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

Guangxuan Xiao\textsuperscript{*,1}, Ji Lin\textsuperscript{*,4},
Mickael Seznec\textsuperscript{2}, Hao Wu\textsuperscript{2}, Julien Demouth\textsuperscript{2}, Song Han\textsuperscript{1}

\textsuperscript{1} MIT \quad \textsuperscript{2} NVIDIA
\textsuperscript{*} Equal contribution
Quantization for LLMs is Important

- NLP model size and computation are increasing exponentially. Model Compression with:
  - Quantization (SmoothQuant) <= today’s focus: training-free, model-in & model-out.
  - Token pruning (SpAtten)
  - Neural architecture search (HAT, Lite-Transformer)
Linear Quantization

An affine mapping of integers to real numbers $r = S(q - Z)$

$$r = S(q - Z)$$
Existing Quantization Method is Slow or Inaccurate

- Systematic outliers emerge in activations when we scale up LLMs beyond 6.7B. Naive but efficient quantization methods will destroy the accuracy.
- The accuracy-preserving baseline, LLM.int8() uses FP16 to represent outliers, which needs runtime outlier detection, scattering and gathering. It is slower than FP16 inference.
SmoothQuant: Accurate and Efficient Post-Training Quantization for LLMs

<table>
<thead>
<tr>
<th></th>
<th>LLM (100B+) \hspace{1cm} Accuracy</th>
<th>Hardware \hspace{1cm} Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZeroQuant</td>
<td>❌</td>
<td>✔</td>
</tr>
<tr>
<td>Outlier Suppression</td>
<td>❌</td>
<td>✔</td>
</tr>
<tr>
<td>LLM.int8()</td>
<td>✔</td>
<td>❌</td>
</tr>
<tr>
<td>SmoothQuant</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

• We propose SmoothQuant, an **accurate** and **efficient** post-training-quantization (PTQ) method to enable 8-bit weight, 8-bit activation (W8A8) quantization for LLMs.

• Since **weights are easy** to quantize while **activations are not**, SmoothQuant smooths the activation outliers by migrating the quantization difficulty from activations to weights with a mathematically equivalent transformation.
Review the Quantization Difficulty of LLMs

LLMs are notoriously difficult to quantize because:

- Activations are harder to quantize than weights
- Outliers make activation quantization difficult
- Outliers persist in fixed channels
Review the Quantization Difficulty of LLMs

- Activations are harder to quantize than weights
  Previous work has shown quantizing the weights of LLMs with INT8 or even INT4 doesn’t degrade accuracy.

Activation (Original) 
**Hard** to quantize

Weight (Original) 
**Very easy** to quantize

LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale (Dettmers et al., 2022)
GLM-130b: An open bilingual pre-trained model (Zeng et al., 2022)
Review the Quantization Difficulty of LLMs

- Outliers make activation quantization difficult
  The scale of outliers is ~100x larger than most of the activation values.
  If we use INT8 quantization, most values will be zeroed out.

Understanding and overcoming the challenges of efficient transformer quantization (Bondarenko et al., 2021)
LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale (Dettmers et al., 2022)
Review the Quantization Difficulty of LLMs

- Outliers persist in *fixed* channels
  
  Fixed channels have outliers, and the outlier channels are persistently large.

![Activation (Original)](image)

**Hard** to quantize
Among different activation quantization schemes, only per-channel quantization preserves the accuracy, but it is not compatible with INT8 GEMM kernels.

\[
\Delta X = \max(|X|) \quad \Delta = \frac{2^{N-1} - 1}{|X|} \\
Y = \text{diag}(\Delta_{FP16}) \cdot (\tilde{X}_{INT8} \cdot \tilde{W}_{INT8}) \cdot \text{diag}(\Delta_{FP16})
\]
Review the Quantization Difficulty of LLMs

- Activations are harder to quantize than weights
- Outliers make activation quantization difficult
- Outliers persist in fixed channels

We can smooth the outlier channels in activations by migrating their magnitudes into the following weights!
Activation Smoothing

Original:

\[
\begin{array}{ccc}
X & \text{Abs Max} & W \\
1 & -16 & 2 & 6 \\
-2 & 8 & -1 & -9 \\
2 & 16 & 2 & 9 \\
\end{array}
\]

SmoothQuant:

\[
\begin{array}{ccc}
\hat{X} = X \ diag(s)^{-1} \\
1 & -4 & 2 & 2 \\
-2 & 2 & -1 & -3 \\
1 & 4 & 1 & 3 \\
\end{array}
\]

\[
s = \frac{\text{max} |X|}{\text{max} |W|} \quad \hat{W} = diag(s)W
\]

\[
s_j = \max(|X_j|)^\alpha / \max(|W_j|)^{1-\alpha}, \ j = 1, 2, \ldots, C_i
\]

\[
Y = (X \ diag(s)^{-1}) \cdot (\ diag(s)W) = \hat{X}\hat{W}
\]

\[\alpha: \ \text{Migration Strength}\]
Activation Smoothing

1. Calibration Stage (Offline):

\[ s_j = \frac{\max(\|X_j\|)^\alpha}{\max(\|W_j\|)^{1-\alpha}}, \quad j = 1, 2, \ldots, C_i \]

\( \alpha \): Migration Strength
Activation Smoothing

2. Smoothing Stage (Offline):

\[
\hat{X} = X \, \text{diag}(s)^{-1}
\]

\[
\begin{bmatrix}
1 & -16 & 2 & 6 \\
-2 & 8 & -1 & -9
\end{bmatrix}
\div
\begin{bmatrix}
1 & 4 & 1 & 3
\end{bmatrix}
\]

\[
s_j = \max(|X_j|)^{\alpha}/\max(|W_j|)^{1-\alpha}, \ j = 1,2,\ldots, C_i
\]

\[
Y = (X\, \text{diag}(s)^{-1}) \cdot (\text{diag}(s)W) = \hat{X}\hat{W}
\]

\[\alpha: \text{Migration Strength}\]
3. Inference (deployed model):

At runtime, the activations are smooth and easy to quantize.

\[ Y = \hat{X}\hat{W} \]
• SmoothQuant’s precision mapping for a Transformer block.
• All compute-intensive operators, such as linear layers and batched matrix multiplications (BMMs) use INT8 arithmetic.

Quantization setting of the baselines and SmoothQuant. All weight and activations use INT8 representations unless specified.
• We implement three efficiency levels of quantization settings for SmoothQuant. The efficiency improves from O1 to O3.
### Accuracy on OPT-175B

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

SmoothQuant maintains the accuracy of OPT-175B model after INT8 quantization, even with the most aggressive and most efficient O3 setting.
SmoothQuant works for different LLMs. We can quantize the 3 largest, openly available LLM models into INT8 without degrading the accuracy.
Scaling Up: 530B Model Within a Single Node

### MT-NLG 530B Accuracy

<table>
<thead>
<tr>
<th></th>
<th>LAMBADA</th>
<th>HellaSwag</th>
<th>PIQA</th>
<th>WinoGrande</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FP16</strong></td>
<td>76.6%</td>
<td>62.1%</td>
<td>81.0%</td>
<td>72.9%</td>
<td>73.1%</td>
</tr>
<tr>
<td><strong>INT8</strong></td>
<td>77.2%</td>
<td>60.4%</td>
<td>80.7%</td>
<td>74.1%</td>
<td>73.1%</td>
</tr>
</tbody>
</table>

### MT-NLG 530B Efficiency

<table>
<thead>
<tr>
<th>SeqLen</th>
<th>Prec.</th>
<th>#GPUs</th>
<th>Latency</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>FP16</td>
<td>16</td>
<td>232ms</td>
<td>1040GB</td>
</tr>
<tr>
<td></td>
<td>INT8</td>
<td>8</td>
<td>253ms</td>
<td>527GB</td>
</tr>
<tr>
<td>256</td>
<td>FP16</td>
<td>16</td>
<td>451ms</td>
<td>1054GB</td>
</tr>
<tr>
<td></td>
<td>INT8</td>
<td>8</td>
<td>434ms</td>
<td>533GB</td>
</tr>
<tr>
<td>512</td>
<td>FP16</td>
<td>16</td>
<td>838ms</td>
<td>1068GB</td>
</tr>
<tr>
<td></td>
<td>INT8</td>
<td>8</td>
<td>839ms</td>
<td>545GB</td>
</tr>
<tr>
<td>1024</td>
<td>FP16</td>
<td>16</td>
<td>1707ms</td>
<td>1095GB</td>
</tr>
<tr>
<td></td>
<td>INT8</td>
<td>8</td>
<td>1689ms</td>
<td>570GB</td>
</tr>
</tbody>
</table>

SmoothQuant can accurately quantize MT-NLG 530B model and reduce the serving GPU numbers by half at a similar latency, which allows serving the 530B model within a single node.
Ablation Study on the Migration Strength $\alpha$

Migration strength $\alpha$ controls the amount of quantization difficulty migrated from activations to weights.

A suitable migration strength (sweet spot) makes both activations and weights easy to quantize.

If the $\alpha$ is too large, weights will be hard to quantize; if too small, activations will be hard to quantize.

$$s_j = \max(|X_j|)^\alpha / \max(|W_j|)^{1-\alpha}, \quad j = 1, 2, \ldots, C_i$$

$$Y = (X \text{diag}(s)^{-1}) \cdot (\text{diag}(s)W) = \hat{X}\hat{W}$$
The PyTorch implementation of SmoothQuant achieves up to $1.51\times$ speedup and $1.96\times$ memory saving for OPT models on a single NVIDIA A100-80GB GPU, while LLM.int8() slows down the inference in most cases.
We integrate SmoothQuant into FasterTransformer, a state-of-the-art Transformer serving framework. For smaller models, the latency can be significantly reduced with SmoothQuant by up to 1.56x compared to FP16. For the bigger models (OPT-66B and 175B), we can achieve similar or even faster inference using only half number of GPUs. Memory footprint is almost halved compared to FP16.
Conclusion

• We propose SmoothQuant, a turn-key solution to enable accurate W8A8 quantization for large language models.

• SmoothQuant is accurate and efficient on existing hardware. We can implement SmoothQuant with off-the-shelf kernels to achieve high speedup and memory saving.

• Paper: https://arxiv.org/abs/2211.10438

• Code: https://github.com/mit-han-lab/smoothquant
Integration

- NVIDIA: FasterTransformer
- Intel Neural Compressor

- Code: https://github.com/mit-han-lab/smoothquant