# An Introduction to Quantization of Large Language Models

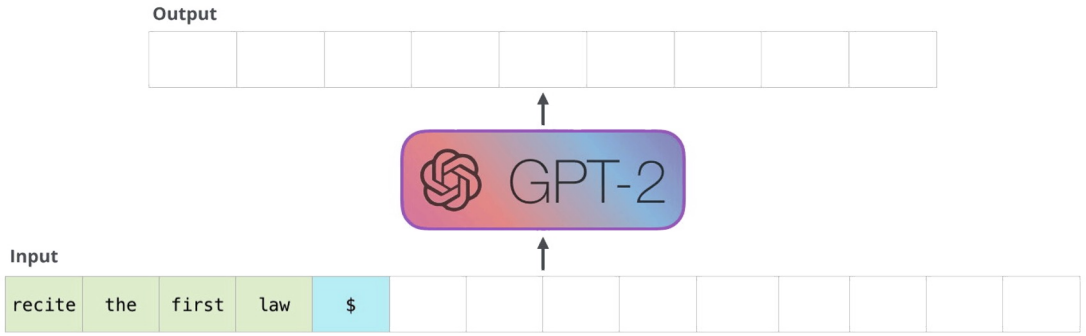
**Xuefei Ning**

Department of Electronic Engineering, Tsinghua University

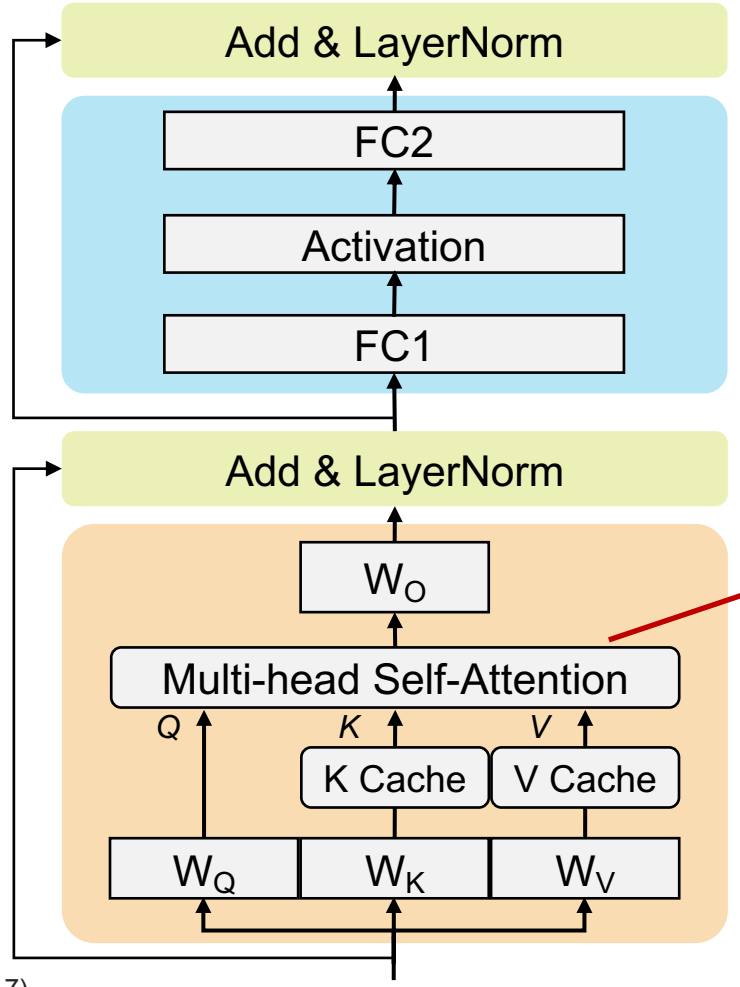

# Menu

# Large Language Model (LLM)

- Most large language models are based on the Transformer architecture[1].
- A Transformer block consists of :
  - Attention-Linear (generate matrix Q, K, V)
  - Multi-Head Attention
  - Feed Forward Network
  - Layer Norm
- A typical LLM inference process:




Example of Decoder's word-by-word translation




Feed Forward Network

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

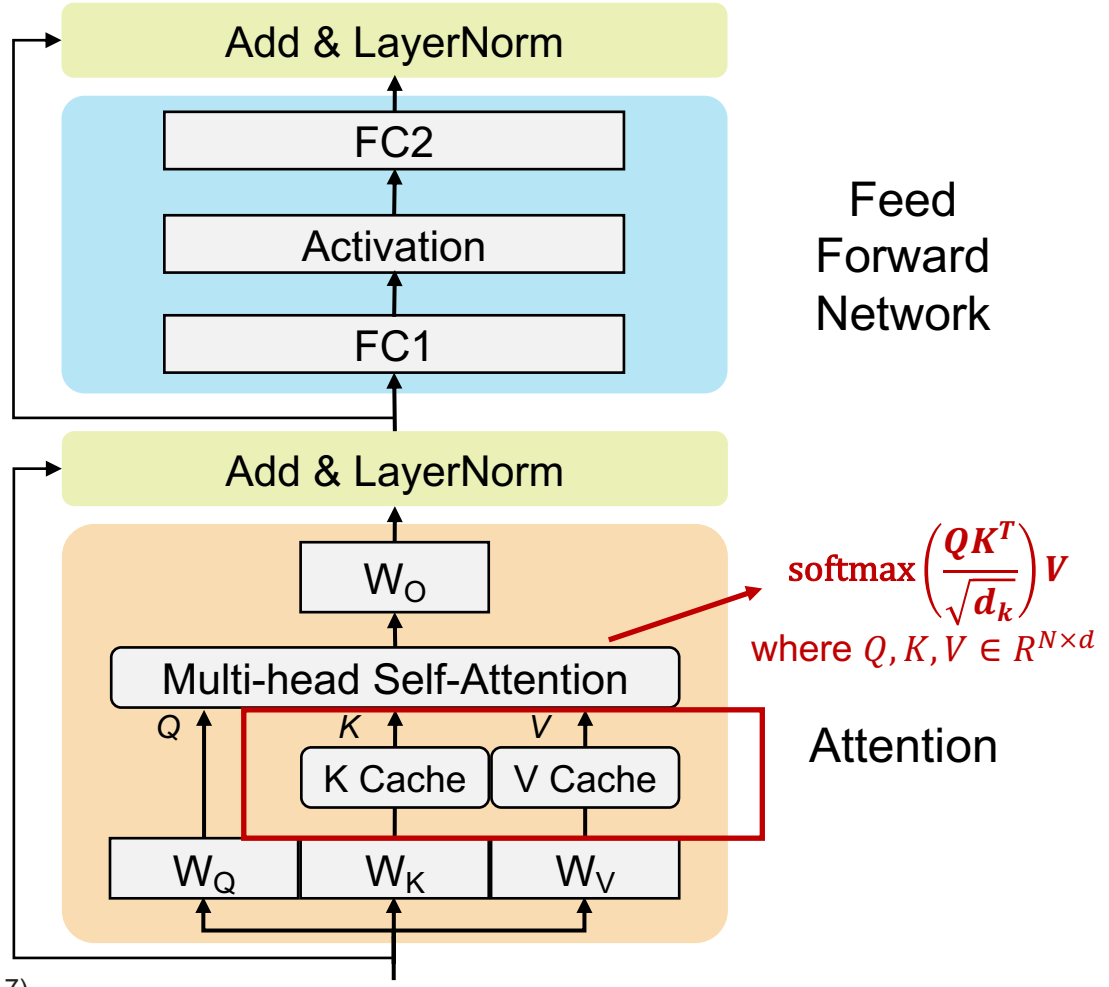
where $Q, K, V \in R^{N \times d}$

Attention

[1] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

# Large Language Model (LLM)

- Most large language models are based on the Transformer architecture[1].
- A Transformer block consists of :
  - Attention-Linear (generate matrix Q, K, V)
  - Multi-Head Attention
  - Feed Forward Network
  - Layer Norm
- A typical LLM inference consists of two stages:
  - Prefill Stage: takes a prompt sequence to generate the key-value cache
  - Decode Stage: utilizes and updates the KV cache to generate tokens one by one, where the current token generation depends on all the previously generated tokens




[1] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

# Root causes of LLM's slow inference or training

1. Model scale: A large number of weights and computations
2. Architecture: Attention operation has quadratic complexity (computation & memory & memory access) w.r.t. input token length
3. Decoding approach in inference: Generate tokens one by one (fully sequential)

**Let's do some rough estimation to get some sense**

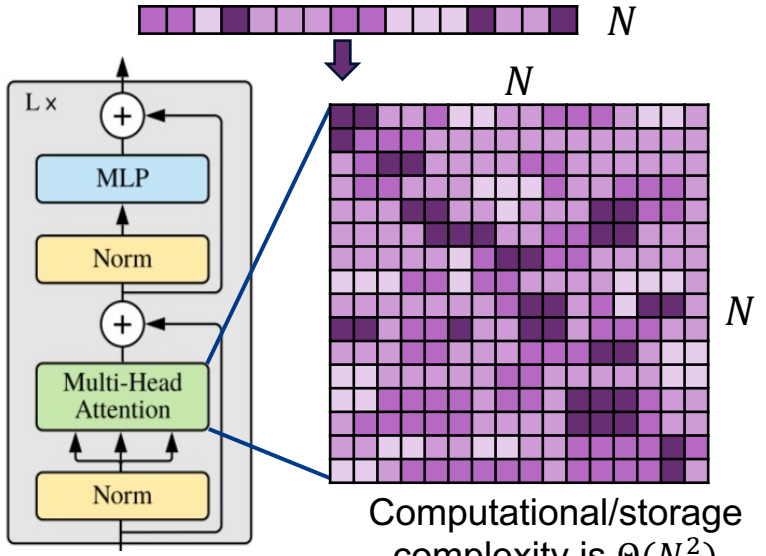
GPT-3

**LLM Model:**
**GPT-3 (175B)**
Computation: 663 TFLOPs
Storage: 350 GB (FP16)

**RTX 3090 GPU**
Mem. Capacity: 24 GB
Computing Capacity:
35.58 TFLOPS

- Memory requirements: solely storing all the parameters on GPU HBM requires **15 * 3090 GPUs**
- First-token latency: with 2048 input tokens, it needs at least **> 19 GPU** seconds to generate the first output token. (estimated using the ***peak performance*** of 3090 GPU)

**Attention overhead increases quadratically with input token length**




Computational/storage complexity is $\Theta(N^2)$

# Efforts towards more efficient LLM inference

Lower Latency | Lower Storage | Higher Throughput | Lower Power Consumption

| What algorithm property? | Cause what? | Solutions |
|---|---|---|
| Model Scale | • Large computation<br>• Large memory access<br>• Large memory footprint | Structure Design |
| Attention Operation | • Input-quadratic computation<br>• Input-quadratic memory access<br>• Input-quadratic memory footprint | Model Compression |
| Decoding Approach | • Low arithmetic intensity (i.e., computation / memory access) cause under-utilization<br>• Varying length -> Dynamically increasing KV cache cause fragmented memory, increasing both footprint and access | Inference Engine<br>Serving Framework |

# Efforts towards more efficient LLM inference

Lower Latency | Lower Storage | Higher Throughput | Lower Power Consumption

| What algorithm property? | Cause what? | Solutions |
|---|---|---|
| Model Scale | • Large computation<br>• Large memory access<br>• Large memory footprint | • Dynamic MoE<br>• Low-complexity attention<br>• Multi-query attention<br>• … |
| Attention Operation | • Input-quadratic computation<br>• Input-quadratic memory access<br>• Input-quadratic memory footprint | |
| Decoding Approach | • Low arithmetic intensity (i.e., computation / memory access) cause under-utilization<br>• Varying length -> Dynamically increasing KV cache cause fragmented memory, increasing both footprint and access | |

Structure Design

MoE
- Gshard (ICLR'21) [1]
- SwitchTransformers (JMLR'22) [2]

Low-complexity Attention
- Linformer (arXiv'20) [3]
- Performer (arXiv'20) [4]

Multi-query Attention
- Shazeer (arXiv'19) [5]


[1] Lepikhin, Dmitry, et al. "Gshard: Scaling giant models with conditional computation and automatic sharding." *ICLR* (2020).
[2] Fedus, William, et al. "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity." *JMLR* (2022).
[3] Wang, Sinong, et al. "Linformer: Self-attention with linear complexity." arXiv (2020).
[4] Choromanski, Krzysztof et al. "Rethinking Attention with Performers." arXiv (2020).
[5] Shazeer et al. "Fast Transformer Decoding: One Write-Head is All You Need." arXiv (2019)

# Efforts towards more efficient LLM inference

Lower Latency | Lower Storage | Higher Throughput | Lower Power Consumption

| What algorithm property? | Cause what? | Solutions |
|---|---|---|
| Model Scale | • Large computation<br>• Large memory access<br>• Large memory footprint | • Dynamic MoE<br>• Low-complexity attention<br>• Multi-query attention<br>• … → Structure Design |
| Attention Operation | • Input-quadratic computation<br>• Input-quadratic memory access<br>• Input-quadratic memory footprint | • Quantization<br>• Weight sparsification<br>• Attention sparsification<br>• Other: structure factorization… → Model Compression |
| Decoding Approach | • Low arithmetic intensity (i.e., computation / memory access) cause under-utilization<br>• Varying length -> Dynamically increasing KV cache cause fragmented memory, increasing both footprint and access | |

Quantization
- Smoothquant (ICML'23) [1]
- AWQ (arXiv'23) [2]

Weight / Attention Sparsification
- DynaBERT (NIPS'20) [3]
- Big Bird (NIPS'20) [4]


[1] Xiao, Guangxuan, et al. "Smoothquant: Accurate and efficient post-training quantization for large language models." . *ICML* (2023)
[2] Lin, Ji, et al. "AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration." *arXiv* (2023).
[3] Hou, Lu et al. "DynaBERT: Dynamic BERT with Adaptive Width and Depth." *NIPS* (2020)
[4] Zaheer, Manzil, et al. "Big bird: Transformers for longer sequences." Advances in neural information processing systems 33 (2020)

# Efforts towards more efficient LLM inference

Lower Latency | Lower Storage | Higher Throughput | Lower Power Consumption

| What algorithm property? | Cause what? |
|---|---|
| Model Scale | • Large computation<br>• Large memory access<br>• Large memory footprint |
| Attention Operation | • Input-quadratic computation<br>• Input-quadratic memory access<br>• Input-quadratic memory footprint |
| Decoding Approach | • Low arithmetic intensity (i.e., computation / memory access) cause under-utilization<br>• Varying length -> Dynamically increasing KV cache cause fragmented memory, increasing both footprint and access |

## Solutions

**Structure Design**
- Dynamic MoE
- Low-complexity attention
- Multi-query attention
- …

**Model Compression**
- Quantization
- Weight sparsification
- Attention sparsification
- Other: structure factorization…

**Inference Engine**
- Graph and Operator Optimization / Implementation
- Speculative decoding
- Memory Management

Graph/Operator Optimization
- Flash Attention (NIPS'22) [1]
- FasterTransformer [2]

Memory Management
- vLLM [3], lightLLM [4]

Speculative decoding
- [5]


[1] Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness." *Advances in Neural Information Processing Systems* 35 (2022)
[2] NVIDIA, FasterTransformer, 2019
[3] UCB, https://github.com/vllm-project/vllm, 2023
[4] Sensetime, https://github.com/ModelTC/lightllm, 2023
[5] Stern et al., "Blockwise parallel decoding for deep autoregressive models ", NeruIPS'18.

# Efforts towards more efficient LLM inference

Lower Latency | Lower Storage | Higher Throughput | Lower Power Consumption

| What algorithm property? | Cause what? | Solutions |
|---|---|---|
| Model Scale | • Large computation<br>• Large memory access<br>• Large memory footprint | • Dynamic MoE<br>• Low-complexity attention<br>• Multi-query attention<br>• … → Structure Design |
| Attention Operation | • Input-quadratic computation<br>• Input-quadratic memory access<br>• Input-quadratic memory footprint | Split/Assignment/Schedule<br>• FlexGen [1]<br>• Alpa [2]<br>• DeepSpeed [3]<br>→ Model Compression |
| Decoding Approach | • Low arithmetic intensity (i.e., computation / memory access) cause under-utilization<br>• Varying length -> Dynamically increasing KV cache cause fragmented memory, increasing both footprint and access | Multi-query batching/caching<br>• vLLM [4], lightLLM [5]<br>• NVIDIA Triton [6]<br>→ Inference Engine |
| | | Mainly **throughput-oriented**<br>• Split & Assignment & Schedule<br>• Multi-query/task Batching & Caching<br>→ Serving Framework |


[1] Sheng, Ying, et al. "High-throughput generative inference of large language models with a single gpu." *arXiv* (2023).
[2] Zheng, Lianmin, et al. "Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning." *OSDI* (2022).
[3] Li, Conglong, et al. "DeepSpeed Data Efficiency: Improving Deep Learning Model Quality and Training Efficiency via Efficient Data Sampling and Routing." *arXiv* (2022).
[4] UCB, https://github.com/vllm-project/vllm, 2023
[5] Sensetime, https://github.com/ModelTC/lightllm, 2023
[6] NVIDIA, Triton inference server, 2021.

# Efforts towards more efficient LLM inference

Lower Latency | Lower Storage | Higher Throughput | Lower Power Consumption

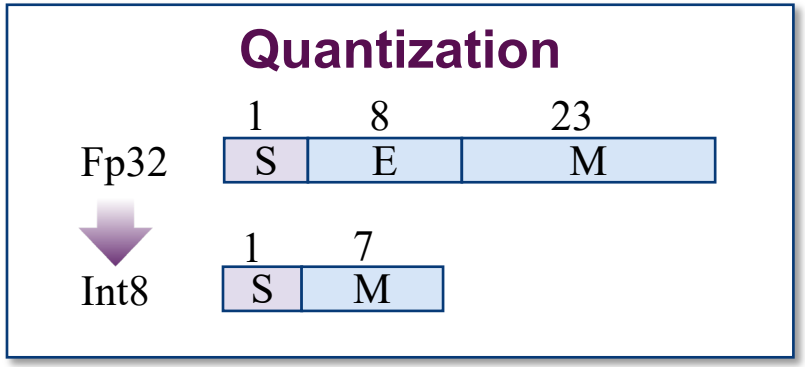
| What algorithm property? | Cause what? | Solutions | |
|---|---|---|---|
| Model Scale | • Large computation<br>• Large memory access<br>• Large memory footprint | • Dynamic MoE<br>• Low-complexity attention<br>• Multi-query attention<br>• … | Structure Design |
| Attention Operation | • Input-quadratic computation<br>• Input-quadratic memory access<br>• Input-quadratic memory footprint | • Quantization<br>• Weight sparsification<br>• Attention sparsification<br>• Other: structure factorization… | Model Compression |
| Decoding Approach | • Low arithmetic intensity (i.e., computation / memory access) cause under-utilization<br>• Varying length -> Dynamically increasing KV cache cause fragmented memory, increasing both footprint and access | • Graph and Operator Optimization / Implementation<br>• Speculative decoding<br>• Memory Management<br>Mainly **throughput-oriented**<br>• Split & Assignment & Schedule<br>• Multi-query/task Batching & Caching | Inference Engine<br>Serving Framework |

**Lossy model-level techniques**

**Lossless system-level techniques**

# Let’s zoom into quantization

- This tutorial focuses on a specific and important type of lossy model-level technique -- Quantization



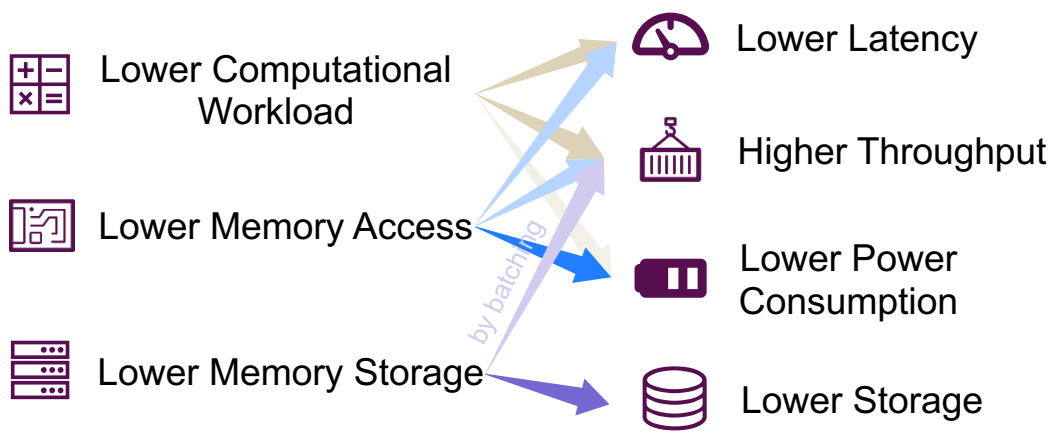
# Menu

1. Overview
2. **Quantization**
3. Summary of Researches & Demos

# Why Quantization & What is Quantization

- We want **lower latency**, **higher throughput**, and other goods (e.g., **lower power consumption**, **fewer GPUs**, etc.).

## How Quantization Benefits Model Deployment




- e.g., quantization speeds up the model inference*

| | Batch size 1 | | | Batch size 8 | | | Batch size 128 | | |
|---|---|---|---|---|---|---|---|---|---|
| | FP32 | FP16 | Int8 | FP32 | FP16 | Int8 | FP32 | FP16 | Int8 |
| MobileNet v1 | 1 | 1.91 | 2.49 | 1 | 3.03 | 5.50 | 1 | 3.03 | 6.21 |
| MobileNet v2 | 1 | 1.50 | 1.90 | 1 | 2.34 | 3.98 | 1 | 2.33 | 4.58 |
| ResNet50 (v1.5) | 1 | 2.07 | 3.52 | 1 | 4.09 | 7.25 | 1 | 4.27 | 7.95 |
| VGG-16 | 1 | 2.63 | 2.71 | 1 | 4.14 | 6.44 | 1 | 3.88 | 8.00 |
| VGG-19 | 1 | 2.88 | 3.09 | 1 | 4.25 | 6.95 | 1 | 4.01 | 8.30 |
| Inception v3 | 1 | 2.38 | 3.95 | 1 | 3.76 | 6.36 | 1 | 3.91 | 6.65 |
| Inception v4 | 1 | 2.99 | 4.42 | 1 | 4.44 | 7.05 | 1 | 4.59 | 7.20 |
| ResNext101 | 1 | 2.49 | 3.55 | 1 | 3.58 | 6.26 | 1 | 3.85 | 7.39 |

* Relative speed-up w.r.t. FP32, tested on Tesla T4 GPU, input size 224x224

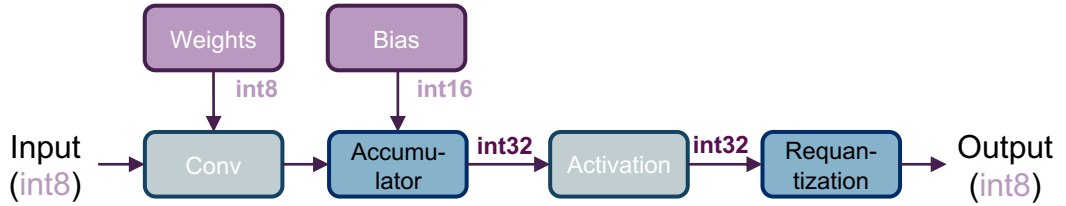
## Two Procedures

### (Offline Stage) Model Transformation

- Convert the **FP weights** into **low-bit-width weights**
- Determine the quantization parameters for activations (*optional, if using low-precision activation and not using online quant*)


### (Online Stage) Quantized Inference

- **Low-Precision computation**: LP arithmetic -> Requantization

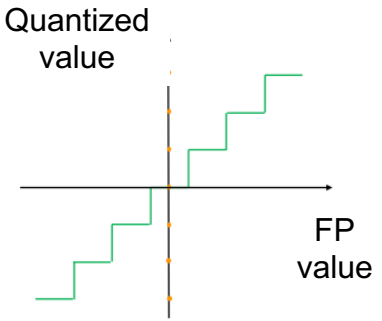



- **High-Precision computation**: Dequant -> HP arithmetic -> (optional) Quant

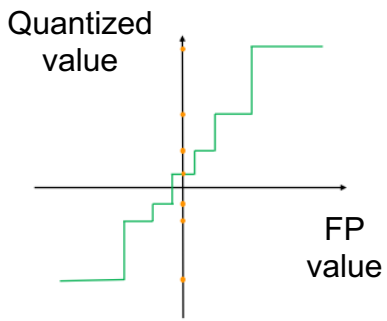
# Formula & Basic Concepts of Quantization

- **Quantization Formats**
  - An 8-bit machine number has 256 value choices, corresponding to 256 **quantization levels**.
  - What actual value (i.e., quantization level) does each choice represent? -> Depends on quantization format and parameters.

Whether the **quantization levels** are evenly spaced

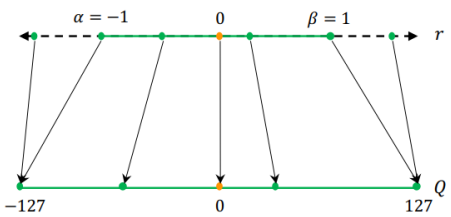



Uniform Quantization

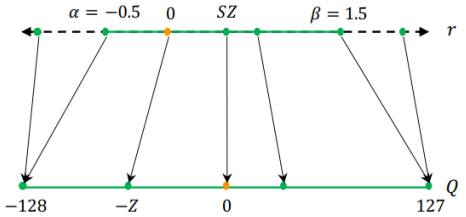



Non-uniform Quantization

Whether the **representation range** is symmetric with respect to the origin




Symmetric Quantization




Asymmetric Quantization

- The formula for converting FP weights to low-bit-width representation
  - Uniform Quantization can be formulated as

$$x_{\text{int}} = \text{clip}\left(\left[\frac{x}{s_x}\right] + z; q_{\min}, q_{\max}\right)$$

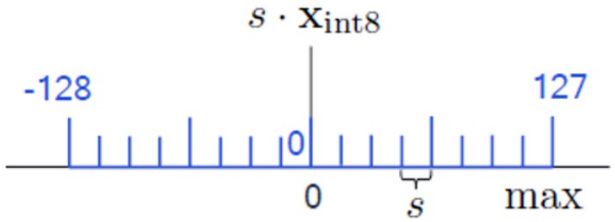
[1] Gholami, Amir, et al. "A survey of quantization methods for efficient neural network inference." arXiv preprint arXiv:2103.13630 (2021).

# Formula & Basic Concepts of Quantization

- **Quantization Parameters**
  - Taking **signed uniform** quantization as an example, quantization parameters include

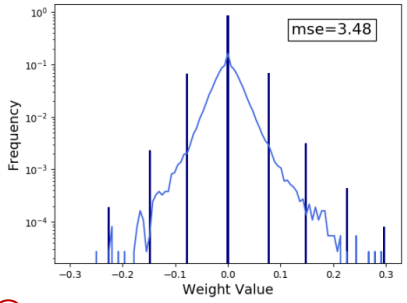
**Scaling Factor** (usually FP), **Zero Point**, **Bitwidth**

$$x_{\text{int}} = \text{clip}\left(\left\lfloor \frac{x}{s_x} \right\rfloor + z; q_{\min}, q_{\max}\right), \quad \text{where } q_{\max} = 2^{b-1} - 1, \; q_{\min} = -2^{b-1}$$

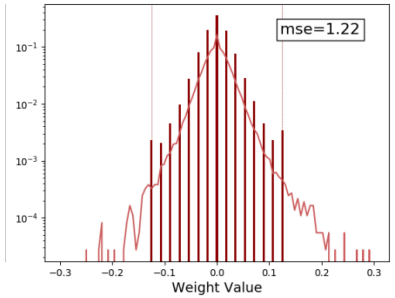



  - For actual storage decrease and speed-up, a group of values needs to share the same quantization parameters (the group size is called **quantization granularity**, e.g., tensor-wisely, channel-wisely).

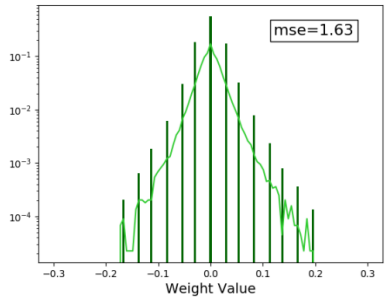
- A key question of the **offline quantization** is to decide the parameters to properly **balance representation range and precision** (i.e., balance clipping and rounding error).




☹ **Large rounding error**
(large scale factor)




☹ **Large clipping error**
(small scale factor)

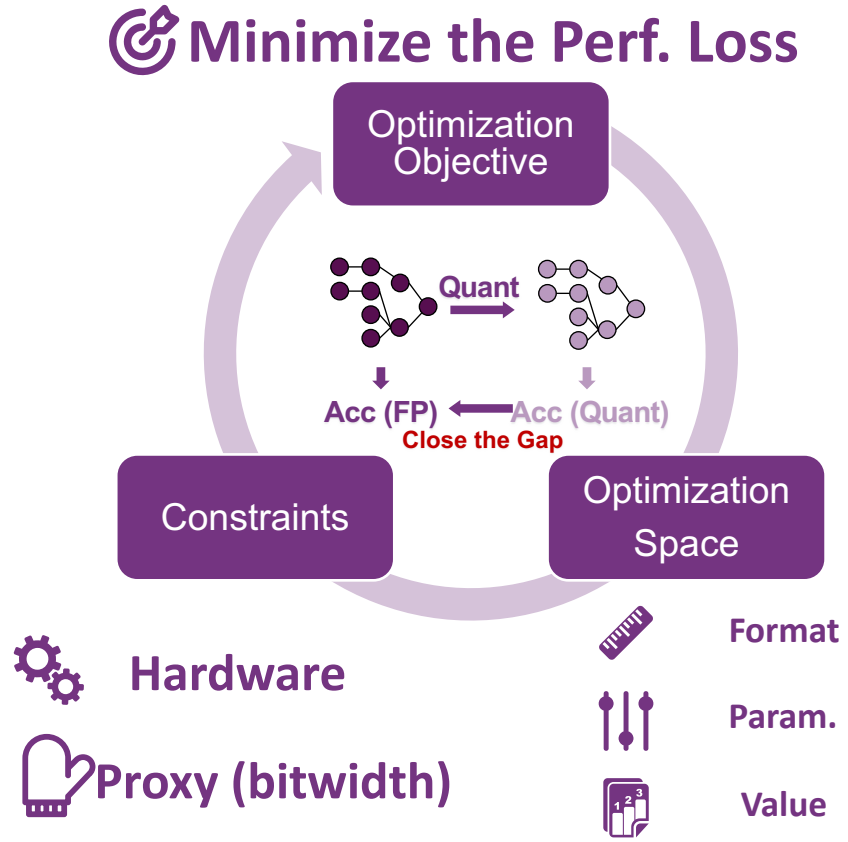



☺ **Well-balanced**
(appropriate scale factor)

[1] Zhao, Ritchie, et al. "Improving neural network quantization without retraining using outlier channel splitting." International conference on machine learning. PMLR, 2019.

# The Quantization Optimization Problem

- Quantization is a lossy transform of neural networks.
  Should **trade-off between efficiency & task performance**.
- The optimization problem of offline quantization:
  - Objective:
    - Minimize the performance loss
  - Constraints:
    - Hardware objectives / Proxy (e.g., bitwidth)
  - Optimization Space:
    - Quantization Formats
    - Quantization Parameters
    - Quantization Values

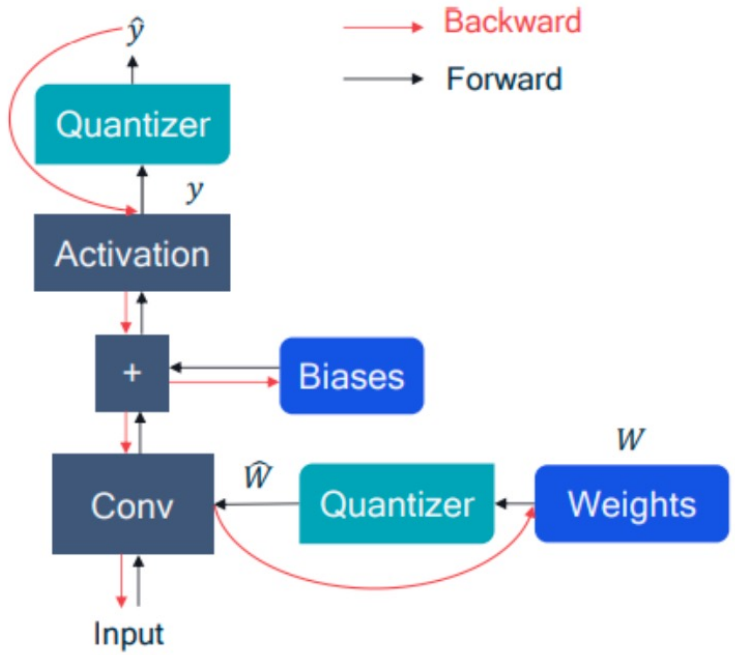


# Quantization Workflow

- Two quantization workflows suitable for different scenarios
  - Post-Training Quantization (PTQ)
    - Determine the quantization format, parameters, and values for a pre-trained model. Do not need the original training pipeline or full dataset.
    - **Need fewer data and time**, **Less accurate** Suitable for scenarios where training is prohibitive or data is scarce.
  - Quantization-Aware Training (QAT)
    - Let the model weights adapt to quantization error through training. Training and dataset needed.
    - **Need more data and time**, **More accurate** Suitable for scenarios where the training budget is enough and data is available / can be synthesized.

**QAT**

1. Before training: Insert fake quantizer
2. During training: Use Straight-Through Estimator (STE) to calculate gradient



# Quantization Techniques

- Common techniques categorized by optimization subspace
  - Quantization **Formats**[1,2,3,4]
    - Non-uniform quantization (e.g., bf16, fp8, logarithm, other…)
  - Quantization Parameters
    - **Bit-width**: mixed-precision (e.g., HAWQ, HAQ) [5,6]
    - **Scaling factor & Zero point**: learnable scale / zero-points (e.g., LSQ, LSQ++, ReactNet)[7,8]
  - Quantization **Value**
    - **Value transformation**: reparameterization, smart rounding (e.g., Stochastic, AdaRound)[9,10]
    - **Training techniques**: advanced gradient estimators beyond STE in QAT[11]


[1] Kalamkar, D., et al. "A study of bfloat16 for deep learning training." arXiv preprint arXiv:2307.15337 (2023).

[2] Li, Y., et al. "Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. " ICLR20.

[3] Zhong, K. et al. "Exploring the Potential of Low-bit Training of Convolutional Neural Networks." TCAD'22.

[4] Zhang, D. et al. "Lqnets:Learnedquantizationforhighlyaccurate and compact deep neural networks." ECCV'18.

[5] Dong, Z., et al. "Hawq: Hessian aware quantization of neural networks with mixedprecision." ICCV'19.

[6] Wang, K. et al. "Haq: Hardwareaware automated quantization with mixed precision." CVPR'19.

[7] Liu, Z. et al. "Reactnet: Towards precise binary neural network with generalized activation functions." ECCV'20.

[8] Esser, S. et al. "Learned step size quantization." ICLR'20.

[9] Gupta, S. et al., "Deep Learning with Limited Numerical Precision" ICML'15.

[10] Nagel, M. et al. "Up or down? adaptive rounding for post-training quantization" ICML'20.

[11] Liu, Z. et al. "Bireal net: Enhancing the per formance of 1bit cnns with improved representational capability and advanced training algorithm." ECCV'18.

# Special Properties to Consider for Quantizing LLMs

- Special considerations for LLM quantization

**(Efficiency)** Weight loading is the efficiency bottleneck of the LLM generation (data from [1])

- The decoding phase accounts for the major e2e latency.

| Model | Prefill/Decode Latency (ms) | Prefill/Decode GPU Perf. (TFLOPS) |
|---|---|---|
| LLaMA-7B | 40 / 2735 | 43 / 0.31 |
| LLaMA-13B | 54 / 3725 | 62 / 0.44 |
| LLaMA-33B | 100 / 5506 | 85 / 0.15 |

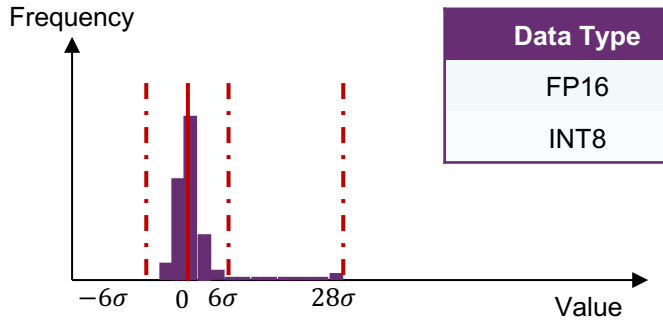
Table 1: The latency and average GPU performance of the prefilling and decoding phases when inferencing LLMs. The prefilling token length is 128, the decoding token length is 64, and the batch size is 1. The test is run on one NVIDIA A100 GPU.

**0.1%** utilization w.r.t. the peak FP16 GPU perf.

- Memory-bounded: During the decoding phase, the GPU is under-utilized.
- Weight-loading-bounded: The amount of weight loading is orders of magnitudes larger than activation loading, especially when the decoding length and batch size are not large.

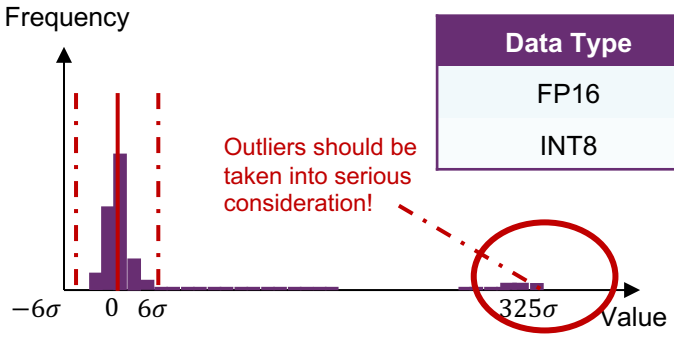
**(Algorithm)** LLM weights have a wider dynamic range

When keeping the dynamic range, the rounding error is not so large that the quantized ResNet18 performs well, while the rounding error is too large for the LLM. (data from [2])




| Data Type | Algo Perf. |
|---|---|
| FP16 | 69.7% |
| INT8 | 69.7% |

ResNet18
- Max Value: $28\sigma$
- Proportion of $6\sigma$: 0.05%




| Data Type | Algo Perf. |
|---|---|
| FP16 | 68.6% |
| INT8 | 0.0% |

OPT-13B
- Max Value: $325\sigma$
- Proportion of $6\sigma$: 0.2%


[1] Ning, X., et al. "Skeleton-of-Thought: Large Language Models Can Do Parallel Decoding." arXiv preprint arXiv:2307.15337 (2023).

[2] Guo, Cong, et al. "OliVe: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization." ISCA (2023).

# LLM Quantization Techniques and Results

- LLM quantization related work

| Name | Format | Parameter | Value | Online Quantized Inference Acceleration | Performance |
|---|---|---|---|---|---|
| LLM.int8() (NIPS'22) [1] | Uniform | Range-preserving; Mixed-precision | Nearest rounding | No | 8/16-bit, No speedup |
| GPTQ (ICLR'23) [2] | Uniform | Range-preserving | Smart rounding | Fuse dequant & gemm | W4A16, 2× speedup |
| LUT-GEMM (Arxiv'22) [3] | Non-uniform | Range-preserving | Nearest rounding | LUT-based dequant & kernel fusion | W3A16, 2.1x speedup |
| SmoothQuant (ICML'23) [4] | Uniform | Range-preserving | Reparameterization Nearest rounding | INT8 Tensor Core | W8A8, 1.56× speedup |
| Olive (ISCA'23) [5] | Non-uniform | Scale search (reconstruction err) | Nearest rounding | New Tensor Core HW | W4A4, 4.5× speedup |
| AWQ (Arxiv'23) [6] | Uniform | Scale search (reconstruction err) | Reparameterization Nearest rounding | Fuse dequant & gemm | W4A16, 1.85× speedup |
| NICSEFC (Ours) | Uniform | Mixed-precision; Smart grouping | Reparameterization Nearest rounding | Fuse dequant & gemm | **W3A16, ? speedup** |


[1] Dettmers, Tim, et al. "Llm. int8 (): 8-bit matrix multiplication for transformers at scale." *NIPS* (2022)
[2] Frantar, Elias, et al. "Gptq: Accurate post-training quantization for generative pre-trained transformers." *ICLR* (2023)
[3] Gunho, Park, et al. nuqmm: Quantized matmul for efficient inference of large-scale generative language models. arXiv, (2022)
[4] Xiao, Guangxuan, et al. "Smoothquant: Accurate and efficient post-training quantization for large language models." . ICML (2023)
[5] Guo, Cong, et al. "OliVe: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization.". ISCA (2023).
[6] Lin, Ji, et al. "AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration." *arXiv* (2023).

# Some Questions

- What ability loss does the quantization of different modules and tensors bring?
  - A recent attempt to empirically quantify the quantization effect: [1]

- For acceleration, how should we further quantize the computation of LLM
  - Application scenarios: Prefill, large batch-size inference, or training / tuning
  - Hardware: those that support efficient low-bit computation, or specialized hardware design

- For peak memory optimization, to what extreme can we quantize the weights and KV cache?
  - Applicable scenarios: Long-sequence / large batch-size inference
  - E.g., Dynamic quantization of KV cache?

[1] Liu, Peiyu, et al. "Do Emergent Abilities Exist in Quantized Large Language Models: An Empirical Study." *arXiv preprint arXiv:2307.08072* (2023).

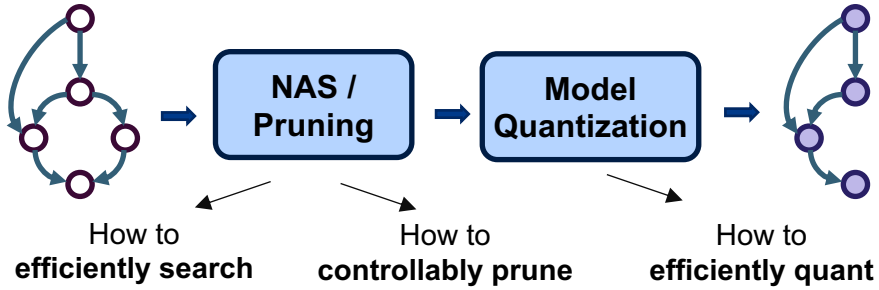
# Menu

1. Overview
2. Quantization
3. **Summary of Researches & Demos**

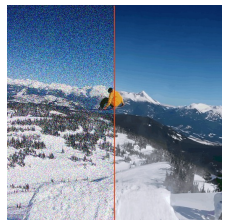
# Some of Our Past Work

- Structure Search
  - **[ECCV20]** A Generic Graph-based Neural Architecture Encoding Scheme for Predictor-based NAS
  - **[NeurIPS21]** Evaluating Efficient Performance Estimators of Neural Architectures
  - **[ICML23]** OMS-DPM: Deciding The Optimal Model Schedule for Diffusion Probabilistic Model
- Pruning
  - **[FPGA17]** ESE: Efficient Speech Recognition Engine with Compressed LSTM on FPGA
  - **[ECCV20]** DSA: More Efficient Budgeted Pruning via Differentiable Sparsity Allocation
  - **[AAAI23]** Memory-Oriented Structural Pruning for Efficient Image Restoration
- Quantization
  - **[FPGA16]** Going Deeper with Embedded FPGA Platform for Convolutional Neural Network
  - **[TCAD22]** Exploring the Potential of Low-bit *Training* of Convolutional Neural Networks
- Dynamic inference
  - **[ICCV23]** Ada3D : Exploiting the Spatial Redundancy with Adaptive Inference for Efficient 3D Object Detection
- Sparse Operator Design
  - **[DAC22]** Heuristic Adaptability to Input Dynamics for SpMM on GPUs
  - **[MLSys23]** Exploiting Hardware Utilization and Adaptive Dataflow for Sparse Convolution in 3D Point Clouds

## Research Direction: Model Compression

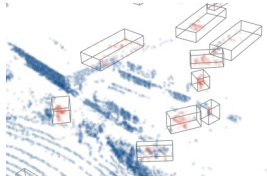



## Apply to practical scenarios



**Low-level Vision**
**2x** peak mem. reduce

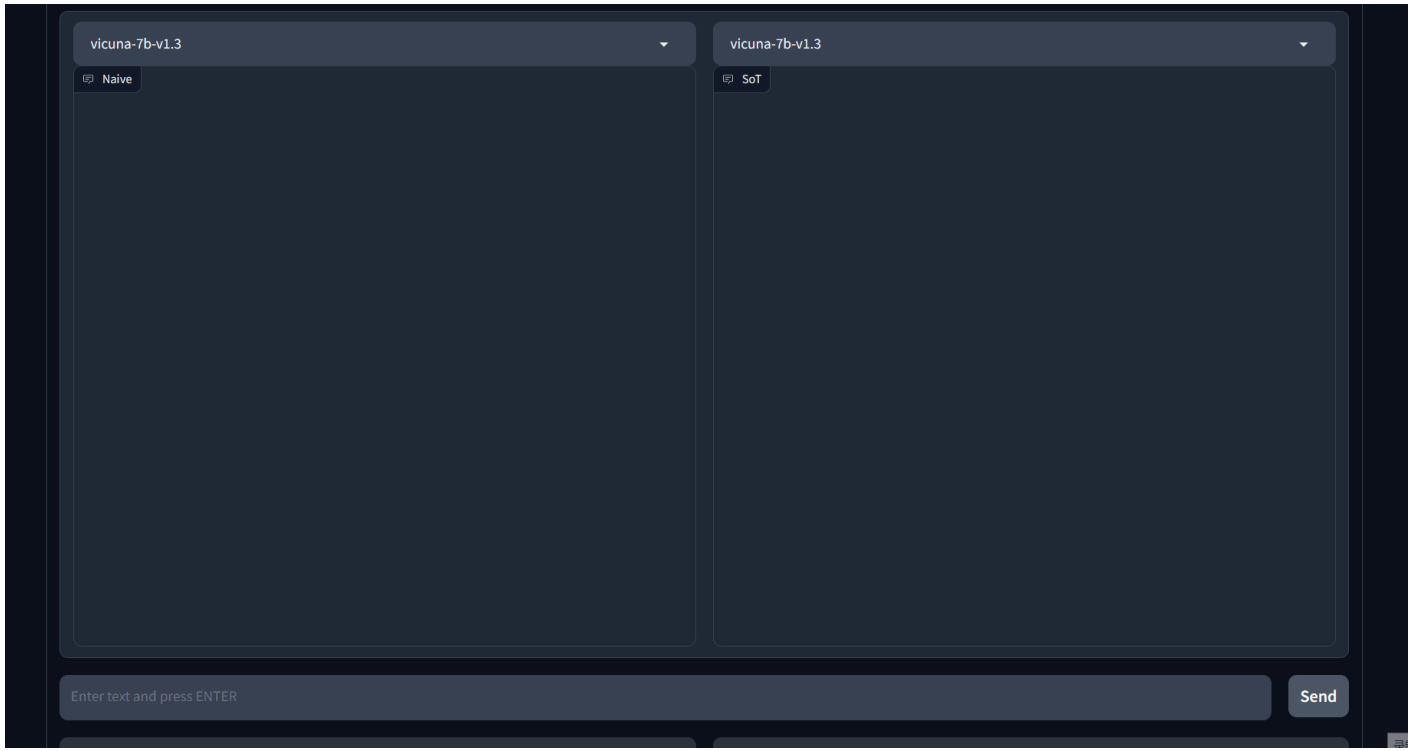
**Image Generation**
**2~5x** speedup



**3D Perception**
**1.38x** speedup & **2.21x** peak mem. reduce

# Some Past Demo on Efficient AIGC – Efficient LLM

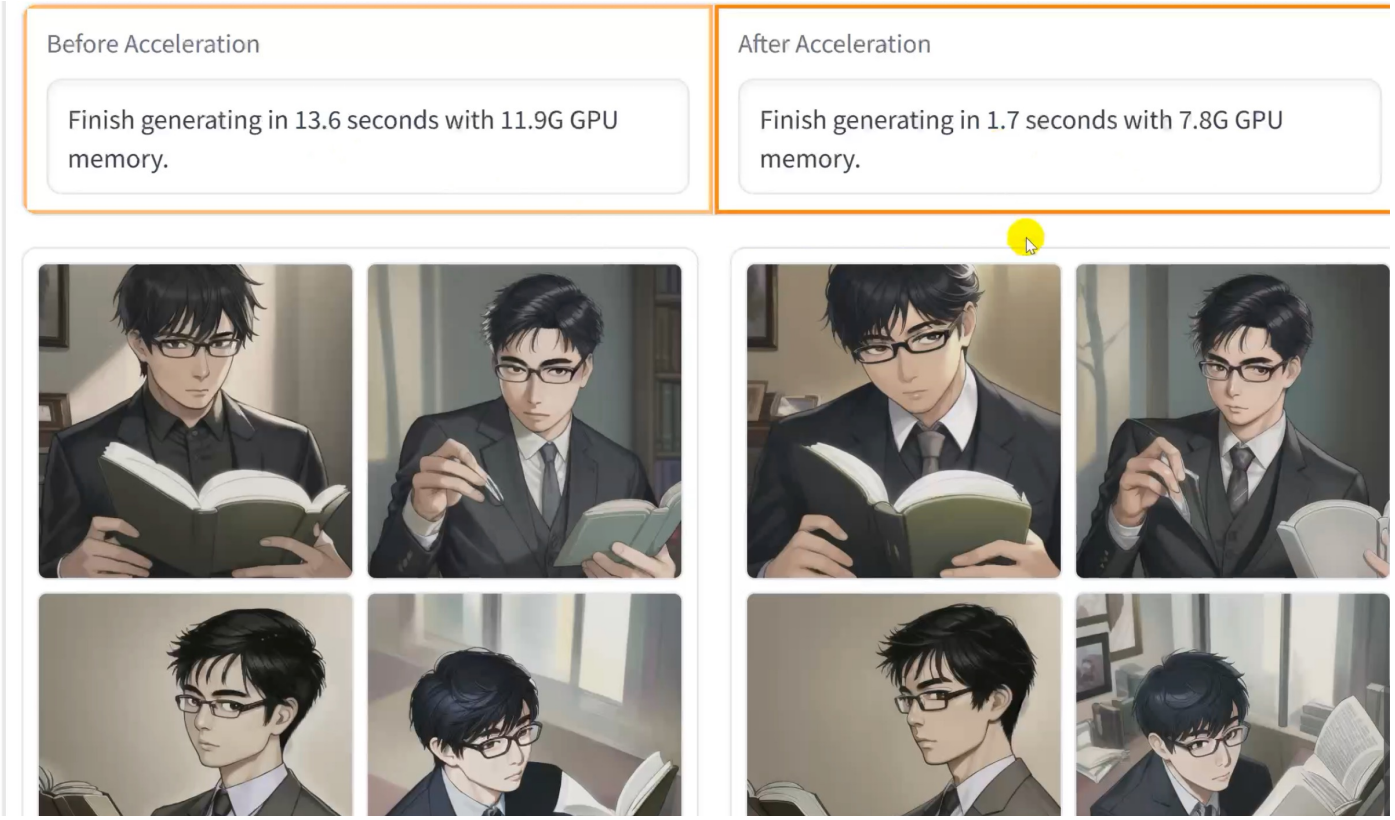
- **Data-level optimization**: **New possibilities for efficiency improvements in the LLM era!**
  - Different from model-level and system-level techniques, we demonstrate the potential of "data-level content organization" for decreasing the end-to-end latency
  - Vicuna-7B on A100, **2.1x** end-to-end acceleration




**Project website: https://sites.google.com/view/sot-llm**

# Some Past Demo on Efficient AIGC – Efficient Diffusion

- **Schedule/Model-level optimization**
  - Leverage off-the-shelf models with different sizes or weights at different time steps
  - LoRA finetuned (based on SD 1.5) on A100, **2x** end-to-end acceleration




This demonstrated speedup also includes TensorRT's operator fusion and selection

**Project website:** https://sites.google.com/view/oms-dpm/ In ICML'23.

# Thanks!

## Researchers and Engineers Wanted!

- No matter whether you're an application dreamer, a DL practitioner, a software geek, or a hardware fever.
- No matter whether you're interested in how humans and AI "think" or how to enable AI to create perceptual assets.

If you're interested in making AIGC more efficient, welcome to join us on efficient AIGC projects!

- For **research at NICS-EFC lab@Tsinghua**, email Prof. Yu Wang and Dr. Xuefei Ning for discussion.

foxdoraame@gmail.com Dr. Xuefei Ning

yu-wang@tsinghua.edu.cn Prof. Yu Wang