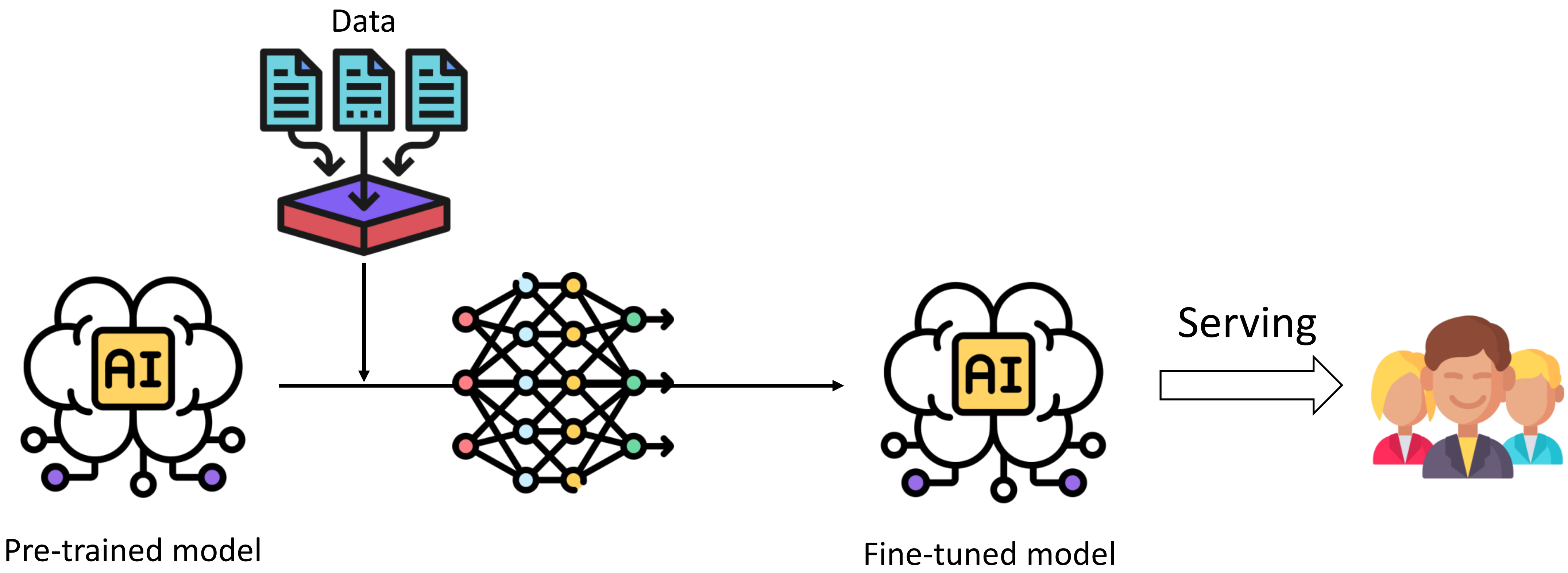
CS 598
AI Efficiency: Systems and Algorithms
Overview & Key Challenges in AI Model Serving

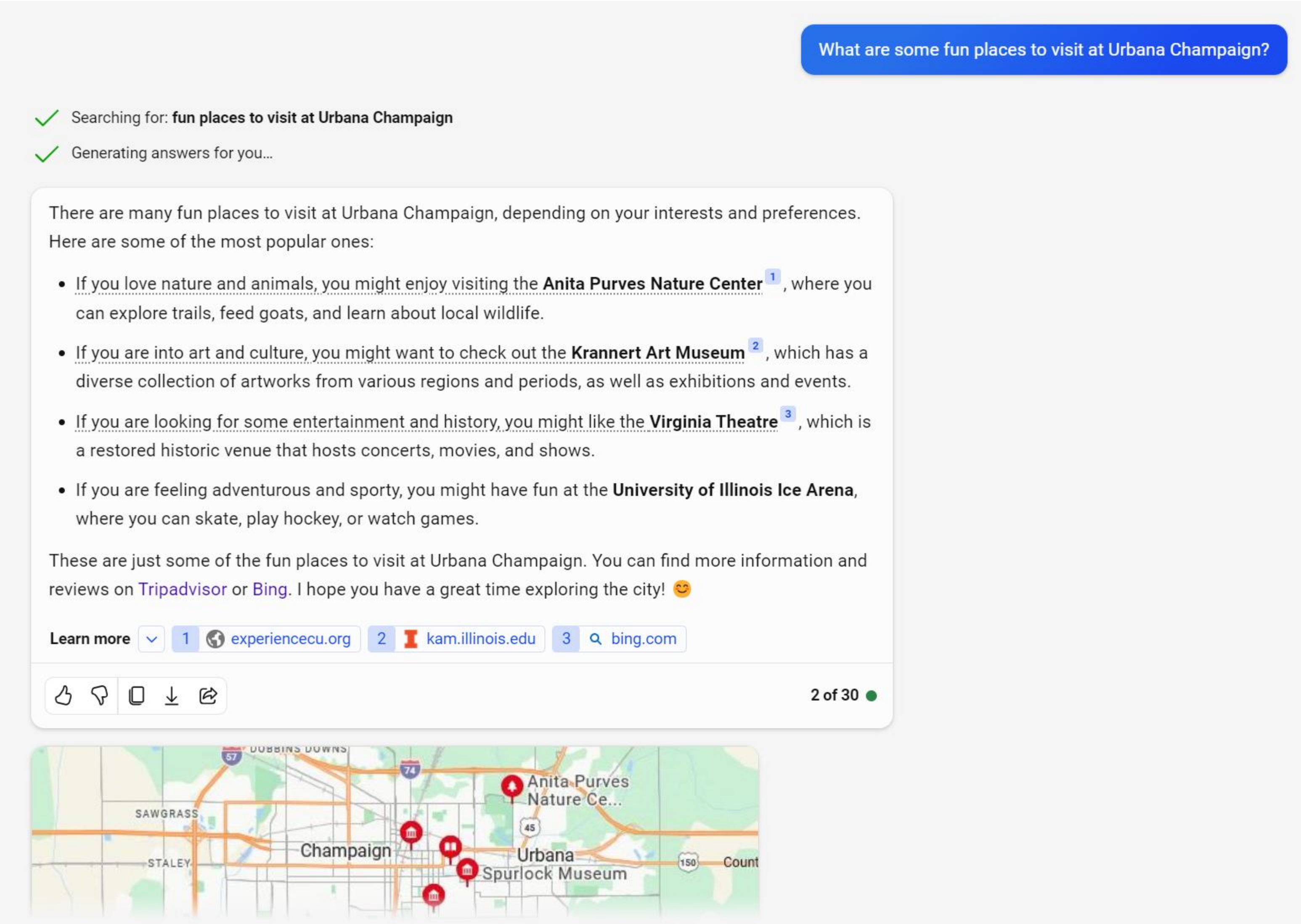
Minjia Zhang

Computer Science Department

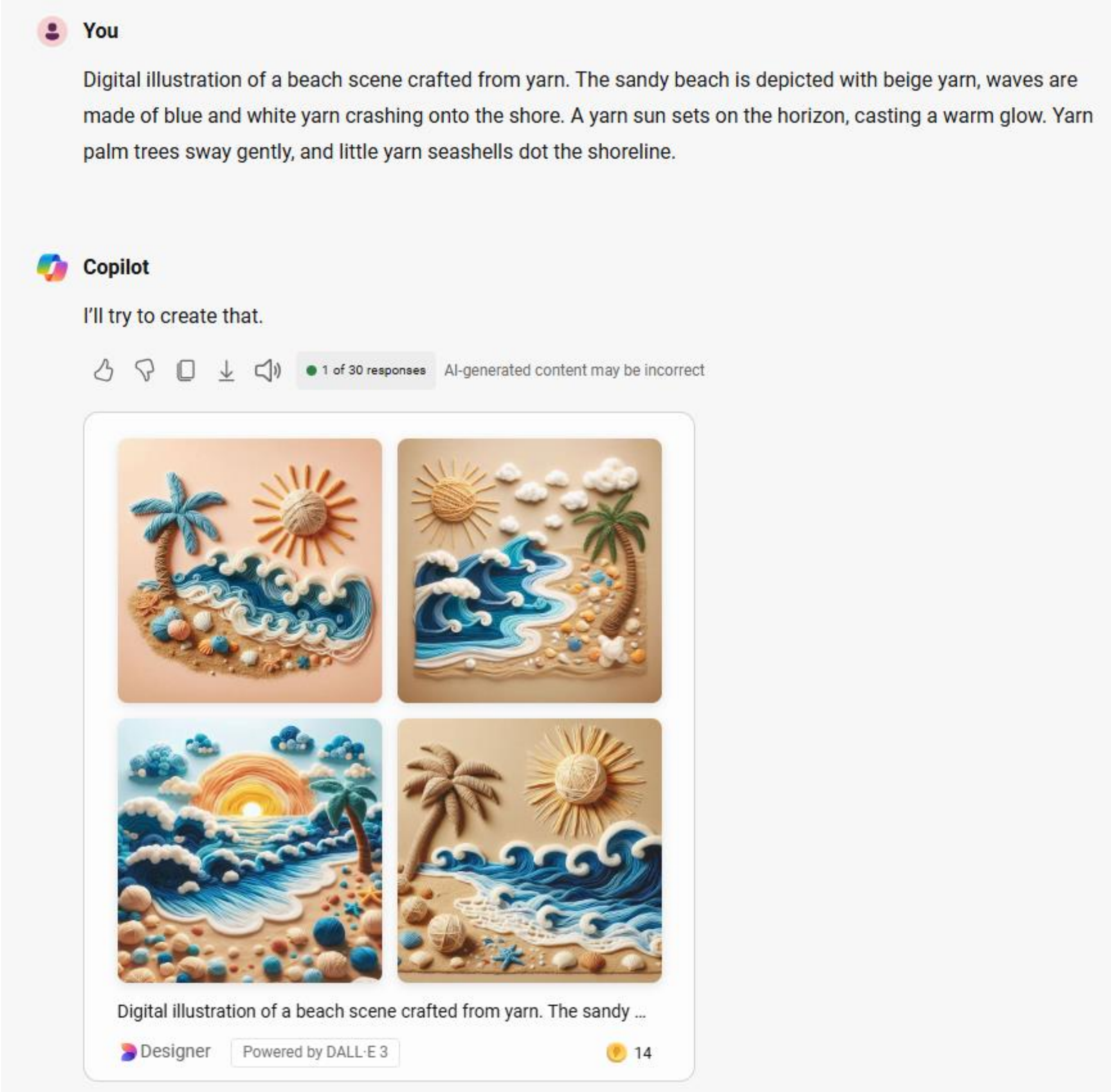
What is Model Serving?



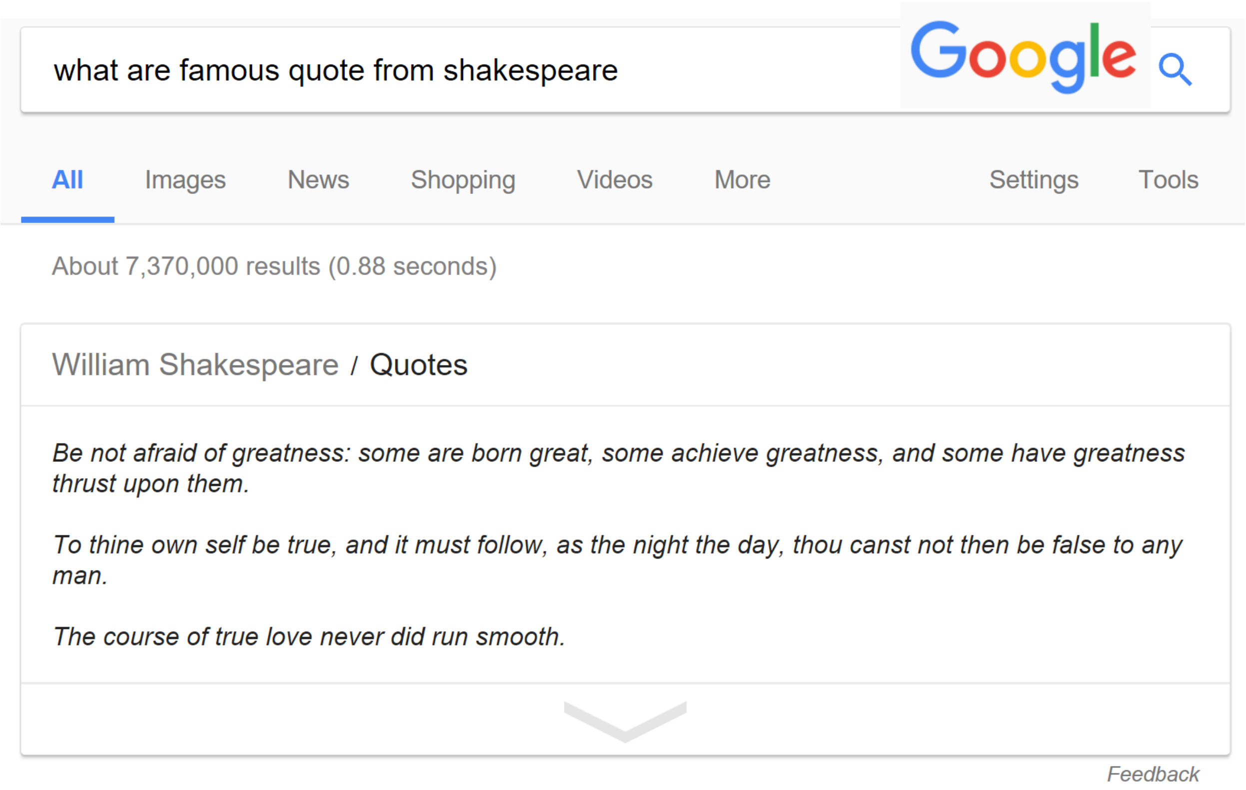
Serving Scenario 1: Online ChatBot



Serving Scenario 2: Online Image Generation

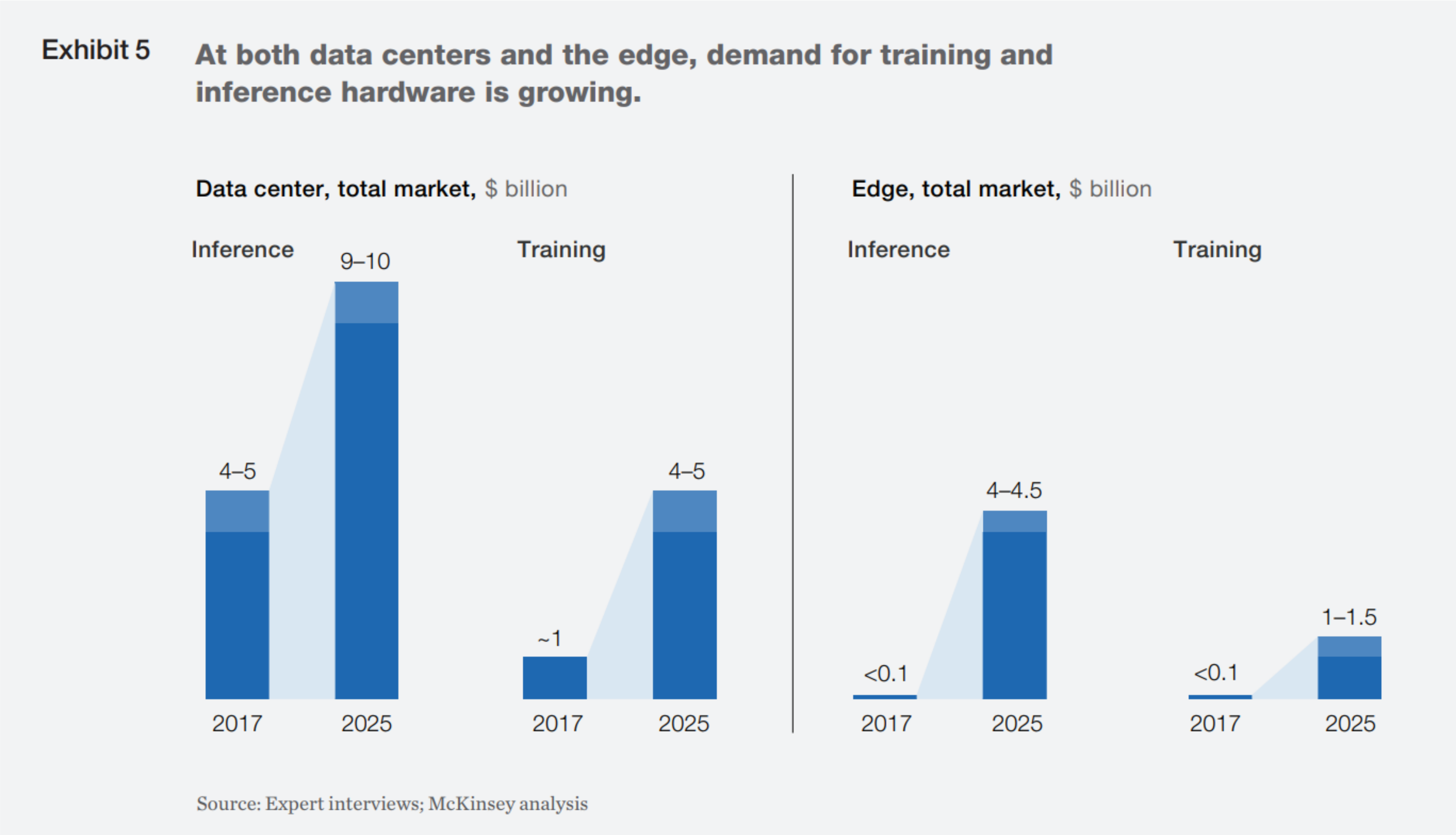


Serving Scenario 3: Online Q&A



- Question and Answer Scenario
    - Direct answer not a list of webpages
    - Good quality of answer

Training -> Inference



Inference Challenges

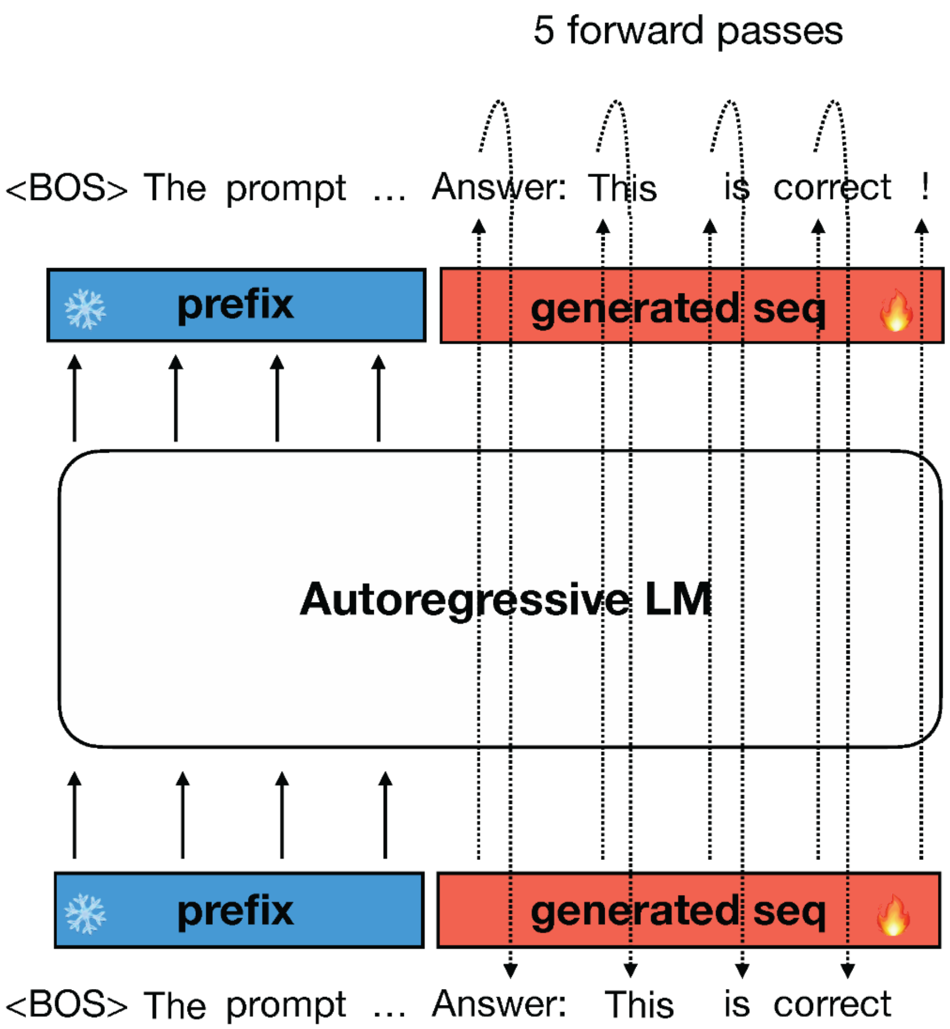
	Training	vs	Inference
Runtime	Weeks or months		Milliseconds or seconds
Challenges	TCO (Cost, Energy)		TCO (Cost, Energy)

Speed (LLM: token rates)

Model size

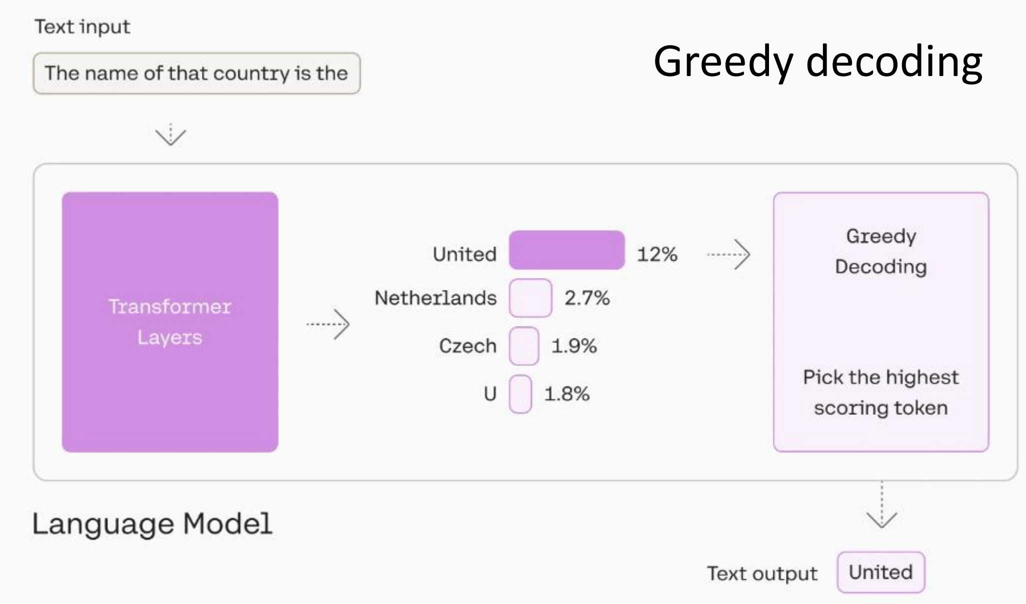
- Parameter volume
- LLM: Context length

LLM Autoregressive Generation



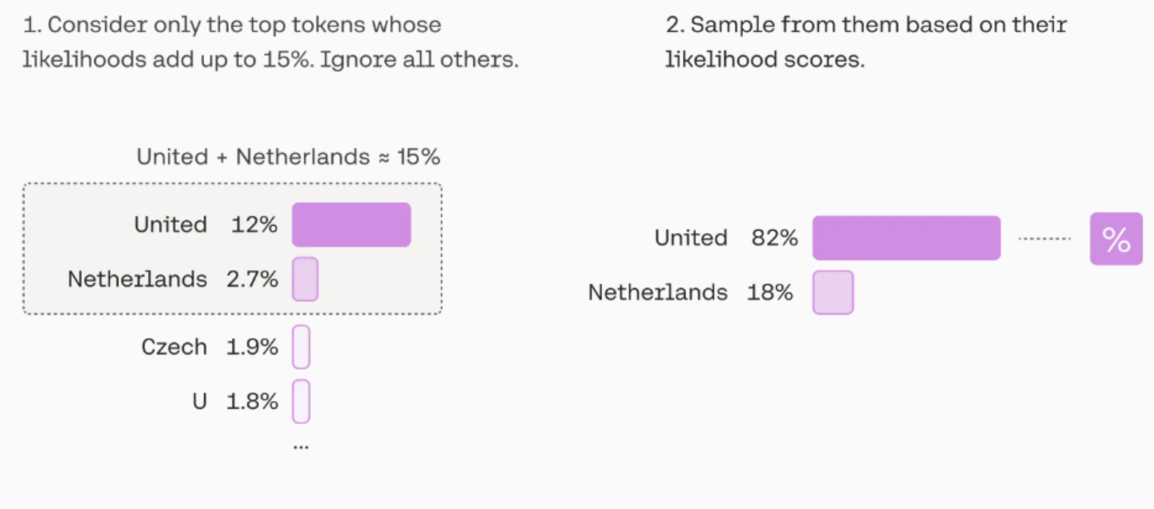
Decoding Strategies

Greedy decoding



Top-k decoding

Top-p decoding (Nucleus Sampling)

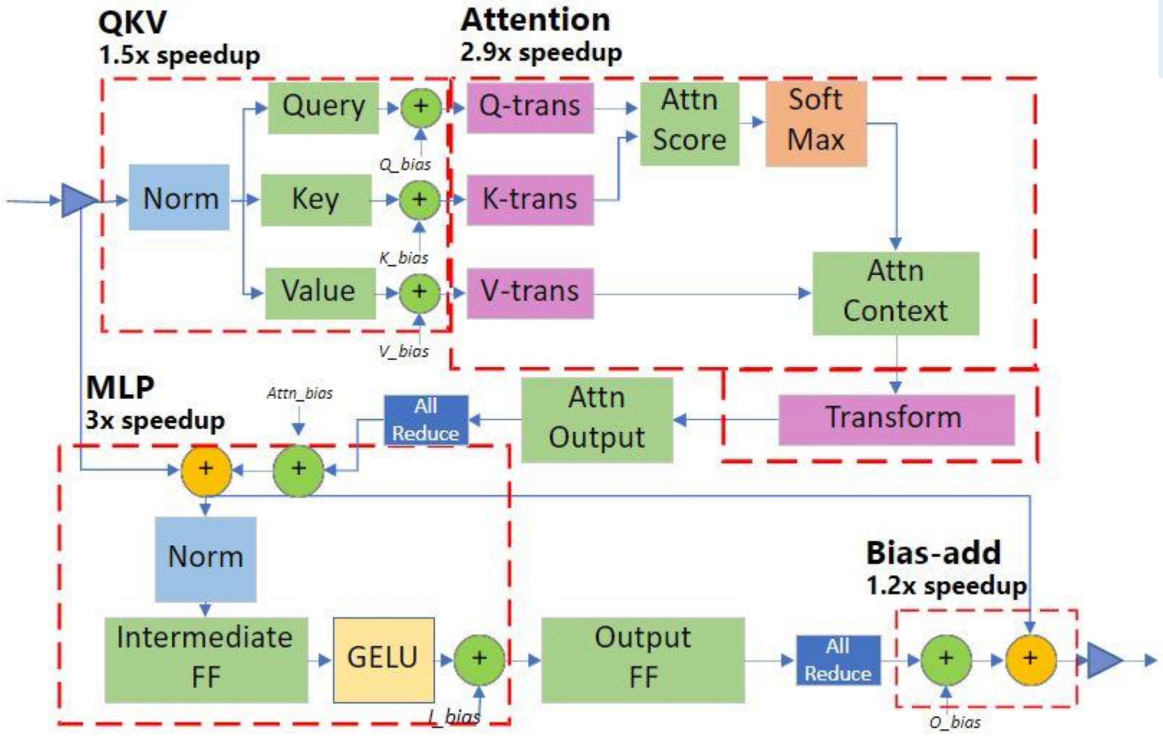


Serving Challenge: Long Latency

- Long serving latency blocks deployment
- Support advance models while meeting latency SLA and saving cost

DL Scenarios	Original Latency	Latency Target
Turing Prototype 2	~100ms	< 10ms
Turing Prototype 3	~107ms	< 10ms
Deep Query Document Similarity	10~12ms for [query, 1 doc] x 33 docs	< 6ms
Malta Click Features	10ms for [query, 1 passage] x 150 passages	< 5ms
Ads seq2seq model for query rewriting	~51ms	< 5ms

Customized Kernels



DeepSpeed-Inference: enabling efficient inference of transformer models at unprecedented scale, SC 2022

ByteTransformer: A High-Performance Transformer Boosted for Variable-Length Inputs, 2023

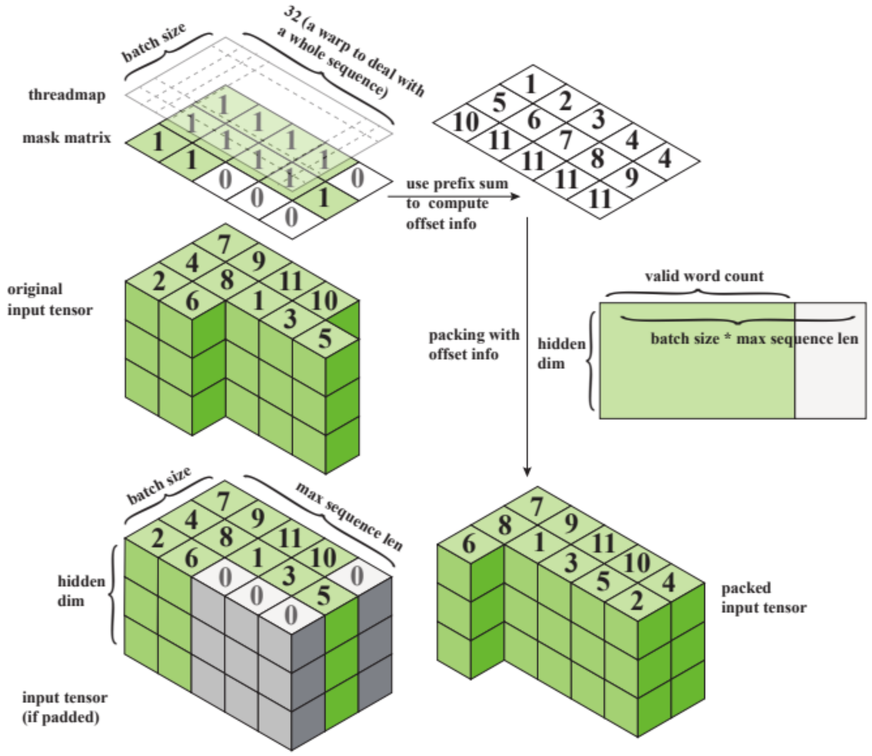
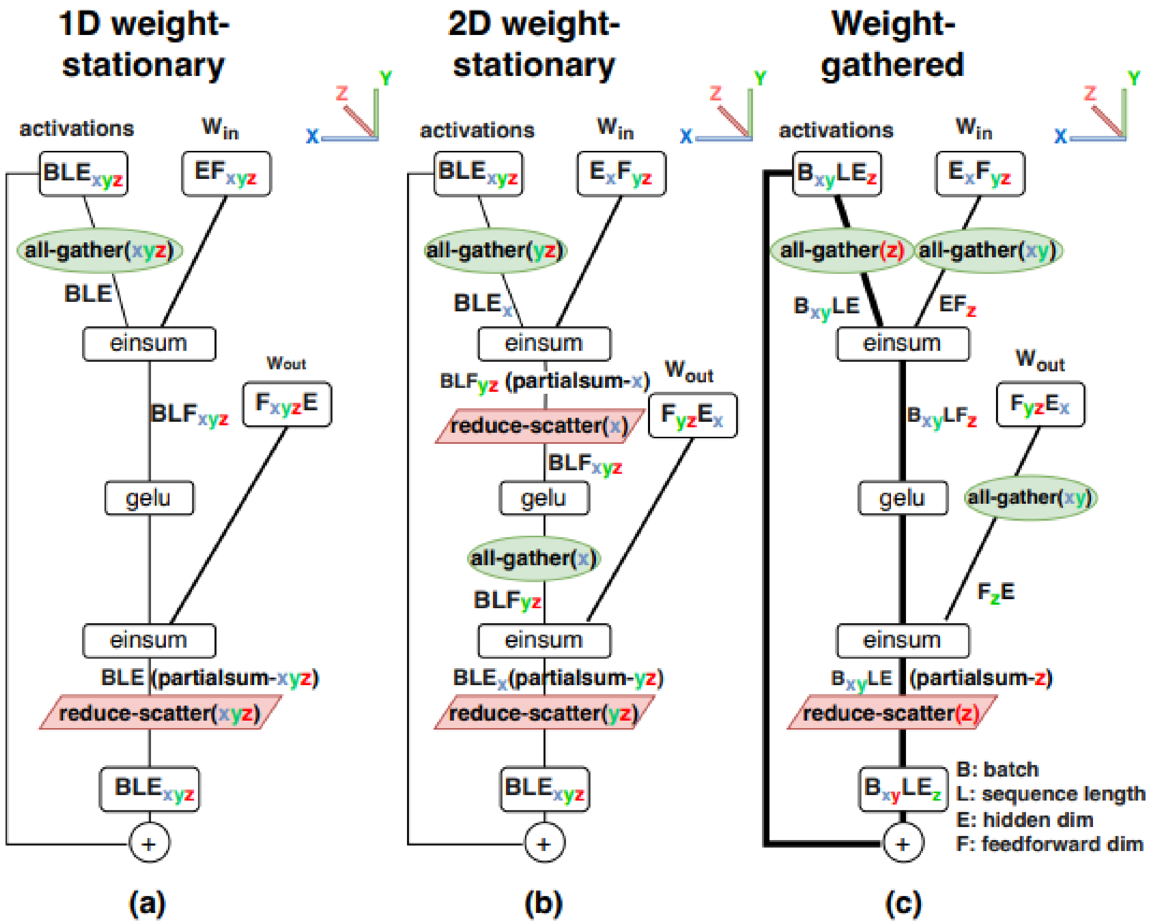


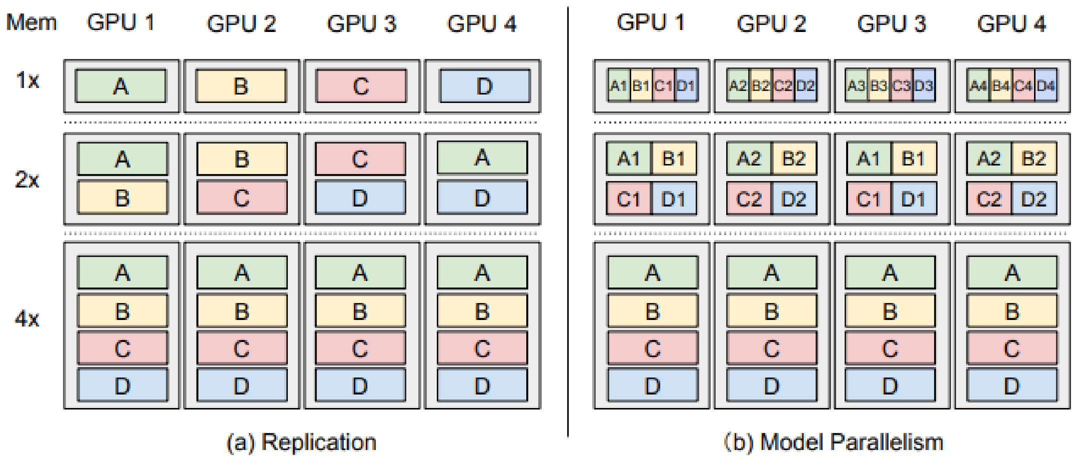
Fig. 4: The zero padding algorithm.

Multi-GPU Inference via Partitioned Layouts



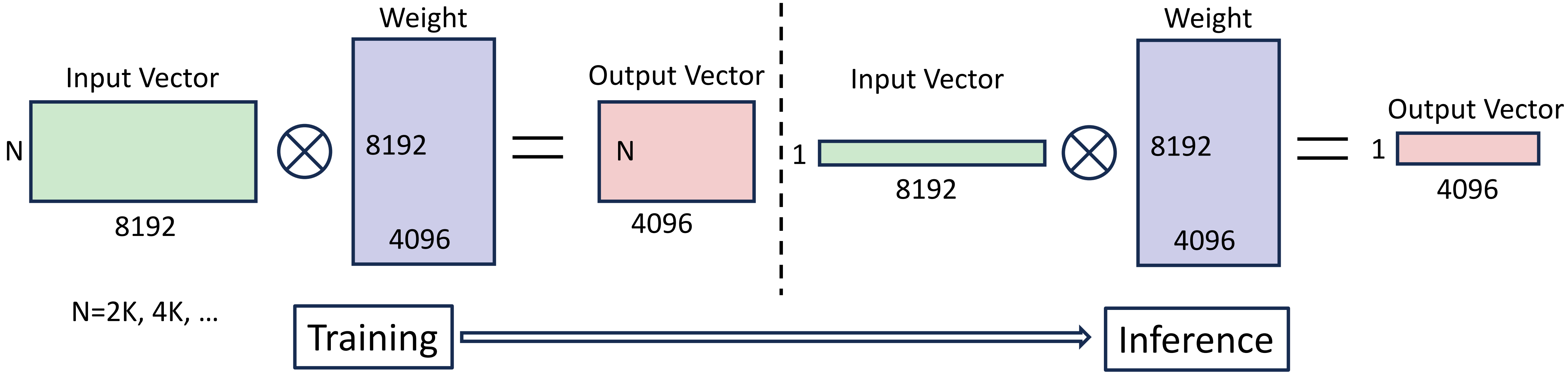
Efficiently Scaling Transformer Inference, MLSys 2023

AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving, OSDI 2023

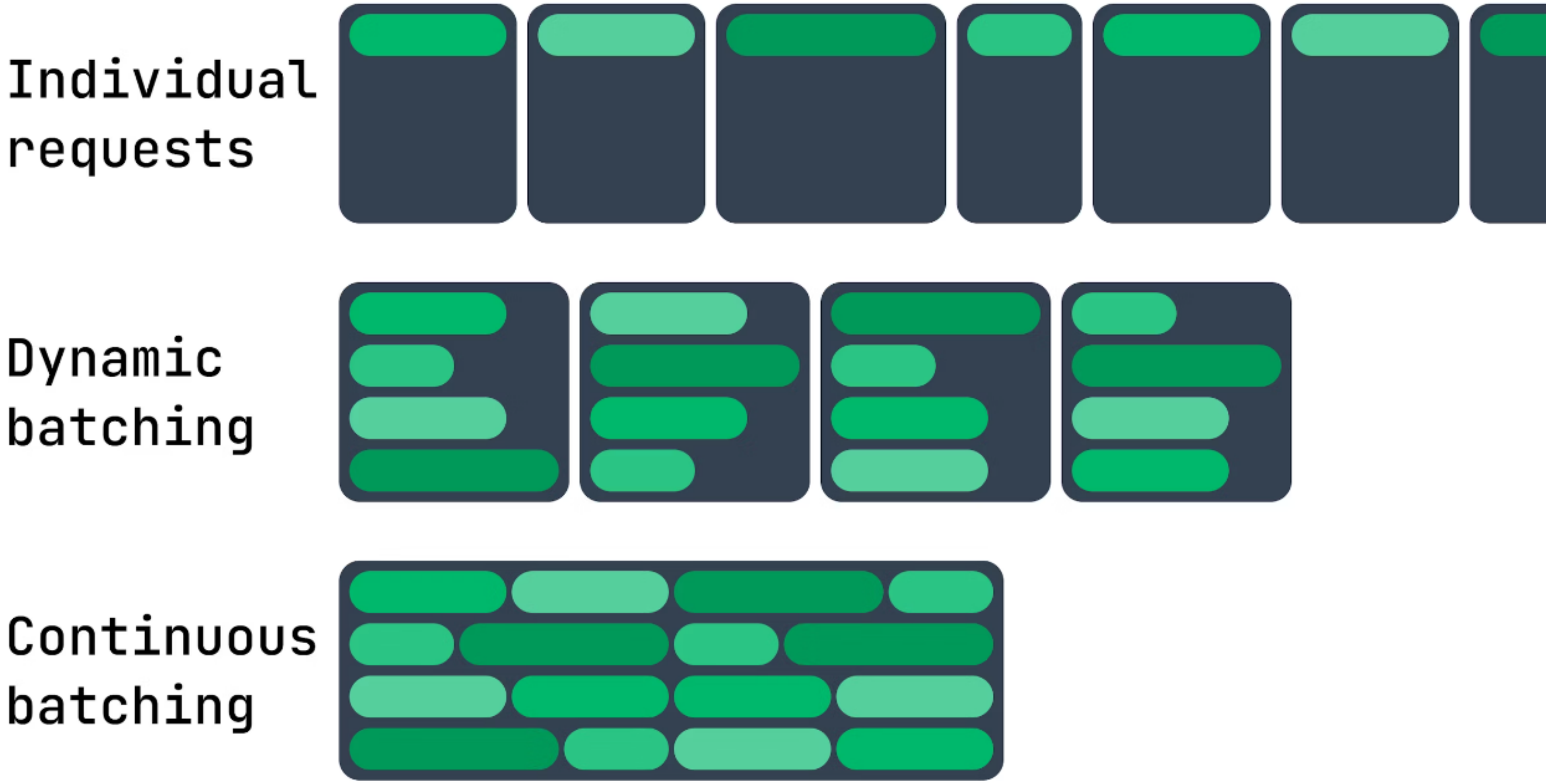


Inference Challenge: Limited Parallelism

- Small batch size ⟹ Low data reuse
- Autoregressive generation ⟹ Sequential dependency



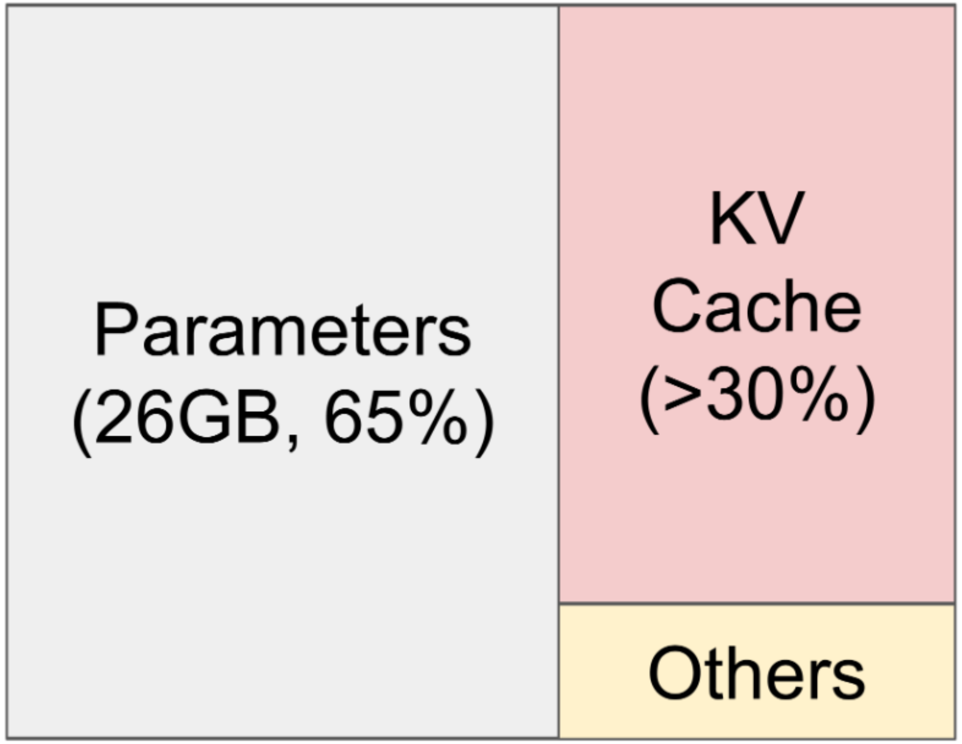
Batching Strategies for LLM Inference



Orca: A Distributed Serving System for Transformer-Based Generative Models, OSDI 2022

Inference Challenge: Large Memory Footprint

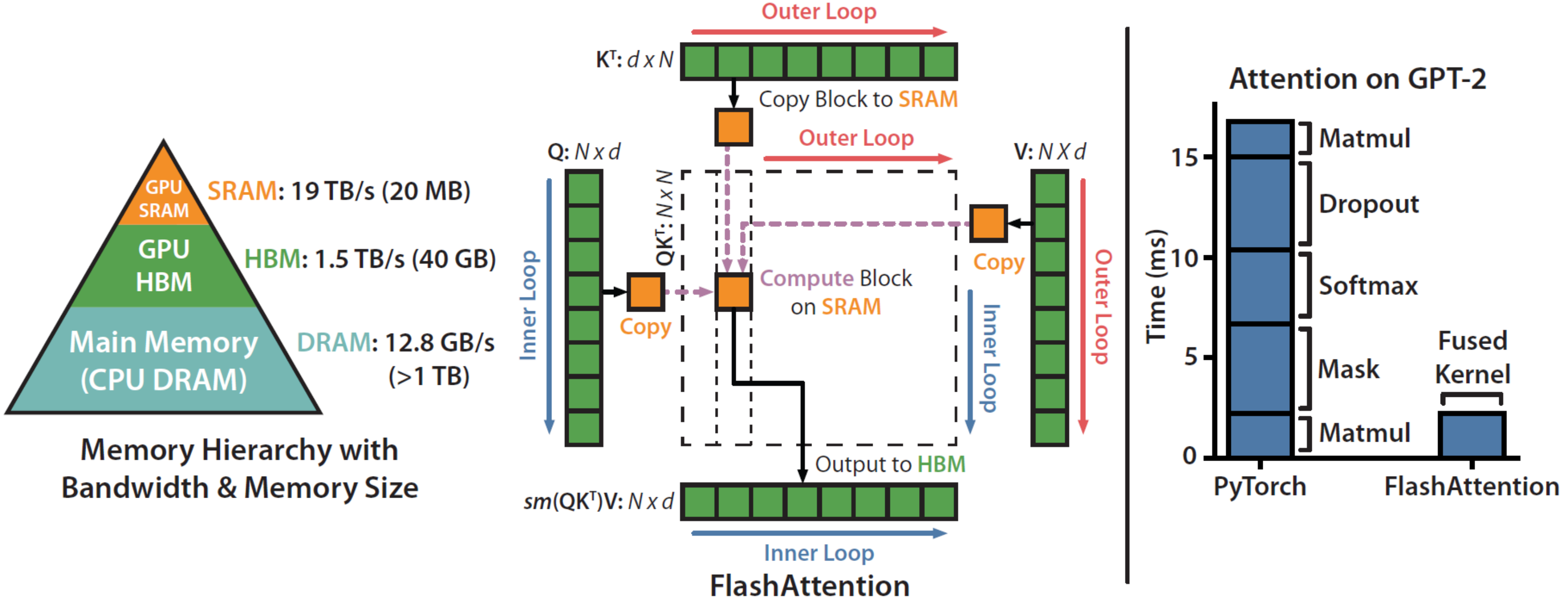
- Model parameters
  - # Layers
  - # Hidden dim
- KV cache
  - Batch size
  - Sequence length
  - # Layers
  - # Hidden
- Activation and others



OPT-13B on A100 40 GB

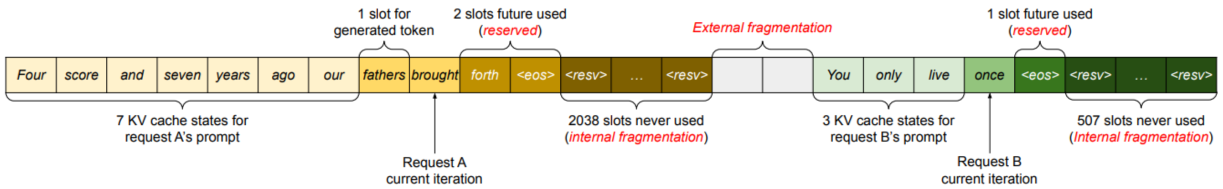
Efficient Memory Management for Large Language Model Serving with PagedAttention, by Kwon et al., 2023

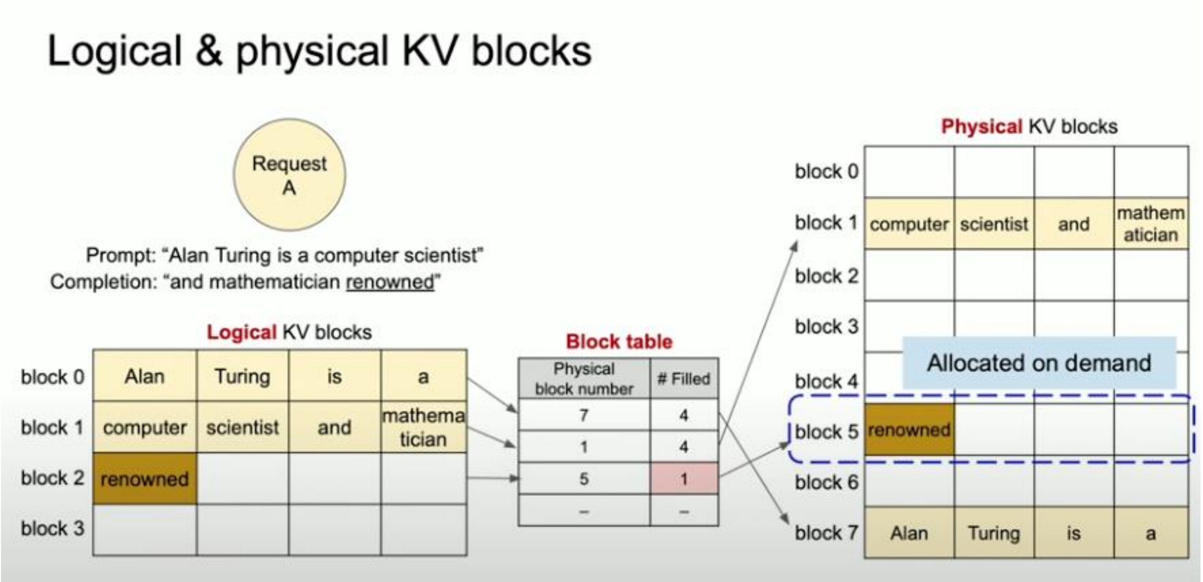
FlashAttention



Fast and Memory-Efficient Exact Attention with IO-Awareness, 2023

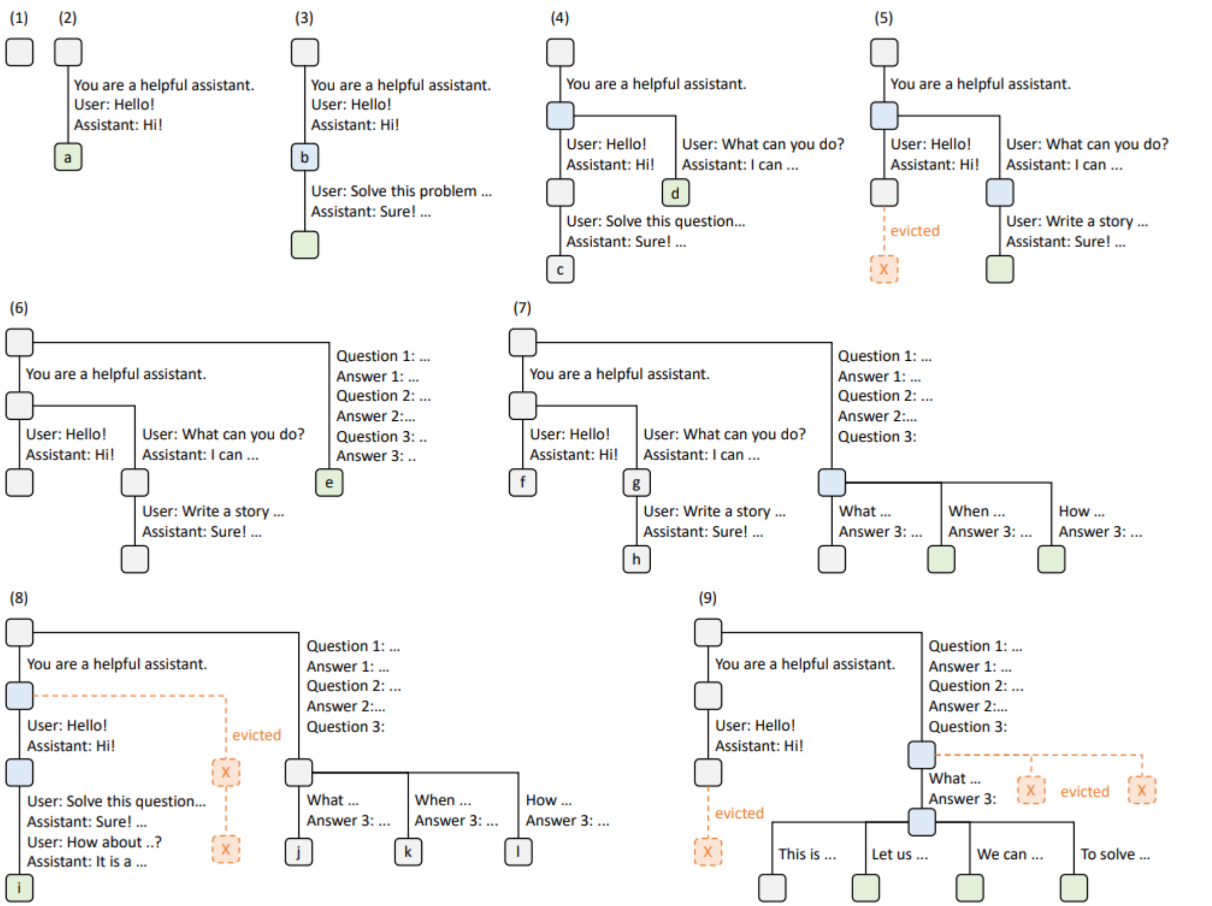
PagedAttention





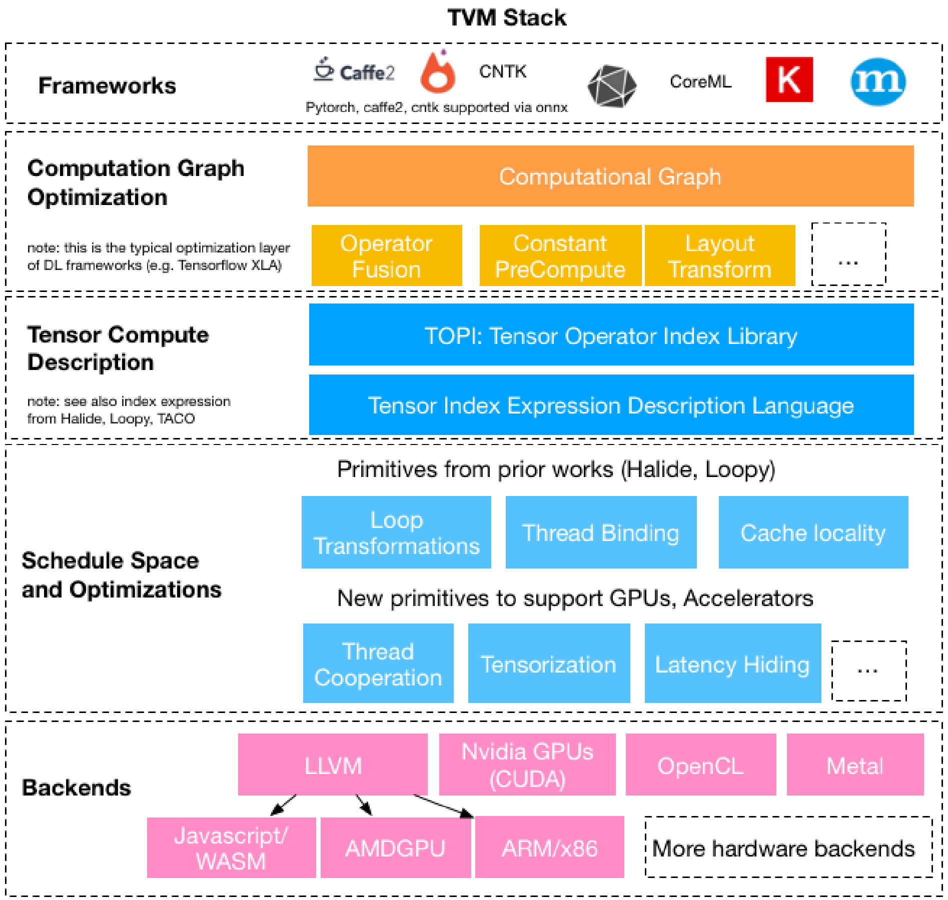
Efficient Memory Management for Large Language Model Serving with PagedAttention, 2023

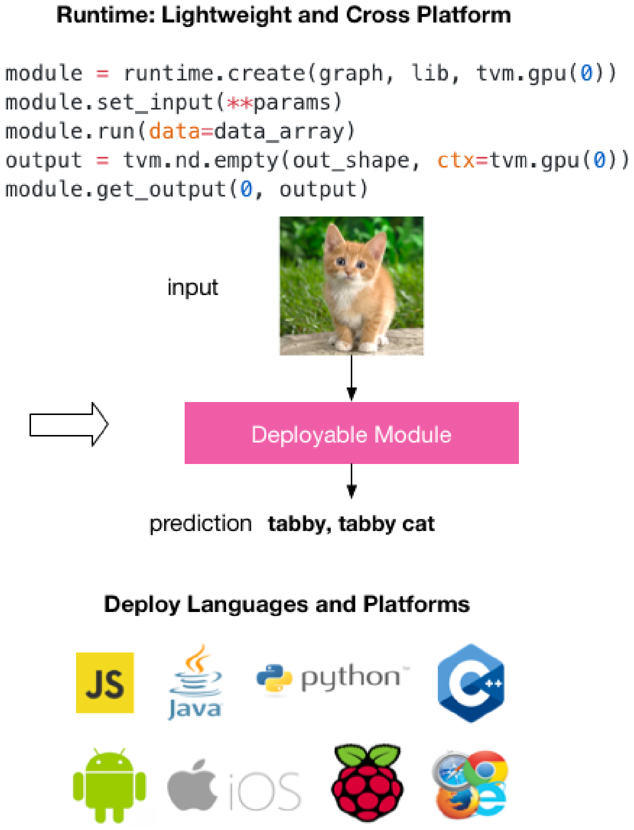
SGLang: Efficient Execution of Structured Language Model Programs, 2024

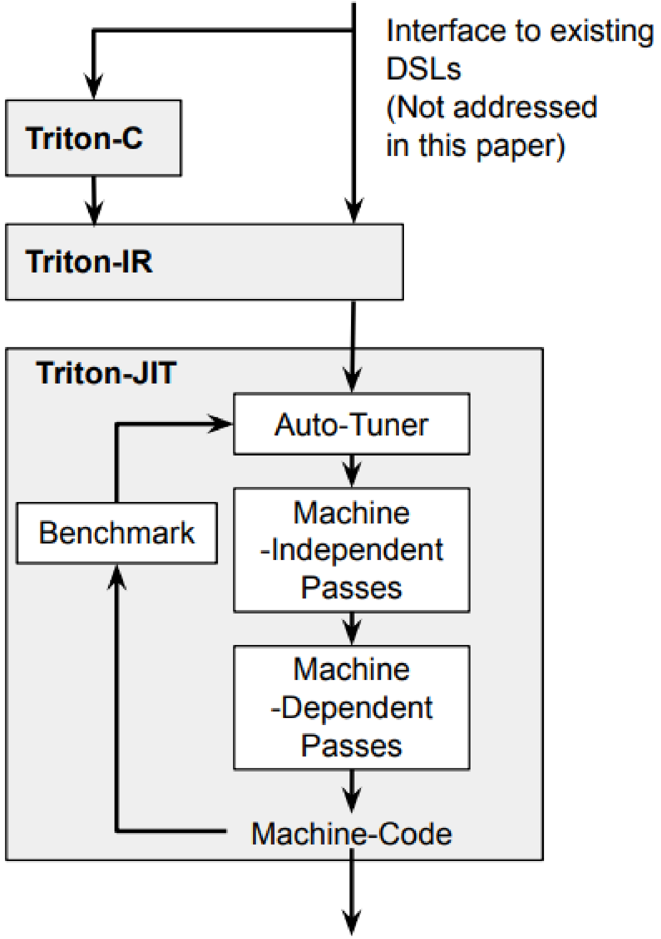


DL Compilation

Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations, 2019







TVM: An Automated End-to-End Optimizing Compiler for Deep Learning, 2018

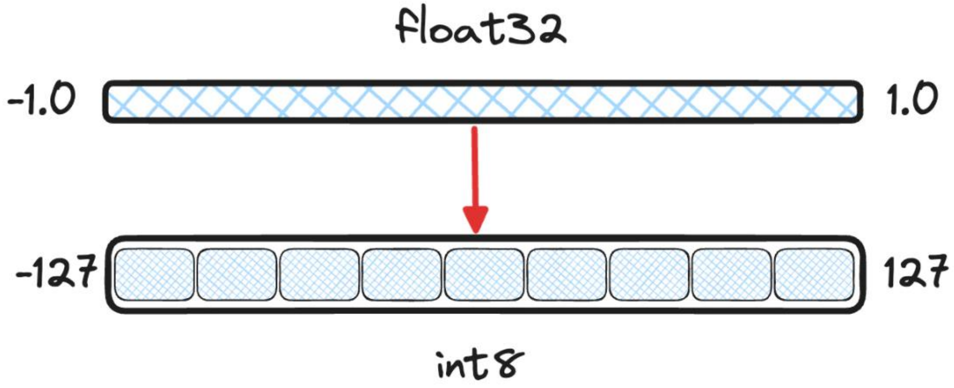
Efficient and Effective Algorithms

Compression Strategies

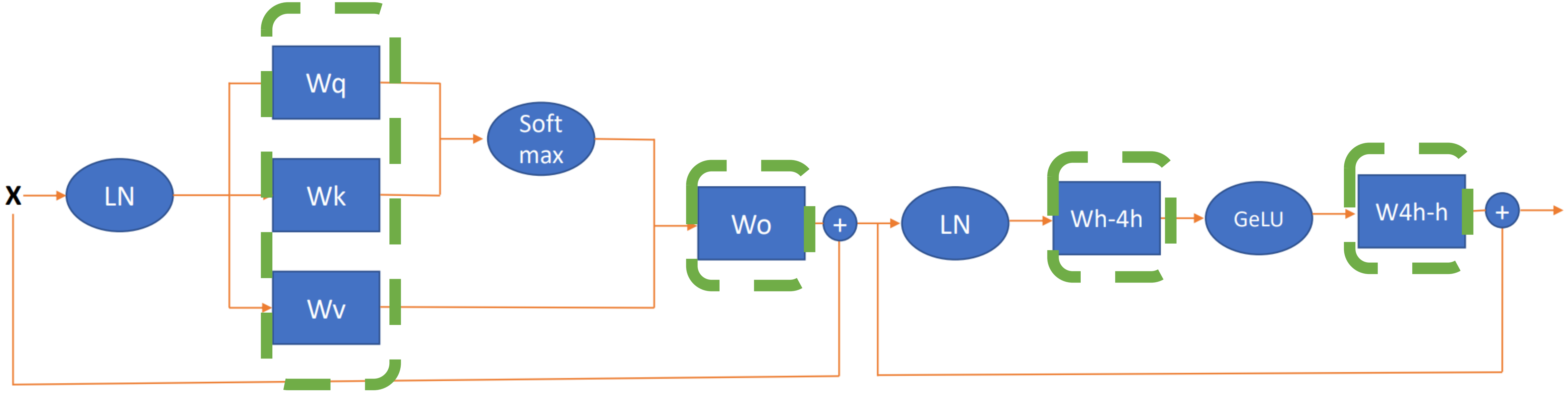
- Quantization
- Sparsification
- Distillation
- Low-Rank Decomposition
- …

Quantization: Quick Recap

- Reduce the bits per weight, saving memory consumption
- Accelerate inference speed on supporting hardware



8-bit Weight Quantization



- 8-bit weight quantization

$$\mathbf{x}_{quantize} = round\left(clamp(\frac{\mathbf{x}}{S}, -2^{bit-1}, 2^{bit-1} - 1)\right)$$

FP32 weight matrix

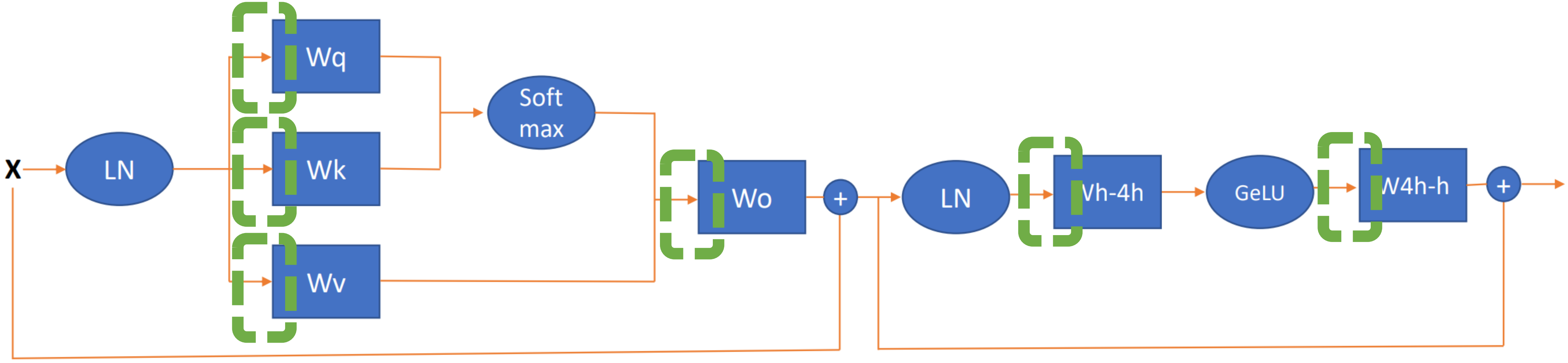
1.1	2.2	0.1	-0.1	-5.5	-6.6
…					
…					
…					
…					
1.1	2.1	0.1	-0.1	-4.8	-6.6

≈ Scaling Factor $1/S$ 0.05 *

8-bit quantization

21	42	2	-2	-106	-127
…					
…					
…					
…					
21	40	2	-2	-92	-127

8-bit Activation Quantization



- 8-bit activation

  (Input to the linear layer)

$$\mathbf{x}_{quantize} = round\left(clamp(\frac{\mathbf{x}}{S}, -2^{bit-1}, 2^{bit-1}-1)\right)$$

FP32 input matrix

1.1	2.2	0.1	-0.1	-5.5	-6.6
…					
…					
…					
…					
1.1	2.1	0.1	-0.1	-4.8	-6.6

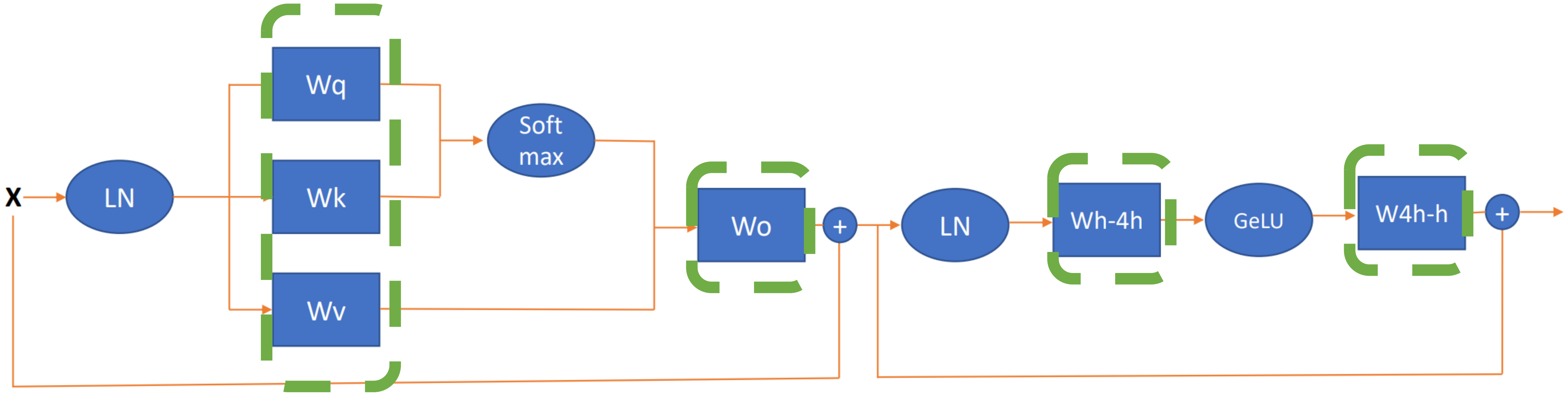
Scaling Factor $1/S$

$\approx$ 0.05*

8-bit quantization

21	42	2	-2	-106	-127
…					
…					
…					
…					
21	40	2	-2	-92	-127

Weight Ternarization



- Ternarization (weight)

  W: weight matrix, FP32.

  Q(W): Quantization mapping, 2-bit.

  With α=$\|W\|_1$/n, for some scalar s

$$Q(W_{ij}) = \begin{cases} \alpha \cdot \mathrm{sign}(W_{ij}) & \text{when } |W_{ij}| > s \\ 0 & \text{when } |W_{ij}| < s \end{cases}$$

FP32 weight matrix

1.1	2.2	0.1	-0.1	-5.5	-6.6
…					
…					
…					
…					
1.1	2.1	0.1	-0.1	-4.8	-6.0

$\approx$

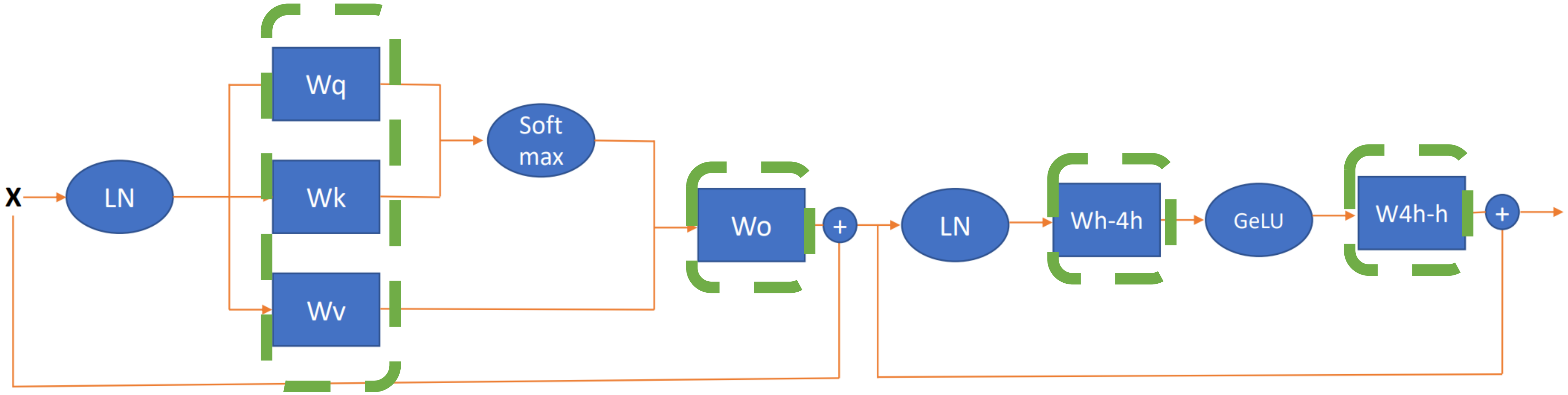
Scaling Factor α

2.06 *

2-bit quantization

1	1	0	0	-1	-1
…					
…					
…					
…					
1	1	0	0	-1	-1

Weight Binarization



- Binarization (weight)

  W: weight matrix, FP32.

  Q(W): Quantization mapping, 1-bit.

  With α= $\|W\|_1$/n

$$Q(W_{ij}) = \alpha \cdot \text{sign}(W_{ij})$$

FP32 weight matrix

1.1	2.2	0.1	-0.1	-5.5	-6.6
…					
…					
…					
…					
1.1	2.1	0.1	-0.1	-4.8	-6.0

≈ 1.4 *

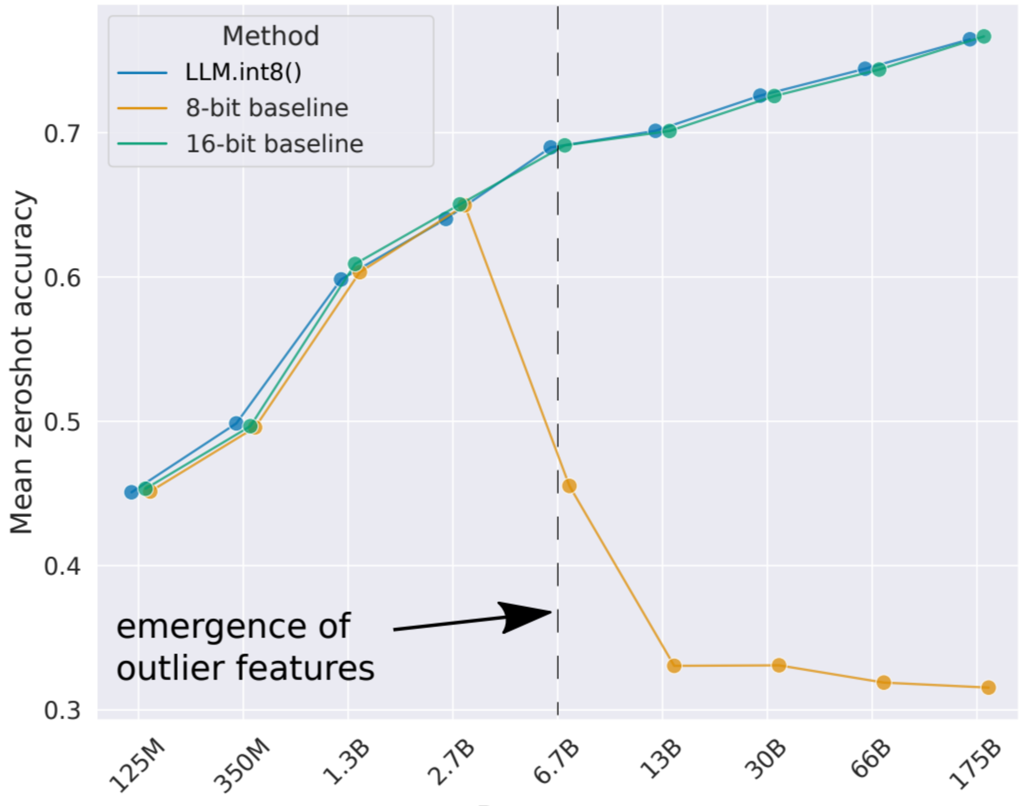
Scaling Factor α

1-bit quantization

1	1	1	-1	-1	-1
…					
…					
…					
…					
1	1	1	-1	-1	-1

Challenges to Quantize LLMs

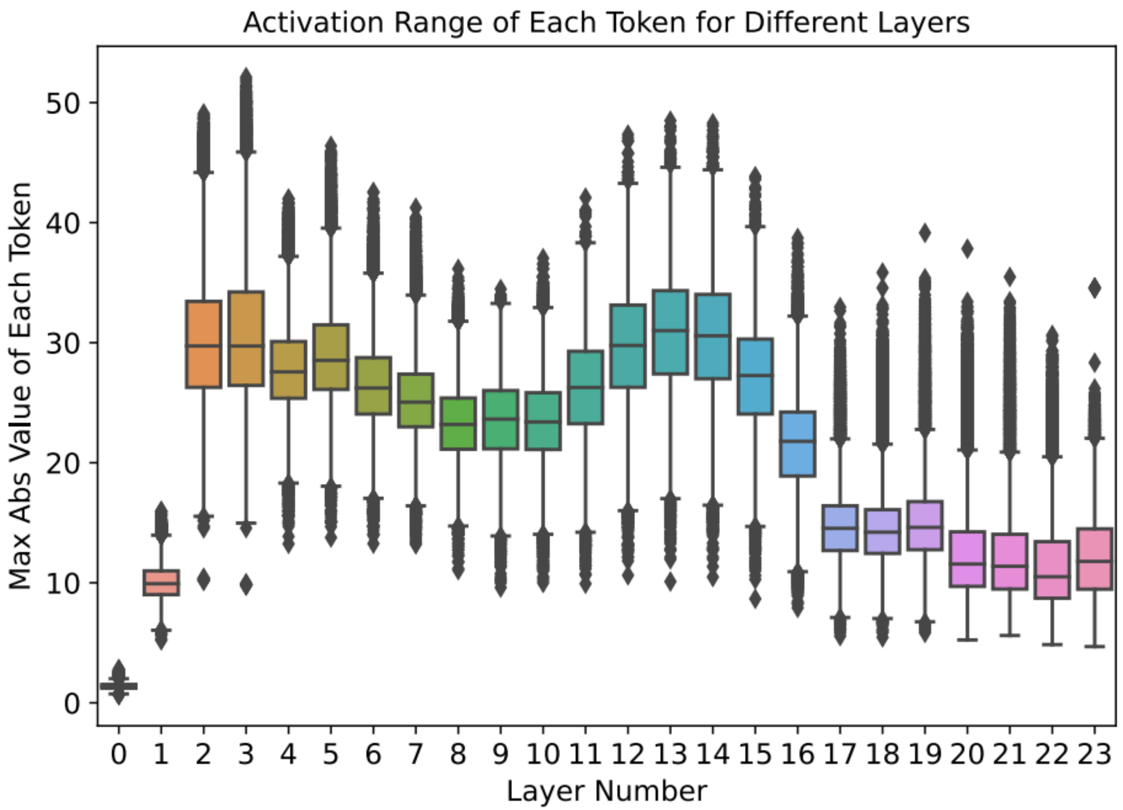
- Standard quantization strategy leads to catastrophic accuracy drop



LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale, 2023

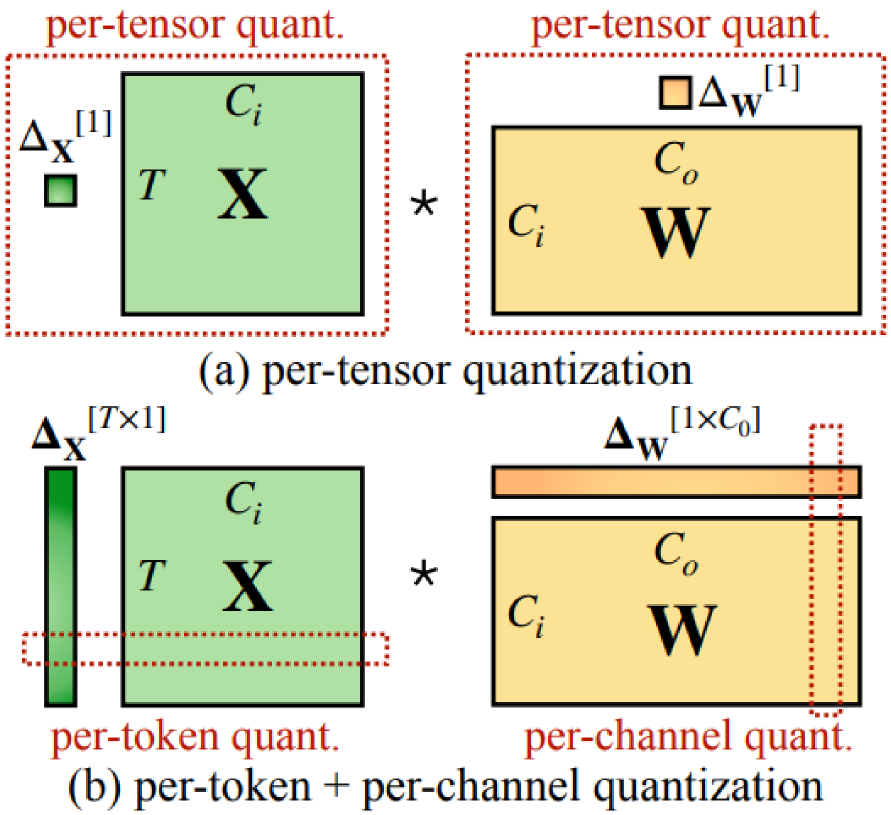
Challenges to Quantize LLMs

- High dynamic ranges of activation, leading to large quantization errors



Fine-grained Quantization

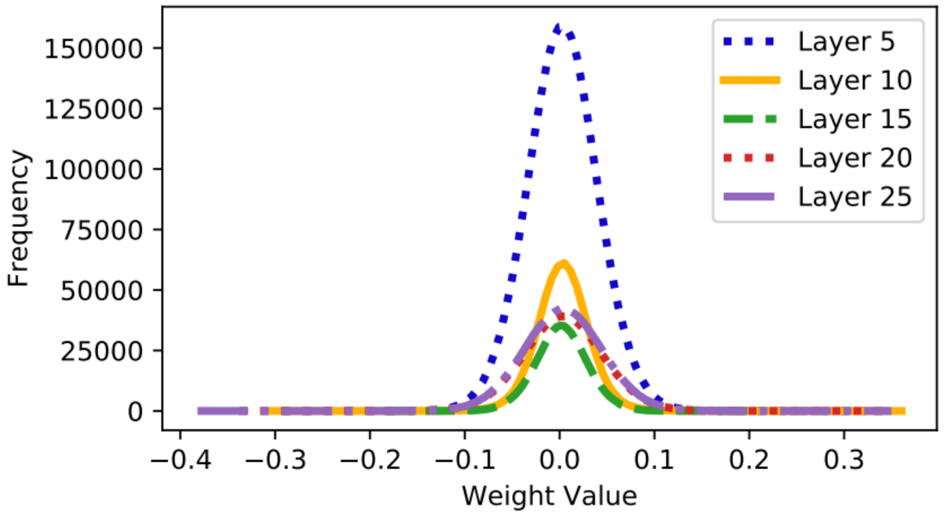
- Per-tensor quantization
  - Low accuracy
  - Fast to quantize/dequantize
- Per-token/channel quantization
  - High accuracy
  - Slower to quantize/dequantize
  - Custom kernels required



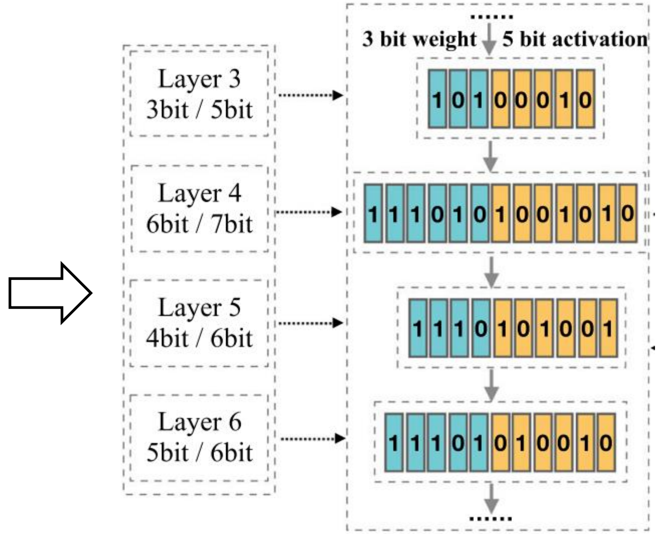
ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers, NeurIPS 2022

Mixed Precision Quantization

- Weights follow Gaussian distribution
- Outliers remain in original form, quantize the rest of the values
- Different bits for different layers



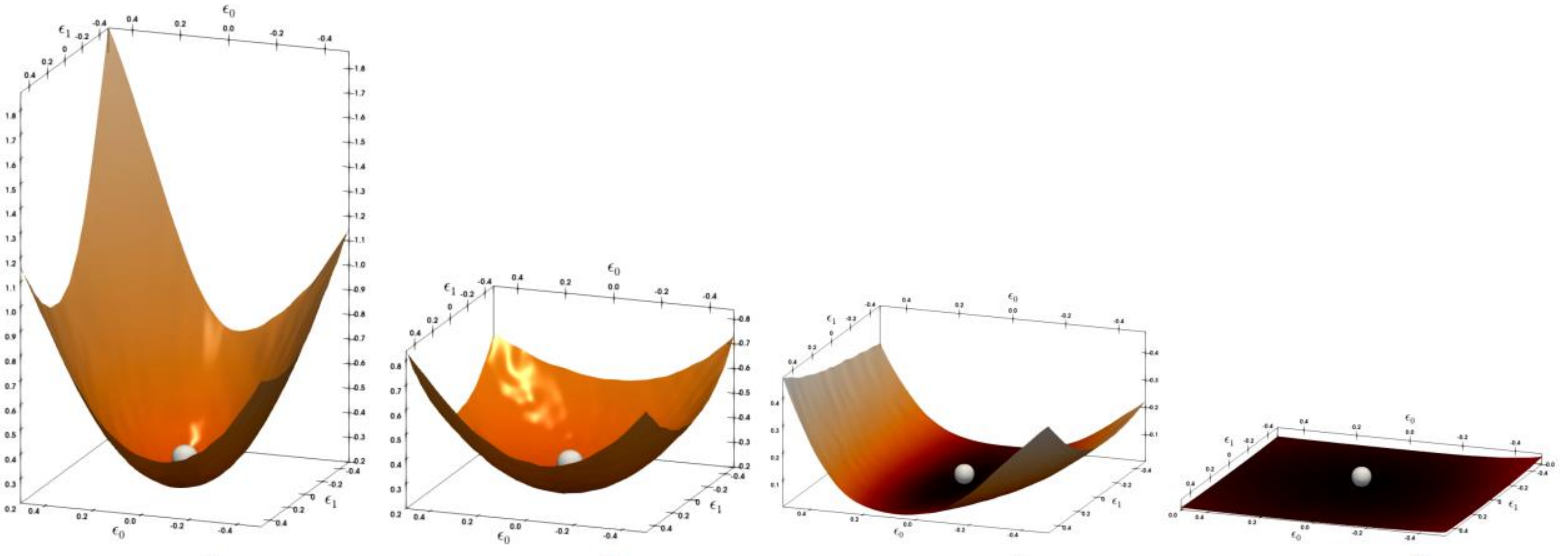
Per-layer weight distribution of BERT model



GOBO: Quantizing Attention-Based NLP Models for Low Latency and Energy Efficient Inference, MICRO 2020

Second Order Information

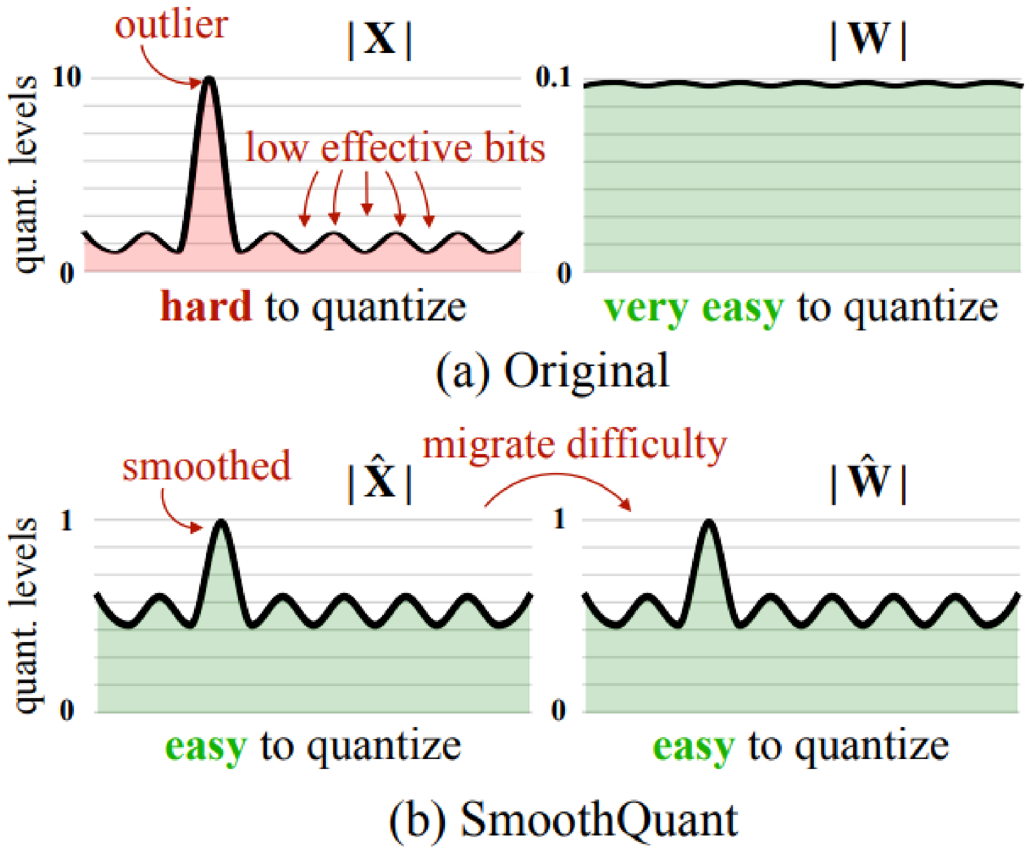
- Analyze the loss curvature (Hessian matrices) to help identify layer sensitivity



(a) MNLI 4th layer (b) MNLI 10th layer (c) CoNLL-03 4th layer (d) CoNLL-03 11th layer

GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers, ICLR 2023

Outlier Smoothing



SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models, ICML 2023

Model Pruning

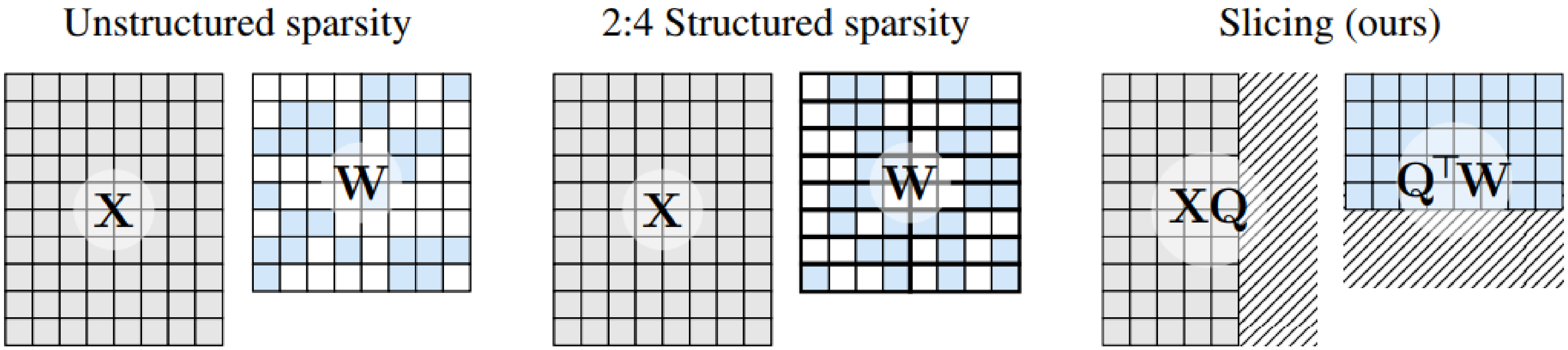
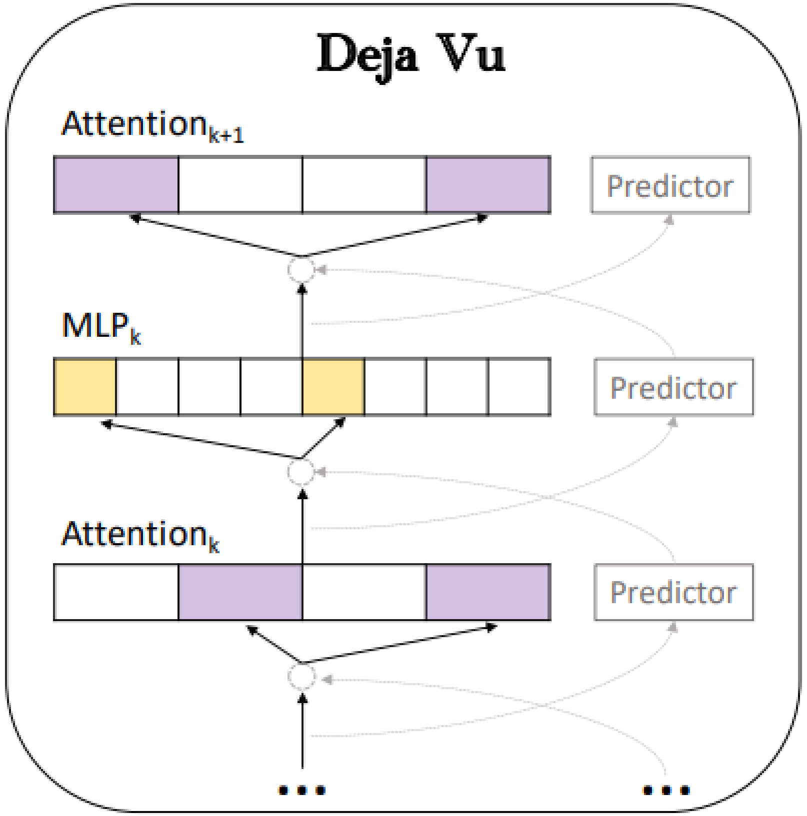


Figure 1: Matrix multiplication of the signal **X** and a weight matrix **W** under different types of sparsity. **Left**: unstructured sparsity, where some elements of **W** are zero, and **X** is dense. **Middle**: 2:4 structured sparsity, where each block of four weight matrix entries contains two zeros, and **X** is dense. **Right**: SliceGPT, where after introducing transformation **Q**, all the sparsity is arranged to the bottom rows of **W** and the corresponding columns of **X** are removed.

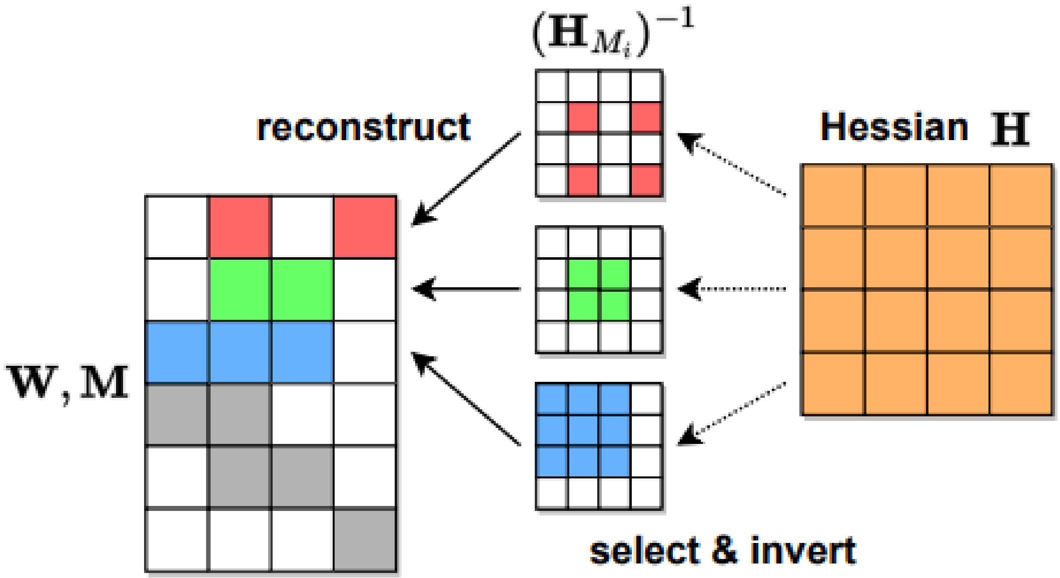
SliceGPT: Compress Large Language Models by Deleting Rows and Columns, ICLR 2024

Model Pruning

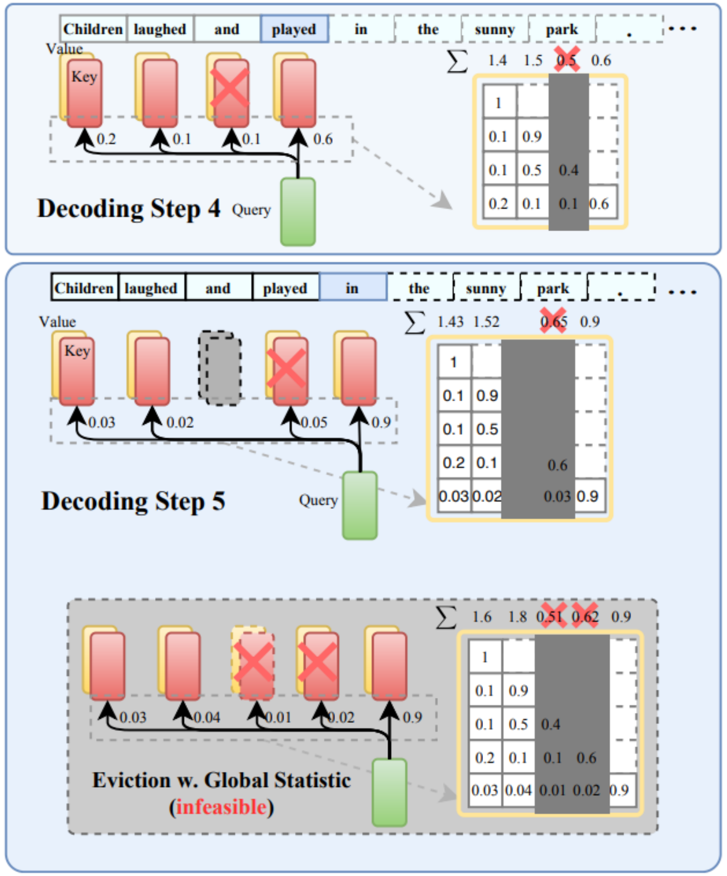


Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time, 2023

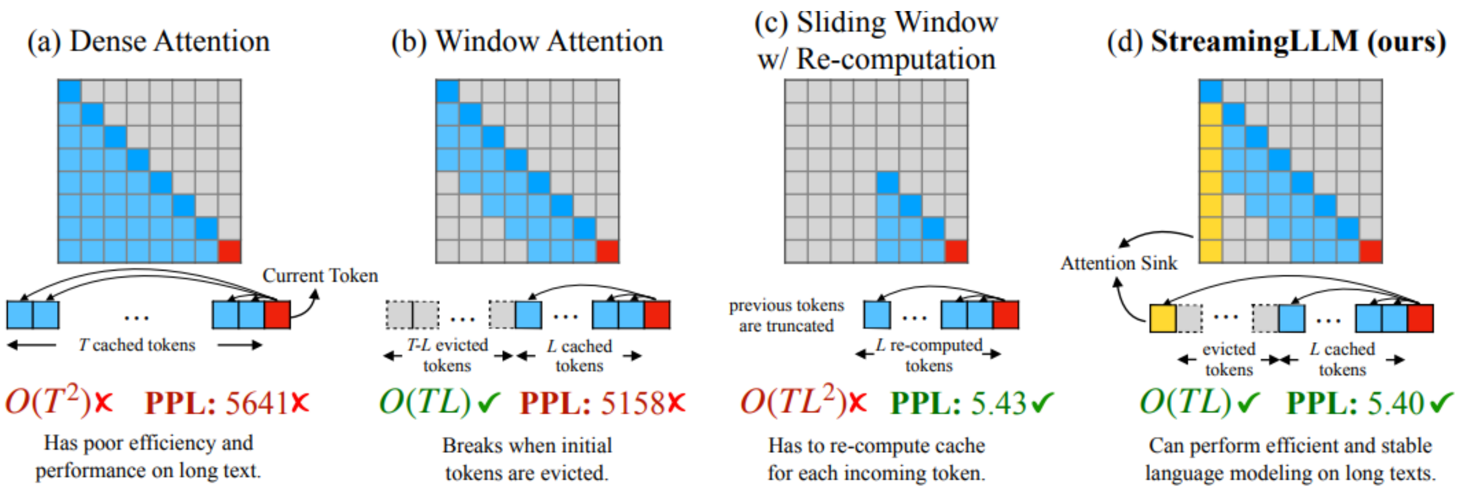
SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot, 2023



KV Cache Compression

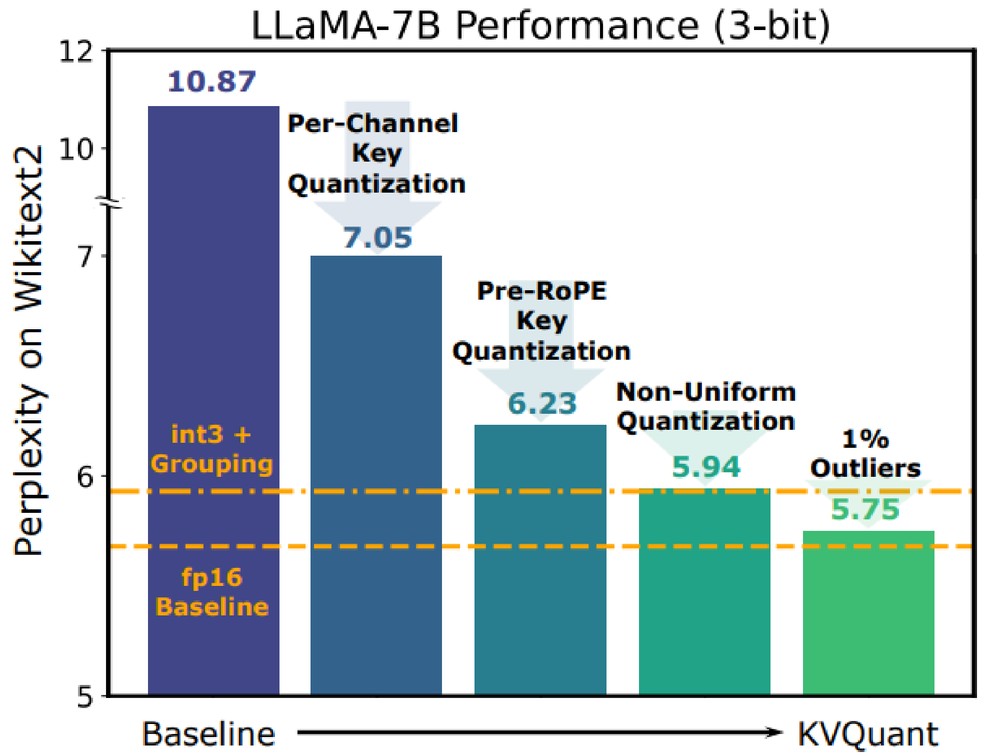


Efficient Streaming Language Models with Attention Sinks, ICL 2024



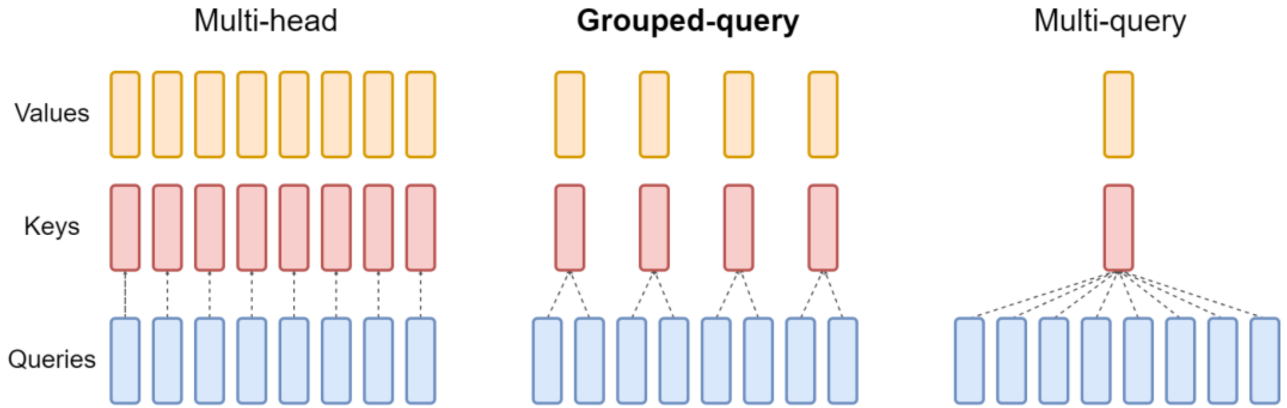
H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models, 2023

KV Cache Compression



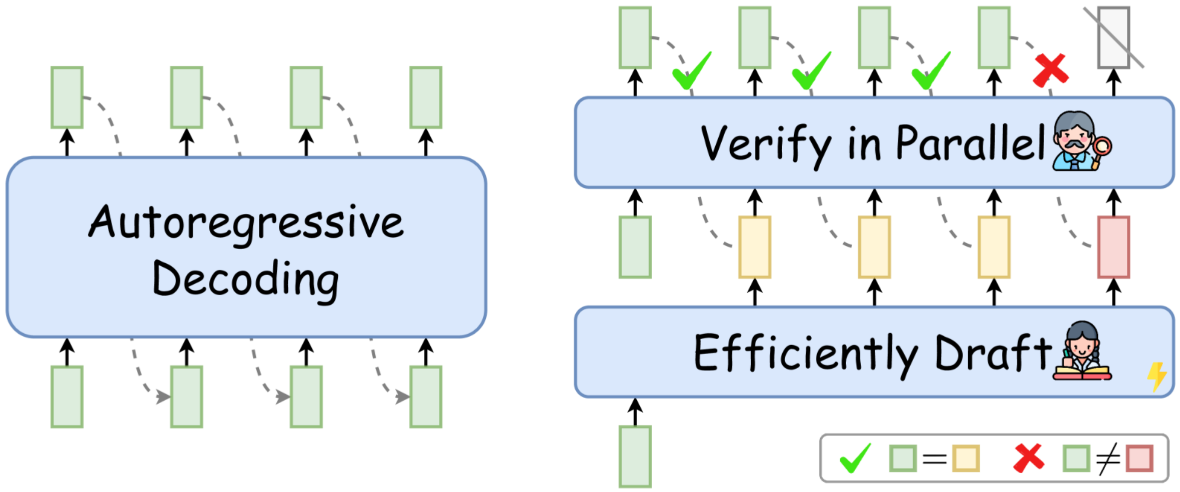
KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization, 2024

GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints, 2023

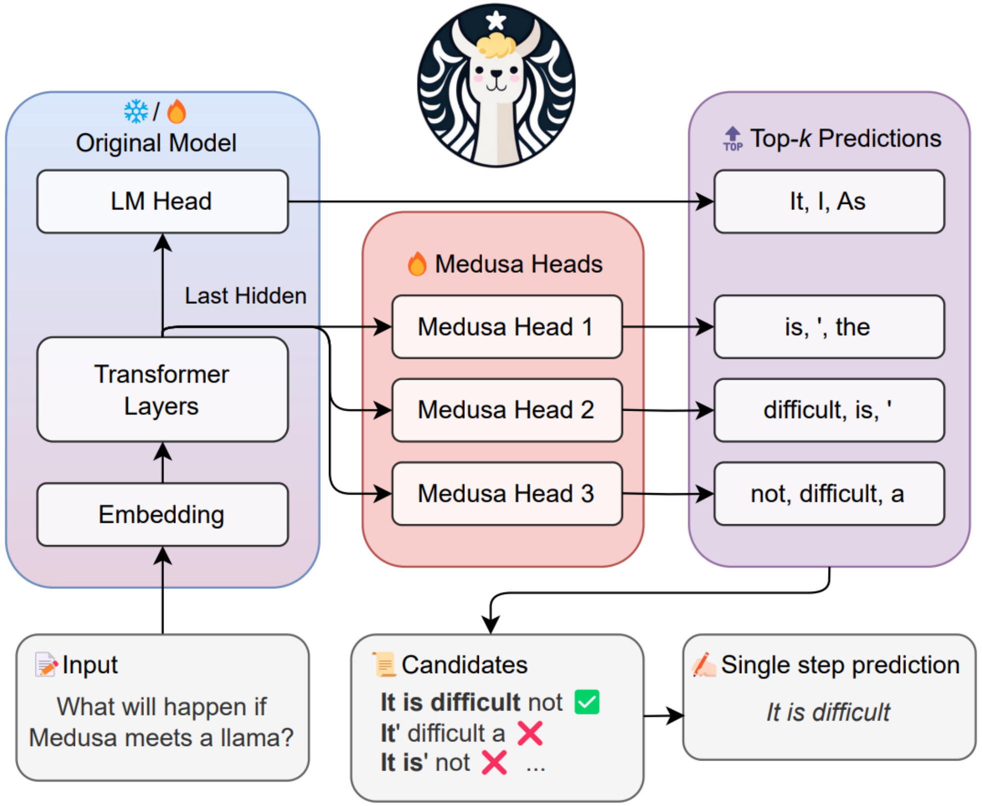


Speculative/Parallel Decoding

MEDUSA: Simple LLM Inference Acceleration Framework with Multiple, 2024



Fast Inference from Transformers via Speculative Decoding, 2023



Inference Optimizations

- Reduce the inference latency to satisfy latency SLA
- Improve the inference throughput to save cost
- Reduce the memory footprint of the model by using fewer GPU devices and less GPU memory
- Improve agility from DNN prototype to deployment

QA