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

Tim Dettmers[λ*] **Mike Lewis**[†] **Younes Belkada**[§∓] **Luke Zettlemoyer**[†λ]

University of Washington[λ]
Facebook AI Research[†]
Hugging Face[§]
ENS Paris-Saclay[∓]

Abstract

Large language models have been widely adopted but require significant GPU memory for inference. We develop a procedure for Int8 matrix multiplication for feed-forward and attention projection layers in transformers, which cut the memory needed for inference by half while retaining full precision performance. With our method, a 175B parameter 16/32-bit checkpoint can be loaded, converted to Int8, and used immediately without performance degradation. This is made possible by understanding and working around properties of highly systematic emergent features in transformer language models that dominate attention and transformer predictive performance. To cope with these features, we develop a two-part quantization procedure, **LLM.int8()**. We first use vector-wise quantization with separate normalization constants for each inner product in the matrix multiplication, to quantize most of the features. However, for the emergent outliers, we also include a new mixed-precision decomposition scheme, which isolates the outlier feature dimensions into a 16-bit matrix multiplication while still more than 99.9% of values are multiplied in 8-bit. Using LLM.int8(), we show empirically it is possible to perform inference in LLMs with up to 175B parameters without any performance degradation. This result makes such models much more accessible, for example making it possible to use OPT-175B/BLOOM on a single server with consumer GPUs. We open source our software.

1 Introduction

Large pretrained language models are widely adopted in NLP (Vaswani et al., 2017; Radford et al., 2019; Brown et al., 2020; Zhang et al., 2022) but require significant memory for inference. For large transformer language models at and beyond 6.7B parameters, the feed-forward and attention projection layers and their matrix multiplication operations are responsible for 95%[2] of consumed parameters and 65-85% of all computation (Ilharco et al., 2020). One way to reduce the size of the parameters is to quantize them to less bits and use low-bit-precision matrix multiplication. With this goal in mind, 8-bit quantization methods for transformers have been developed (Chen et al., 2020; Lin et al., 2020; Zafrir et al., 2019; Shen et al., 2020). While these methods reduce memory use, they degrade performance, usually require tuning quantization further after training, and have only been studied for models with less than 350M parameters. Degradation-free quantization up to 350M parameters is poorly understood, and multi-billion parameter quantization remains an open challenge.

[*]Majority of research done as a visiting researcher at Facebook AI Research.
[2]Other parameters come mostly from the embedding layer. A tiny amount comes from norms and biases.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

In this paper, we present the first multi-billion-scale Int8 quantization procedure for transformers that does not incur any performance degradation. Our procedure makes it possible to load a 175B parameter transformer with 16 or 32-bit weights, convert the feed-forward and attention projection layers to 8-bit, and use the resulting model immediately for inference without any performance degradation. We achieve this result by solving two key challenges: the need for higher quantization precision at scales beyond 1B parameters and the need to explicitly represent the sparse but systematic large magnitude outlier features that ruin quantization precision once they emerge in *all* transformer layers starting at scales of 6.7B parameters. This loss of precision is reflected in C4 evaluation perplexity (Section 3) as well as zeroshot accuracy as soon as these outlier features emerge, as shown in Figure 1.

We show that with the first part of our method, vector-wise quantization, it is possible to retain performance at scales up to 2.7B parameters. For vector-wise quantization, matrix multiplication can be seen as a sequence of independent inner products of row and column vectors. As such, we can use a separate quantization normalization constant for each inner product to improve quantization precision. We can recover the output of the matrix multiplication by denormalizing by the outer product of column and row normalization constants before we perform the next operation.

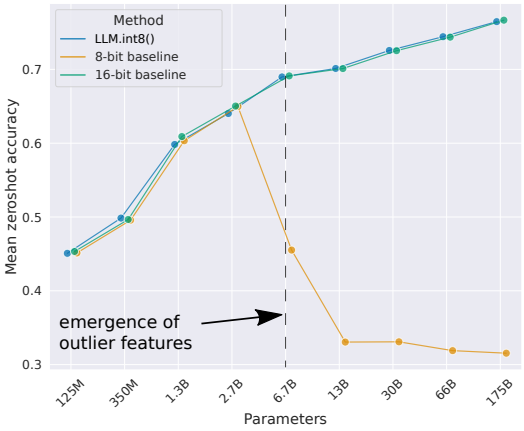

Figure 1: OPT model mean zeroshot accuracy for WinoGrande, HellaSwag, PIQA, and LAMBADA datasets. Shown is the 16-bit baseline, the most precise previous 8-bit quantization method as a baseline, and our new 8-bit quantization method, LLM.int8(). We can see once systematic outliers occur at a scale of 6.7B parameters, regular quantization methods fail, while LLM.int8() maintains 16-bit accuracy.

To scale beyond 6.7B parameters without performance degradation, it is critical to understand the emergence of extreme outliers in the feature dimensions of the hidden states during inference. To this end, we provide a new descriptive analysis which shows that large features with magnitudes up to 20x larger than in other dimensions first appear in about 25% of all transformer layers and then gradually spread to other layers as we scale transformers to 6B parameters. At around 6.7B parameters, a phase shift occurs, and *all* transformer layers and 75% of all sequence dimensions are affected by extreme magnitude features. These outliers are highly systematic: at the 6.7B scale, 150,000 outliers occur per sequence, but they are concentrated in only 6 feature dimensions across the entire transformer. Setting these outlier feature dimensions to zero decreases top-1 attention softmax probability mass by more than 20% and degrades validation perplexity by 600-1000% despite them only making up about 0.1% of all input features. In contrast, removing the same amount of random features decreases the probability by a maximum of 0.3% and degrades perplexity by about 0.1%.

To support effective quantization with such extreme outliers, we develop mixed-precision decomposition, the second part of our method. We perform 16-bit matrix multiplication for the outlier feature dimensions and 8-bit matrix multiplication for the other 99.9% of the dimensions. We name the combination of vector-wise quantization and mixed precision decomposition, **LLM.int8()**. We show that by using LLM.int8(), we can perform inference in LLMs with up to 175B parameters without any performance degradation. Our method not only provides new insights into the effects of these outliers on model performance but also makes it possible for the first time to use very large models, for example, OPT-175B/BLOOM, on a single server with consumer GPUs. While our work focuses on making large language models accessible without degradation, we also show in Appendix D that we maintain end-to-end inference runtime performance for large models, such as BLOOM-176B and provide modest matrix multiplication speedups for GPT-3 models of size 6.7B parameters or larger. We open-source our software[3] and release a Hugging Face Transformers (Wolf et al., 2019) integration making our method available to all hosted Hugging Face Models that have linear layers.

[3] `https://github.com/TimDettmers/bitsandbytes`

Figure 2: Schematic of LLM.int8(). Given 16-bit floating-point inputs $\mathbf{X}_{f16}$ and weights $\mathbf{W}_{f16}$, the features and weights are decomposed into sub-matrices of large magnitude features and other values. The outlier feature matrices are multiplied in 16-bit. All other values are multiplied in 8-bit. We perform 8-bit vector-wise multiplication by scaling by row and column-wise absolute maximum of $\mathbf{C}_x$ and $\mathbf{C}_w$ and then quantizing the outputs to Int8. The Int32 matrix multiplication outputs $\mathbf{Out}_{i32}$ are dequantization by the outer product of the normalization constants $\mathbf{C}_x \otimes \mathbf{C}_w$. Finally, both outlier and regular outputs are accumulated in 16-bit floating point outputs.

2 Background

In this work, push quantization techniques to their breaking point by scaling transformer models. We are interested in two questions: at which scale and why do quantization techniques fail and how does this related to quantization precision? To answer these questions we study high-precision asymmetric quantization (zeropoint quantization) and symmetric quantization (absolute maximum quantization). While zeropoint quantization offers high precision by using the full bit-range of the datatype, it is rarely used due to practical constraints. Absolute maximum quantization is the most commonly used technique.

2.1 8-bit Data Types and Quantization

Absmax quantization scales inputs into the 8-bit range $[-127, 127]$ by multiplying with $s_{x_{f16}}$ which is 127 divided by the absolute maximum of the entire tensor. This is equivalent to dividing by the infinity norm and multiplying by 127. As such, for an FP16 input matrix $\mathbf{X}_{f16} \in \mathbb{R}^{s \times h}$ Int8 absmax quantization is given by:

$$\mathbf{X}_{i8} = \left\lfloor \frac{127 \cdot \mathbf{X}_{f16}}{\max_{ij}(|\mathbf{X}_{f16_{ij}}|)} \right\rceil = \left\lfloor \frac{127}{\|\mathbf{X}_{f16}\|_\infty} \mathbf{X}_{f16} \right\rceil = \left\lfloor s_{x_{f16}} \mathbf{X}_{f16} \right\rceil,$$

where $\lfloor \rceil$ indicates rounding to the nearest integer.

Zeropoint quantization shifts the input distribution into the full range $[-127, 127]$ by scaling with the normalized dynamic range nd_x and then shifting by the zeropoint zp_x. With this affine transformation any input tensors will use all bits of the data type, thus *reducing the quantization error for asymmetric distributions*

$$\mathbf{X}_{f16} = \left\lfloor s_{x_{f16}} \mathbf{X}_{f16} \right\rceil,$$

to the nearest integer.

To use zeropoint quantization in an operation we feed both the tensor $\mathbf{X}_{i8}$ and the zeropoint $zp_{x_{i16}}$ into a special instruction[4] which adds $zp_{x_{i16}}$ to each element of $\mathbf{X}_{i8}$ before performing a 16-bit integer operation. For example, to multiply two zeropoint quantized numbers A_{i8} and B_{i8} along with their zeropoints $zp_{a_{i16}}$ and $zp_{b_{i16}}$ we calculate:

$$C_{i32} = \text{multiply}_{i16}(A_{zp_{a_{i16}}}, B_{zp_{b_{i16}}}) = (A_{i8} + zp_{a_{i16}})(B_{i8} + zp_{b_{i16}}) \tag{4}$$

where unrolling is required if the instruction multiply_{i16} is not available such as on GPUs or TPUs:

$$C_{i32} = A_{i8}B_{i8} + A_{i8}zp_{b_{i16}} + B_{i8}zp_{a_{i16}} + zp_{a_{i16}}zp_{b_{i16}}, \tag{5}$$

where $A_{i8}B_{i8}$ is computed with Int8 precision while the rest is computed in Int16/32 precision. As such, zeropoint quantization can be slow if the multiply_{i16} instruction is not available. In both cases, the outputs are accumulated as a 32-bit integer C_{i32}. To dequantize C_{i32}, we divide by the scaling constants $nd_{a_{f16}}$ and $nd_{b_{f16}}$.

Int8 Matrix Multiplication with 16-bit Float Inputs and Outputs. Given hidden states $\mathbf{X}_{f16} \in \mathbb{R}^{s \times h}$ and weights $\mathbf{W}_{f16} \in \mathbb{R}^{h \times o}$ with sequence dimension s, feature dimension h, and output dimension o we perform 8-bit matrix multiplication with 16-bit inputs and outputs as follows:

$$\begin{aligned} \mathbf{X}_{f16}\mathbf{W}_{f16} = \mathbf{C}_{f16} &\approx \frac{1}{c_{x_{f16}}c_{w_{f16}}}\mathbf{C}_{i32} = S_{f16} \cdot \mathbf{C}_{i32} \\ &\approx S_{f16} \cdot \mathbf{A}_{i8}\mathbf{B}_{i8} = S_{f16} \cdot Q(\mathbf{A}_{f16})\, Q(\mathbf{B}_{f16}), \end{aligned} \tag{6}$$

Where $Q(\cdot)$ is either absmax or zeropoint quantization and $c_{x_{f16}}$ and $c_{w_{f16}}$ are the respective tensor-wise scaling constants s_x and s_w for absmax or nd_x and nd_w for zeropoint quantization.

3 Int8 Matrix Multiplication at Scale

The main challenge with quantization methods that use a single scaling constant per tensor is that a single outlier can reduce the quantization precision of all other values. As such, it is desirable to have multiple scaling constants per tensor, such as block-wise constants (Dettmers et al., 2022), so that the effect of that outliers is confined to each block. We improve upon one of the most common ways of blocking quantization, row-wise quantization (Khudia et al., 2021), by using vector-wise quantization, as described in more detail below.

To handle the large magnitude outlier features that occur in all transformer layers beyond the 6.7B scale, vector-wise quantization is no longer sufficient. For this purpose, we develop mixed-precision decomposition, where the small number of large magnitude feature dimensions (≈0.1%) are represented in 16-bit precision while the other 99.9% of values are multiplied in 8-bit. Since most entries are still represented in low-precision, we retain about 50% memory reduction compared to 16-bit. For example, for BLOOM-176B, we reduce the memory footprint of the model by 1.96x.

Vector-wise quantization and mixed-precision decomposition are shown in Figure 2. The **LLM.int8()** method is the combination of absmax vector-wise quantization and mixed precision decomposition.

3.1 Vector-wise Quantization

One way to increase the number of scaling constants for matrix multiplication is to view matrix multiplication as a sequence of independent inner products. Given the hidden states $\mathbf{X}_{f16} \in \mathbb{R}^{b \times h}$ and weight matrix $\mathbf{W}_{f16} \in \mathbb{R}^{h \times o}$, we can assign a different scaling constant $c_{x_{f16}}$to each row of $\mathbf{X}_{f16}$ and c_w to each column of $\mathbf{W}_{f16}$. To dequantize, we denormalize each inner product result by $1/(c_{x_{f16}}c_{w_{f16}})$. For the whole matrix multiplication this is equivalent to denormalization by the outer product $\mathbf{c}_{x_{f16}} \otimes \mathbf{c}_{w_{f16}}$, where $\mathbf{c}_x \in \mathbb{R}^s$ and $\mathbf{c}_w \in \mathbb{R}^o$. As such the full equation for matrix multiplication with row and column constants is given by:

$$\mathbf{C}_{f_{16}} \approx \frac{1}{\mathbf{c}_{x_{f16}} \otimes \mathbf{c}_{w_{f16}}}\mathbf{C}_{i32} = S \cdot \mathbf{C}_{i32} = \mathbf{S} \cdot \mathbf{A}_{i8}\mathbf{B}_{i8} = \mathbf{S} \cdot Q(\mathbf{A}_{f16})\, Q(\mathbf{B}_{f16}), \tag{7}$$

which we term *vector-wise quantization* for matrix multiplication.

[4] `https://www.felixcloutier.com/x86/pmaddubsw`

3.2 The Core of LLM.int8(): Mixed-precision Decomposition

In our analysis, we demonstrate that a significant problem for billion-scale 8-bit transformers is that they have large magnitude features (*columns*), which are important for transformer performance and require high precision quantization. However, vector-wise quantization, our best quantization technique, quantizes each *row* for the hidden state, which is ineffective for outlier features. Luckily, we see that these outlier features are both incredibly sparse and systematic in practice, making up only about 0.1% of all feature dimensions, thus allowing us to develop a new decomposition technique that focuses on high precision multiplication for these particular dimensions.

We find that given input matrix $\mathbf{X}_{f16} \in \mathbb{R}^{s \times h}$, these outliers occur systematically for almost all sequence dimensions s but are limited to specific feature/hidden dimensions h. As such, we propose *mixed-precision decomposition* for matrix multiplication where we separate outlier feature dimensions into the set $O = \{i | i \in \mathbb{Z}, 0 \leq i \leq h\}$, which contains all dimensions of h which have at least one outlier with a magnitude larger than the threshold α. In our work, we find that $\alpha = 6.0$ is sufficient to reduce transformer performance degradation close to zero. Using Einstein notation where all indices are superscripts, given the weight matrix $\mathbf{W}_{f16} \in \mathbb{R}^{h \times o}$, mixed-precision decomposition for matrix multiplication is defined as follows:

$$\mathbf{C}_{f16} \approx \sum_{h \in O} \mathbf{X}^h_{f16} \mathbf{W}^h_{f16} + \mathbf{S}_{f16} \cdot \sum_{h \notin O} \mathbf{X}^h_{i8} \mathbf{W}^h_{i8} \tag{8}$$

where $\mathbf{S}_{f16}$ is the denormalization term for the Int8 inputs and weight matrices $\mathbf{X}_{i8}$ and $\mathbf{W}_{i8}$.

This separation into 8-bit and 16-bit allows for high-precision multiplication of outliers while using memory-efficient matrix multiplication with 8-bit weights of more than 99.9% of values. Since the number of outlier feature dimensions is not larger than 7 ($|O| \leq 7$) for transformers up to 13B parameters, this decomposition operation only consumes about 0.1% additional memory.

3.3 Experimental Setup

We measure the robustness of quantization methods as we scale the size of several publicly available pretrained language models up to 175B parameters. The key question is not how well a quantization method performs for a particular model but the trend of how such a method performs as we scale.

We use two setups for our experiments. One is based on language modeling perplexity, which we find to be a highly robust measure that is very sensitive to quantization degradation. We use this setup to compare different quantization baselines. Additionally, we evaluate zeroshot accuracy degradation on OPT models for a range of different end tasks, where we compare our methods with a 16-bit baseline.

For the language modeling setup, we use dense autoregressive transformers pretrained in fairseq (Ott et al., 2019) ranging between 125M and 13B parameters. These transformers have been pretrained on Books (Zhu et al., 2015), English Wikipedia, CC-News (Nagel, 2016), OpenWebText (Gokaslan and Cohen, 2019), CC-Stories (Trinh and Le, 2018), and English CC100 (Wenzek et al., 2020). For more information on how these pretrained models are trained, see Artetxe et al. (2021).

To evaluate the language modeling degradation after Int8 quantization, we evaluate the perplexity of the 8-bit transformer on validation data of the C4 corpus (Raffel et al., 2019) which is a subset of the Common Crawl corpus.[5] We use NVIDIA A40 GPUs for this evaluation.

To measure degradation in zeroshot performance, we use OPT models (Zhang et al., 2022), and we evaluate these models on the EleutherAI language model evaluation harness (Gao et al., 2021).

3.4 Main Results

The main language modeling perplexity results on the 125M to 13B Int8 models evaluated on the C4 corpus can be seen in Table 1. We see that absmax, row-wise, and zeropoint quantization fail as we scale, where models after 2.7B parameters perform worse than smaller models. Zeropoint quantization fails instead beyond 6.7B parameters. Our method, LLM.int8(), is the only method that preserves perplexity. As such, LLM.int8() is the only method with a favorable scaling trend.

[5] `https://commoncrawl.org/`

Table 1: C4 validation perplexities of quantization methods for different transformer sizes ranging from 125M to 13B parameters. We see that absmax, row-wise, zeropoint, and vector-wise quantization leads to significant performance degradation as we scale, particularly at the 13B mark where 8-bit 13B perplexity is worse than 8-bit 6.7B perplexity. If we use LLM.int8(), we recover full perplexity as we scale. Zeropoint quantization shows an advantage due to asymmetric quantization but is no longer advantageous when used with mixed-precision decomposition.

Parameters	125M	1.3B	2.7B	6.7B	13B
32-bit Float	25.65	15.91	14.43	13.30	12.45
Int8 absmax	87.76	16.55	15.11	14.59	19.08
Int8 zeropoint	56.66	16.24	14.76	13.49	13.94
Int8 absmax row-wise	30.93	17.08	15.24	14.13	16.49
Int8 absmax vector-wise	35.84	16.82	14.98	14.13	16.48
Int8 zeropoint vector-wise	25.72	15.94	14.36	13.38	13.47
Int8 absmax row-wise + decomposition	30.76	16.19	14.65	13.25	12.46
Absmax LLM.int8() (vector-wise + decomp)	25.83	15.93	14.44	**13.24**	**12.45**
Zeropoint LLM.int8() (vector-wise + decomp)	**25.69**	**15.92**	**14.43**	**13.24**	**12.45**

When we look at the scaling trends of zeroshot performance of OPT models on the EleutherAI language model evaluation harness in Figure 1, we see that LLM.int8() maintains full 16-bit performance as we scale from 125M to 175B parameters. On the other hand, the baseline, 8-bit absmax vector-wise quantization, scales poorly and degenerates into random performance.

Although our primary focus is on saving memory, we also measured the run time of LLM.int8(). The quantization overhead can slow inference for models with less than 6.7B parameters, as compared to a FP16 baseline. However, models of 6.7B parameters or less fit on most GPUs and quantization is less needed in practice. LLM.int8() run times is about two times faster for large matrix multiplications equivalent to those in 175B models. Appendix D provides more details on these experiments.

4 Emergent Large Magnitude Features in Transformers at Scale

As we scale transformers, outlier features with large magnitudes emerge and strongly affect *all* layers and their quantization. Given a hidden state $\mathbf{X} \in \mathbb{R}^{s \times h}$ where s is the sequence/token dimension and h the hidden/feature dimension, we define a feature to be a particular dimension h_i. **Our analysis looks at a particular feature dimension h_i across all layers of a given transformer.**

We find that outlier features strongly affect attention and the overall predictive performance of transformers. While up to 150k outliers exist per 2048 token sequence for a 13B model, these outlier features are highly systematic and only representing at most 7 unique feature dimensions h_i. Insights from this analysis were critical to developing mixed-precision decomposition. Our analysis explains the advantages of zeropoint quantization and why they disappear with the use of mixed-precision decomposition and the quantization performance of small vs. large models.

4.1 Finding Outlier Features

The difficulty with the quantitative analysis of emergent phenomena is

Table 4: Summary statistics of outliers with a magnitude of at least 6 that occur in at least 25% of all layers and at least 6% of all sequence dimensions. We can see that the lower the C4 validation perplexity, the more outliers are present. Outliers are usually one-sided, and their quartiles with maximum range show that the outlier magnitude is 3-20x larger than the largest magnitude of other feature dimensions, which usually have a range of [-3.5, 3.5]. With increasing scale, outliers become more and more common in all layers of the transformer, and they occur in almost all sequence dimensions. A phase transition occurs at 6.7B parameters when the same outlier occurs in all layers in the same feature dimension for about 75% of all sequence dimensions (SDim). Despite only making up about 0.1% of all features, the outliers are essential for large softmax probabilities. The mean top-1 softmax probability shrinks by about 20% if outliers are removed. Because the outliers have mostly asymmetric distributions across the sequence dimension s, these outlier dimensions disrupt symmetric absmax quantization and favor asymmetric zeropoint quantization. This explains the results in our validation perplexity analysis. These observations appear to be universal as they occur for models trained in different software frameworks (fairseq, OpenAI, Tensorflow-mesh), and they occur in different inference frameworks (fairseq, Hugging Face Transformers). These outliers also appear robust to slight variations of the transformer architecture (rotary embeddings, embedding norm, residual scaling, different initializations).

			Outliers		Frequency			Top-1 softmax p	
Model	PPL↓	Params	Count	1-sided	Layers	SDims	Quartiles	w/ Outlier	No Outlier
GPT2	33.5	117M	1	1	25%	6%	(-8, -7, -6)	45%	19%
GPT2	26.0	345M	2	1	29%	18%	(6, 7, 8)	45%	19%
FSEQ	25.7	125M	2	2	25%	22%	(-40, -23, -11)	32%	24%
GPT2	22.6	762M	2	0	31%	16%	(-9, -6, 9)	41%	18%
GPT2	21.0	1.5B	2	1	41%	35%	(-11, -9, -7)	41%	25%
FSEQ	15.9	1.3B	4	3	64%	47%	(-33, -21, -11)	39%	15%
FSEQ	14.4	2.7B	5	5	52%	18%	(-25, -16, -9)	45%	13%
GPT-J	13.8	6.0B	6	6	62%	28%	(-21, -17, -14)	55%	10%
FSEQ	13.3	6.7B	6	6	100%	75%	(-44, -40, -35)	35%	13%
FSEQ	12.5	13B	7	6	100%	73%	(-63, -58, -45)	37%	16%

C Detailed Outlier Feature Data

Table 4 provides tabulated data from our outlier feature analysis. We provide the quartiles of the most common outlier in each transformer and the number of outliers that are one-sided, that is, which have asymmetric distributions which do not cross zero.

D Inference Speedups and Slowdowns

D.1 Matrix Multiplication benchmarks

While our work focuses on memory efficiency to make models accessible, Int8 methods are also often used to accelerate inference. We find that the quantization and decomposition overhead is significant, and Int8 matrix multiplication itself only yields an advantage if the entire GPU is well saturated, which is only true for large matrix multiplication. This occurs only in LLMs with a model dimension of 4096 or larger.

Detailed benchmarks of raw matrix multiplication and quantization overheads are seen in Table 5. We see that raw Int8 matrix multiplication in cuBLASLt begins to be two times faster than cuBLAS at a model size of 5140 (hidden size 20560). If inputs need to be quantized and outputs dequantized – a strict requirement if not the entire transformer is done in Int8 – then the speedups compared to 16-bit is reduced to 1.6x at a model size of 5140. Models with model size 2560 or smaller are slowed down. Adding mixed precision decomposition slows inference further so that only the 13B and 175B models have speedups.

These numbers could be improved significantly with optimized CUDA kernels for the mixed precision decomposition. However, we also see that existing custom CUDA kernels are much faster than when we use default PyTorch and NVIDIA-provided kernels for quantization which slow down all matrix multiplications except for a 175B model.

Table 5: Inference speedups compared to 16-bit matrix multiplication for the first hidden layer in the feed-forward of differently sized GPT-3 transformers. The hidden dimension is 4x the model dimension. The 8-bit without overhead speedups assumes that no quantization or dequantization is performed. Numbers small than 1.0x represent slowdowns. Int8 matrix multiplication speeds up inference only for models with large model and hidden dimensions.

GPT-3 Size	Small	Medium	Large	XL	2.7B	6.7B	13B	175B
Model dimension	768	1024	1536	2048	2560	4096	5140	12288
FP16-bit baseline	1.00x	1.00x	1.00x	1.00x	1.00x	1.00x	1.00x	1.00x
Int8 without overhead	0.99x	1.08x	1.43x	1.61x	1.63x	1.67x	2.13x	2.29x
Absmax PyTorch+NVIDIA	0.25x	0.24x	0.36x	0.45x	0.53x	0.70x	0.96x	1.50x
Vector-wise PyTorch+NVIDIA	0.21x	0.22x	0.33x	0.41x	0.50x	0.65x	0.91x	1.50x
Vector-wise	**0.43x**	**0.49x**	**0.74x**	**0.91x**	**0.94x**	**1.18x**	**1.59x**	**2.00x**
LLM.int8() (vector-wise+decomp)	0.14x	0.20x	0.36x	0.51x	0.64x	0.86x	1.22x	1.81x

D.2 End-to-end benchmarks

Besides matrix multiplication benchmarks, we also test the end-to-end inference speed of BLOOM-176B in Hugging Face. Hugging Face uses an optimized implementation with cached attention values. Since this type of inference is distributed and, as such, communication dependent, we expect the overall speedup and slowdown due to Int8 inference to be smaller since a large part of the overall inference runtime is the fixed communication overhead.

We benchmark vs. 16-bit and try settings that use a larger batch size or fewer GPUs in the case of Int8 inference, since we can fit the larger model on fewer devices. We can see results for our benchmark in Table 6. Overall Int8 inference is slightly slower but close to the millisecond latency per token compared to 16-bit inference.

Table 6: Ablation study on the number of GPUs used to run several types of inferences of BLOOM-176B model. We compare the number of GPUs used by our quantized BLOOM-176B model together with the native BLOOM-176B model. We also report the *per-token* generation speed in milliseconds for different batch sizes. We use our method integrated into transformers(Wolf et al., 2019) powered by accelerate library from HuggingFace to deal with multi-GPU inference. Our method reaches a similar performance to the native model by fitting into fewer GPUs than the native model.

Batch Size	Hardware	1	8	32
bfloat16 baseline	8xA100 80GB	**239**	**32**	9.94
LLM.int8()	8xA100 80GB	253	34	10.44
LLM.int8()	4xA100 80GB	246	33	9.40
LLM.int8()	3xA100 80GB	247	33	**9.11**

E Training Results

We test Int8 training on a variety of training settings and compare to 32-bit baselines. We test separate settings for running the transformer with 8-bit feed-forward networks with and without 8-bit linear projections in the attention layer, as well at the attention iteself in 8-bit and compare against 32-bit performance. We test two tasks (1) language modeling on part of the RoBERTa corpus including Books (Zhu et al., 2015), CC-News (Nagel, 2016), OpenWebText (Gokaslan and Cohen, 2019), and CC-Stories (Trinh and Le, 2018); and (2) neural machine translation (NMT) (Ott et al., 2018) on WMT14+WMT16 (Macháček and Bojar, 2014; Sennrich et al., 2016).

The results are shown in Table 7 and Table 8. We can see that for training, using the attention linear projections with Int8 data types and vector-wise quantization leads to degradation for NMT and for 1.1B language model but not for 209M language modeling. The results improve slightly if mixed-precision decomposition is used but is not sufficient to recover full performance in most cases. These suggests that training with 8-bit FFN layers is straightforward while other layers require

additional techniques or different data types than Int8 to do 8-bit training at scale without performance degradation.

Table 7: Initial results on small and large-scale language modeling. Doing attention in 8-bit severely degrades performance and performance cannot fully recovered with mixed-precision decomposition. While small-scale language models is close to baseline performance for both 8-bit FFN and 8-bit linear projects in the attention layers performance degrades at the large scale.

	Is 8-bit				
Params	FFN	Linear	Attention	Decomp	PPL
209M				0%	16.74
209M	✓			0%	16.77
209M	✓	✓.		0%	16.83
209M	✓	✓		2%	16.78
209M	✓	✓		5%	16.77
209M	✓	✓		10%	16.80
209M	✓	✓	✓	2%	24.33
209M	✓	✓	✓	5%	20.00
209M	✓	✓	✓	10%	19.00
1.1B				0%	9.99
1.1B	✓			0%	9.93
1.1B	✓	✓		0%	10.52
1.1B	✓	✓		1%	10.41

F Fine-tuning Results

We also test 8-bit finetuning on RoBERTa-large finetuned on GLUE. We run two different setups: (1) we compare with other Int8 methods, and (2) we compare degradation of finetuning with 8-bit FFN layers as well as 8-bit attention projection layers comparel to 32-bit. We finetune with 5 random seeds and report median performance.

Table 9 compares with different previous 8-bit methods for finetuning and shows that vector-wise quantization improves on other methods. Table 10 shows the performance of FFN and/or linear attention projections in 8-bit as well as improvements if mixed-precision decomposition is used. We find that 8-bit FFN layers lead to no degradation while 8-bit attention linear projections lead to degradation if not combined with mixed-precision decomposition where at least the top 2% magnitude dimensions are computed in 16-bit instead of 8-bit. These results highlight the critical role of mixed-precision decomposition for finetuning if one wants to not degrade performance.

Table 8: Neural machine translation results for 8-bit FFN and linear attention layers for WMT14+16. Decomp indicates the percentage that is computed in 16-bit instead of 8-bit. The BLEU score is the median of three random seeds.

Is 8-bit			
FFN	Linear	Decomp	BLEU
		0%	28.9
✓		0%	28.8
✓	✓	0%	unstable
✓	✓	2%	28.0
✓	✓	5%	27.6
✓	✓	10%	27.5

Table 9: GLUE finetuning results for quantization methods for the feedforward layer in 8-bit while the rest is in 16-bit. No mixed-precision decomposition is used. We can see that vector-wise quantization improve upon the baselines.

Method	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	Mean
32-bit Baseline	90.4	94.9	92.2	84.5	96.4	90.1	67.4	93.0	88.61
32-bit Replication	90.3	94.8	92.3	85.4	96.6	90.4	68.8	92.0	88.83
Q-BERT (Shen et al., 2020)	87.8	93.0	90.6	84.7	94.8	88.2	65.1	91.1	86.91
Q8BERT (Zafrir et al., 2019)	85.6	93.0	90.1	84.8	94.7	89.7	65.0	91.1	86.75
PSQ (Chen et al., 2020)	89.9	94.5	92.0	**86.8**	96.2	90.4	67.5	91.9	88.65
Vector-wise	**90.2**	**94.7**	**92.3**	85.4	**96.4**	**91.0**	**68.6**	**91.9**	**88.81**

Table 10: Breakdown for 8-bit feedforward network (FFN) and linear attention layers for GLUE. Scores are median of 5 random seeds. Decomp indicates the percentage that is decomposed into 16-bit matrix multplication. Compared to inference, fine-tuning appears to need a higher decomp percentage if the linear attention layers are also converted to 8-bit.

Is 8-bit										
FFN	Linear	Decomp	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	MEAN
		0%	90.4	94.9	92.2	84.5	96.4	90.1	67.4	93.0	88.6
✓		0%	90.2	94.7	92.3	85.4	96.4	91.0	68.6	91.9	88.8
✓	✓	0%	90.2	94.4	92.2	84.1	96.2	89.7	63.6	91.6	87.7
✓	✓	1%	90.0	94.6	92.2	83.0	96.2	89.7	65.8	91.8	87.9
✓	✓	2%	90.0	94.5	92.2	85.9	96.7	90.4	68.0	91.9	88.7
✓	✓	3%	90.0	94.6	92.2	86.3	96.4	90.2	68.3	91.8	88.7