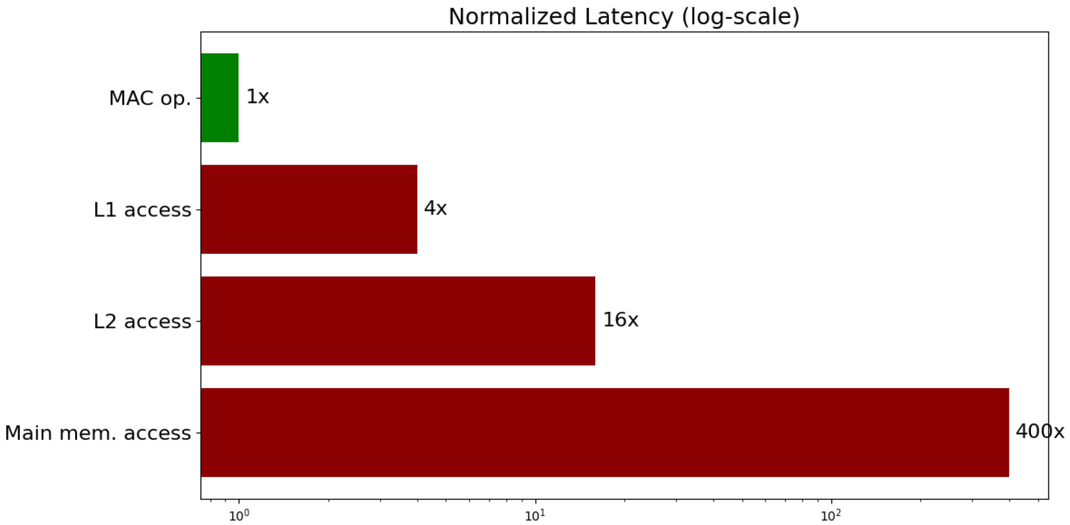
[Hennesy19]

Conclusion

Tensor Accelerators are essential to meet compute demands

Data movement is the primary bottleneck





Conclusion

Tensor Accelerators are essential to meet compute demands

Data movement is the primary bottleneck

Timeloop & Accelergy ecosystem enables fair comparison