Inference Optimization for Foundation Models – Part 2. Model Compression

Yan Xu

Houston Machine Learning

Sep 20, 2024

Reference: KDD tutorial - Inference Optimization of Foundation Models on AI Accelerators.

YanAITalk

@yanaitalk · 1.65K subscribers · 56 videos

Make machine learning easy to understand! ...more

Customize channel | Manage videos

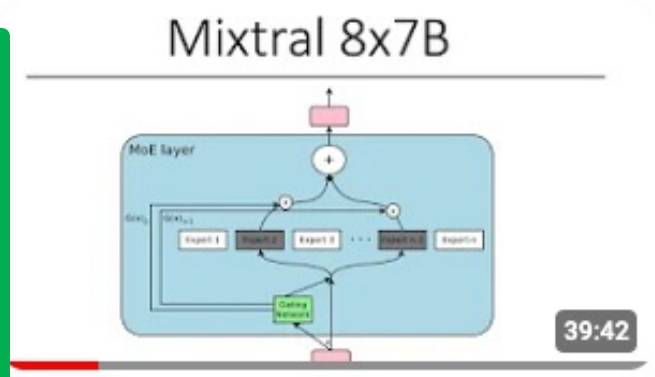
Home | Videos | Playlists | Community

Latest | Popular | Oldest

Part 1. KW cache, architecture, FlashAttention

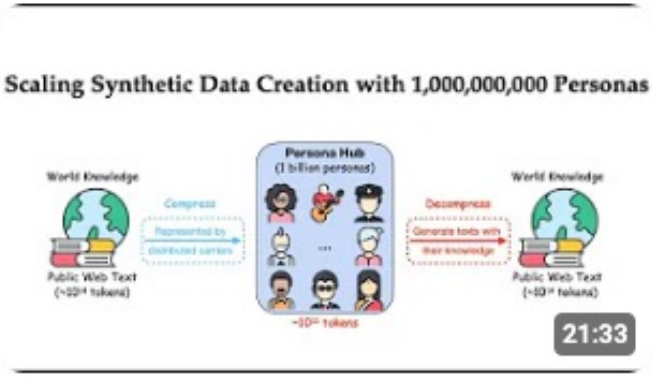
LLM inference optimization: Architecture, KV cache and Flash attention

263 views • 13 days ago



Mixture of Experts: Mixtral 8x7B

12 views • 1 month ago



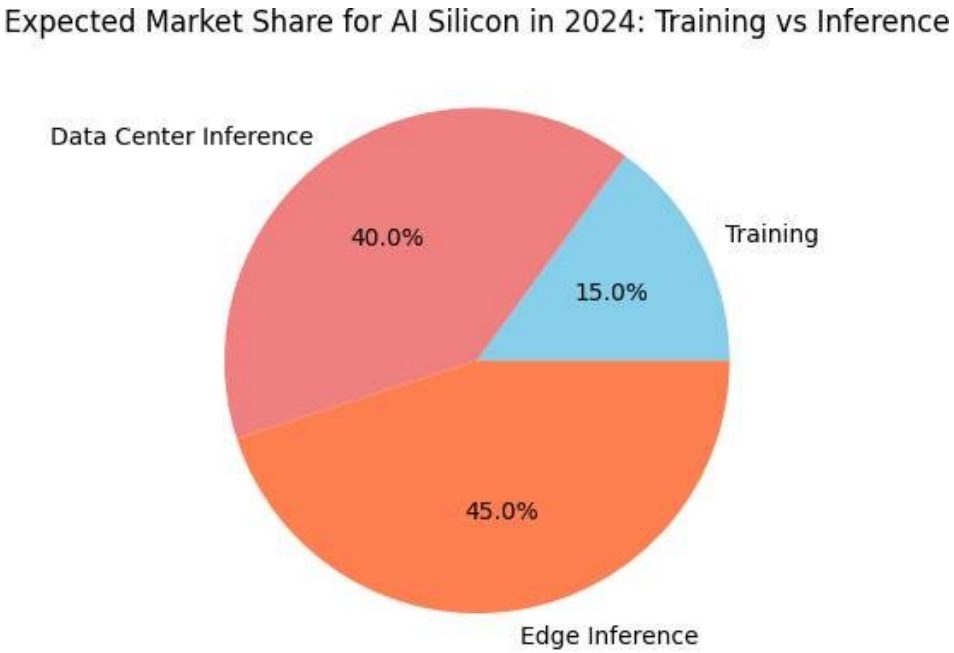
Scaling Synthetic Data Creation with 1,000,000,000 Personas

54 views • 1 month ago

Recap: Why inference optimization?

- Projected market size of inference vs training
  - LLM pre-training: one time, or update every month
  - LLM inference: far more frequent for serving millions of users with low latency.

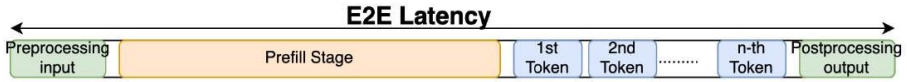
- Inference optimization is essential to meet serving criteria of latency and energy cost
  - E.g., GPT3 (175 billion parameters), inference optimization can reduce from a few second to millisecond or lower to serve ChatGPT.



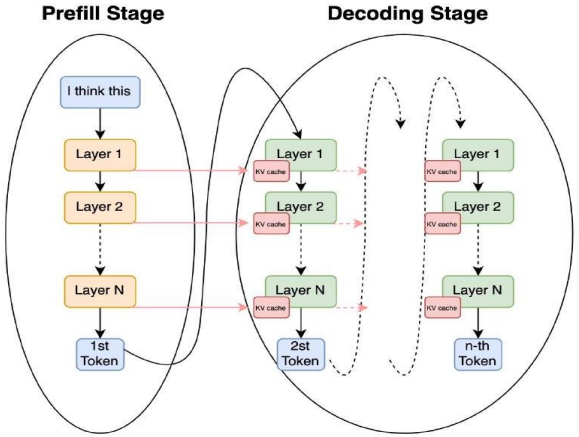
[Source: “The AI chip market landscape”, TECHSPOT, 2023]

Recap: The space of inference optimizations

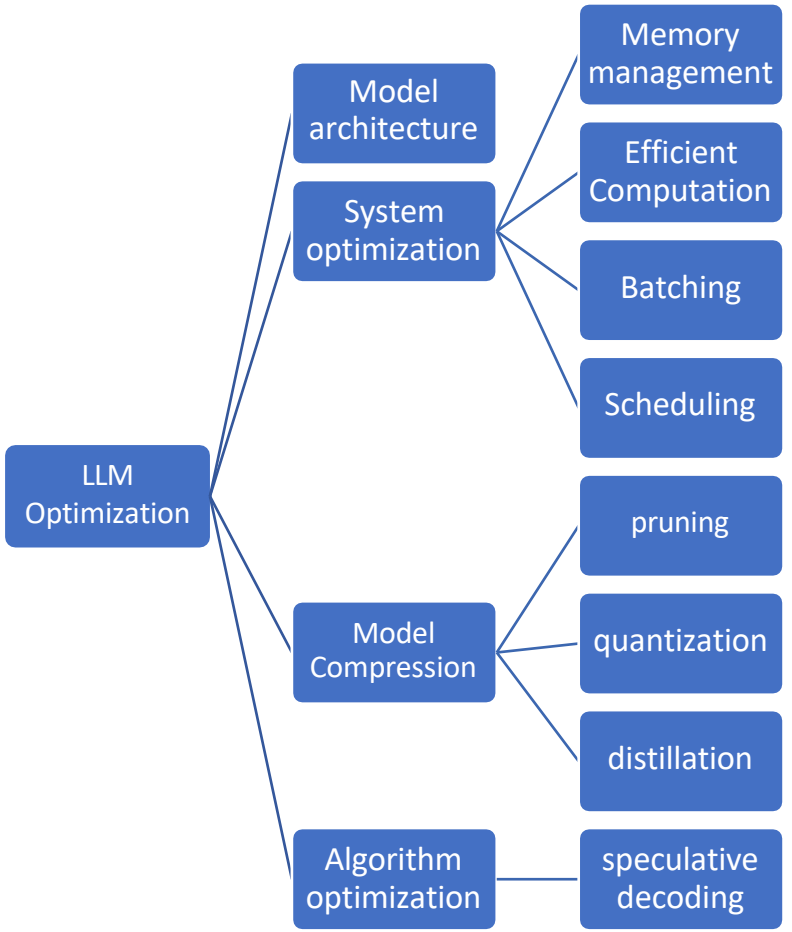
Metric:



Inference Stages:



- With these metrics, optimize prefill, decoding and other stages of LLM via
  - Model architecture choice
  - System optimizations
  - **Model compressions**
  - Algorithmic optimization



Primer on model compression

1. Motivation
2. Quantization
3. Distillation
4. Pruning

Motivation

Challenge: Inference is *expensive* and *slow* with *large* models

- **Need multiple AI accelerator chips**

  Llama-3.1 405B
  - ≈ 810 GB in native precision (FP16)

AWS EC2 Trainium Source: https://aws.amazon.com/ec2/instance-types/trn1/ (date: 8/18/2024)

Key Idea

- Compress a large model into a smaller model
  - Use low-precision data types
  - Use fewer parameters
- Trade-off between accuracy and inference improvement

Benefits

- Reduces the number of flops
  - Applies to *some* model compression approaches
    (e.g., pruning: less important parameters are set
    to zero or removed)
  - Speeds up prefill
    Removing 50% of the parameters of the model reduces the flops by a half

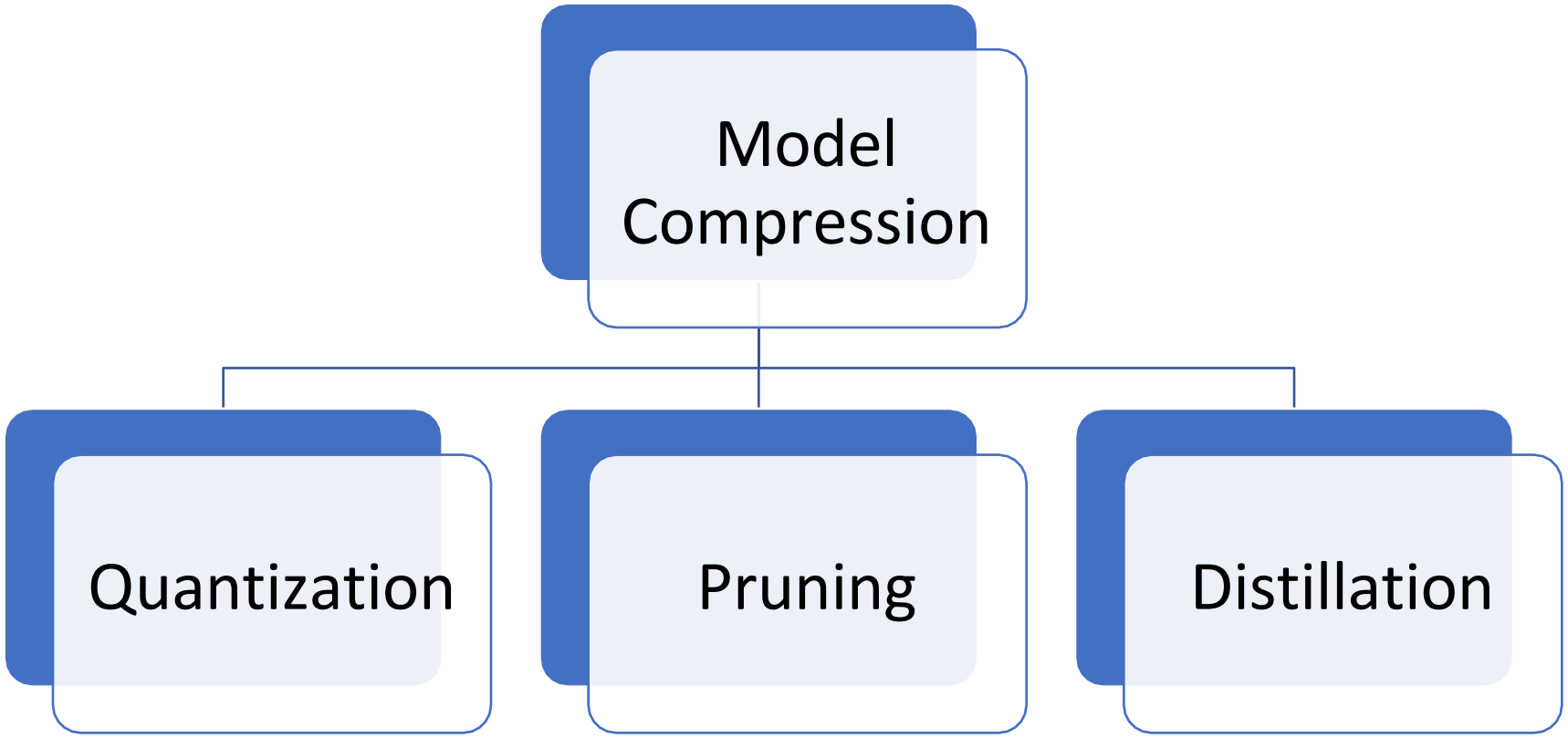
- Reduces the memory footprint of LLMs
  - Applies to *all* model compression methods
  - Number of AI accelerators to serve the model goes down
  - Speeds up decoding

Model compression measurement

$$\text{Compression Ratio } (r) = \frac{Uncompressed\ Model\ Size}{Compressed\ Model\ Size}$$

$r > 1 \Rightarrow$ effective compression

Model compression methods



Model quantization

Rich data types

Data Type	Bit Width	Range	Precision	Use Case
FP32	32 bits	$\sim -3.4 \times 10^{38}$ to 3.4×10^{38}	~6-7 decimal digits	Standard for training neural networks
FP16	16 bits	$\sim -6.55 \times 10^{4}$ to 6.55×10^{4}	~3-4 decimal digits	Faster inference, especially on GPUs and TPUs
INT8	8 bits	-128 to 127	Integer precision	Efficient inference, commonly used in deployment
UINT8	8 bits	0 to 255	Integer precision	Used for quantized image data or non-negative values
INT4	4 bits	-8 to 7	Integer precision (very low)	Research on ultra-lightweight models
UINT4	4 bits	0 to 15	Integer precision (very low)	Rarely used, explored in highly constrained devices

Quantization

- Computes and stores *tensors* at lower bit-widths

16 bit (e.g., FP16) → 8 bit (e.g., INT8) ⇒ r = 2

16 bit (e.g., FP16) → 4 bit (e.g., INT4) ⇒ r = 4

Typically quantized components in LLMs

- Key-Value Cache
- Weights of linear layers in attention and feedforward layers
- Activations in feedforward layers

How to quantize different components?

Aspect	Static Quantization	Dynamic Quantization
Quantization of	Weights and activations	Only weights (activations dynamically)
When applied	Pre-inference (calibration needed)	During inference (no calibration)
Performance	Typically better	Moderate
Accuracy	Higher due to calibrated activations	Slightly lower
Calibration	Requires a representative dataset	No calibration needed
Use case	High-performance, optimized inference	Quick deployment, flexible use

Quantization formula (One example)

From float32 to int8: Only 256 values can be represented in int8, while float32 can represent a very wide range of values. The idea is to find the best way to project our range [a, b] of float32 values to the int8 space of [-128, 127].

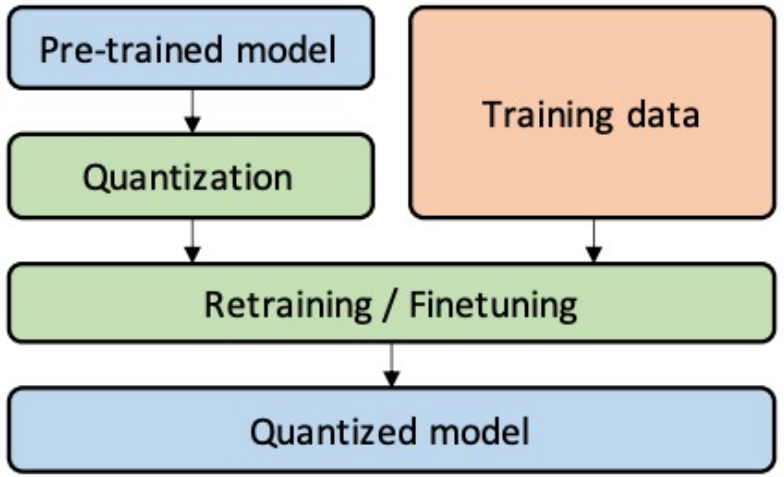
```
x_q = clip(round(x/S + Z), round(a/S + Z), round(b/S + Z))
```

where:

- `x_q` is the quantized `int8` value associated to `x`
- `S` and `Z` are the quantization parameters
  - `S` is the scale, and is a positive `float32`
  - `Z` is called the zero-point, it is the `int8` value corresponding to the value `0` in the `float32` realm. This is important to be able to represent exactly the value `0` because it is used everywhere throughout machine learning models.

When to quantize?

Quantization-aware training



Post-training quantization

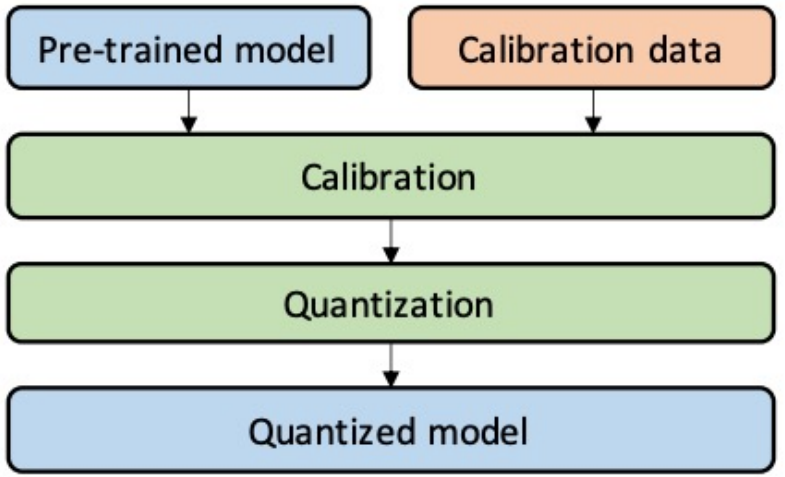


Figure 4: *Comparison between Quantization-Aware Training (QAT, Left) and Post-Training Quantization (PTQ, Right). In QAT, a pre-trained model is quantized and then finetuned using training data to adjust parameters and recover accuracy degradation. In PTQ, a pre-trained model is calibrated using calibration data (e.g., a small subset of training data) to compute the clipping ranges and the scaling factors. Then, the model is quantized based on the calibration result. Note that the calibration process is often conducted in parallel with the finetuning process for QAT.*

1.5-bit LLM

BitNet Architecture: Transformer with nn.Linear replaced by BitLinear

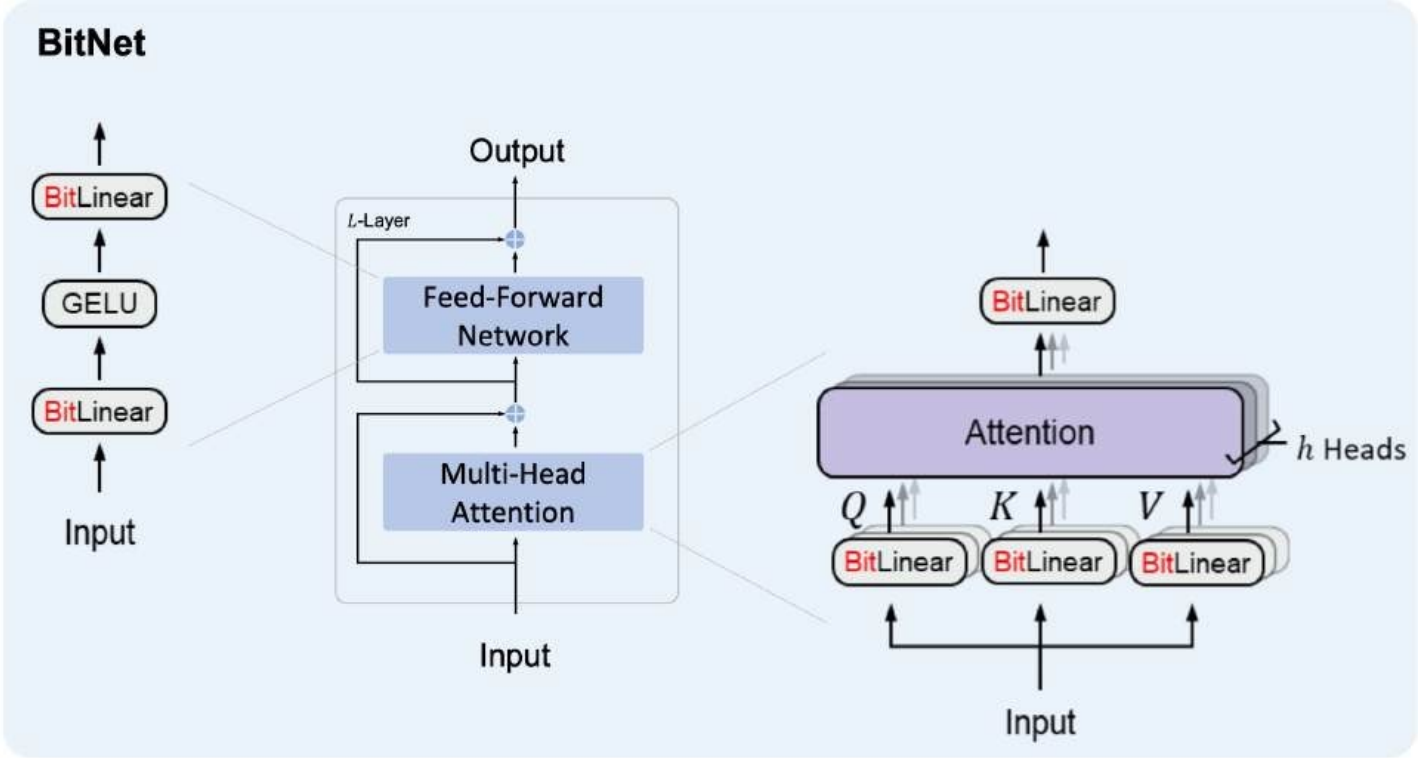
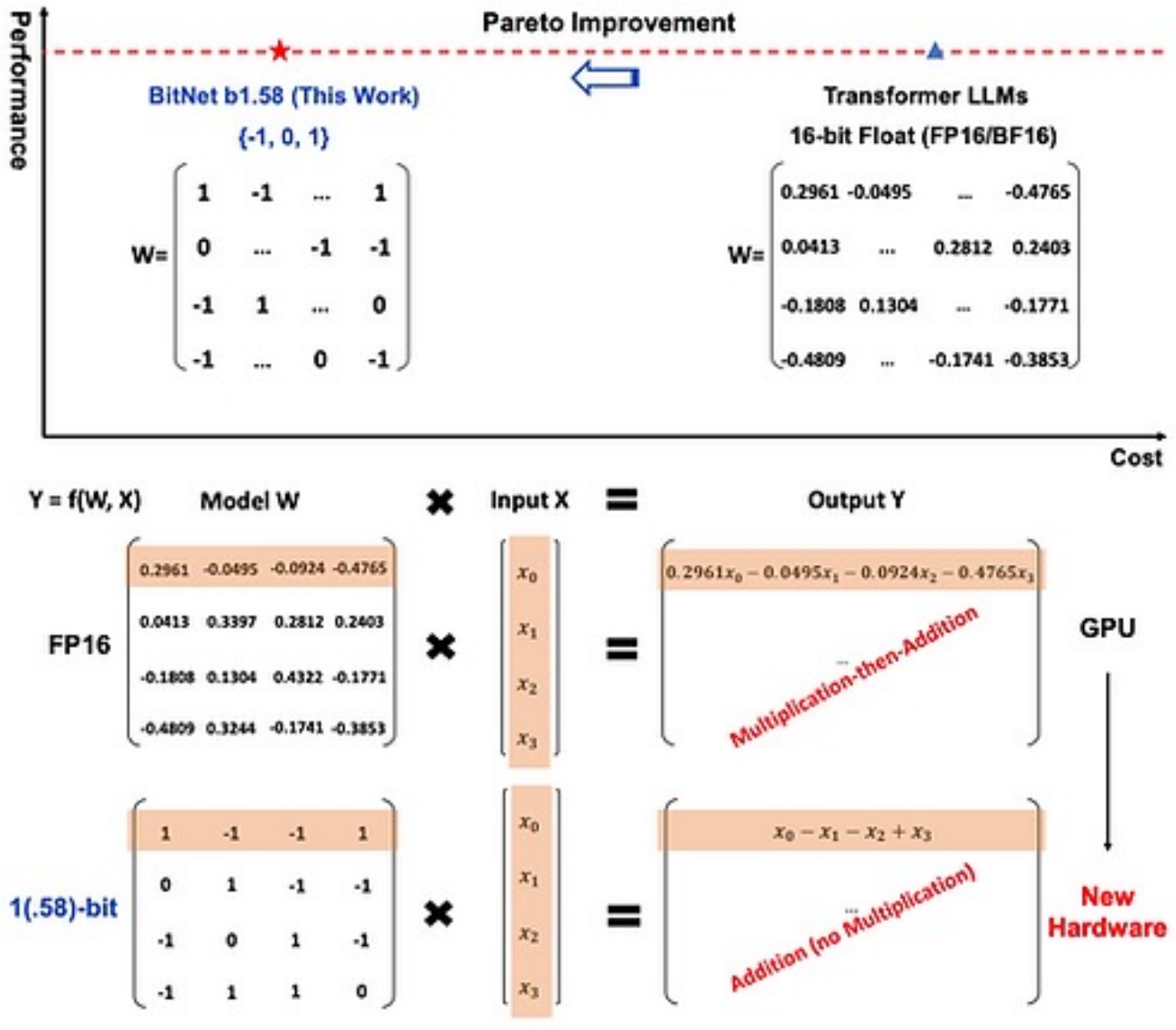


Image Source: Ma & Wang et al., The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits, arXiv 2024

1.5-bit LLM


Performance
Pareto Improvement
BitNet b1.58 (This Work)
{-1, 0, 1}
W=
1 -1 ... 1
0 ... -1 -1
-1 1 ... 0
-1 ... 0 -1
Transformer LLMs
16-bit Float (FP16/BF16)
W=
0.2961 -0.0495 ... -0.4765
0.0413 ... 0.2812 0.2403
-0.1808 0.1304 ... -0.1771
-0.4809 ... -0.1741 -0.3853
Cost
Y = f(W, X)
Model W
Input X
Output Y
FP16
0.2961 -0.0495 -0.0924 -0.4765
0.0413 0.3397 0.2812 0.2403
-0.1808 0.1304 0.4322 -0.1771
-0.4809 0.3244 -0.1741 -0.3853
x_0 x_1 x_2 x_3
$0.2961x_0 - 0.0495x_1 - 0.0924x_2 - 0.4765x_3$
...
Multiplication-then-Addition
GPU
1(.58)-bit
1 -1 -1 1
0 1 -1 -1
-1 0 1 -1
-1 1 1 0
x_0 x_1 x_2 x_3
$x_0 - x_1 - x_2 + x_3$
...
Addition (no Multiplication)
New Hardware

1.5-bit LLM deep dive – accuracy results

- BitNet b1.58 can match the performance of the full precision baseline starting from a 3B size.

Models	Size	ARCe	ARCc	HS	BQ	OQ	PQ	WGe	Avg.
LLaMA LLM	700M	54.7	23.0	37.0	60.0	20.2	68.9	54.8	45.5
BitNet b1.58	700M	51.8	21.4	35.1	58.2	20.0	68.1	55.2	44.3
LLaMA LLM	1.3B	56.9	23.5	38.5	59.1	21.6	70.0	53.9	46.2
BitNet b1.58	1.3B	54.9	24.2	37.7	56.7	19.6	68.8	55.8	45.4
LLaMA LLM	3B	62.1	25.6	43.3	61.8	24.6	72.1	58.2	49.7
BitNet b1.58	3B	**61.4**	**28.3**	**42.9**	**61.5**	**26.6**	**71.5**	**59.3**	**50.2**
BitNet b1.58	3.9B	**64.2**	**28.7**	**44.2**	**63.5**	**24.2**	**73.2**	**60.5**	**51.2**

Image Source: Ma & Wang et al., The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits, arXiv 2024

Decoding latency and memory consumption

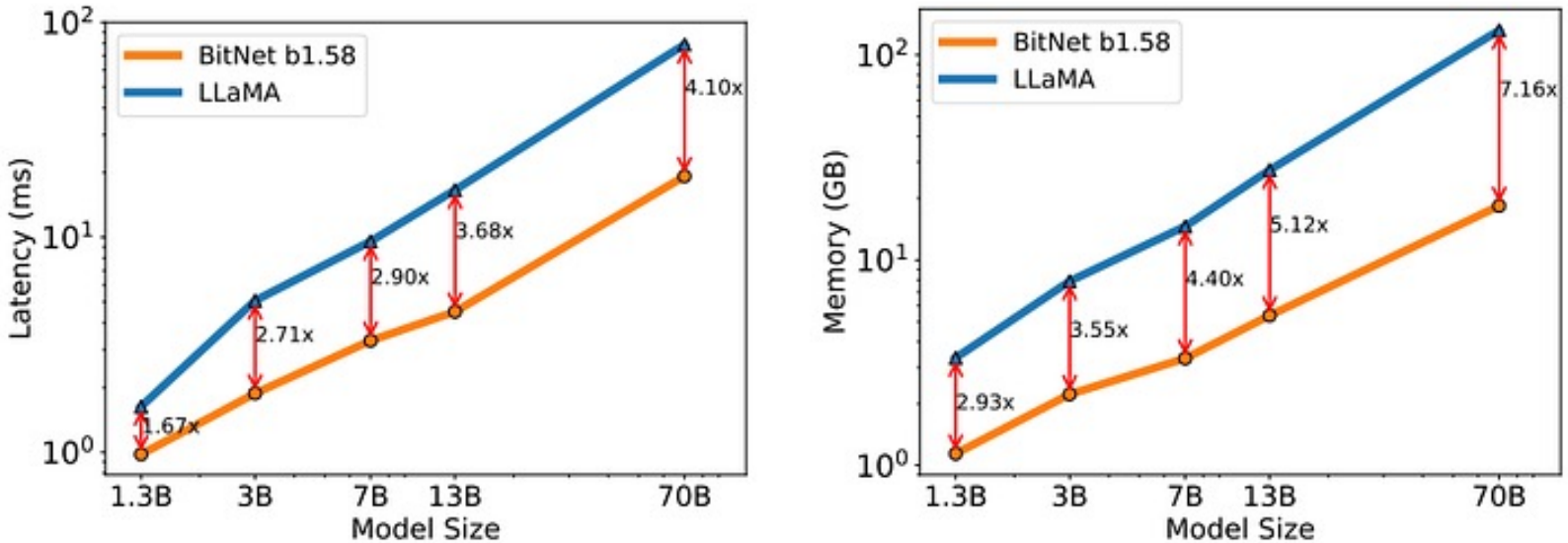


Figure 2: Decoding latency (Left) and memory consumption (Right) of BitNet b1.58 varying the model size.

Models	Size	Max Batch Size	Throughput (tokens/s)
LLaMA LLM	70B	16 (1.0x)	333 (1.0x)
BitNet b1.58	70B	**176 (11.0x)**	**2977 (8.9x)**

Table 3: Comparison of the throughput between BitNet b1.58 70B and LLaMA LLM 70B.

Model distillation

Model distillation history

Ensemble of models:

Performance(Ensemble of models) > Performance(single model)
Inference Cost(Ensemble of models) > Inference Cost(single model)

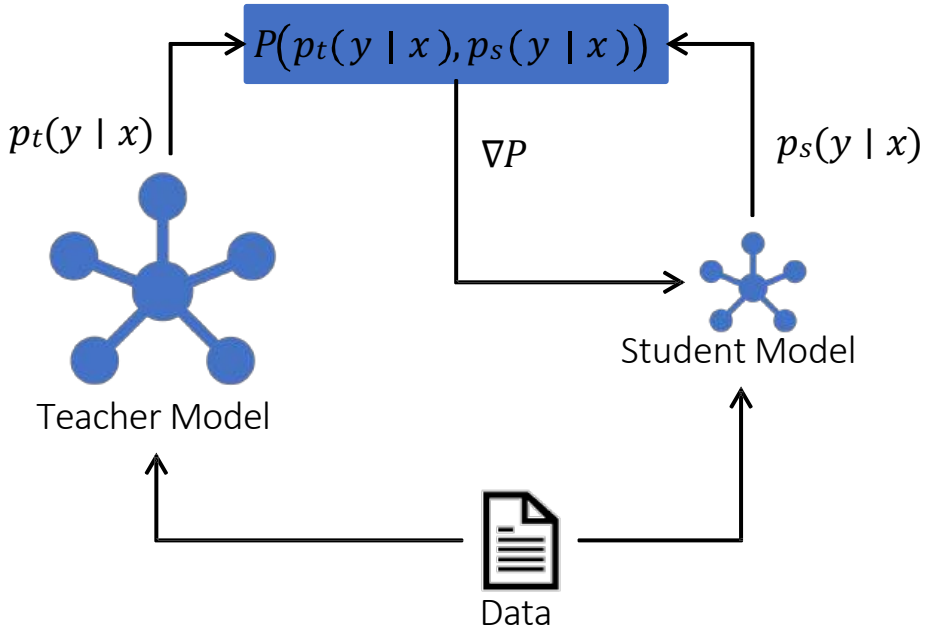
Model distillation:
Compress ensemble of models / Large model → smaller model
(Bucila et al., KDD 2006, Hinton et al., NeurIPS 2014)

Similar scenario with LLMs with billions of parameters

Key idea

Analogy from formal education:
Larger model (**teacher**) teaches the smaller model (**student**)

Train the student to match the teacher's perf. via a **distillation loss**



Benefits

- Train a smaller model (of same/different arch.) with fewer parameters
- Smaller memory footprint
  - Number of AI accelerators to host the model goes down
- Speeds up prefill
  - fewer flops during prefill
- Speeds up decoding
  - reduces the memory I/O for loading parameters and intermediate states

How to match a student to a teacher?

- Output logits (Hinton et al, NeurIPS DL Workshop 2014)
  - Compute probabilities $\left[p^{(1)}, \ldots, p^{(v)}\right]$ from logits $[z^{(1)}, \ldots, z^{(v)}]$ using softmax (for teacher & student) with v being the vocabulary size
  - Compute the cross-entropy loss

Geoffrey Hinton, Oriol Vinyals, Jeff Dean et al., **Distilling the Knowledge in a Neural Network**, NeurIPS 2014 Deep Learning Workshop

How to match a student to a teacher?

Softmax output of teacher:

$$q_i^T = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

Softmax output of student:

$$p_i^T = \frac{\exp(s_i/T)}{\sum_j \exp(s_j/T)}$$

Cross-entropy loss between teacher and student:

$$L_{\text{soft}} = -\sum_i q_i^T \log(p_i^T)$$

Cross-entropy loss between student and ground truth labels y:

$$L_{\text{hard}} = -\sum_i y_i \log(p_i)$$

Total loss for model distillation:

$$L_{\text{total}} = \alpha \cdot L_{\text{hard}} + (1-\alpha) \cdot T^2 \cdot L_{\text{soft}}$$

How to match a student to a teacher?

- Intermediate Weights (Romero et al., ICLR 2015)
  - Add an L_2 loss between teacher and student weights in addition to cross entropy loss
- Intermediate Features (Huang and Wang, arXiv 2017)
  - Minimize the distance between intermediate activations of teacher/student
  - Maximum Mean Discrepancy, L_2 distance

Huang and Wang, Like What You Like: Knowledge Distill via Neuron Selectivity Transfer, arXiv 2017

Model distillation knowledge transfer set

Typical approach:

Use an external dataset also called transfer set

Another approach (Symbolic distillation):

Teacher generates synthetic data (West et al., ACL 2022)

West et al., Symbolic Knowledge Distillation: from General Language Models to Commonsense Models, ACL 2022

Microsoft Phi-1/1.5/2 Models trained with synthetic textbook data

Breaking existing scaling laws with high-quality data

Use high-quality textbook-style data that have higher education values

Use synthetic textbook data with an existing LLM (e.g. GPT-3.5)

Train a smaller model

Achieve comparable or better performance

Small Language Model with big ambition:

Can we achieve ChatGPT's level of capability at the one billion parameters scale?

Phi series: Textbooks are all you need

```
To begin, let us define singular and nonsingular matrices. A matrix is said to be singular if its
determinant is zero. On the other hand, a matrix is said to be nonsingular if its determinant is not
 zero. Now, let's explore these concepts through examples.

Example 1: Consider the matrix A = np.array([[1, 2], [2, 4]]). We can check if this matrix is
singular or nonsingular using the determinant function. We can define a Python function, `
is_singular(A)`, which  returns true if the determinant of A is zero, and false otherwise.

import numpy as np
def is_singular(A):
    det = np.linalg.det(A)
    if det == 0:
        return True
    else:
        return False

A = np.array([[1, 2], [2, 4]])
print(is_singular(A)) # True
```

1B tokens of GPT-3.5 generated Python textbooks, synthesized to provide a high-quality source of natural language heavy text interleaved with relevant code snippets. We further targeted the content of these textbooks to cover topics that promote reasoning and basic algorithmic skills.

Phi-2 outperforms Mistral-7B, and the latter outperforms the Llama-2 models (7B, 13B, and 70B)

Model	Size	BBH	Commonsense Reasoning	Language Understanding	Math	Coding
Llama-2	7B	40.0	62.2	56.7	16.5	21.0
	13B	47.8	65.0	61.9	34.2	25.4
	70B	66.5	69.2	67.6	64.1	38.3
Mistral	7B	57.2	66.4	63.7	46.4	39.4
Phi-2	2.7B	59.2	68.8	62.0	61.1	53.7

Table 1. Averaged performance on grouped benchmarks compared to popular open-source SLMs.

Model	Size	BBH	BoolQ	MBPP	MMLU
Gemini Nano 2	3.2B	42.4	79.3	27.2	55.8
Phi-2	2.7B	59.3	83.3	59.1	56.7

Table 2. Comparison between Phi-2 and Gemini Nano 2 Model on Gemini's reported benchmarks.

Conclusion

YanAITalk

@yanaitalk · 1.65K subscribers · 56 videos

Make machine learning easy to understand! ...more

Customize channel Manage videos