# SmoothQuant

MIT, NVIDIA 2023

# Introduction

## Serving Large Language Models Is Expensive

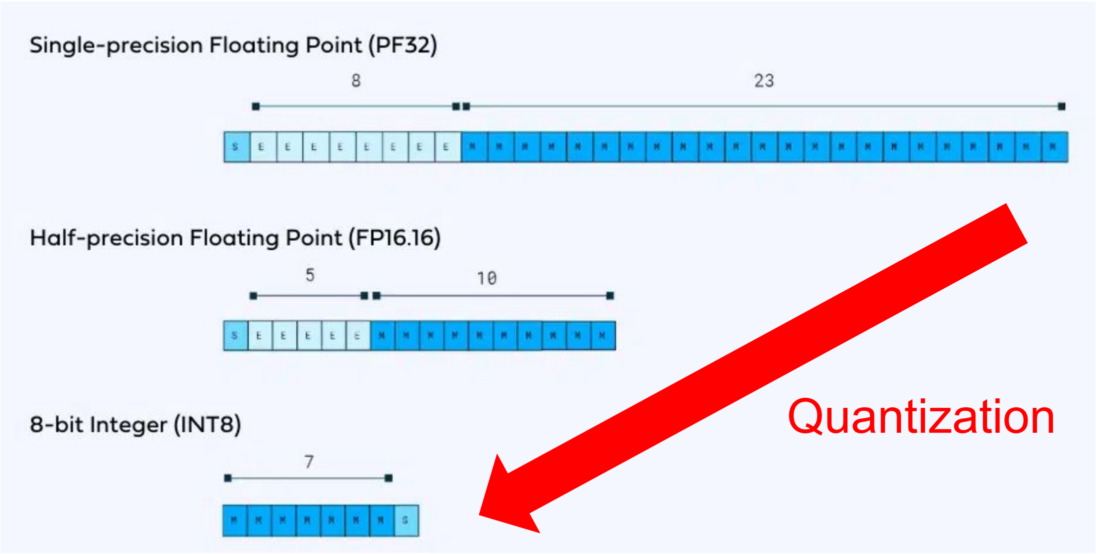
- Pre-trained language models have achieved remarkable performance

Amazon Q ChatGPT GitHub Copilot

- Large language models often come in high precision formats such as FP16
    - Significant GPU memory requirements (size and bandwidth)
    - Slow matrix multiplication operations

# Post-Training Quantization Reduces The Cost

- Post-training quantization (PTQ) reduces the cost of LLMs.
  - “Post-training:” no modifications to training => easy to implement and wide applicability
  - “Quantization:” lowers the bit-width and improves efficiency
  - Mitigate memory consumption and reduce computational overhead => higher performance



# Existing methods cannot maintain accuracy and hardware efficiency at the same time

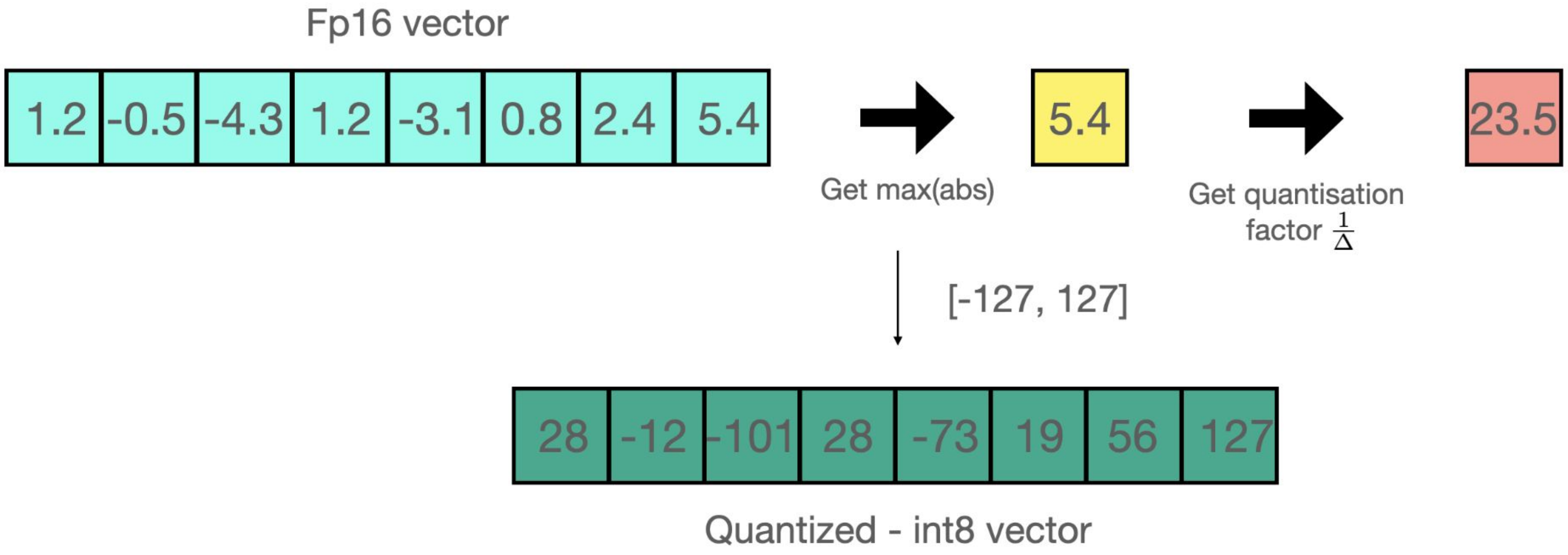
- ZeroQuant
  - Uses layer-by-layer knowledge distillation without the original training data
  - Delivers good accuracy for GPT-3-350M and GPT-J-6B
  - Can not maintain the accuracy for the large OPT model with 175 billion parameters
- LLM.int8()
  - Increases accuracy by keeping outliers in FP16 and uses INT8 for the other activations
  - The mixed-precision decomposition is not hardware friendly

# Paper Key Takeaways

- Per-channel quantization is infeasible
  - GEMM kernels rely on a sequence of operations at a high throughput
  - But operations that apply different scales for each channel have a lower throughput
- Preprocessing the weights and activations is the solution
  - Activations are hard to quantize and weights are easy to quantize
  - Exploit the linearity of matrix multiplication to offload the quantization difficulty
  - Uniform quantization is supported by hardware
- SmoothQuant enables an INT8 quantization of both weights and activations for all the matrix multiplications in LLMs
  - Preserves accuracy
  - Hardware-friendly
  - Up to 1.56x speedup and 2x memory reduction

# The Quantization Process

$$\bar{\mathbf{X}}^{\mathrm{INT8}} = \lceil \frac{\mathbf{X}^{\mathrm{FP16}}}{\Delta} \rfloor, \quad \Delta = \frac{\max(|\mathbf{X}|)}{2^{N-1}-1}$$



# The Quantization Process

$$\bar{\mathbf{X}}^{\mathrm{INT8}} = \lceil \frac{\mathbf{X}^{\mathrm{FP16}}}{\Delta} \rfloor, \quad \Delta = \frac{\max(|\mathbf{X}|)}{2^{N-1} - 1}$$

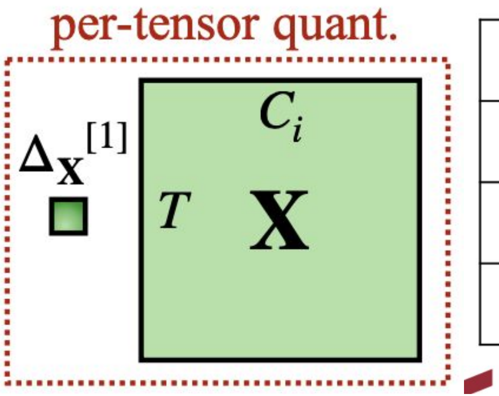
How to get $\Delta$?

- Dynamic Range Quantization
  - At runtime
  - Use the runtime statistics of activations to get $\Delta$
- Static Quantization
  - Before runtime
  - Calculate $\Delta$ offline with the activations of some calibration samples
  - This paper gets activation statistics from 512 random sentences from the pre-training dataset Pile

# Quantization Granularity

- Per-Tensor Quantization
  - Uses a single step size for the entire matrix
- Per-Token Quantization
- Per-Channel Quantization
- Group-Wise Quantization

$$S = \frac{|r|_{\max}}{q_{\max}} = \frac{2.12}{2^{2-1} - 1} = 2.12$$




| | | | |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 0 | 0 | -1 | 1 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |

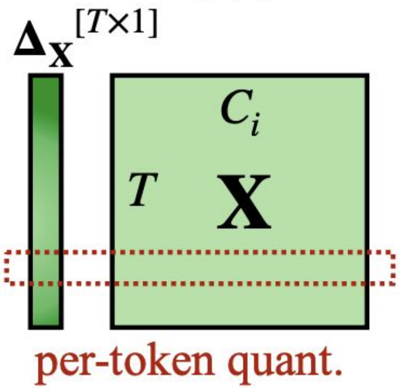
**Quantized**

| | | | |
|---|---|---|---|
| 2.12 | 0 | 2.12 | 0 |
| 0 | 0 | -2.12 | 2.12 |
| 0 | 2.12 | 0 | 0 |
| 2.12 | 0 | 2.12 | 2.12 |

**Reconstructed**

# Quantization Granularity

- Per-Tensor Quantization
- Per-Token Quantization
  - Uses different quantization step sizes for activations associated with each token
- Per-Channel Quantization
- Group-Wise Quantization




| | | | |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 0 | 0 | -1 | 1 |
| 0 | 1 | 0 | -1 |
| 1 | 0 | 1 | 1 |

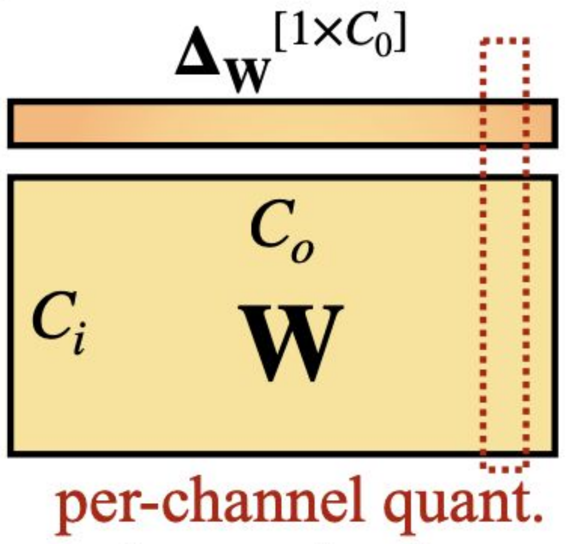
**Quantized**

| | | | |
|---|---|---|---|
| 2.09 | 0 | 2.09 | 0 |
| 0 | 0 | -2.12 | 2.12 |
| 0 | 1.92 | 0 | -1.92 |
| 1.87 | 0 | 1.87 | 1.87 |

**Reconstructed**

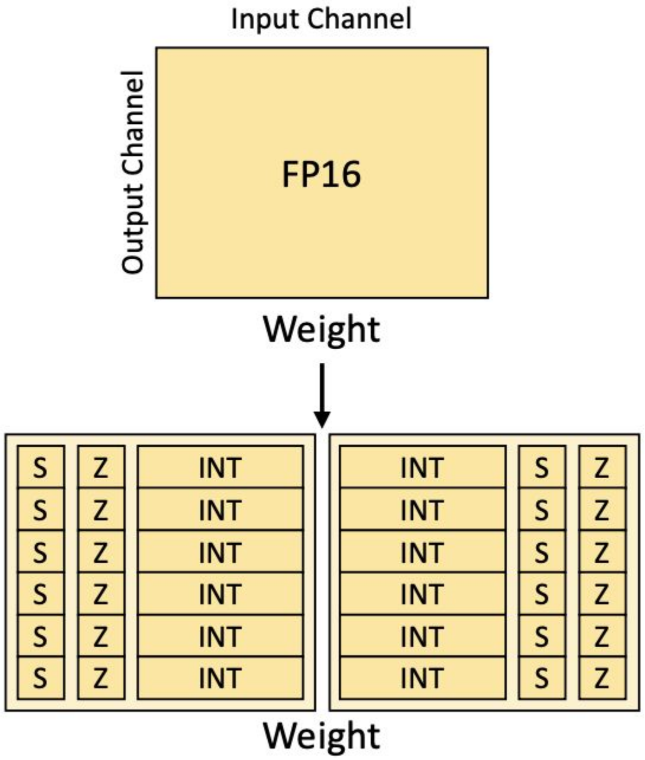
# Quantization Granularity

- Per-Tensor Quantization
- Per-Token Quantization
- Per-Channel Quantization
  - Uses different quantization step sizes for activations associated with each output channel of weights
- Group-Wise Quantization



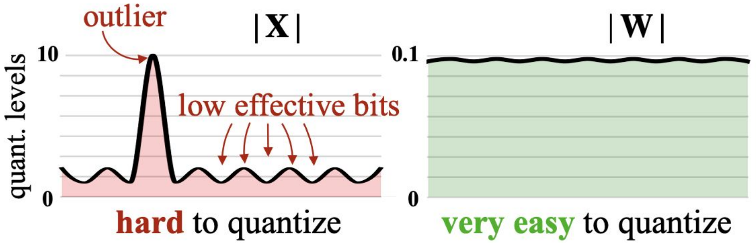
# Quantization Granularity

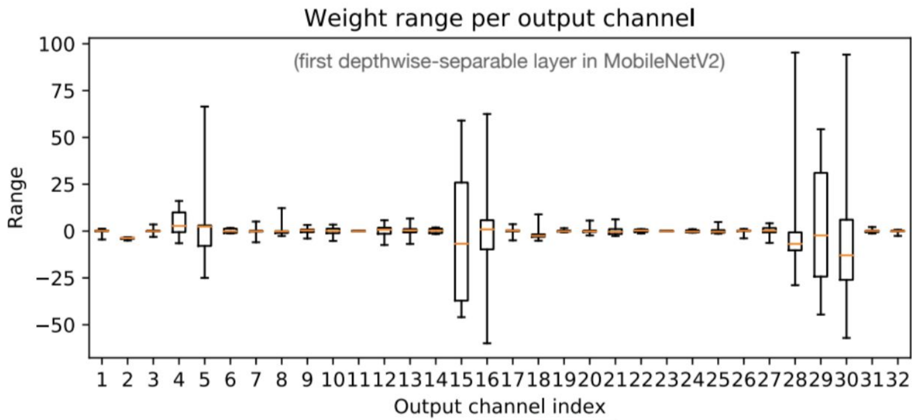
- Per-Tensor Quantization
- Per-Token Quantization
- Per-Channel Quantization
- Group-Wise Quantization
    - Different quantization steps for different channel groups



# Key Idea #1: Per-Channel Quantization is Infeasible

- Observations:
  - Outliers lead to low effective quantization bits
  - Outliers exist in a small fraction of channels
- Reasonable thought:
  - If we could perform per-channel quantization, the quantization error would be much smaller compared to per-tensor quantization



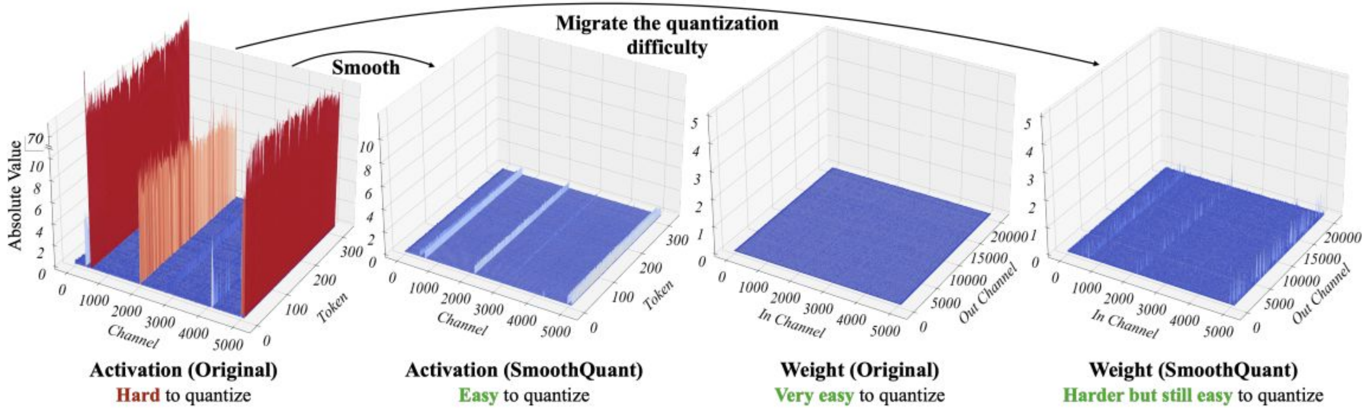
# Key Idea #1: Per-Channel Quantization is Infeasible

- Issue:
  - Hardware-accelerated GEMM kernels, that rely on a sequence of operations executed at a high throughput
  - Per-channel activation quantization relies on insertion of instructions with a lower throughput to apply different scales for each channel
  - GEMM kernels do not tolerate the insertion of instructions with a lower throughput

# Key Idea #2: Migrating the quantization difficulty

Weights are easy to quantize, but activations are hard due to outliers

$$\mathbf{Y} = \mathbf{X}\mathbf{W} = (0.01\mathbf{X})(100\mathbf{W})$$



# Key Idea #2: Migrating the quantization difficulty

- Issue from before: Activation outliers persist in fixed channels
  - Per-tensor quantization is limited
  - But per-channel quantization was infeasible.
- After preprocessing (smoothing): Linearity is exploited so that weights and activations can have similar degrees of outliers
  - Per-tensor quantization is effective again
  - Bake smoothing factor into previous layers (or residual branch for residual add)

# SmoothQuant's Per-Channel Smoothing Factor

$$\mathbf{Y} = (\mathbf{X}\text{diag}(\mathbf{s})^{-1}) \cdot (\text{diag}(\mathbf{s})\mathbf{W}) = \hat{\mathbf{X}}\hat{\mathbf{W}}$$

Push all quantization difficulty from activations to weights (all channels have same maximum magnitude):

$$\mathbf{s}_j = \max(|\mathbf{X}_j|), j = 1, 2, ..., C_i,$$

Push all quantization difficulty from weights to activations:

$$\mathbf{s}_j = 1/\max(|\mathbf{W}_j|).$$

Share difficulty according to α:

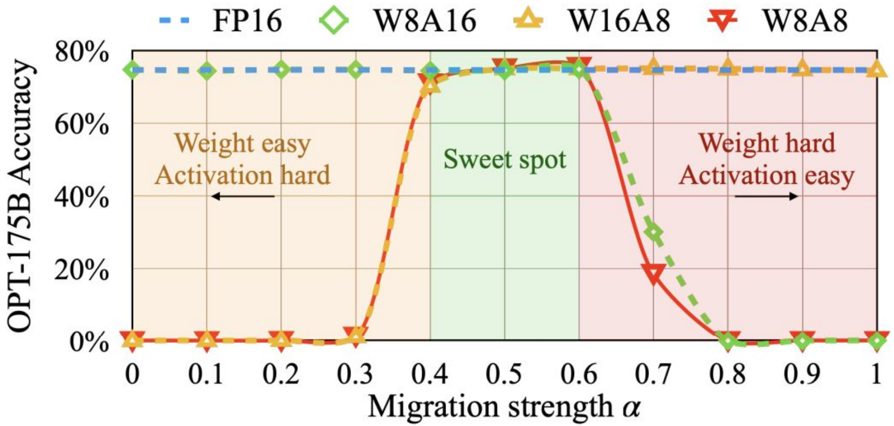
$$\mathbf{s}_j = \max(|\mathbf{X}_j|)^\alpha / \max(|\mathbf{W}_j|)^{1-\alpha}$$

# Choosing α

Case-by-case decision

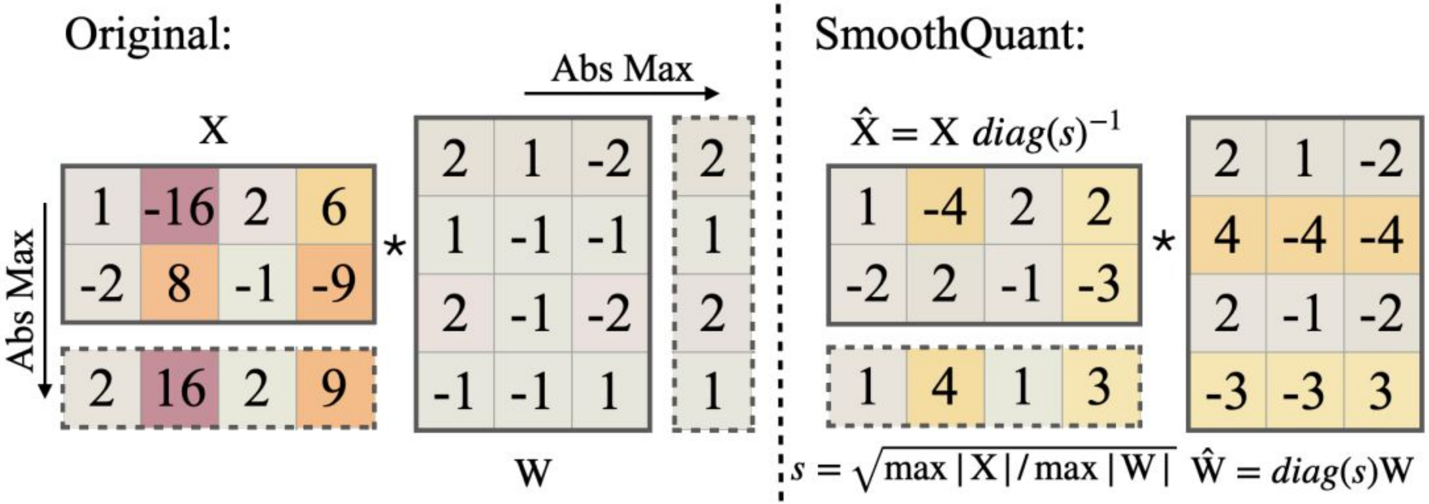
If the α is too large, weights will be hard to quantize. If too small, activations will be hard to quantize.

Goal: make activations and weights both easy to quantize.



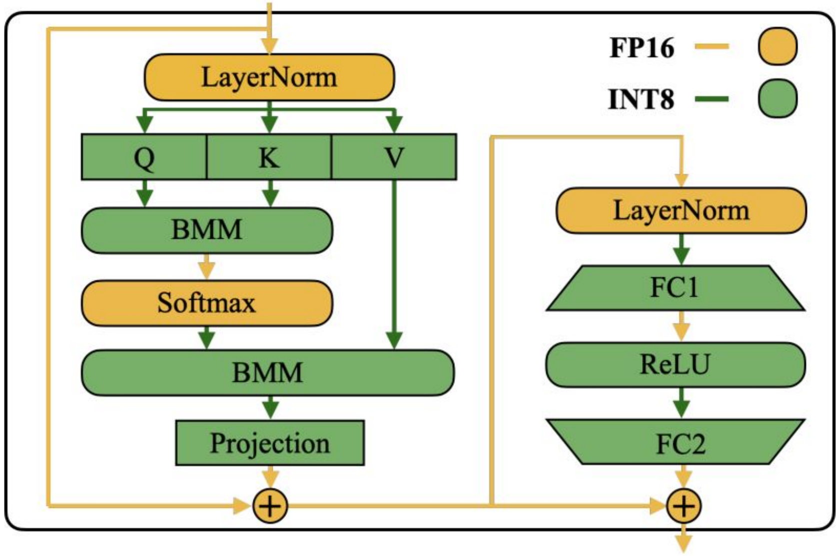
# Example of SmoothQuant

1) Apply Smoothing Factor
2) Quantize (constant step size)



# SmoothQuant Hardware Efficiency

- Applying SmoothQuant to transformer blocks
  - Linear layers take up most of the parameters and computation
  - Smoothing factor can be fused into previous layers' parameters offline
  - All linear layers are quantized with W8A8, as well as BMM operators in Attention computation



# Four Baselines

LLM.int8 keeps outliers in FP16 (large latency overhead). W8A8 is the naive implementation. Outlier suppression uses token-wise clipping

| Method | Weight | Activation |
|---|---|---|
| W8A8 | per-tensor | per-tensor dynamic |
| ZeroQuant | group-wise | per-token dynamic |
| `LLM.int8()` | per-channel | per-token dynamic+FP16 |
| Outlier Suppression | per-tensor | per-tensor static |

# SmoothQuant O1 to O3

Gradually aggressive and efficient (lower latency) quantization levels

| Method | Weight | Activation |
| --- | --- | --- |
| SmoothQuant-O1 | per-tensor | per-token dynamic |
| SmoothQuant-O2 | per-tensor | per-tensor dynamic |
| SmoothQuant-O3 | per-tensor | per-tensor static |

# Evaluation

- Three families of LLMs
  - OPT
    - $\alpha = 0.5$
  - BLOOM
    - $\alpha = 0.5$
  - GLM-130B
    - $\alpha$ is set to 0.75 since its activations are more difficult to quantize
- Seven zero-shot evaluation tasks e.g. LAMBADA, WikiText
- Focus on *relative* performance chance before/after quantization

# OPT-175B results

| *OPT-175B* | LAMBADA | HellaSwag | PIQA | WinoGrande | OpenBookQA | RTE | COPA | **Average↑** | **WikiText↓** |
|---|---|---|---|---|---|---|---|---|---|
| FP16 | 74.7% | 59.3% | 79.7% | 72.6% | 34.0% | 59.9% | 88.0% | 66.9% | 10.99 |
| W8A8 | 0.0% | 25.6% | 53.4% | 50.3% | 14.0% | 49.5% | 56.0% | 35.5% | 93080 |
| ZeroQuant | 0.0%* | 26.0% | 51.7% | 49.3% | 17.8% | 50.9% | 55.0% | 35.8% | 84648 |
| `LLM.int8()` | 74.7% | 59.2% | 79.7% | 72.1% | 34.2% | 60.3% | 87.0% | 66.7% | 11.10 |
| Outlier Suppression | 0.00% | 25.8% | 52.5% | 48.6% | 16.6% | 53.4% | 55.0% | 36.0% | 96151 |
| SmoothQuant-O1 | 74.7% | 59.2% | 79.7% | 71.2% | 33.4% | 58.1% | 89.0% | 66.5% | 11.11 |
| SmoothQuant-O2 | 75.0% | 59.0% | 79.2% | 71.2% | 33.0% | 59.6% | 88.0% | 66.4% | 11.14 |
| SmoothQuant-O3 | 74.6% | 58.9% | 79.7% | 71.2% | 33.4% | 59.9% | 90.0% | 66.8% | 11.17 |

# Results On Different LLMs

| Method | OPT-175B | BLOOM-176B | GLM-130B* |
|---|---|---|---|
| FP16 | 71.6% | 68.2% | 73.8% |
| W8A8 | 32.3% | 64.2% | 26.9% |
| ZeroQuant | 31.7% | 67.4% | 26.7% |
| `LLM.int8()` | 71.4% | 68.0% | 73.8% |
| Outlier Suppression | 31.7% | 54.1% | 63.5% |
| SmoothQuant-O1 | **71.2%** | 68.3% | **73.7%** |
| SmoothQuant-O2 | 71.1% | **68.4%** | 72.5% |
| SmoothQuant-O3 | 71.1% | 67.4% | 72.8% |

# Lossless W8A8 quantization for LLaMA models

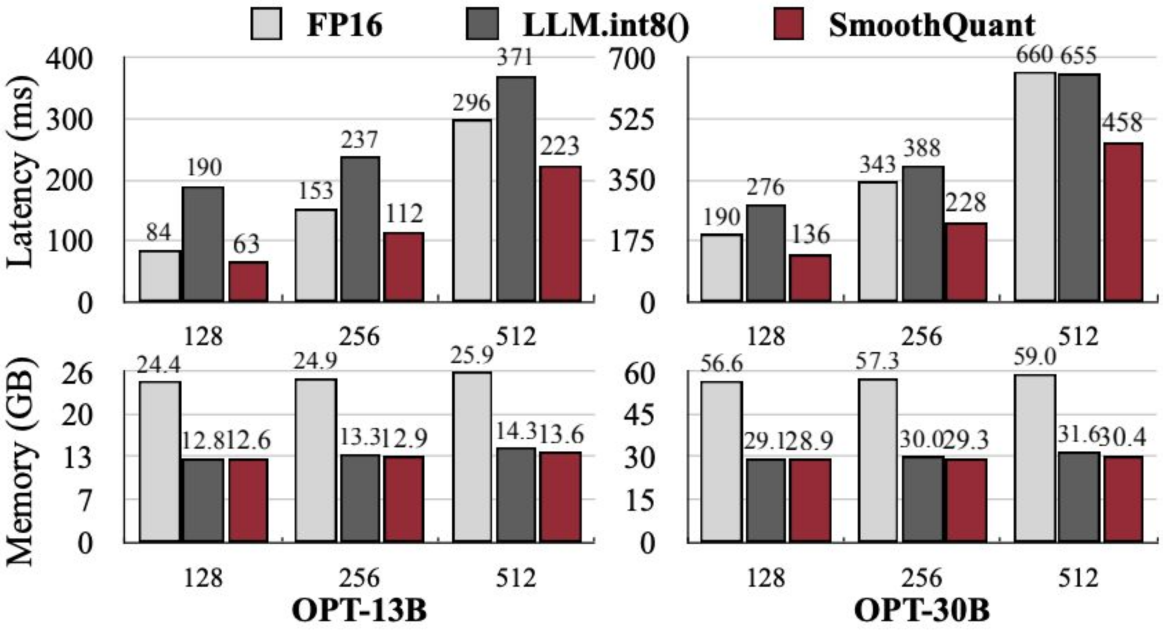
Lower perplexity is better

| Wiki PPL↓ | 7B | 13B | 30B | 65B |
|---|---|---|---|---|
| FP16 | 11.51 | 10.05 | 7.53 | 6.17 |
| W8A8 SmoothQuant | 11.56 | 10.08 | 7.56 | 6.20 |

Similar!

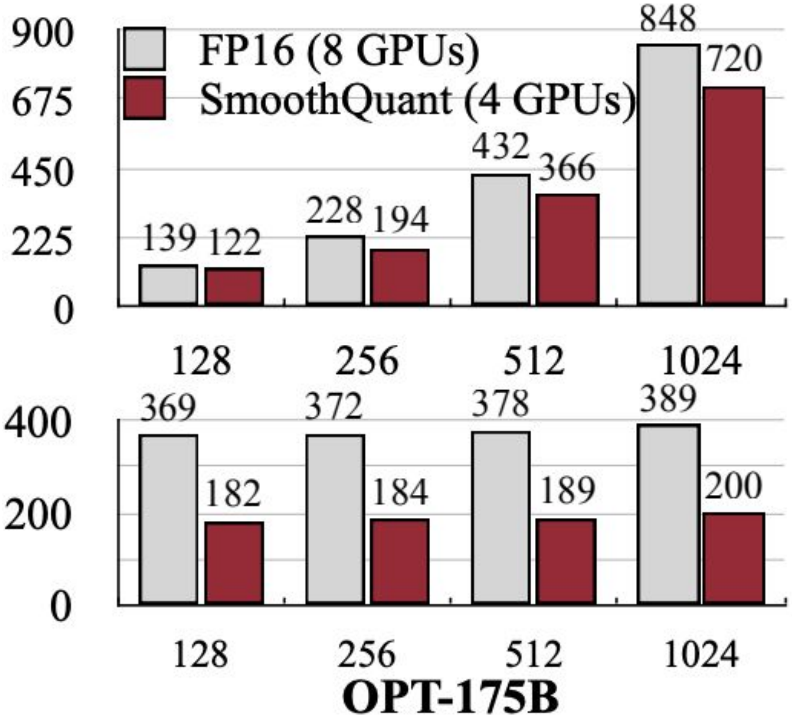
# Memory/Latency Savings

Only compares with LLM.int8() because it is the only method that maintains accuracy.



# Hardware Efficiency

Similar or faster latency with half # GPUs



# Overall Results

- SmoothQuant is faster than FP16 baseline under all settings
- LLM.int8() is usually slower than SmoothQuant
- Additionally: SmoothQuant can serve a >500B model within a single node (8×A100 80GB GPUs) at a negligible accuracy loss

# My Thoughts

- Strengths
  - Novel suggestion of migrating quantization difficulty to weights
  - Surpasses SOTA PTQ methods in high accuracy and low latency
  - High quality of evaluation - many families of LLMs including LLaMA
  - Easy to use (no training required)
- Weaknesses
  - Selecting the α hyperparameter is a little difficult to get right
  - Getting activation statistics requires some work
- Design Choices
  - Very thoughtful and simple use of matmul’s linearity
  - Efficiency relies on access to hardware accelerators
- Future Directions
  - Getting down to 4 bits for weights and/or activations
  - Could FP4 be viable?

# Discussion

- Can SmoothQuant scale for even larger LLMs (e.g. 100 trillion parameters)?
- Are there any specific types of applications where SmoothQuant wouldn't be the best choice?
- What prevents quantizing to reduced bit-widths of size 4, 2, or even 1?

# Q&A