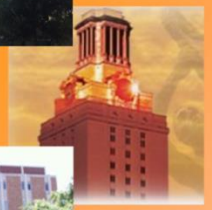
AI for Performance Engineering and Performance Engineering for AI

When AI meets Performance Engineering

Prof. Lizy K. John
Laboratory for Computer Architecture (LCA)
The University of Texas at Austin
ece.utexas.edu/~ljohn
lca.ece.utexas.edu

Thanks to University of Texas that gave me the chance to have colleagues such as 2019 Nobel winner, Prof. John Goodenough and 2023 Turing award winner Bob Metcalfe.

Laboratory for Computer Architecture



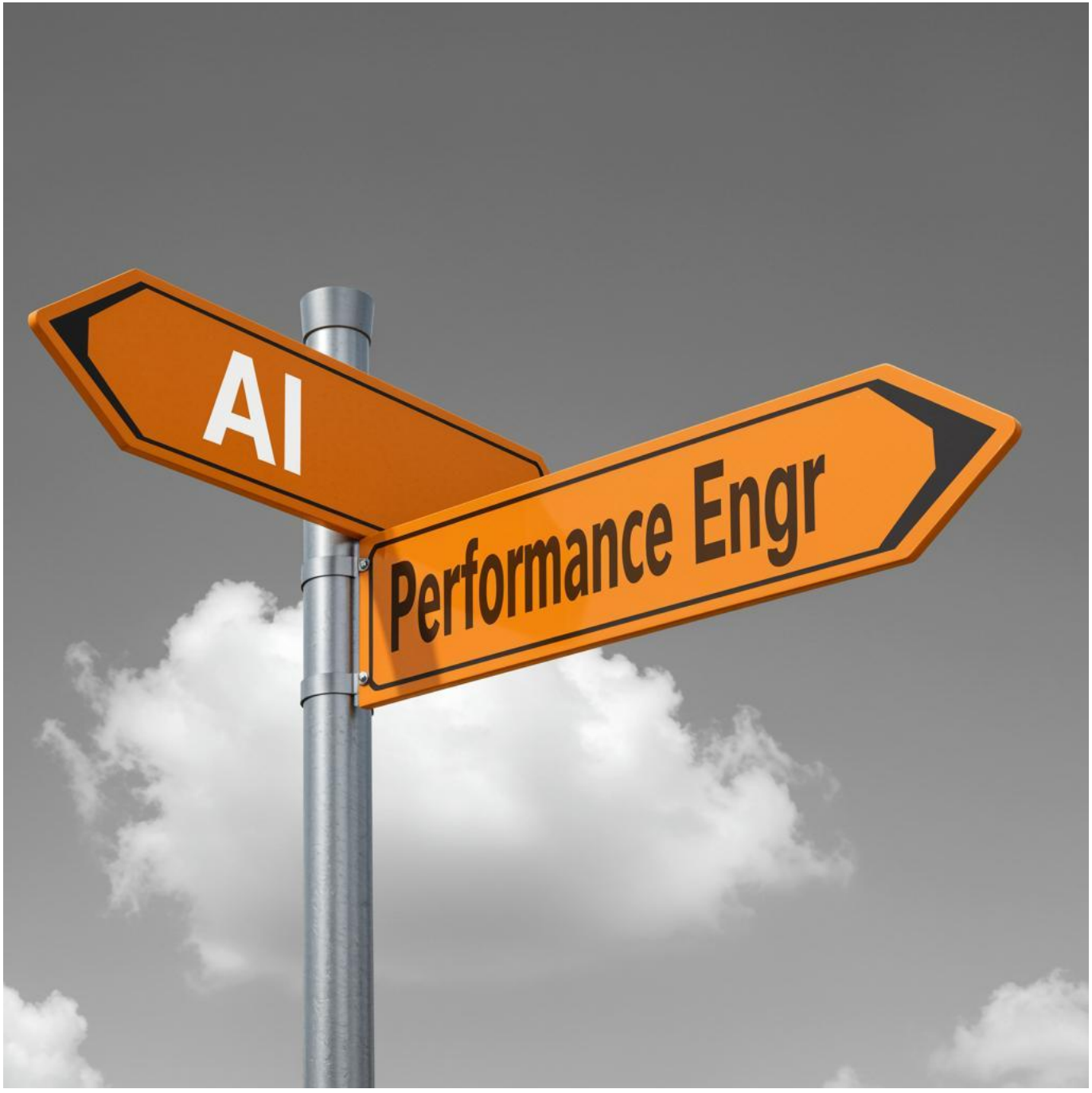
The Laboratory for Computer Architecture (LCA) is a research group within the Department of Electrical and Computer Engineering at The University of Texas at Austin. The lab is directed by Dr. Lizy Kurian John and is part of the Computer Engineering Research Center (CERC).

The members of the Laboratory for Computer Architecture are investigating several avenues in computer architecture. Some of our current research interests include:

- Cloud and Big Data Architecture
- Memory Systems for Multicore and Many- core Architectures
- Workload Characterization
- Proxies for Computer Performance/ Power Evaluation
- Low Power Architectures
- Development of Energy-efficient, High-Performance Codes
- Compiler Support for Innovative Micro-architectures

UT ECE

LCA research was supported in part by
Semiconductor Research Corporation (SRC) Tasks 3015.001 and 3148.001,
National Science Foundation (NSF) Grant #2326894,
and NVIDIA Applied Research Accelerator Program Grant.


AI
Performance Engr

When AI meets Performance Engineering

1. Performance Engineering for AI

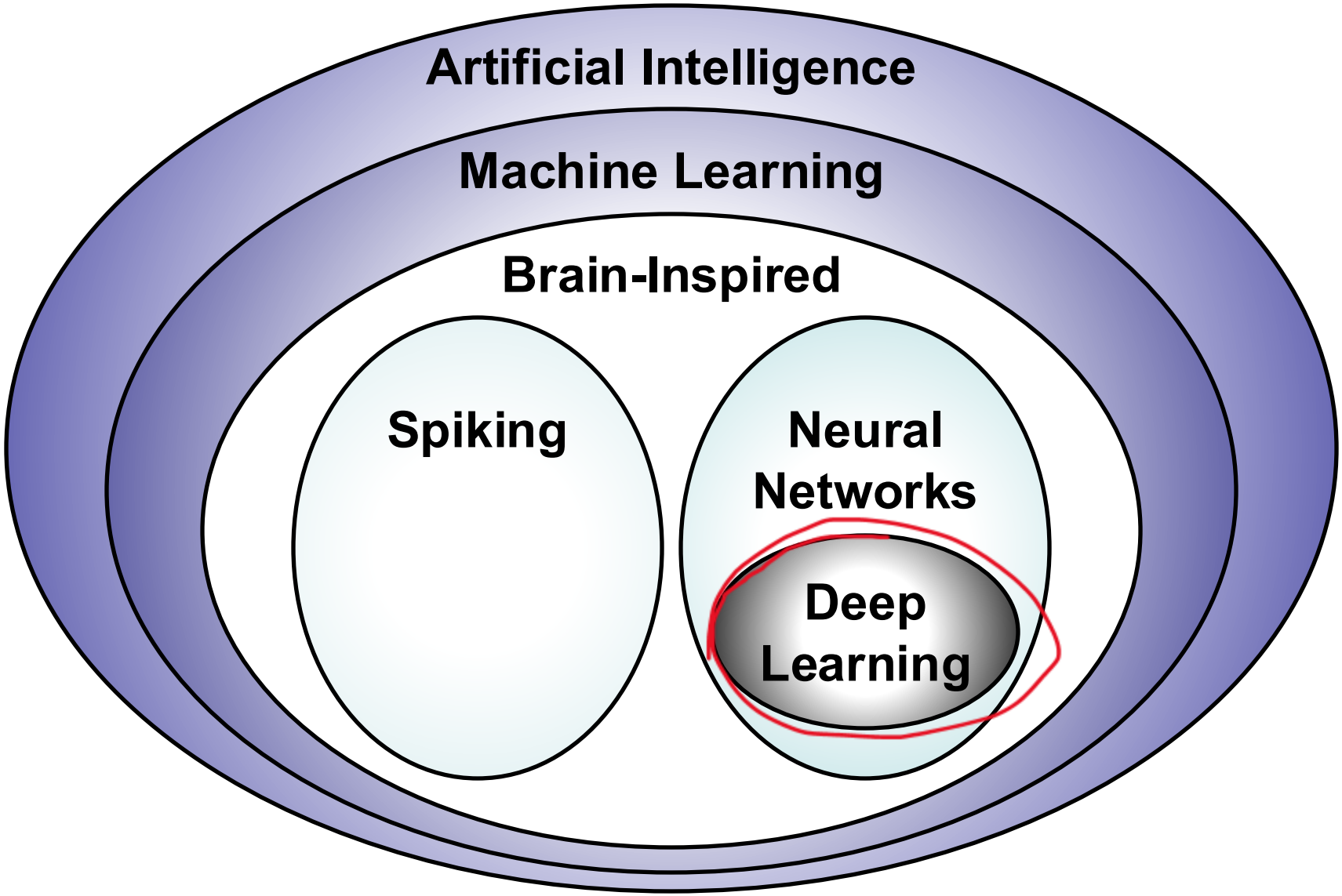
2. AI for Performance Engineering

Performance Engineering for AI

What is the AI workload of today?

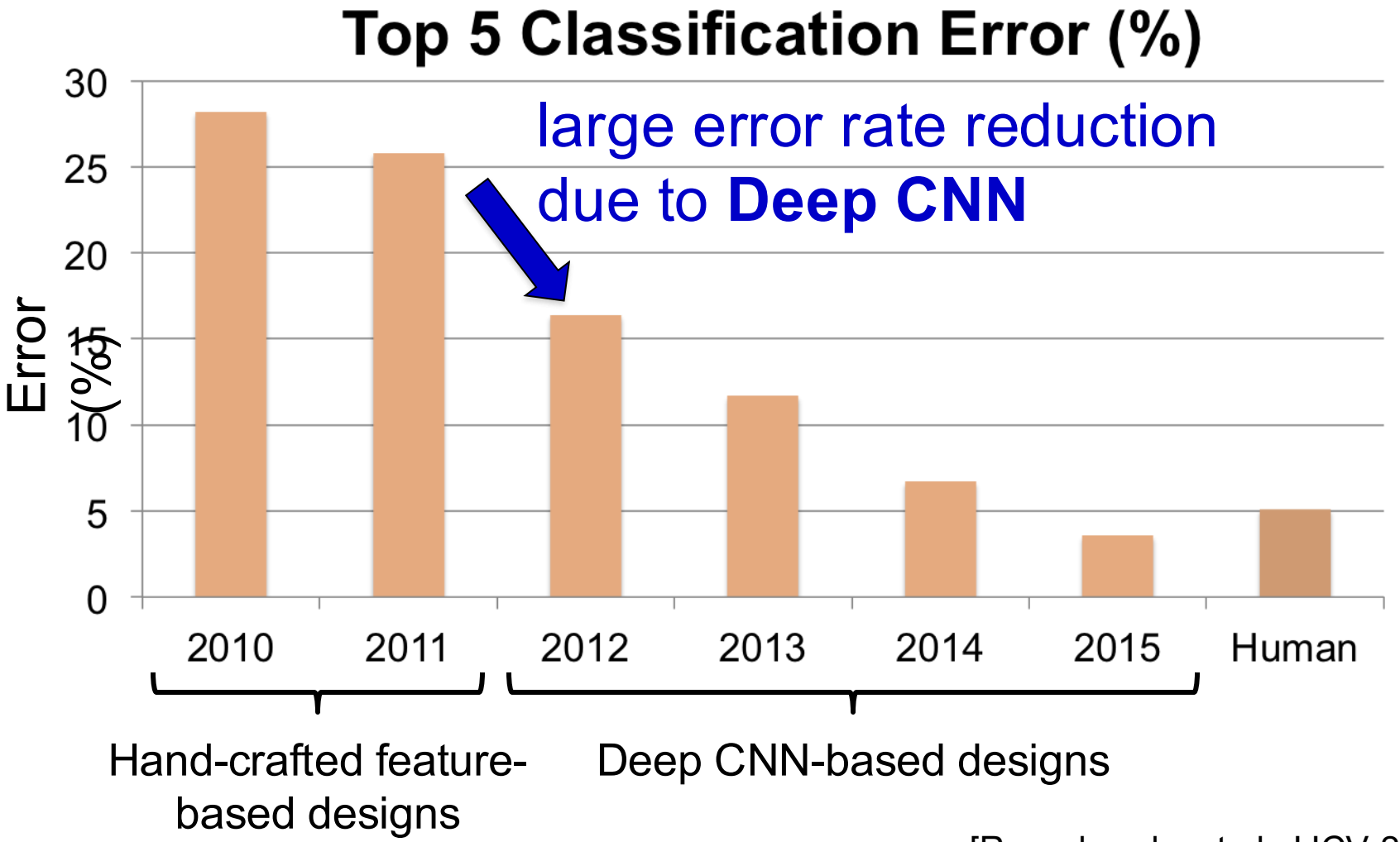
How do we do performance engineering for AI?

What is the AI workload of today?



Slide Courtesy: Joel Emer and Vivienne Sze

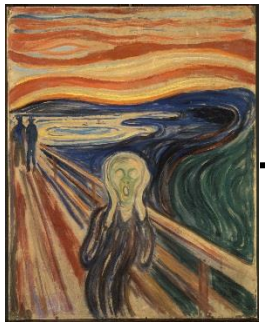
ImageNet: Image Classification Task



[Russakovsky et al., IJCV 2015]

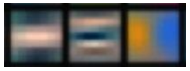
Slide Courtesy: Joel Emer and Vivienne Sze

Convolutional Neural Networks



CONV Layer

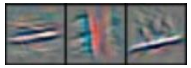
Low-Level Features



...

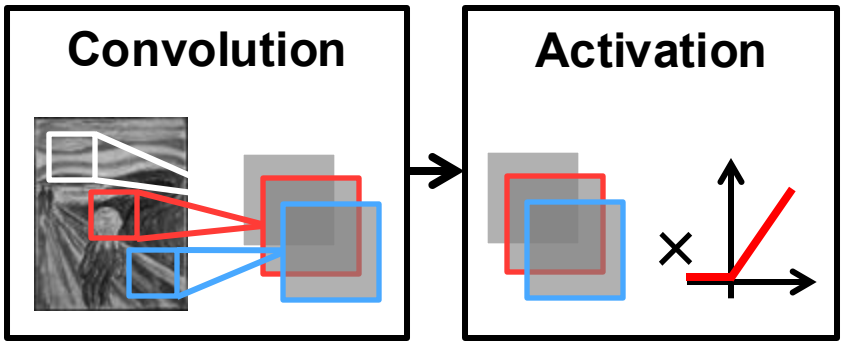
CONV Layer

High-Level Features

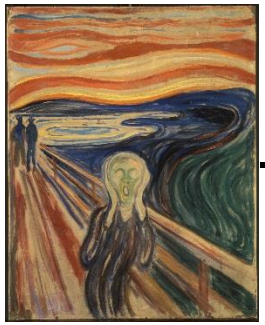


FC Layer

Classes

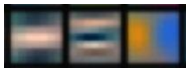


Convolutional Neural Networks



CONV Layer

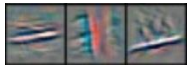
Low-Level Features



…

CONV Layer

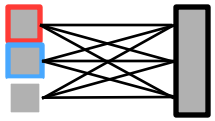
High-Level Features



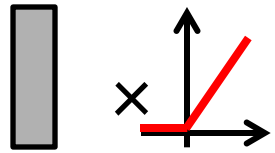
FC Layer

Classes

Fully Connected



Activation



Convolution Code – Nested Loops

7th (outermost) loop used during training

```
for(n=0; n<N; n++) { // Input feature maps (IFMaps)
  for(m=0; m<M; m++) { // Weight Filters
    for(c=0; c<C; c++) { // IFMap/Weight Channels
      for(y=0; y<H; y++) {  // Input feature map row
        for(x=0; x<H; x++) {  // Input feature map column
          for(j=0; j<R; j++) {  // Weight filter row
            for(i=0; i<R; i++) {  // Weight filter column
              O[n][m][x][y] += W[m][c][i][j] * I[n][c][y][x]}}}}}}}
```

Performance Engineering for AI

But is Convolution the popular AI workload of today?

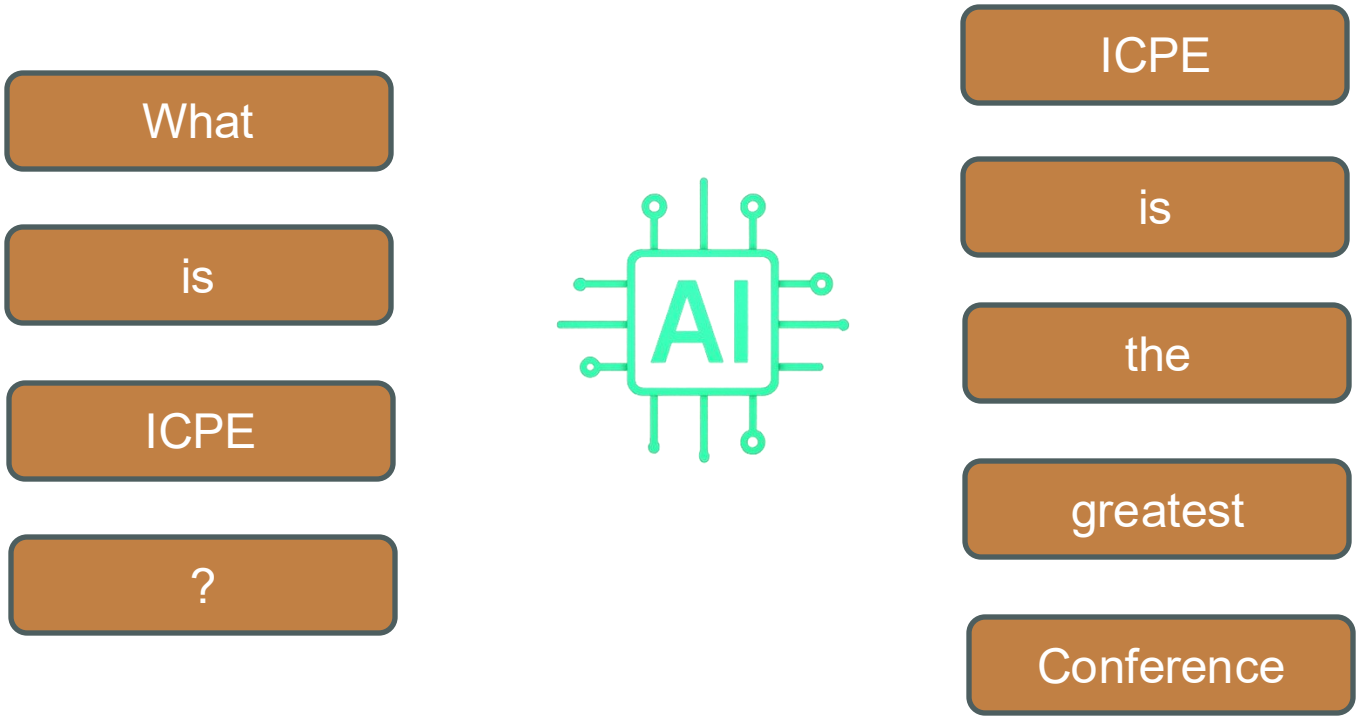
No.

It is GenAI.

It is ChatGPT and DeepSeek and …..

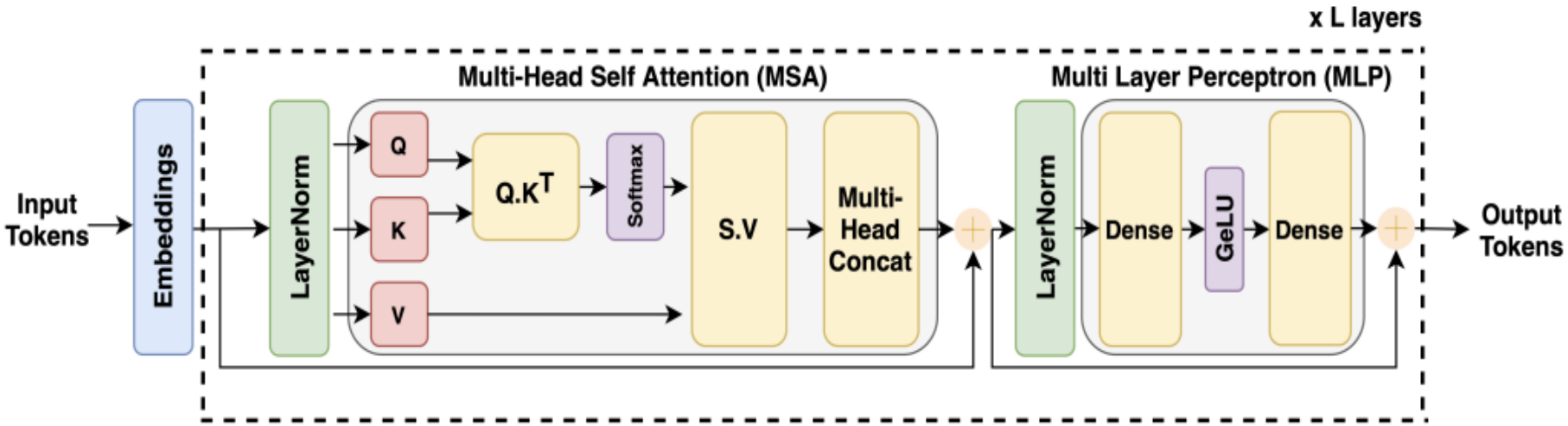
Tokens the LLM Input Building Blocks

Enables handling diverse and complex text.

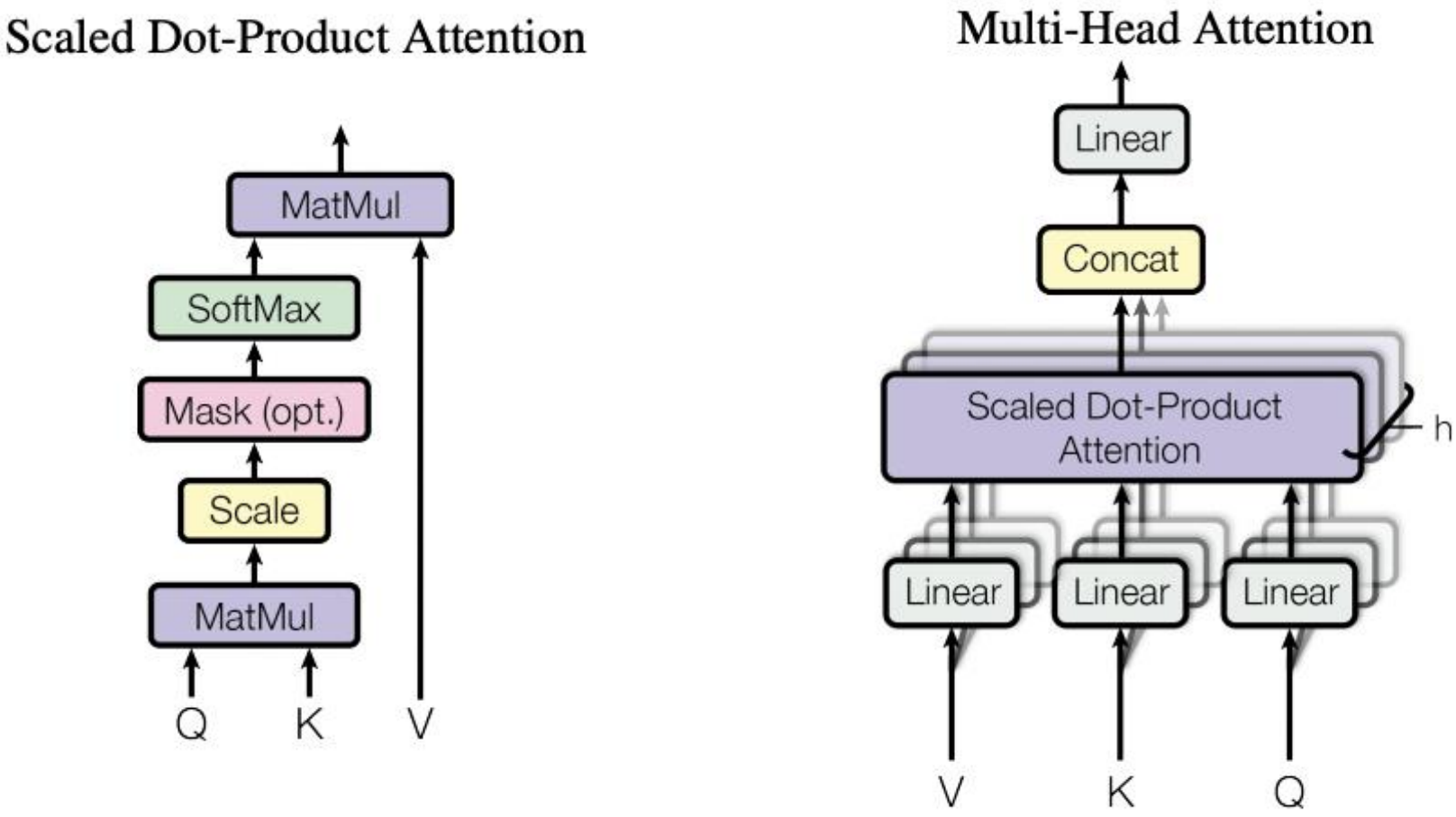


Transformers

- Tokenization
- Attention and MLP layers

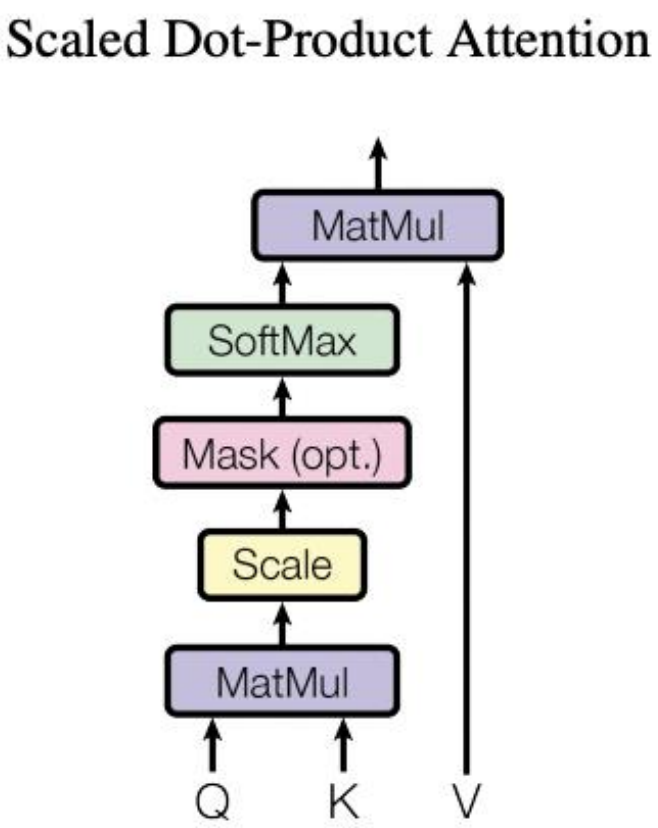


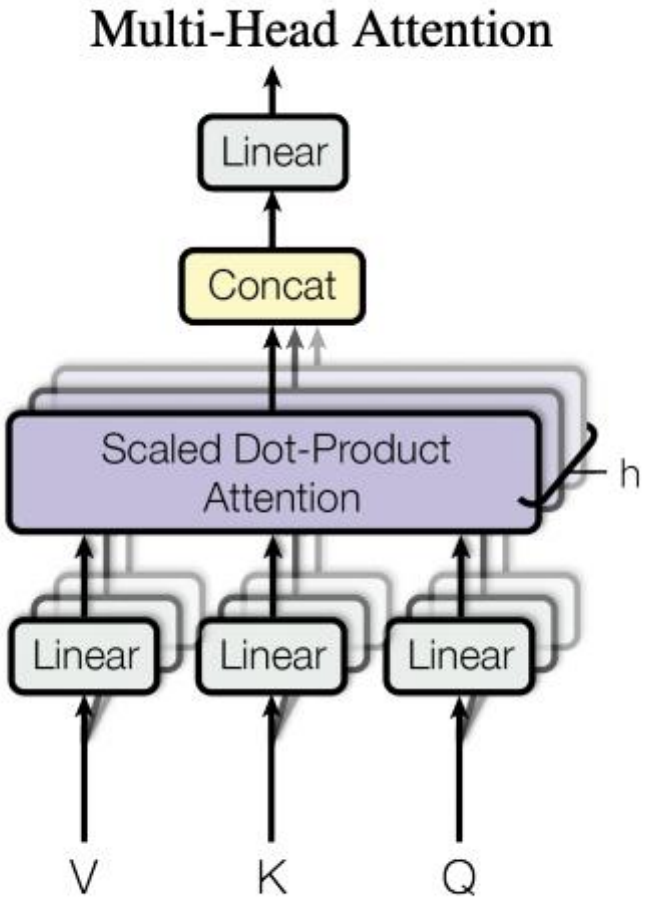
Gen AI and Transformers



Attention is all you need

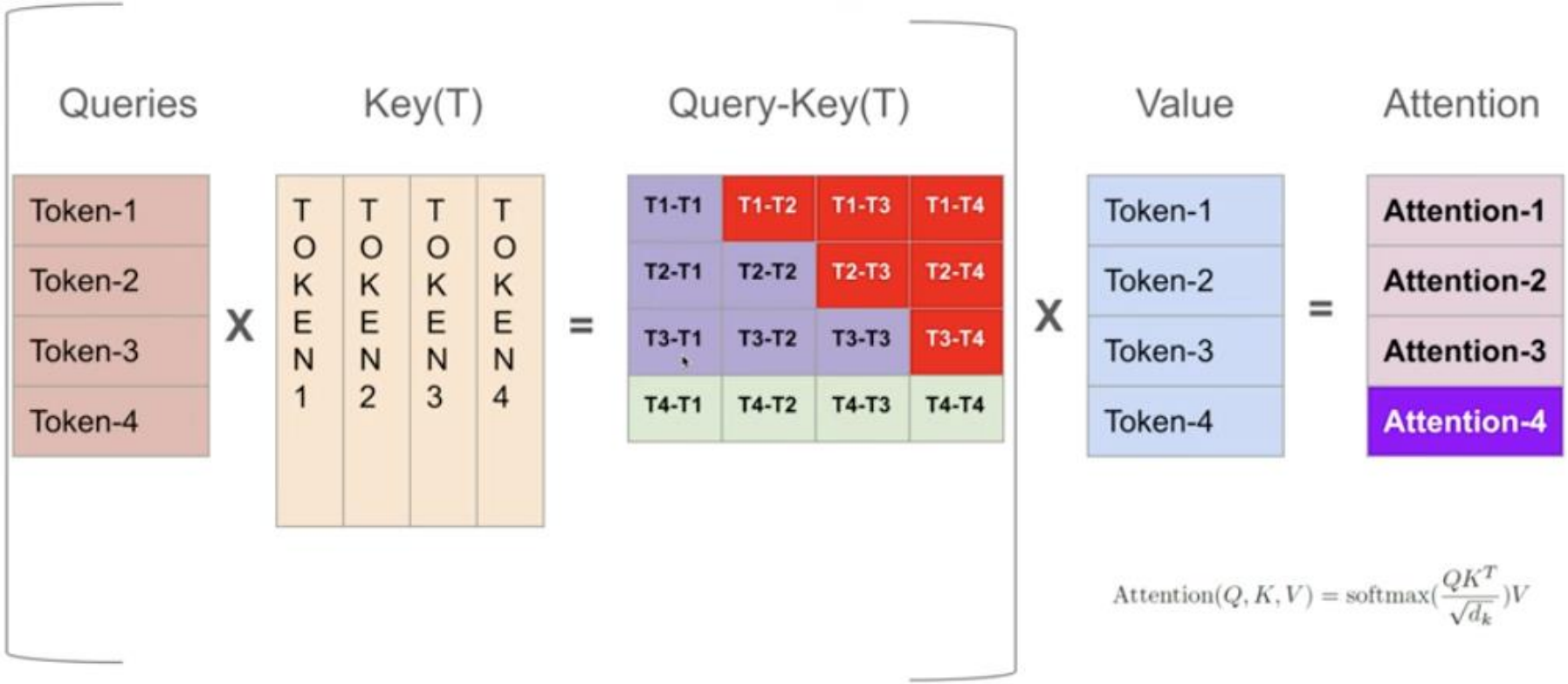
Gen AI and Transformers





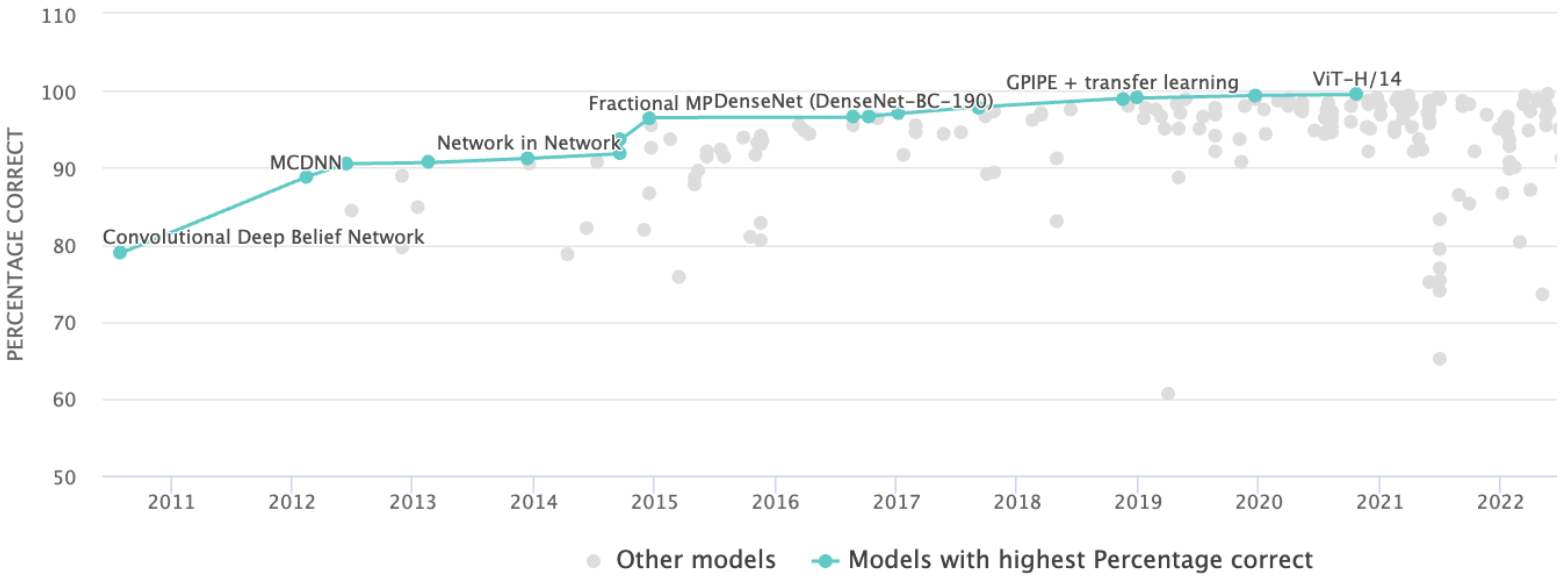
What is the dominant operation in GenAI?

What is the dominant operation in GenAI?



Transformers are not only for LLMs

Vision Transformers (ViTs) occupy leaderboard position for Vision, Object Recognition/Classification Tasks.



Tokens the New Unit of Work

Tokens/Second Valuebar

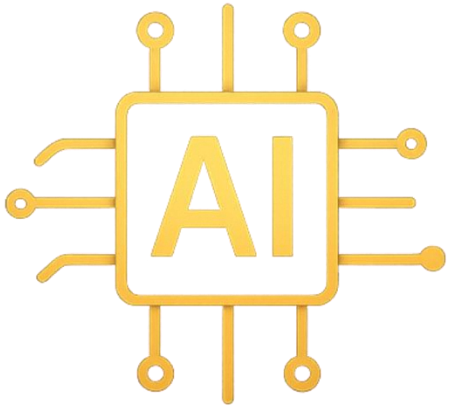
2 tokens/sec

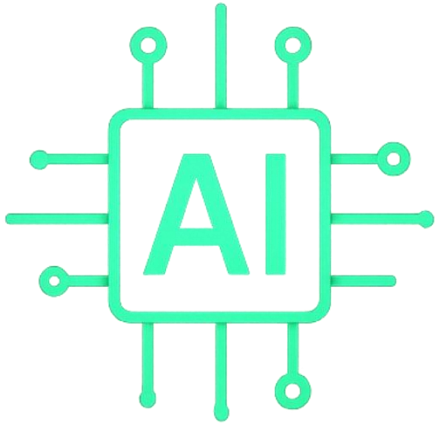
100 tokens/sec

60,000 tokens/second at 1 ms/token latency
for Llama3 8B in a single server

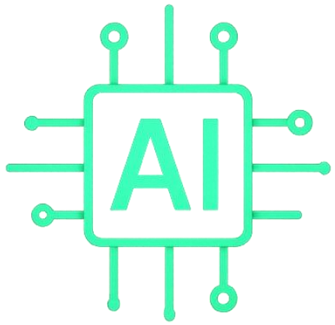
30,000 tokens/second at 2 ms/token latency
for Llama3 70B in a single rack

Power: Watts & Battery Life Matter





Holistic View of Systems



Tokens/Second

Cost

Battery

On What Benchmarks?

When you analyze LLMs

All Roads Lead to GEMM

Batch size	1			
Stages	Matrix Multiplication			No of Macs
	M	K	N	~M*K*N
Tokenization and Word Embeddings				
One Hot Mat * Embedded Weight Mat	2048	51,200	12288	1.28849E+12
Positional Encoding				
Word Embedding Mat+ Positional Encoded Ma	2048	12288		
Muli Head Attention Block				
	Number of Blocks		96	
X*WQ = Q	2048	12288	128	3221225472
X*WK = K	2048	12288	128	3221225472
X*WV = V	2048	12288	128	3221225472
Q*KT = QK	2048	12288	2048	51539607552
Softmax	2048	2048		
QK*V	2048	2048	128	536870912
Concate Heads	2048	12288		
Linear Tranformation	2048	12288	12288	3.09238E+11
Feed forward Neural Network				
Linear Tranformation + Bias	2048	12288	49152	1.23695E+12
Linear Transformation + Bias	2048	49152	12288	1.23695E+12
FFN + Input	2048	12288		
Normalize	2048	12288		
Decoding				
	2048	12288	51200	1.28849E+12
Softmax	2048	51200		
Output	2048	51200		
Total Model Parameters	1.7461E+11			

But GEMMs are highly sensitive to Optimizations

GEMM using C

```
void multiply_matrices(float A[N][N], float
B[N][N], float C[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            C[i][j] = 0;
            for (int k = 0; k < N; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

- AMD EPYC 7742 64-Core Processor
- Single thread
- gcc 11.4.0, -O1
- 478.89 s

C Loop –O3 on 64-core

```
void multiply_matrices(float A[N][N], float
B[N][N], float C[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            C[i][j] = 0;
            for (int k = 0; k < N; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

- AMD EPYC 7742 64-Core Processor
- Single thread
- gcc 11.4.0, **-O3**
- 262.83 s
- Speedup: 1.82x

Python with Numpy

```
A = np.random.rand(M, K)
B = np.random.rand(K, N)
# Perform GEMM
for i in range(iter):
    C = np.matmul(A, B)
```

- AMD EPYC 7742 64-Core Processor
- Multi-thread
- 1000 iterations
- 276.97 ms per iteration
- Speedup: 1729x

Python with Torch

```
A = torch.randn(M, K).to(device)
B = torch.randn(K, N).to(device)
# Perform GEMM
for i in range(iter):
    C = torch.matmul(A, B)
```

- AMD EPYC 7742 64-Core Processor
- Multi-thread
- 1000 iterations
- 49.79 ms per iteration
- Speedup: 9618x

Hardware	Config	Language/Library	Latency (ms)	Speedup
AMD EPYC	Single Thread	C –O1 C –O3	478890 262830	1 1.82x
	MultiThread	Py (NumPy)	276.97	1729x
		Py (Torch)	49.79	9618x

On GPU with CUDA

```
template <int BLOCK_SIZE, typename T> __global__ void
MatrixMulCUDA(T *C, T *A,
    T *B, int wA,
    int wB) {
  ...

  T Csub = 0;

  for (int a = aBegin, b = bBegin;
       a <= aEnd;
       a += aStep, b += bStep) {
    __shared__ T As[BLOCK_SIZE][BLOCK_SIZE];
    __shared__ T Bs[BLOCK_SIZE][BLOCK_SIZE];

    As[ty][tx] = A[a + wA * ty + tx];
    Bs[ty][tx] = B[b + wB * ty + tx];

    __syncthreads();

#pragma unroll

    for (int k = 0; k < BLOCK_SIZE; ++k) {
      Csub += As[ty][k] * Bs[k][tx];
    }

    __syncthreads();
  }

  int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;
  C[c + wB * ty + tx] = Csub;
}
```

- Nvidia A100 GPU
- CUDA Core
- 1000 iterations with warm-up
- 23.82ms per iteration

On GPU - Python with Torch

```
A = torch.randn(M, K).to(device)
B = torch.randn(K, N).to(device)
# Perform GEMM
for i in range(iter):
    C = torch.matmul(A, B)
torch.cuda.synchronize()
```

- **Nvidia A100 GPU**
- FP32-> Cuda Core
- 1000 iterations with warm-up
- 10.94 ms per iteration

CUDA with CuBLAS on CUDA cores

```
for(int iter=0;iter<gnum_iter;iter++)
   {
      gpuErrchk(cublasGemmEx(handle,                  // handle to cuBLAS
library context
                        matA_op,                 // CUBLAS_OP_N,
CUBLAS_OP_T, CUBLAS_OP_C
                        matB_op,                 // CUBLAS_OP_N,
CUBLAS_OP_T, CUBLAS_OP_C
                        gdim_M,                  // dimension M
                        gdim_N,                  // dimension N
                        gdim_K,                  // dimension K
                        &alpha,                  // Scaling factor alpha where
(alpha)x(AxB)
                        dev_matA,                // Pointer to Matrix A on Device
                        mulDataType,             // Data type of Matrix A
                        gdim_M,                  // Leading Dimension of Matrix A
                        dev_matB,                // Pointer to Matrix B on Device
                        mulDataType,             // Data Type of Matrix B
                        gdim_K,                  // Leading Dimension of Matrix B
                        &beta,                   // Scaling factor beta where (beta)xC
                        dev_matC,                // Pointer to Matrix C on Device
                        accDataType,             // Data Type of Matrix C
                        gdim_M,                  // Leading Dimension of Matrix C
                        computeType,             // Computation Type
                        algoType                 // Computation Algorithm
      ));
   }
```

- Nvidia A100 GPU
- CUDA Core
- 1000 iterations with warm up
- 7.14ms per iteration
- Speedup: 3.34x

CUDA with Cutlass

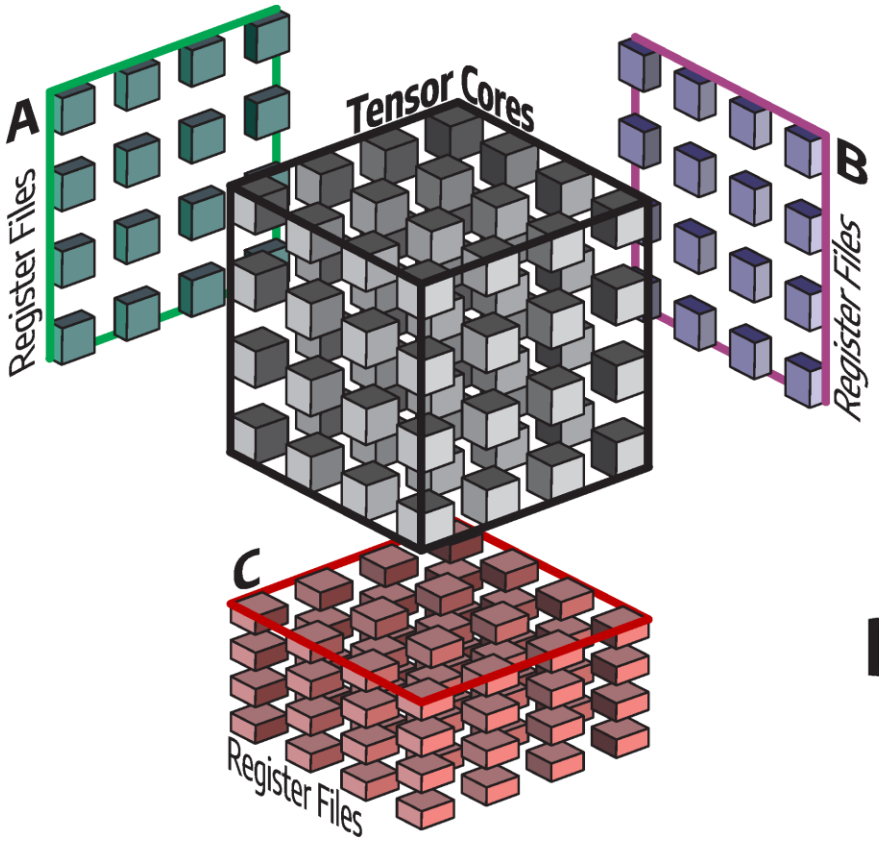
```
#include <cutlass/cutlass.h>
#include <cutlass/gemm/device/gemm.h>
// Prepare launch arguments and extra device memory for matrix multiplication
   typename Gemm::Arguments arguments{problem_dim,  // <- problem size of
matrix multiplication
                                      {dev_matA, gdim_K},  // <- reference to matrix A on
device //MxK
                                      {dev_matB, gdim_K},  // <- reference to matrix B on
device //KxN
                                      {dev_matC, gdim_M},  // <- reference to matrix C on
device //MxN
                                      {dev_matC, gdim_M},  // <- reference to matrix D on
device
                                      {alpha, beta}, // <- tuple of alpha and beta
                                      1};          // <- k-dimension split factor
   …

   // Instantiate CUTLASS kernel depending on templates
   Gemm gemm_op;
   …

   // Launch initialized CUTLASS kernel
   for(int iter=0;iter<gnum_iter;iter++)
   {
      gpuErrchk(gemm_op());
   }
   gpuErrchk(cudaDeviceSynchronize());
```

- Nvidia A100 GPU
- CUDA Core
- 1000 iterations with warm-up
- 5.28ms per iteration
- Speedup: 4.51x

Tensor Cores in GPUs



Dedicated unit for accelerating GEMM

$$D = \begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}_{A} \times \begin{bmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{bmatrix}_{B} + \begin{bmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{bmatrix}_{C}$$

Python with Torch on TensorCores

```
A = torch.randn(M, K).to(device)
B = torch.randn(K, N).to(device)
# Perform GEMM
for i in range(iter):
    C = torch.matmul(A, B)
torch.cuda.synchronize()
```

- **Nvidia A100 GPU**
- FP16 -> Tensor Core
- 1000 iterations with warm-up
- 0.64 ms per iteration
- 36.85x Speedup

CUDA with CuBLAS on TensorCores

```
for(int iter=0;iter<gnum_iter;iter++)
   {
      gpuErrchk(cublasGemmEx(handle,                  // handle to cuBLAS
library context
                  matA_op,              // CUBLAS_OP_N, CUBLAS_OP_T,
CUBLAS_OP_C
                  matB_op,              // CUBLAS_OP_N, CUBLAS_OP_T,
CUBLAS_OP_C
                  gdim_M,               // dimension M
                  gdim_N,               // dimension N
                  gdim_K,               // dimension K
                  &alpha,               // Scaling factor alpha where
(alpha)x(AxB)
                  dev_matA,             // Pointer to Matrix A on Device
                  mulDataType,          // Data type of Matrix A
                  gdim_M,               // Leading Dimension of Matrix A
                  dev_matB,             // Pointer to Matrix B on Device
                  mulDataType,          // Data Type of Matrix B
                  gdim_K,               // Leading Dimension of Matrix B
                  &beta,                // Scaling factor beta where (beta)xC
                  dev_matC,             // Pointer to Matrix C on Device
                  accDataType,          // Data Type of Matrix C
                  gdim_M,               // Leading Dimension of Matrix C
                  computeType,          // Computation Type
                  algoType              // Computation Algorithm
      ));
   }
```

- Nvidia A100 GPU
- **Tensor Core**
- 1000 iterations with warm-up
- 0.58ms per iteration
- Speedup: 41.06x

CUDA with Cutlass on TensorCores

```
#include <cutlass/cutlass.h>
#include <cutlass/gemm/device/gemm.h>
// Prepare launch arguments and extra device memory for matrix multiplication
   typename Gemm::Arguments arguments{problem_dim,  // <- problem size of
matrix multiplication
                        {dev_matA, gdim_K},  // <- reference to matrix A on device
//MxK
                        {dev_matB, gdim_K},  // <- reference to matrix B on device
//KxN
                        {dev_matC, gdim_M},  // <- reference to matrix C on
device //MxN
                        {dev_matC, gdim_M},  // <- reference to matrix D on
device
                        {alpha, beta}, // <- tuple of alpha and beta
                        1};          // <- k-dimension split factor
   …

   // Instantiate CUTLASS kernel depending on templates
   Gemm gemm_op;
   …

   // Launch initialized CUTLASS kernel
   for(int iter=0;iter<gnum_iter;iter++)
   {
      gpuErrchk(gemm_op());
   }
   gpuErrchk(cudaDeviceSynchronize());
```

- Nvidia A100 GPU
- **Tensor Core**
- 1000 iterations with warm-up
- 0.58ms per iteration
- Speedup: 41.06x

GEMM Performance Summary

Hardware	Config	Language/Library	Latency (ms)	Speedup
AMD EPYC	Single Thread	C –O1 C –O3	478890 262830	1 1.82x
	MultiThread	Py (NumPy)	276.97	1729x
		Py (Torch)	49.79	9618x
NVIDIA A100 GPU	CUDA Cores	CUDA	23.82	20104 (1)
		Py Torch	10.94	43774 (2.18)
		CUDA CuBLAS	7.14	67071 (3.34)
		Cutlass	5.28	90669 (4.51)
	Tensor Cores	Py Torch	0.64	740857 (36.85)
		Cublas/Cutlass	0.58	825672 (41.06)

Too sensitive

Performance Engineers need to be careful!

SPEC dropped matrix300 from SPEC 89

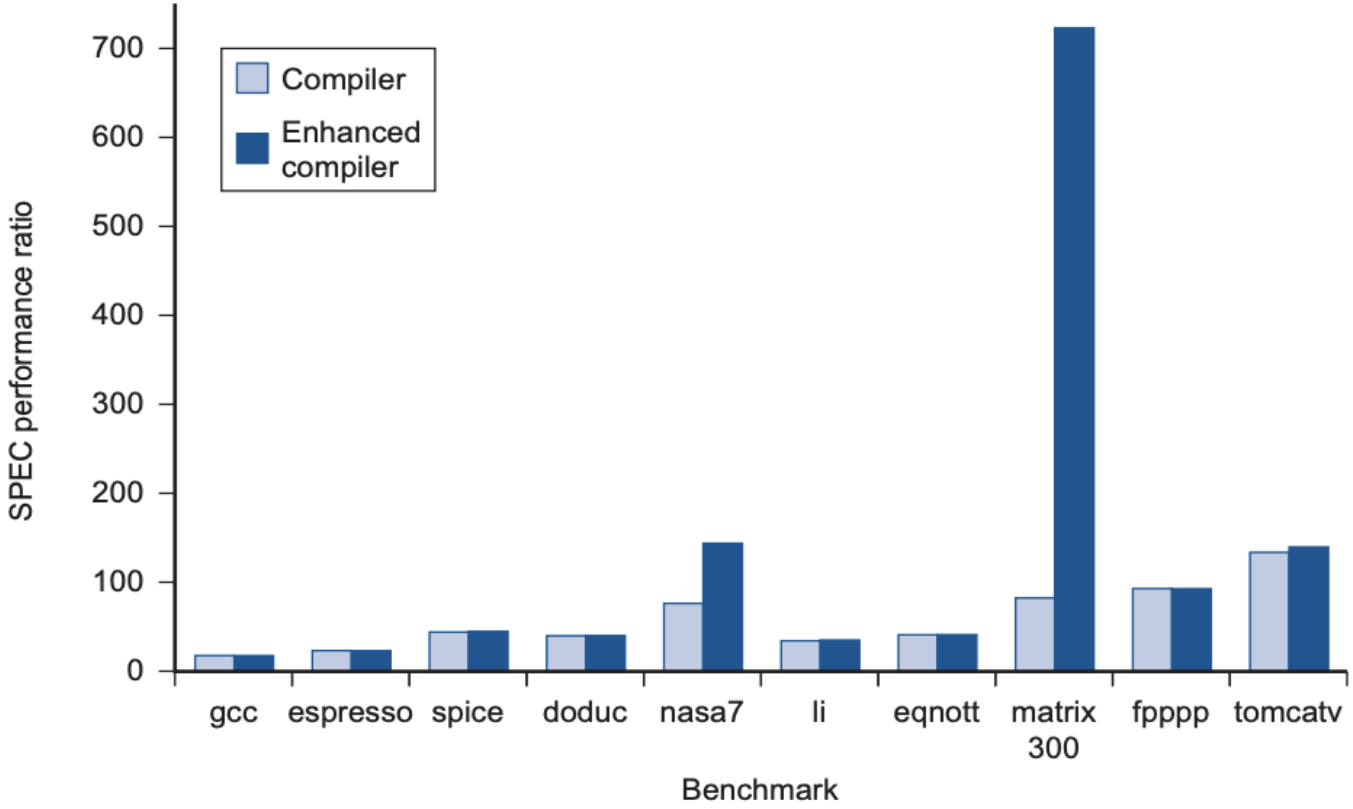
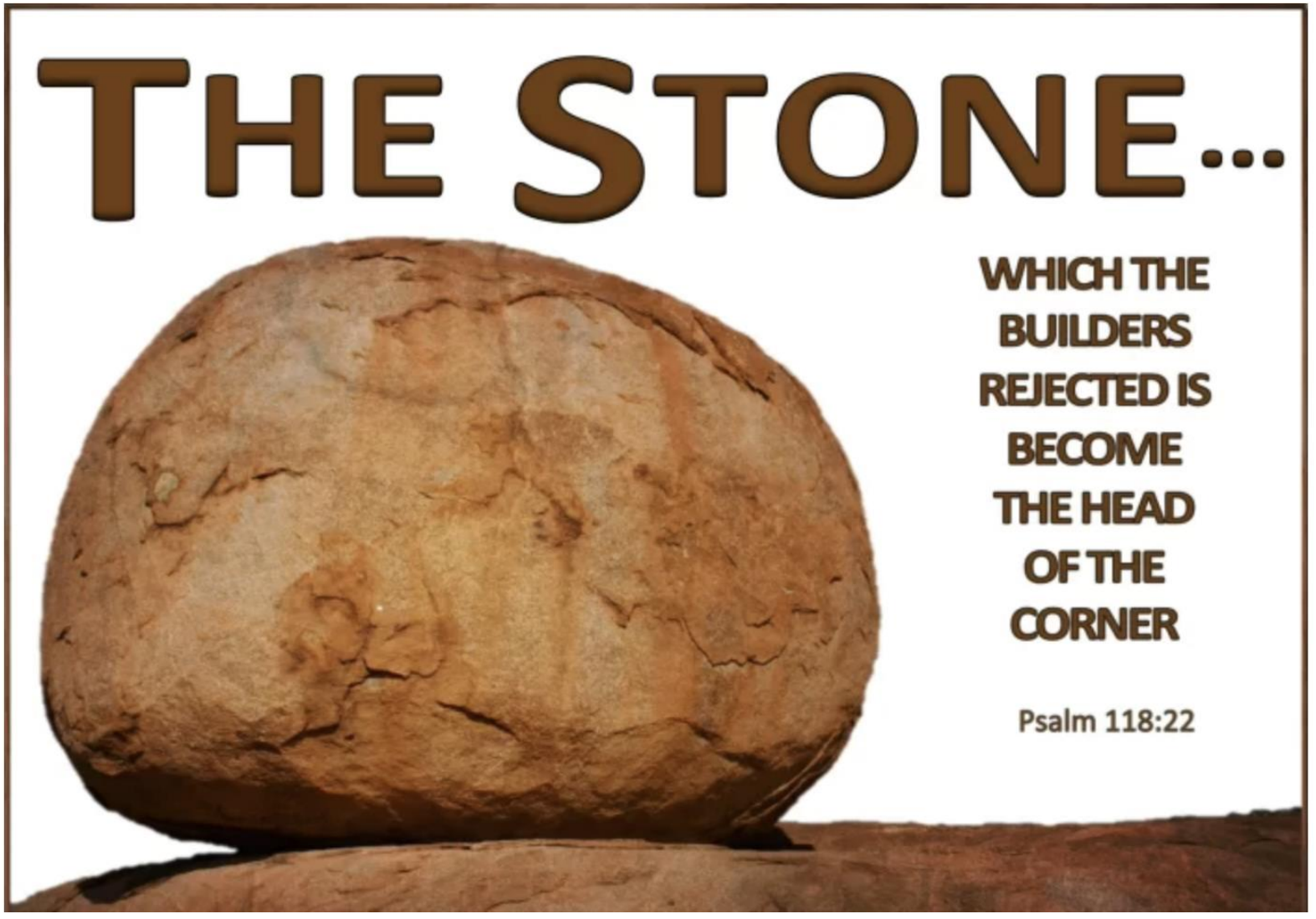


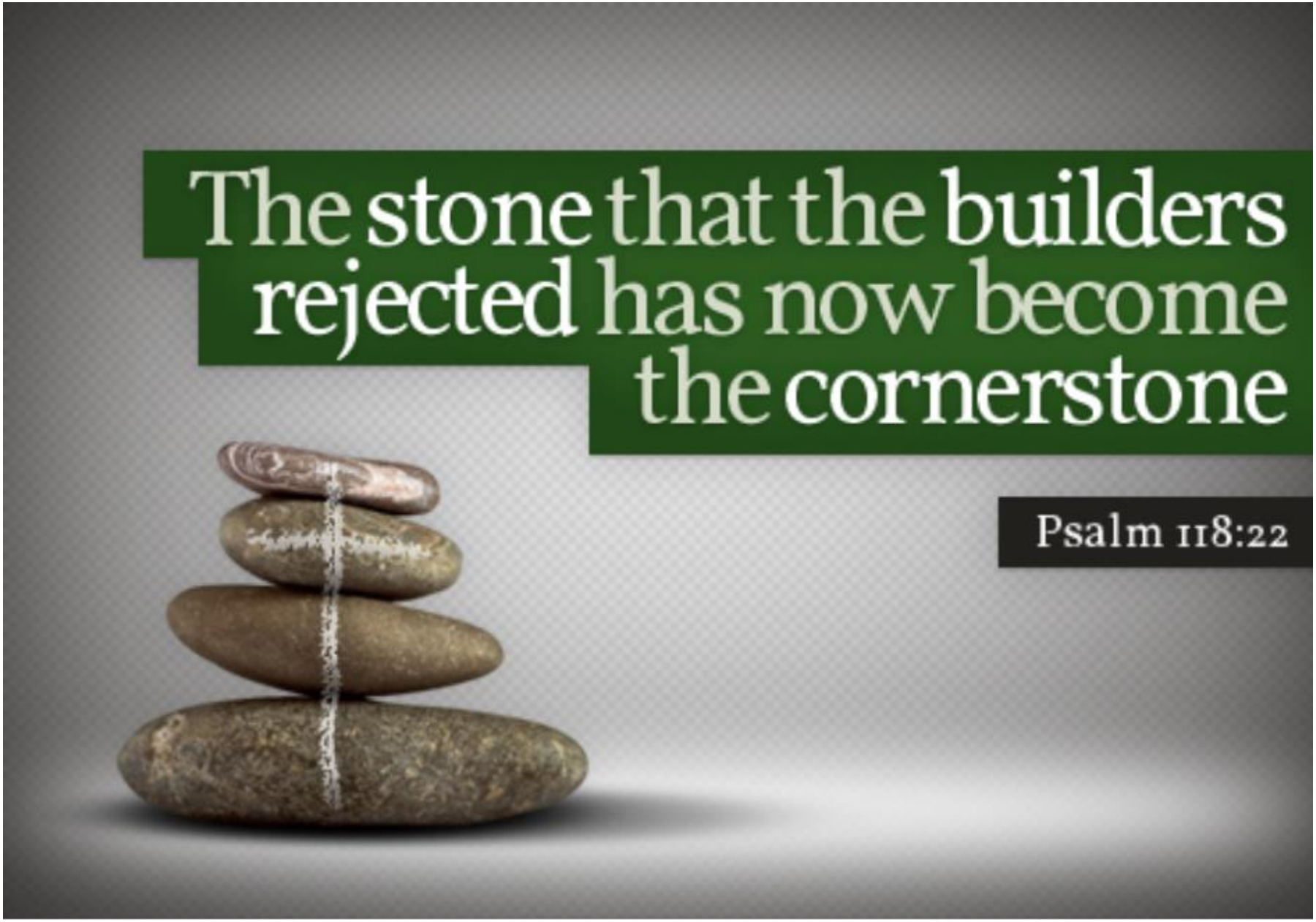
FIGURE 4.8.6 SPEC89 performance ratios for the IBM Powerstation 550 using two different compilers. The higher numbers on matrix300 (and nasa7) result from applying an optimization technique to these two kernel-oriented benchmarks. For the enhanced compiler, special flags are passed to the compiler for both nasa7 and matrix300, which are not used for the other benchmarks. In both programs, the compiler transforms the program by blocking the matrix operations that are in the inner loops. These blocking transformations substantially lower the number of memory accesses required and transform the inner loops from having high cache miss rates to having almost negligible cache miss rates. Interestingly, the original motivation for including matrix300 was to exercise the computer's memory system; however, this optimization basically reorganizes the program to minimize memory usage. This data appeared in two SPEC reports during the fall and winter of 1991. The susceptibility of this benchmark to compiler optimization, and the relatively uninteresting behavior of the benchmark after optimization, led to the elimination of matrix300 from the 1992 release of the SPEC benchmarks.

Is GEMM a reasonable Benchmark if ChatGPT is a Sequence of MatMuls?

Batch size	1			
Stages	Matrix Multiplication			No of Macs
	M	K	N	~M*K*N
Tokenization and Word Embeddings				
One Hot Mat * Embedded Weight Mat	2048	51,200	12288	1.28849E+12
Positional Encoding				
Word Embedding Mat+ Positional Encoded Ma	2048	12288		
Muli Head Attention Block				
	Number of Blocks		96	
X*WQ = Q	2048	12288	128	3221225472
X*WK = K	2048	12288	128	3221225472
X*WV = V	2048	12288	128	3221225472
Q*KT = QK	2048	12288	2048	51539607552
Softmax	2048	2048		
QK*V	2048	2048	128	536870912
Concate Heads	2048	12288		
Linear Tranformation	2048	12288	12288	3.09238E+11
Feed forward Neural Network				
Linear Tranformation + Bias	2048	12288	49152	1.23695E+12
Linear Transformation + Bias	2048	49152	12288	1.23695E+12
FFN + Input	2048	12288		

SPEC dropped matrix300 from SPEC89 but MATMUL is the key computation in LLMs




The stone that the builders rejected has now become the cornerstone
Psalm 118:22

The Benchmark Question is Complicated

Kernel Benchmarking may not work

Application Based AI Benchmarking

MLPerf and MLCommons

mlcommons.org/benchmarks/

ML Commons

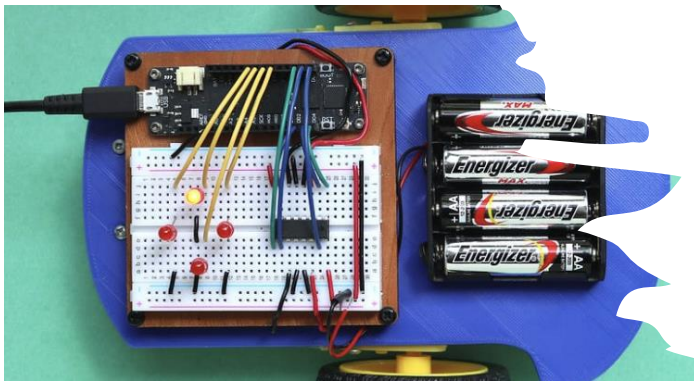
Benchmarks | Working Groups | AILuminate | Datasets | About | Insights | Get Involved

Benchmarks

Delivering open, useful measures of quality, performance and safety to help guide responsible AI development.

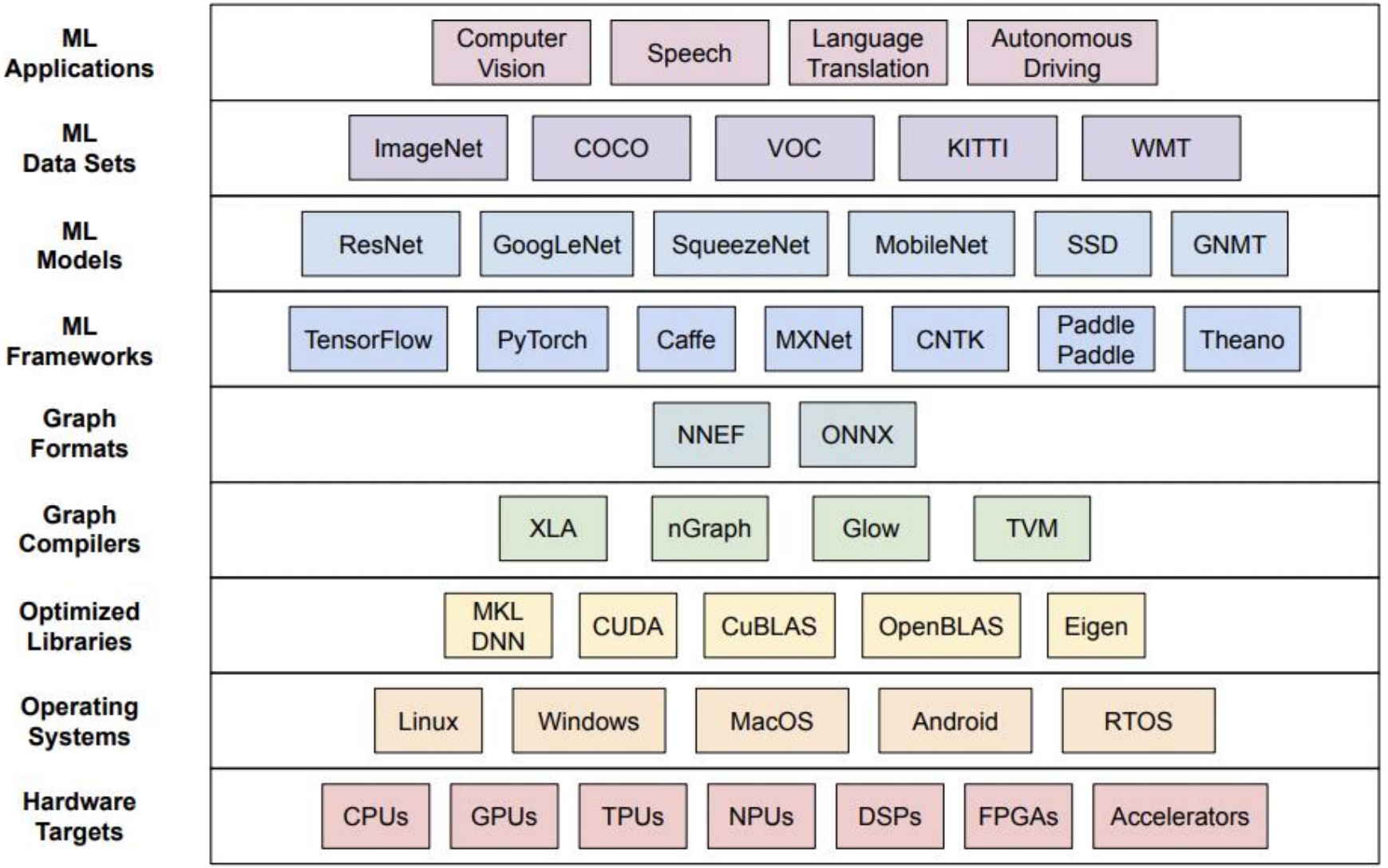
About MLCommons Benchmarks

The foundation for MLCommons benchmark work was derived from and builds upon MLPerf which aims to deliver a representative benchmark suite for AI/ML that fairly evaluates system performance to meet five high-level goals:

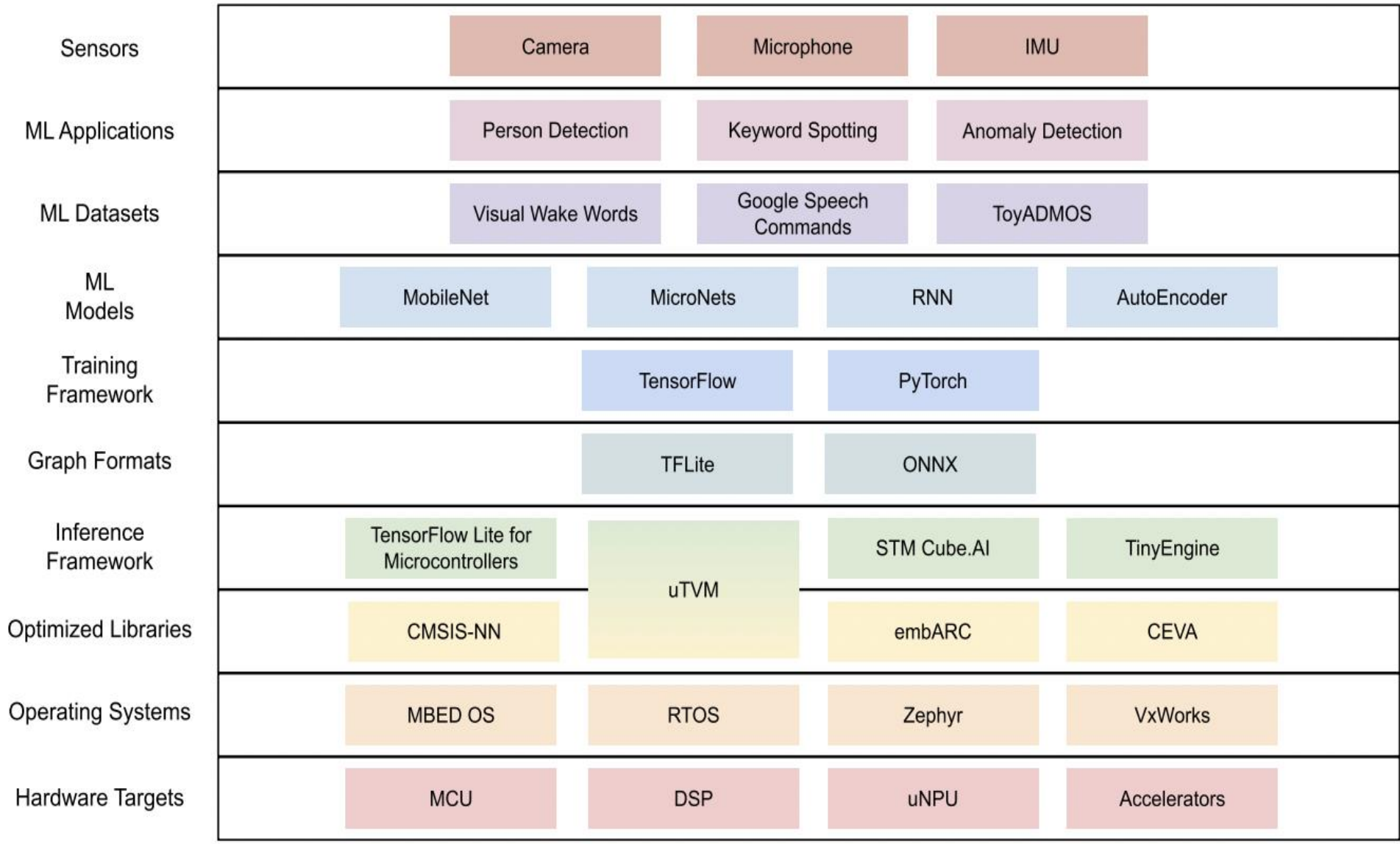


Range of Hardware

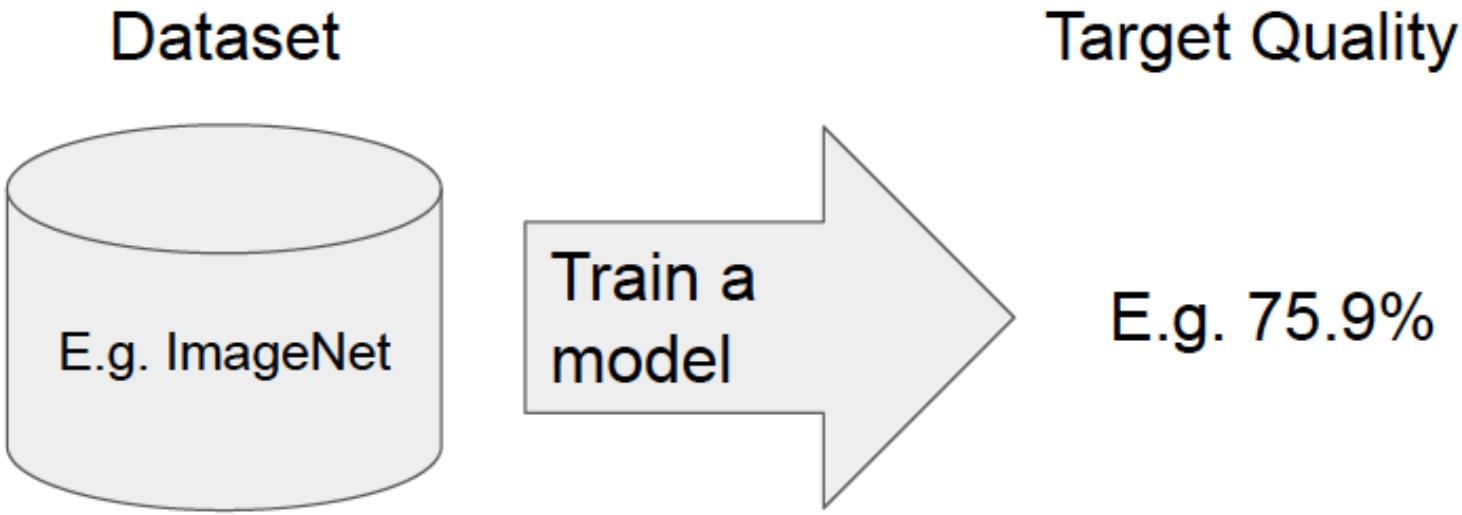
Developing Benchmarks for AI is Challenging



Diversity of Options Even for Tiny and Edge AI

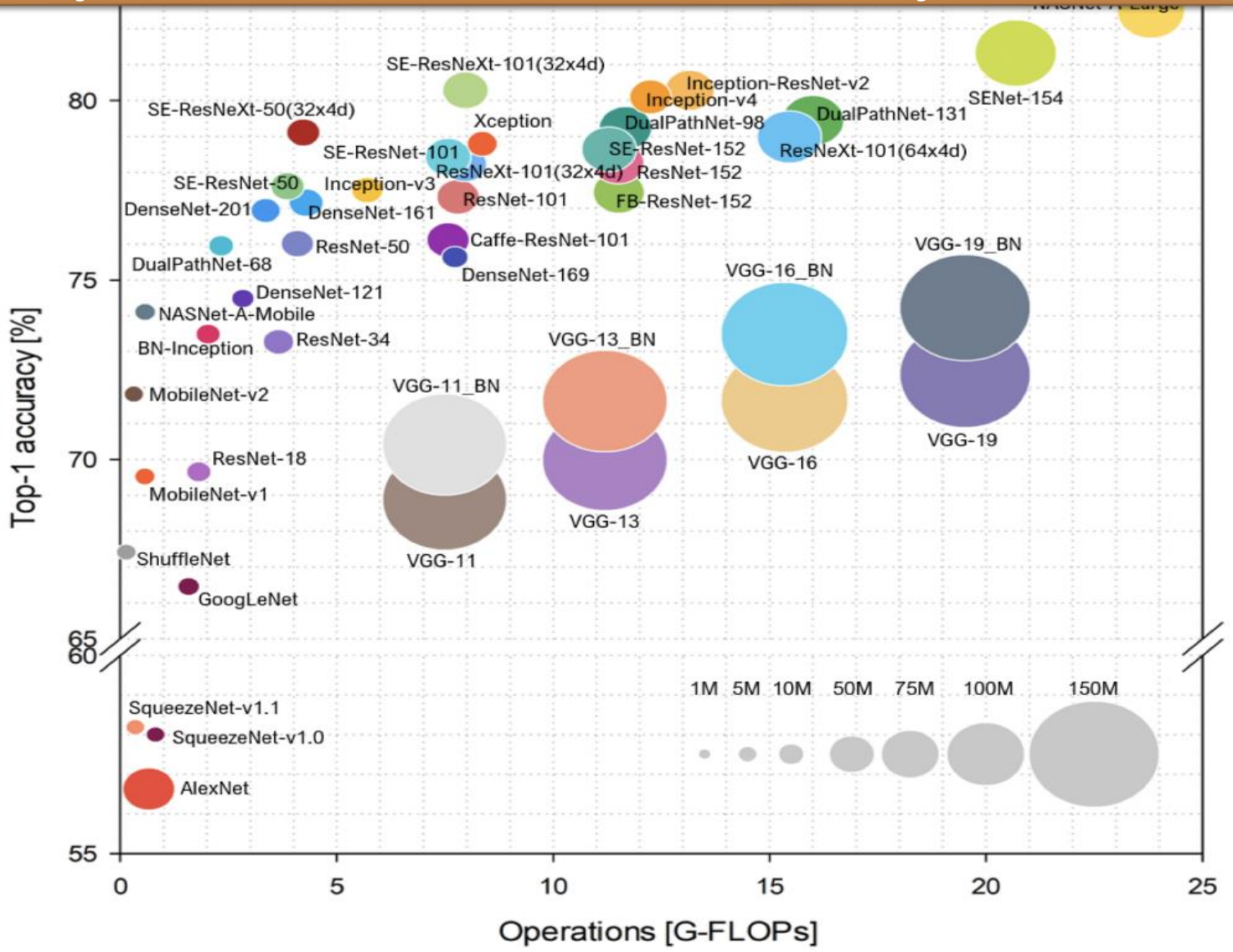


What is an ML benchmark?



MLPERF Benchmarks

Many models with different accuracy and sizes



Selection of Models for Benchmarking

Should we select least common denominator, current or cutting edge model?

Should we have models of different complexity?

Should we have models of different methods?

Allow reimplementation of models?

There are multiple competing ML frameworks.

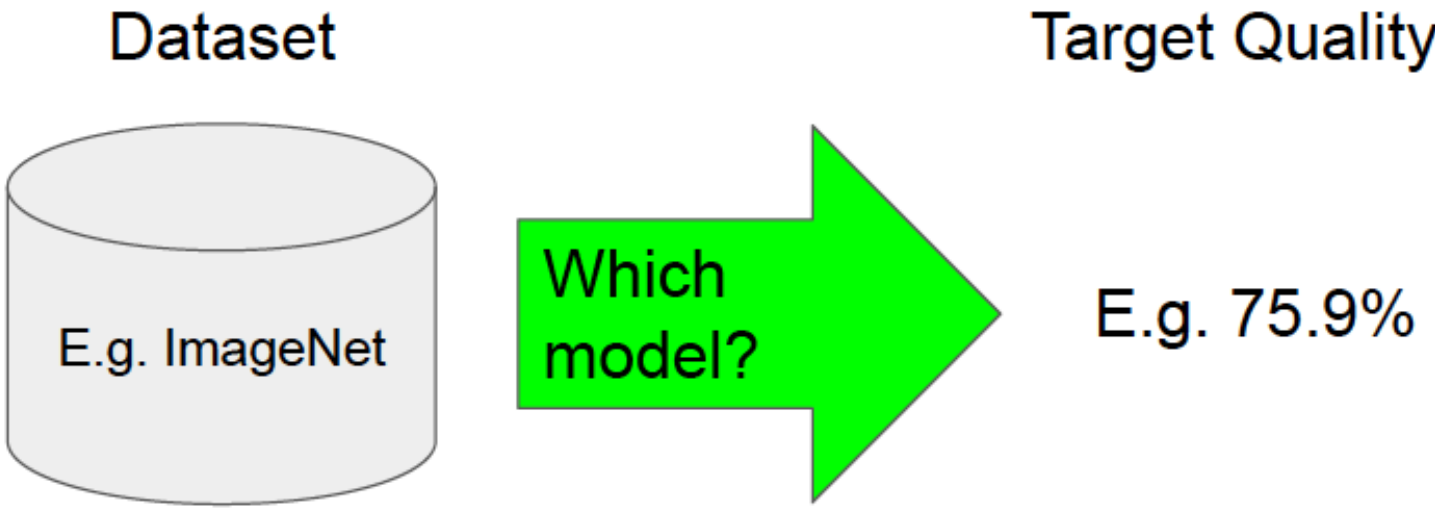
Not all architectures support all frameworks.

Implementations still require some degree of tuning.

Temporary solution – allow to reimplement models.

Require models are mathematically equivalent.

Do we specify the model?



MLPERF

Choice: two divisions

- Closed division: model is specified
- Open division: model is not specified

Metrics for AI Benchmarking

Quality Metrics

- Accuracy, BLEU score, Perplexity

Performance Metrics

Throughput vs Time to Accuracy

Latency – Time to Inference

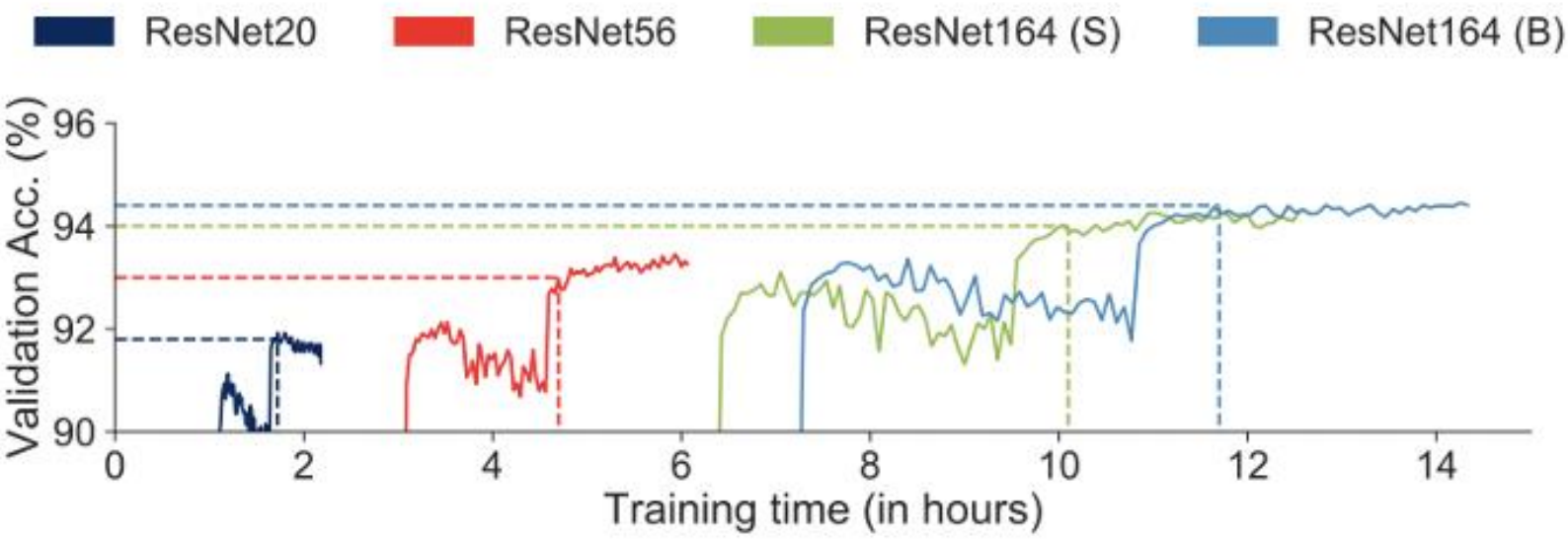
Energy/Inference

Metrics: Time to Accuracy (TTA)

TTT: Time to Threshold

https://dawn.cs.stanford.edu/benchmark/papers/nips17-dawnbench.pdf

Metrics: Time to Accuracy (TTA)



Accuracy threshold	Time to threshold	Fastest model
91.8%	1h 43m	ResNet20
93%	4h 42m	ResNet56
94%	10h 6m	ResNet164 (simple)
94.4%	11h 42m	ResNet164 (bottleneck)

MLPerf Benchmarks

1. Training Benchmark Suite

https://arxiv.org/pdf/1910.01500.pdf

2. Inference Suite – 4 categories of inference
   - Datacenter
   - Edge
   - Mobile
   - Tiny

https://arxiv.org/pdf/1911.02549.pdf

Example Quality Metrics used in MLPerf

Area	Benchmarks	Dataset	Quality Target	Ref Model
Vision	Image Classification	ImageNet	**75.9 classific**	Resnet 50 1.82x
Vision	Image Classification	KiTS19	**0.908 mean DICE score**	3D U-Net
Vision	Object Detection	Open Images	**34,0 mAP**	RetinaNet
Vision	Object Detection	COCO	**0.377 Box min AP/0.339 Mask Min AP**	Mask R CNN
Language	NLP	Wikipedia 2020/01/01	**0.72 Mask LM accuracy**	BERT Large
Commerce	Recommendation	1 TB Clicks	**0.8025 AUC**	DLRM
Research	Reinf Learning	Go	50% win rate	**MiniGo**

Training Benchmarks 2025

Area	Benchmark	Dataset	Quality Target	Reference Implementation Model	Latest Version Available
Vision	Object detection (light weight)	Open Images	34.0% mAP	RetinaNet	v4.1
Language	NLP	Wikipedia 2020/01/01	0.72 Mask-LM accuracy	BERT-large	v4.1
Language	LLM	C4	2.69 log perplexity	GPT3	v4.1
Language	LLM finetuning	SCROLLS GovReport	0.925 cross entropy loss	Llama 2 70B	v4.1
Commerce	Recommendation	Criteo 4TB multi-hot	0.8032 AUC	DLRM-dcnv2	v4.1
Marketing, Art, Gaming	Image Generation	LAION-400M-filtered	FID<=90 and CLIP>=0.15	Stable Diffusionv2	v4.1
Graph neural network	Graph neural network (GNN)*	IGBH-Full	72% classification accuracy	R-GAT	v4.1
Vision	Image classification	ImageNet	75.90% classification	resnet-50	v4.0
Vision	Image segmentation (medical)	KiTS19	0.908 Mean DICE score	3D U-Net	v4.0
Vision	Object detection (heavy weight)	COCO	0.377 Box min AP and 0.339 Mask min AP	Mask R-CNN	v3.1
Language	Speech recognition	LibriSpeech	0.058 Word Error Rate	RNN-T	v3.1
Commerce	Recommendation	1TB Click Logs	0.8025 AUC	DLRM	v2.1
Research	Reinforcement learning	Go	50% win rate vs. checkpoint	Mini Go (based on Alpha Go paper)	v2.1
Vision	Object detection (light weight)	COCO	23.0% mAP	SSD	v1.1
Language	Translation (recurrent)	WMT English-German	24.0 Sacre BLEU	NMT	v0.7
Language	Translation (non-recurrent)	WMT English-German	25.00 BLEU	Transformer	v0.7

Inference Benchmarks 2025

Area	Task	Model	Dataset	QSL Size	Quality	Server latency constraint	Latest Version Available
Vision	Image classification	Resnet50-v1.5	ImageNet (224×224)	1024	99% of FP32 (76.46%)	15 ms	v4.0
Vision	Object detection	Retinanet	OpenImages (800×800)	64	99% of FP32 (0.20 mAP)	100 ms	v4.0
Vision	Medical image segmentation	3D UNET	KITS 2019 (602x512x512)	16	99% of FP32 and 99.9% of FP32 (0.86330 mean DICE score)	N/A	v4.1
Language	LLM - Q&A	Llama 2 70B	OpenOrca	24576	ROUGE-1 = 44.4312 ROUGE-2 = 22.0352 ROUGE-L = 28.6162	TTFT: 2s & TPOT: 200ms	v4.1
Language	LLM - Summarization	GPT-J 6B	CNN-DailyMail News Text Summarization	13368	99.9% or 99% of the original FP32 ROUGE 1 - 42.9865 ROUGE 2 - 20.1235 ROUGE L - 29.9881	20 seconds	v4.1
Language	LLM - Text generation (Question Answering, Math and Code Generation)	Mixtral 8x7B	OpenOrca GSM8K, MBXP	15000	99% or 99.9% of FP32 (ROUGE 1 - 45.4911, ROUGE 2 - 23.2829, ROUGE L 30.3615, (gsm8k)Accuracy 73.78, (mbxp)Accuracy 60.12)	TTFT: 2s & TPOT: 200ms	v4.1
Image	Image Generation	SDXL 1.0	COCO-2014	5000	FID ∈ (23.0108, 23.9501) CLIP ∈ (31.686, 31.813)	20 seconds	v4.1
Language	Language processing	BERT-large	SQuAD v1.1 (max_seq_len=384)	10833	99% of FP32 and 99.9% of FP32 (f1_score=90.874%)	130 ms	v4.1
Commerce	Recommendation	DLRM-DCNv2	Criteo 4TB Multi-hot	204800	99.9% or 99% of the original FP32 AUC metric (80.31%)	60 ms	v4.1
Speech	Speech-to-text	RNNT	Librispeech dev-clean (samples < 15 seconds)	2513	99% of FP32 (1 - WER, where WER=7.452253714852645%)	1000 ms	v4.0
Commerce	Recommendation	DLRM	1TB Click Logs	204800	99% of FP32 and 99.9% of FP32 (AUC=80.25%)	30 ms	v3.0
Vision	Object detection (large)	SSD-ResNet34	COCO (1200×1200)	64	99% of FP32 (0.20 mAP)	100 ms	v2.0
Vision	Medical image segmentation	3D UNET	BraTS 2019 (224x224x160)	16	99% of FP32 and 99.9% of FP32 (0.85300 mean DICE score)	N/A	v1.1
Vision	Image classification	MobileNet-v1	ImageNet (224×224)	1024	99% of FP32 (71.68%)	10 ms	v0.5
Vision	Object detection (small)	SSD-MobileNets-v1	COCO (300×300)	256	99% of FP32 (0.22 mAP)	10 ms	v0.5

The Benchmark Question is Complicated

Kernel Benchmarking may not work

Application Based AI Benchmarking is complicated too!

Calibrate Claims you Hear

What Programming Language?

What Libraries?

What Accuracy?

For what dataset? How is the training set/test set partitioning?

Is latency non-pipelined latency or inverse of throughput?

When AI meets Performance Engineering

1. Performance Engineering for AI

2. AI for Performance Engineering

AI for Performance Engineering

Bottleneck Detection

AI models for Performance

AI models for Power/Energy

Capacity Planning in Data Centers

AI for Performance Engineering

Can AI help to identify performance bottlenecks?

Code Bloat

It takes a million lines of code to open a garage door (IEEE Spectrum article April'24)

Code Bloat

IEEE Spectrum

April 2024

FEATURE | COMPUTING

WHY BLOAT IS STILL SOFTWARE'S BIGGEST VULNERABILITY

A 2024 plea for lean software

BY BERT HUBERT | 08 FEB 2024 | 10 MIN READ

DANIEL ZENDER

This post is dedicated to the memory of Niklaus Wirth, a computing pioneer who passed away 1 January 2024. In 1995 he wrote an influential article called "A Plea for Lean Software," published in Computer, the magazine for members of the IEEE Computer Society, which I read early in my career as an entrepreneur and software developer. In what follows, I try to make the same case nearly 30 years later, updated for today's computing horrors. A version of this post was originally published on my personal blog, Berthub.eu.

Some years ago I did a talk at a local university on cybersecurity, titled "Cyber and Information Security: Have We All Gone Mad?" It is still worth reading today since we *have* gone quite mad collectively.

The way we build and ship software these days is mostly ridiculous, leading to

We are likely looking at over 50 million active lines of code to open a garage door....

These days we often ship software as containers

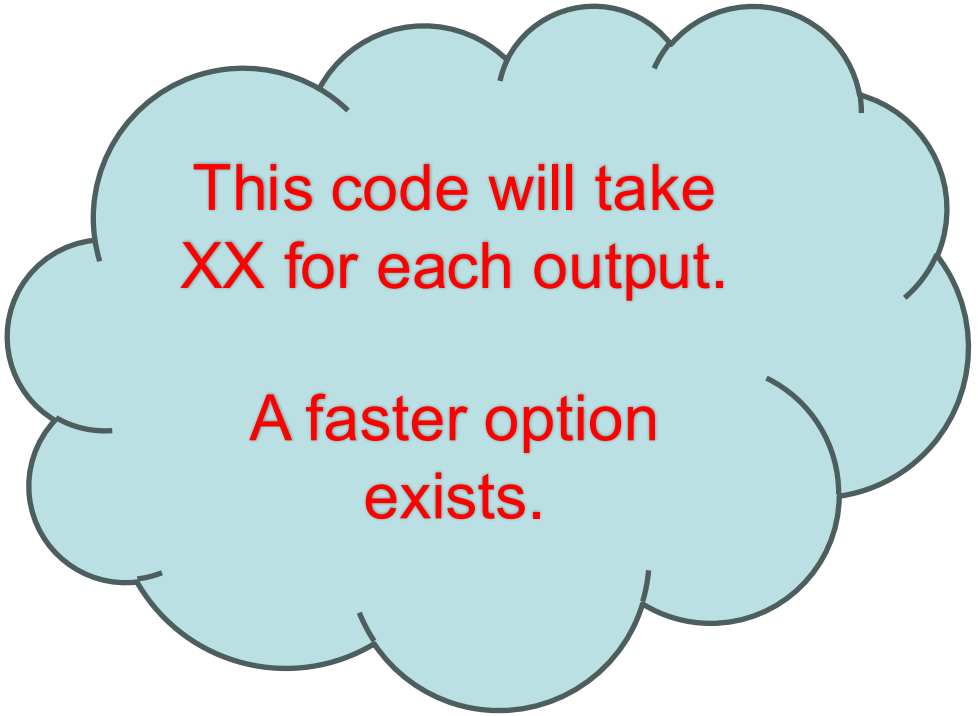
shipping not only the software itself but also including operating system files to make sure the software runs in a well-known environment. This frequently entails effectively shipping a complete computer disk image. This again vastly expands the amount of code being deployed. Note that you can do good things with

If this app controls anything in your house, it will also connect to a software stack over at Amazon, probably also powered by Node.js, also pulling in many dependencies.

AI for Performance Engineering

When you develop code, how do you know it is good? Can AI help?

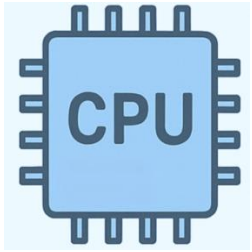
Can you get feedback on performance based on AI?

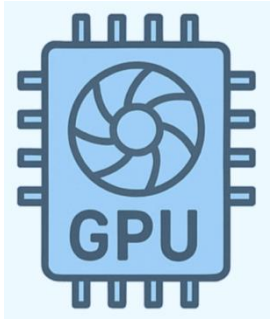


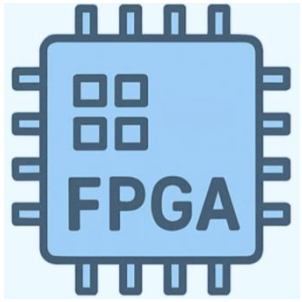
AI for Performance Engineering

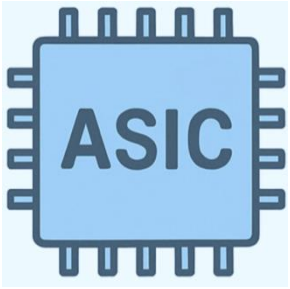
AI to help when to run on GPU/CPU/TPU/ASIC

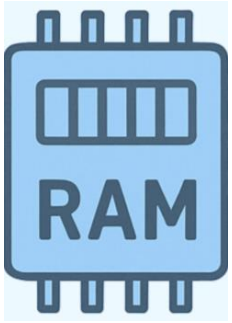
Performance Prediction using Clustering and Near-neighbor techniques



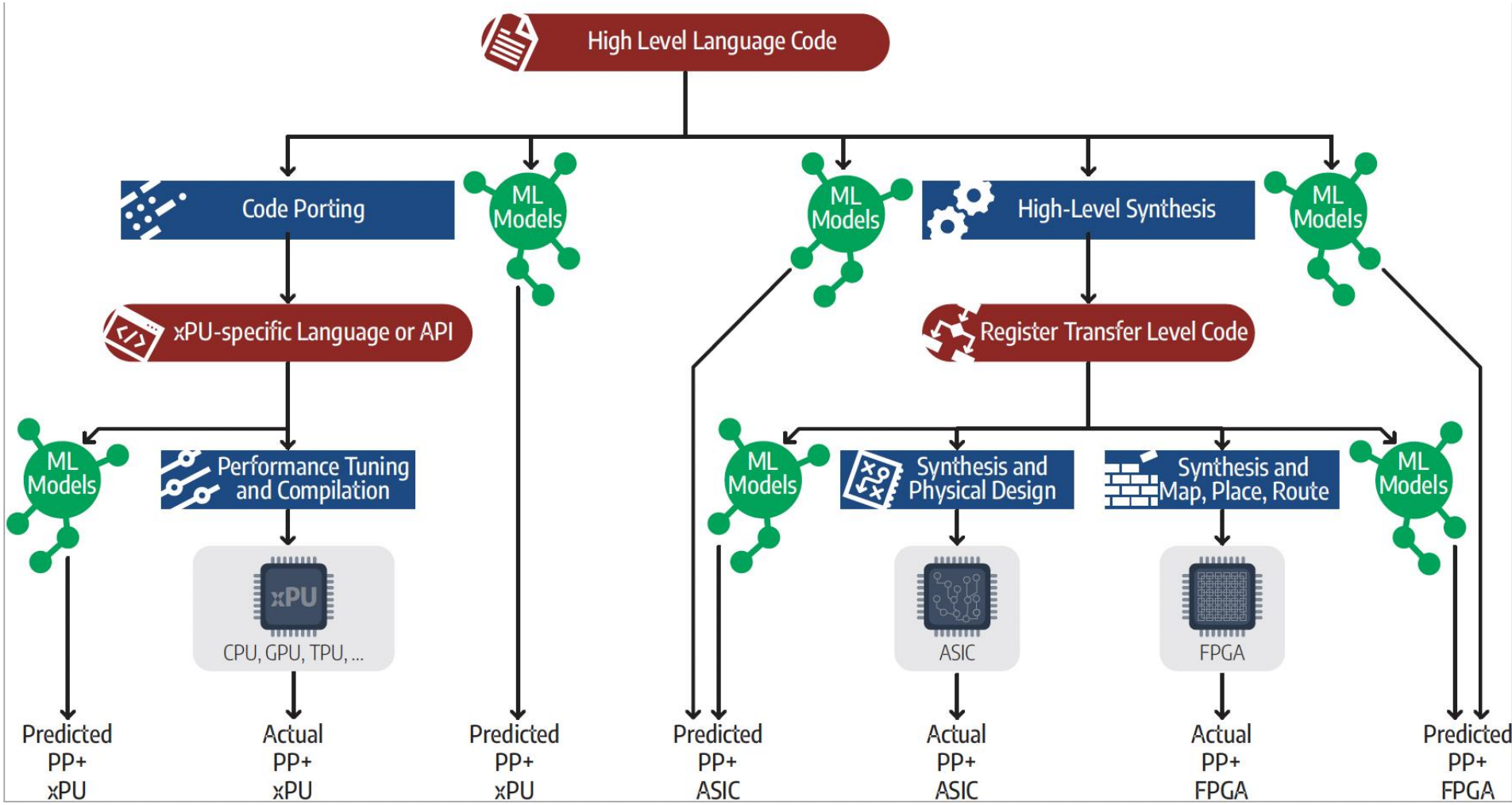




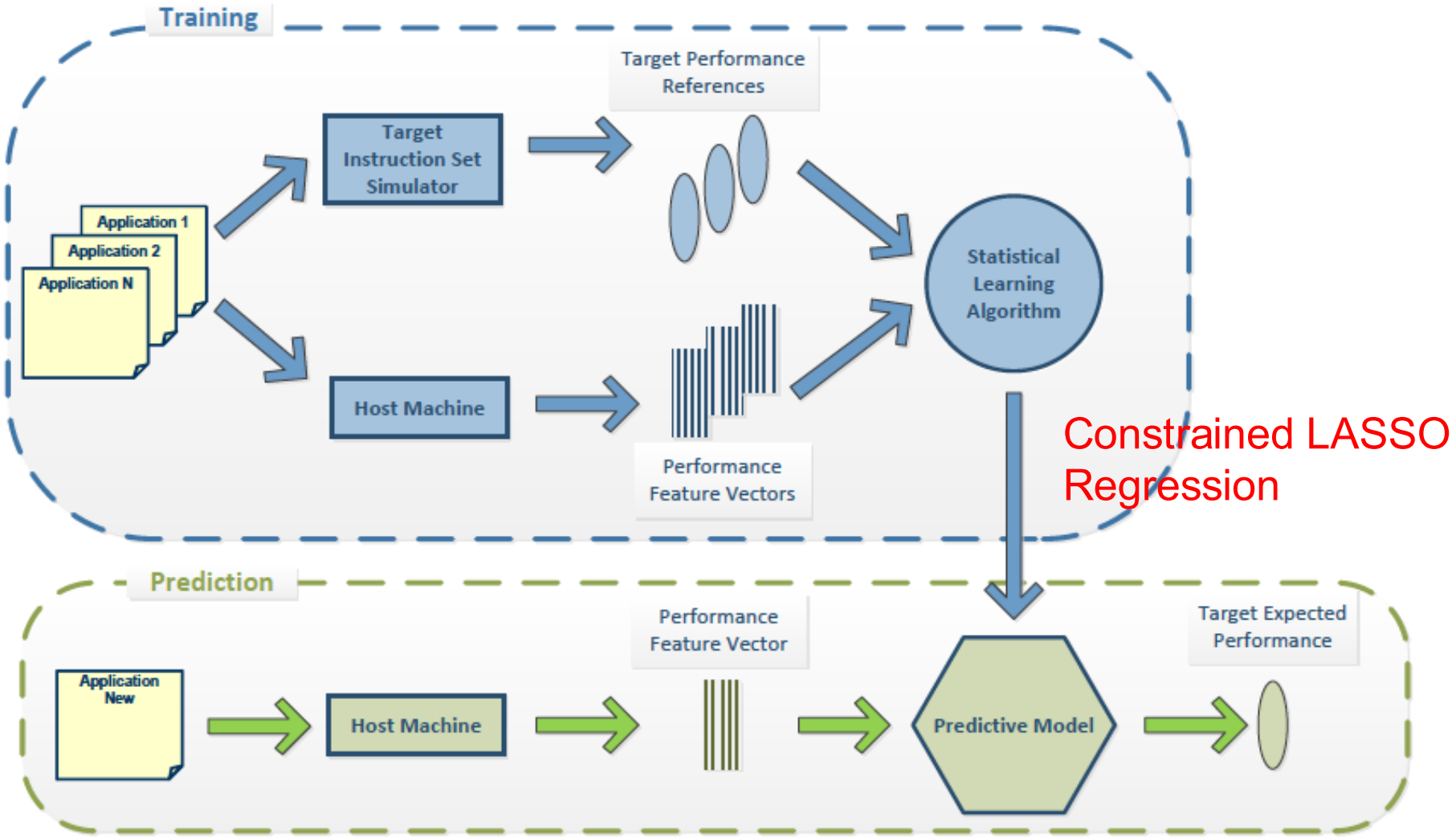




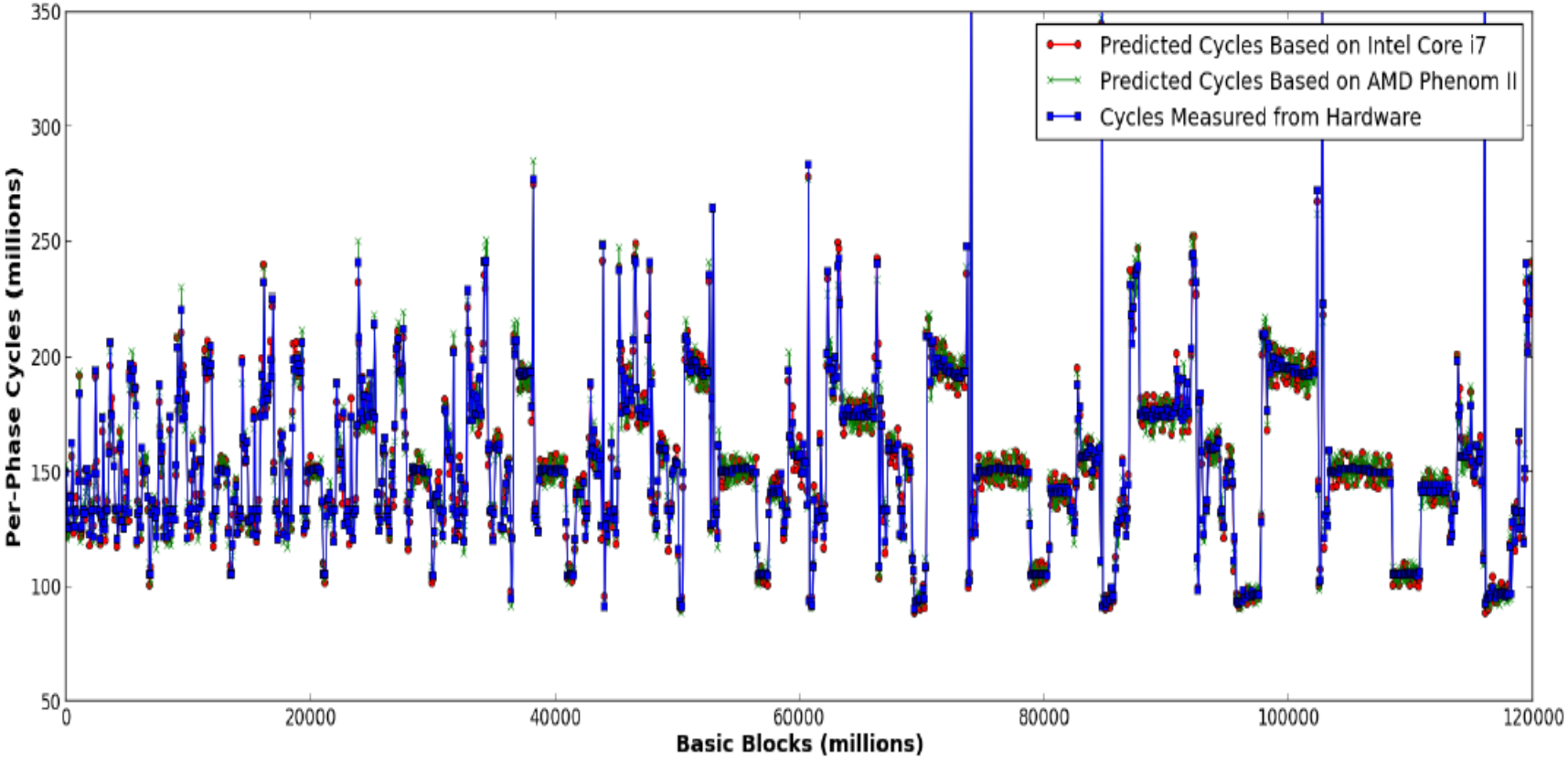
AI for Performance Tradeoff Evaluation



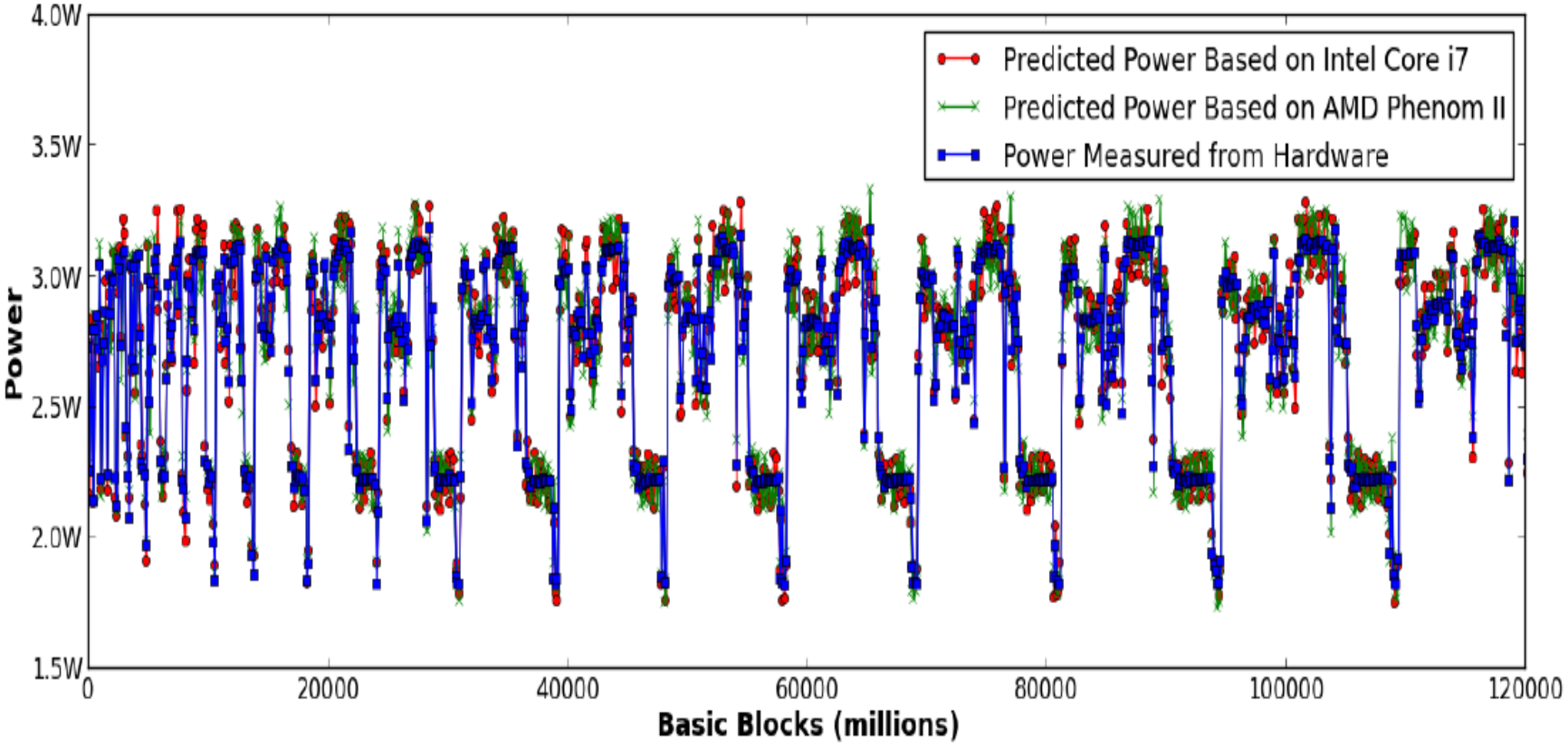
Machine Learning for Cross-Platform Prediction

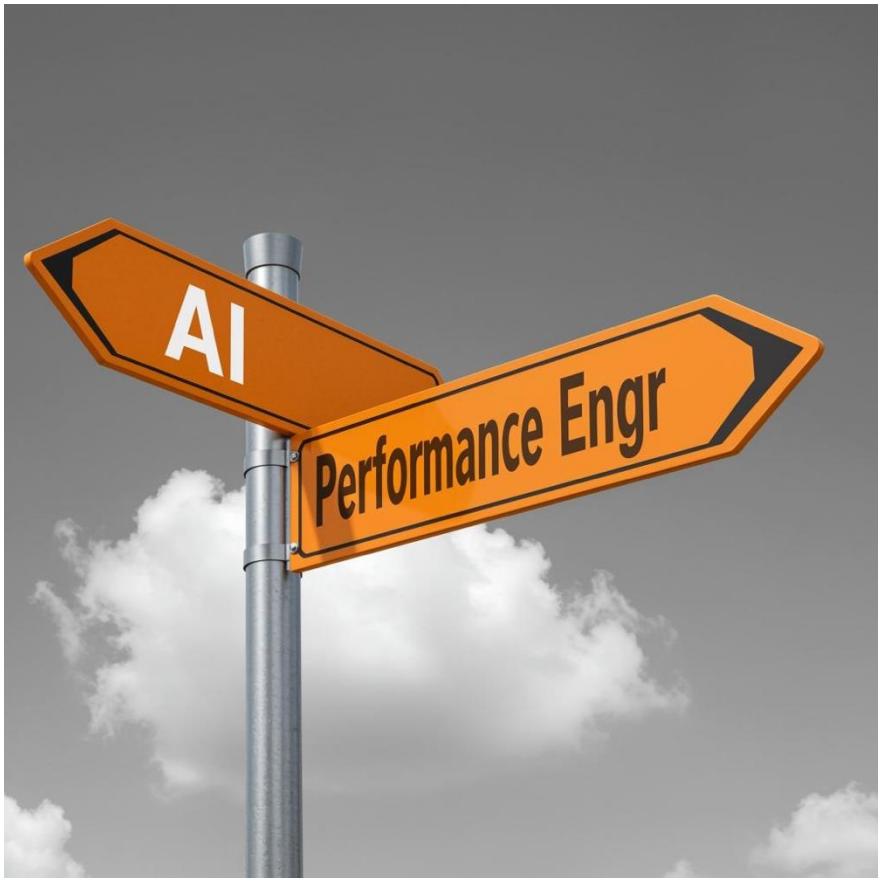


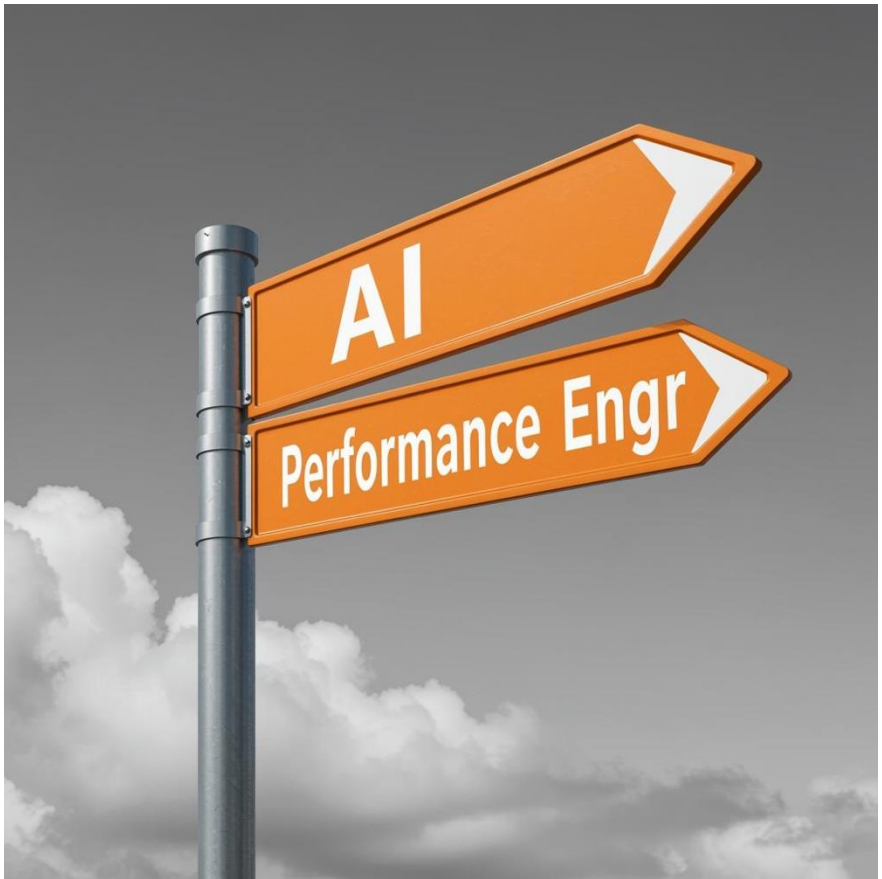
Prediction at Fine-grain



Power Prediction at Fine-grain




AI
Performance Engr


AI
Performance Engr

When AI Meets Performance Engineering

Personally Speaking

The Performance Engineer inside me

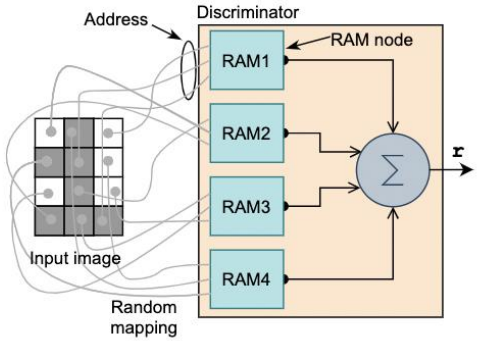
Very dissatisfied with the inefficiency

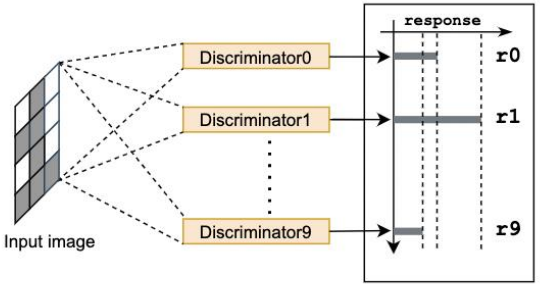
Memory based neurons

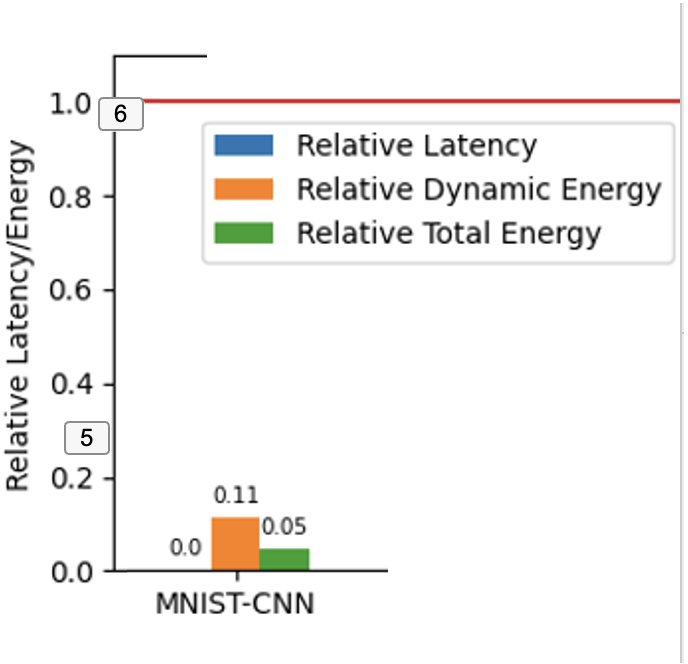
Weightless Neural Networks

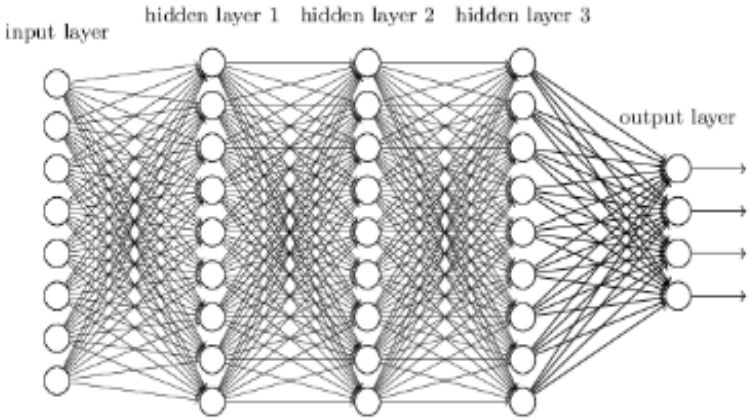
Compute-less LLMs

Weightless Neural Networks









When AI Meets Performance Engineering

Future is Bright

There is something in it for each of you

Whether you are Performance Engineer

Hardware Engineer

Software Engineer

AI Model Developer

Thank you

ICML 2024
Paper &
github

Laboratory for Computer Architecture (LCA)
The University of Texas at Austin
lca.ece.utexas.edu