# 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.

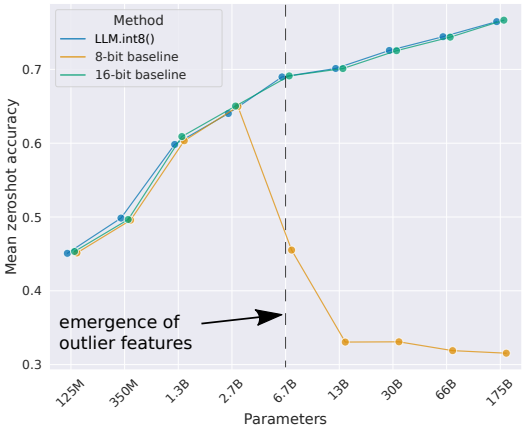



Figure 1: OPT model mean zeroshot accuracy for WinoGrande, HellaSwag, PIQA, and LAMBADA datasets. Shown is the 16-bit baseline, the most precise previous 8-bit quantization method as a baseline, and our new 8-bit quantization method, LLM.int8(). We can see once systematic outliers occur at a scale of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

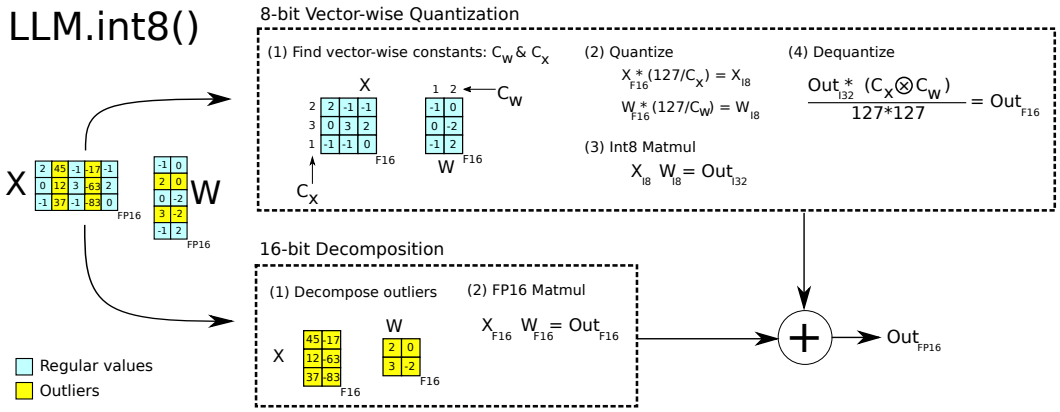



Figure 2: Schematic of LLM.int8(). Given 16-bit floating-point inputs $\mathbf{X}_{f16}$ and weights $\mathbf{W}_{f16}$, the features and weights are decomposed into sub-matrices of large magnitude features and other values. The outlier feature matrices are multiplied in 16-bit. All other values are 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*. For example, for ReLU outputs, in absmax quantization all values in $[-127, 0)$ go unused, whereas in zeropoint quantization the full $[-127, 127]$ range is used. Zeropoint quantization is given by the following equations:

$$nd_{x_{f16}} = \frac{2 \cdot 127}{\max_{ij}(\mathbf{X}_{f16}^{ij}) - \min_{ij}(\mathbf{X}_{f16}^{ij})} \tag{1}$$

$$zp_{x_{i16}} = \left\lfloor \mathbf{X}_{f16} \cdot \min_{ij}(\mathbf{X}_{f16}^{ij}) \right\rceil \tag{2}$$

$$\mathbf{X}_{i8} = \left\lfloor nd_{x_{f16}} \mathbf{X}_{f16} \right\rceil \tag{3}$$

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 ($\approx$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 $\alpha$. 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

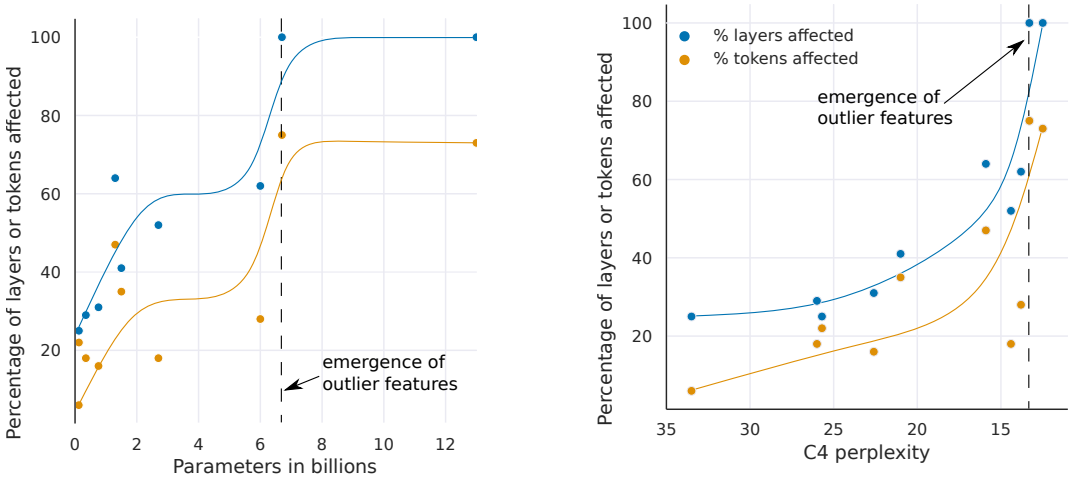
As we scale transformers, outlier features with large magnitudes emerge and strongly affect *all* layers and their quantization. Given a hidden state $\mathbf{X} \in \mathbb{R}^{s \times h}$ where $s$ is the sequence/token dimension and $h$ the hidden/feature dimension, we define a feature to be a particular 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 two-fold. We aim to select a small subset of features for analysis such that the results are intelligible and not to complex while also capturing important probabilistic and structured patterns. We use an empirical approach to find these constraints. We define outliers according to the following criteria: the magnitude of the feature is at least 6.0, affects at least 25% of layers, and affects at least 6% of the sequence dimensions.

More formally, given a transformer with $L$ layers and hidden state $\mathbf{X}_l \in \mathbb{R}^{s \times h}, l = 0...L$ where $s$ is the sequence dimension and $h$ the feature dimension, we define a feature to be a particular dimension $h_i$ in any of the hidden states $\mathbf{X}_{l_i}$. We track dimensions $h_i, 0 \leq i \leq h$, which have at least one value with a magnitude of $\alpha \geq 6$ and we only collect statistics if these outliers occur in the *same* feature dimension $h_i$ in at least 25% of transformer layers $0...L$ and appear in at least 6% of all sequence dimensions $s$ across all hidden states $\mathbf{X}_l$. Since feature outliers only occur in attention projection




(a) (b)

Figure 3: Percentage of layers and all sequence dimensions affected by large magnitude outlier features across the transformer by (a) model size or (b) C4 perplexity. Lines are B-spline interpolations of 4 and 9 linear segments for (a) and (b). Once the phase shift occurs, outliers are present in all layers and in about 75% of all sequence dimensions. While (a) suggest a sudden phase shift in parameter size, (b) suggests a gradual exponential phase shift as perplexity decreases. The stark shift in (a) co-occurs with the sudden degradation of performance in quantization methods.

(key/query/value/output) and the feedforward network expansion layer (first sub-layer), we ignore the attention function and the FFN contraction layer (second sub-layer) for this analysis.

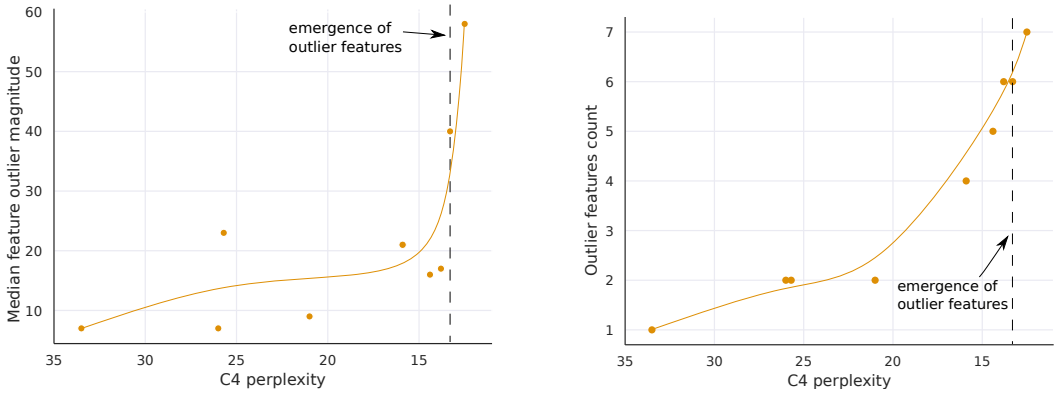
Our reasoning for these thresholds is as follows. We find that using mixed-precision decomposition, perplexity degradation stops if we treat any feature with a magnitude 6 or larger as an outlier feature. For the number of layers affected by outliers, we find that outlier features are *systematic* in large models: they either occur in most layers or not at all. On the other hand, they are *probabilistic* in small models: they occur *sometimes* in *some* layers for each sequence. As such, we set our threshold for how many layers need to be affected to detect an outlier feature in such a way as to limit detection to a *single* outlier in our smallest model with 125M parameters. This threshold corresponds to that at least 25% of transformer layers are affected by an outlier in the same feature dimension. The second most common outlier occurs in only a single layer ( 2% of layers), indicating that this is a reasonable threshold. We use the same procedure to find the threshold for how many sequence dimensions are affected by outlier features in our 125M model: outliers occur in at least 6% of sequence dimensions.

We test models up to a scale of 13B parameters. To make sure that the observed phenomena are not due to bugs in software, we evaluate transformers that were trained in three different software frameworks. We evaluate four GPT-2 models which use OpenAI software, five Meta AI models that use Fairseq (Ott et al., 2019), and one EleutherAI model GPT-J that uses Tensorflow-Mesh (Shazeer et al., 2018). More details can be found in Appendix C. We also perform our analysis in two different inference software frameworks: Fairseq and Hugging Face Transformers (Wolf et al., 2019).

### 4.2 Measuring the Effect of Outlier Features

To demonstrate that the outlier features are essential for attention and predictive performance, we set the outlier features to zero before feeding the hidden states $\mathbf{X}_l$ into the attention projection layers and then compare the top-1 softmax probability with the regular softmax probability with outliers. We do this for all layers independently, meaning we forward the regular softmax probabilities values to avoid cascading errors and isolate the effects due to the outlier features. We also report the perplexity degradation if we remove the outlier feature dimension (setting them to zero) and propagate these altered, hidden states through the transformer. As a control, we apply the same procedure for random non-outlier feature dimensions and note attention and perplexity degradation.

Our main quantitative results can be summarized as four main points.




(a) (b)

Figure 4: The median magnitude of the largest outlier feature in (a) indicates a sudden shift in outlier size. This appears to be the prime reason why quantization methods fail after emergence. While the number of outlier feature dimensions is only roughly proportional to model size, (b) shows that the number of outliers is *strictly monotonic* with respect to perplexity across all models analyzed. Lines are B-spline interpolations of 9 linear segments.

(1) When measured by the number of parameters, the emergence of large magnitude features across *all* layers of a transformer occurs suddenly between 6B and 6.7B parameters as shown in Figure 3a as the percentage of layers affected increases from 65% to 100%. The number of sequence dimensions affected increases rapidly from 35% to 75%. This sudden shift co-occurs with the point where quantization begins to fail.

(2) Alternatively, when measured by perplexity, the emergence of large magnitude features across all layers of the transformer can be seen as emerging smoothly according to an exponential function of decreasing perplexity, as seen in Figure 3b. This indicates that there is nothing sudden about emergence and that we might be able to detect emergent features before a phase shift occurs by studying exponential trends in smaller models. This also suggests that emergence is not only about model size but about perplexity, which is related to multiple additional factors such as the amount of training data used, and data quality (Hoffmann et al., 2022; Henighan et al., 2020).

(3) Median outlier feature magnitude rapidly increases once outlier features occur in all layers of the transformer, as shown in Figure 4a. The large magnitude of outliers features and their asymmetric distribution disrupts Int8 quantization precision. This is the core reason why quantization methods fail starting at the 6.7B scale – the range of the quantization distribution is too large so that most quantization bins are empty and small quantization values are quantized to zero, essentially extinguishing information. We hypothesize that besides Int8 inference, regular 16-bit floating point training becomes unstable due to outliers beyond the 6.7B scale – it is easy to exceed the maximum 16-bit value 65535 by chance if you multiply by vectors filled with values of magnitude 60.

(4) The number of outliers features increases strictly monotonically with respect to decreasing C4 perplexity as shown in Figure 4b, while a relationship with model size is non-monotonic. This indicates that model perplexity rather than mere model size determines the phase shift. We hypothesize that model size is only one important covariate among many that are required to reach emergence.

These outliers features are highly systematic after the phase shift occurred. For example, for a 6.7B transformer with a sequence length of 2048, we find about 150k outlier features per sequence for the entire transformer, but these features are concentrated in only 6 different hidden dimensions.

These outliers are critical for transformer performance. If the outliers are removed, the mean top-1 softmax probability is reduced from about 40% to about 20%, and validation perplexity increases by 600-1000% even though there are at most 7 outlier feature dimensions. When we remove 7 random feature dimensions instead, the top-1 probability decreases only between 0.02-0.3%, and perplexity increases by 0.1%. This highlights the critical nature of these feature dimensions. Quantization precision for these outlier features is paramount as even tiny errors greatly impact model performance.

### 4.3 Interpretation of Quantization Performance

Our analysis shows that outliers in particular feature dimensions are ubiquitous in large transformers, and these feature dimensions are critical for transformer performance. Since row-wise and vector-wise quantization scale each hidden state sequence dimension $s$ (rows) and because outliers occur in the feature dimension $h$ (columns), both methods cannot deal with these outliers effectively. This is why absmax quantization methods fail quickly after emergence.

However, almost all outliers have a strict asymmetric distribution: they are either solely positive or negative (see Appendix C). This makes zeropoint quantization particularly effective for these outliers, as zeropoint quantization is an asymmetric quantization method that scales these outliers into the full $[-127, 127]$ range. This explains the strong performance in our quantization scaling benchmark in Table 1. However, at the 13B scale, even zeropoint quantization fails due to accumulated quantization errors and the quick growth of outlier magnitudes, as seen in Figure 4a.

If we use our full LLM.int8() method with mixed-precision decomposition, the advantage of zeropoint quantization disappears indicating that the remaining decomposed features are symmetric. However, vector-wise still has an advantage over row-wise quantization, indicating that the enhanced quantization precision of the model weights is needed to retain full precision predictive performance.

## 5 Related work

There is closely related work on quantization data types and quantization of transformers, as described below. Appendix B provides further related work on quantization of convolutional networks.

**8-bit Data Types.** Our work studies quantization techniques surrounding the Int8 data type, since it is currently the only 8-bit data type supported by GPUs. Other common data types are fixed point or floating point 8-bit data types (FP8). These data types usually have a sign bit and different exponent and fraction bit combinations. For example, a common variant of this data type has 5 bits for the exponent and 2 bits for the fraction (Wang et al., 2018; Sun et al., 2019; Cambier et al., 2020; Mellempudi et al., 2019) and uses either no scaling constants or zeropoint scaling. These data types have large errors for large magnitude values since they have only 2 bits for the fraction but provide high accuracy for small magnitude values. Jin et al. (2022) provide an excellent analysis of when certain fixed point exponent/fraction bit widths are optimal for inputs with a particular standard deviation. We believe FP8 data types offer superior performance compared to the Int8 data type, but currently, neither GPUs nor TPUs support this data type.

**Outlier Features in Language Models.** Large magnitude outlier features in language models have been studied before (Timkey and van Schijndel, 2021; Bondarenko et al., 2021; Wei et al., 2022; Luo et al., 2021). Previous work proved the theoretical relationship between outlier appearance in transformers and how it relates to layer normalization and the token frequency distribution (Gao et al., 2019). Similarly, Kovaleva et al. (2021) attribute the appearance of outliers in BERT model family to LayerNorm, and Puccetti et al. (2022) show empirically that outlier emergence is related to the frequency of tokens in the training distribution. We extend this work further by showing how the scale of autoregressive models relates to the emergent properties of these outlier features, and showing how appropriately modeling outliers is critical to effective quantization.

**Multi-billion Scale Transformer Quantization.** There are two methods that were developed in parallel to ours: nuQmm (Park et al., 2022) and ZeroQuant (Yao et al., 2022). Both use the same quantization scheme: group-w2ise quantization, which has even finer quantization normalization constant granularity than vector-wise quantization. This scheme offers higher quantization precision but also requires custom CUDA kernels. Both nuQmm and ZeroQuant aim to accelerate inference and reduce the memory footprint while we focus on preserving predictive performance under an 8-bit memory footprint. The largest models that nuQmm and ZeroQuant evaluate are 2.7B and 20B parameter transformers, respectively. ZeroQuant achieves zero-degradation performance for 8-bit quantization of a 20B model. We show that our method allows for zero-degradation quantization of models up to 176B parameters. Both nuQmm and ZeroQuant suggest that finer quantization granularity can be an effective means to quantize large models. These methods are complementary with LLM.int8(). Another parallel work is GLM-130B which uses insights from our work to achieve zero-degradation 8-bit quantization (Zeng et al., 2022). GLM-130B performs full 16-bit precision matrix multiplication with 8-bit weight storage.

## 6 Discussion and Limitations

We have demonstrated for the first time that multi-billion parameter transformers can be quantized to Int8 and used immediately for inference without performance degradation. We achieve this by using our insights from analyzing emergent large magnitude features at scale to develop mixed-precision decomposition to isolate outlier features in a separate 16-bit matrix multiplication. In conjunction with vector-wise quantization that yields our method, LLM.int8(), which we show empirically can recover the full inference performance of models with up to 175B parameters.

The main limitation of our work is that our analysis is solely on the Int8 data type, and we do not study 8-bit floating-point (FP8) data types. Since current GPUs and TPUs do not support this data type, we believe this is best left for future work. However, we also believe many insights from Int8 data types will directly translate to FP8 data types. Another limitation is that we only study models with up to 175B parameters. While we quantize a 175B model to Int8 without performance degradation, additional emergent properties might disrupt our quantization methods at larger scales.

A third limitation is that we do not use Int8 multiplication for the attention function. Since our focus is on reducing the memory footprint and the attention function does not use any parameters, it was not strictly needed. However, an initial exploration of this problem indicated that a solution required additional quantization methods beyond those we developed here, and we leave this for future work.

A final limitation is that we focus on inference but do not study training or finetuning. We provide an initial analysis of Int8 finetuning and training at scale in Appendix E. Int8 training at scale requires complex trade-offs between quantization precision, training speed, and engineering complexity and represents a very difficult problem. We again leave this to future work.

Table 2: Different hardware setups and which methods can be run in 16-bit vs. 8-bit precision. We can see that our 8-bit method makes many models accessible that were not accessible before, in particular, OPT-175B/BLOOM.

| | | | Largest Model that can be run | |
|---|---|---|---|---|
| Class | Hardware | GPU Memory | 8-bit | 16-bit |
| Enterprise | 8x A100 | 80 GB | **OPT-175B / BLOOM** | **OPT-175B / BLOOM** |
| Enterprise | 8x A100 | 40 GB | **OPT-175B / BLOOM** | OPT-66B |
| Academic server | 8x RTX 3090 | 24 GB | **OPT-175B / BLOOM** | OPT-66B |
| Academic desktop | 4x RTX 3090 | 24 GB | **OPT-66B** | OPT-30B |
| Paid Cloud | Colab Pro | 15 GB | **OPT-13B** | GPT-J-6B |
| Free Cloud | Colab | 12 GB | **T0/T5-11B** | GPT-2 1.3B |

## 7 Broader Impacts

The main impact of our work is enabling access to large models that previously could not fit into GPU memory. This enables research and applications which were not possible before due to limited GPU memory, in particular for researchers with the least resources. See Table 3 for model/GPU combinations which are now accessible without performance degradation. However, our work also enables resource-rich organizations with many GPUs to serve more models on the same number of GPUs, which might increase the disparities between resource-rich and poor organizations.

In particular, we believe that the public release of large pretrained models, for example, the recent Open Pretrained Transformers (OPT) (Zhang et al., 2022), along with our new Int8 inference for zero- and few-shot prompting, will enable new research for academic institutions that was not possible before due to resource constraints. The widespread accessibility of such large-scale models will likely have both beneficial and detrimental effects on society that are difficult to predict.

**Acknowledgments** We thank Ofir Press, Gabriel Ilharco, Daniel Jiang, Mitchell Wortsman, Ari Holtzman, Mitchell Gordon for their feedback on drafts of this work. We thank JustHeuristic (Yozh) and Titus von Köller for help with Hugging Face Transformers integration.

## References


Artetxe, M., Bhosale, S., Goyal, N., Mihaylov, T., Ott, M., Shleifer, S., Lin, X. V., Du, J., Iyer, S., Pasunuru, R., et al. (2021). Efficient large scale language modeling with mixtures of experts. *arXiv preprint arXiv:2112.10684*.

Bai, H., Zhang, W., Hou, L., Shang, L., Jin, J., Jiang, X., Liu, Q., Lyu, M. R., and King, I. (2021). Binarybert: Pushing the limit of bert quantization. *ArXiv*, abs/2012.15701.

Bondarenko, Y., Nagel, M., and Blankevoort, T. (2021). Understanding and overcoming the challenges of efficient transformer quantization. *arXiv preprint arXiv:2109.12948*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Cambier, L., Bhiwandiwalla, A., Gong, T., Elibol, O. H., Nekuii, M., and Tang, H. (2020). Shifted and squeezed 8-bit floating point format for low-precision training of deep neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Chen, J., Gai, Y., Yao, Z., Mahoney, M. W., and Gonzalez, J. E. (2020). A statistical framework for low-bitwidth training of deep neural networks. *Advances in Neural Information Processing Systems*, 33:883–894.

Choi, J., Venkataramani, S., Srinivasan, V., Gopalakrishnan, K., Wang, Z., and Chuang, P. (2019). Accurate and efficient 2-bit quantized neural networks. In Talwalkar, A., Smith, V., and Zaharia, M., editors, *Proceedings of Machine Learning and Systems 2019, MLSys 2019, Stanford, CA, USA, March 31 - April 2, 2019*. mlsys.org.

Courbariaux, M. and Bengio, Y. (2016). Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830.

Courbariaux, M., Bengio, Y., and David, J. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3123–3131.

Courbariaux, M., Bengio, Y., and David, J.-P. (2014). Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*.

Dettmers, T., Lewis, M., Shleifer, S., and Zettlemoyer, L. (2022). 8-bit optimizers via block-wise quantization. *9th International Conference on Learning Representations, ICLR*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dong, Z., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. (2019). Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 293–302.

Esser, S. K., McKinstry, J. L., Bablani, D., Appuswamy, R., and Modha, D. S. (2019). Learned step size quantization. *arXiv preprint arXiv:1902.08153*.

Fan, A., Stock, P., Graham, B., Grave, E., Gribonval, R., Jegou, H., and Joulin, A. (2020). Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320*.

Gao, J., He, D., Tan, X., Qin, T., Wang, L., and Liu, T.-Y. (2019). Representation degeneration problem in training natural language generation models. *arXiv preprint arXiv:1907.12009*.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. (2021). A framework for few-shot language model evaluation.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. (2021). A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*.

Gokaslan, A. and Cohen, V. (2019). Openwebtext corpus. *urlhttp://Skylion007. github. io/OpenWebTextCorpus*.

Gong, R., Liu, X., Jiang, S., Li, T., Hu, P., Lin, J., Yu, F., and Yan, J. (2019). Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 4851–4860. IEEE.

Henighan, T., Kaplan, J., Katz, M., Chen, M., Hesse, C., Jackson, J., Jun, H., Brown, T. B., Dhariwal, P., Gray, S., et al. (2020). Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. (2022). Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Ilharco, G., Ilharco, C., Turc, I., Dettmers, T., Ferreira, F., and Lee, K. (2020). High performance natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Tutorial Abstracts*, pages 24–27, Online. Association for Computational Linguistics.

Jin, Q., Ren, J., Zhuang, R., Hanumante, S., Li, Z., Chen, Z., Wang, Y., Yang, K., and Tulyakov, S. (2022). F8net: Fixed-point 8-bit only multiplication for network quantization. *arXiv preprint arXiv:2202.05239*.

Khudia, D., Huang, J., Basu, P., Deng, S., Liu, H., Park, J., and Smelyanskiy, M. (2021). Fbgemm: Enabling high-performance low-precision deep learning inference. *arXiv preprint arXiv:2101.05615*.

Kovaleva, O., Kulshreshtha, S., Rogers, A., and Rumshisky, A. (2021). Bert busters: Outlier dimensions that disrupt transformers. *arXiv preprint arXiv:2105.06990*.

Li, R., Wang, Y., Liang, F., Qin, H., Yan, J., and Fan, R. (2019). Fully quantized network for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 2810–2819. Computer Vision Foundation / IEEE.

Lin, Y., Li, Y., Liu, T., Xiao, T., Liu, T., and Zhu, J. (2020). Towards fully 8-bit integer inference for the transformer model. *arXiv preprint arXiv:2009.08034*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Luo, Z., Kulmizev, A., and Mao, X. (2021). Positional artefacts propagate through masked language model embeddings. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5312–5327, Online. Association for Computational Linguistics.

Macháček, M. and Bojar, O. (2014). Results of the wmt14 metrics shared task. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 293–301.

Mellempudi, N., Srinivasan, S., Das, D., and Kaul, B. (2019). Mixed precision training with 8-bit floating point. *CoRR*, abs/1905.12334.

Nagel, S. (2016). Cc-news.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. (2019). fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*.

Ott, M., Edunov, S., Grangier, D., and Auli, M. (2018). Scaling neural machine translation. *arXiv preprint arXiv:1806.00187*.

Park, G., Park, B., Kwon, S. J., Kim, B., Lee, Y., and Lee, D. (2022). nuqmm: Quantized matmul for efficient inference of large-scale generative language models. *arXiv preprint arXiv:2206.09557*.

Puccetti, G., Rogers, A., Drozd, A., and Dell'Orletta, F. (2022). Outliers dimensions that disrupt transformers are driven by frequency. *arXiv preprint arXiv:2205.11380*.

Qin, H., Gong, R., Liu, X., Bai, X., Song, J., and Sebe, N. (2020). Binary neural networks: A survey. *CoRR*, abs/2004.03333.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). Xnor-net: Imagenet classification using binary convolutional neural networks. In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 525–542. Springer.

Sennrich, R., Haddow, B., and Birch, A. (2016). Edinburgh neural machine translation systems for wmt 16. *arXiv preprint arXiv:1606.02891*.

Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., et al. (2018). Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems*, 31.

Shen, S., Dong, Z., Ye, J., Ma, L., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. (2020). Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821.

Sun, X., Choi, J., Chen, C., Wang, N., Venkataramani, S., Srinivasan, V., Cui, X., Zhang, W., and Gopalakrishnan, K. (2019). Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4901–4910.

Timkey, W. and van Schijndel, M. (2021). All bark and no bite: Rogue dimensions in transformer language models obscure representational quality. *arXiv preprint arXiv:2109.04404*.

Trinh, T. H. and Le, Q. V. (2018). A simple method for commonsense reasoning. *arXiv preprint arXiv:1806.02847*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Wang, N., Choi, J., Brand, D., Chen, C., and Gopalakrishnan, K. (2018). Training deep neural networks with 8-bit floating point numbers. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7686–7695.

Wei, X., Zhang, Y., Zhang, X., Gong, R., Zhang, S., Zhang, Q., Yu, F., and Liu, X. (2022). Outlier suppression: Pushing the limit of low-bit transformer language models. *arXiv preprint arXiv:2209.13325*.

Wenzek, G., Lachaux, M.-A., Conneau, A., Chaudhary, V., Guzmán, F., Joulin, A., and Grave, E. (2020). CCNet: Extracting high quality monolingual datasets from web crawl data. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 4003–4012, Marseille, France. European Language Resources Association.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2019). Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Wu, H., Judd, P., Zhang, X., Isaev, M., and Micikevicius, P. (2020). Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*.

Yao, Z., Aminabadi, R. Y., Zhang, M., Wu, X., Li, C., and He, Y. (2022). Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *arXiv preprint arXiv:2206.01861*.

Yao, Z., Dong, Z., Zheng, Z., Gholami, A., Yu, J., Tan, E., Wang, L., Huang, Q., Wang, Y., Mahoney, M., et al. (2021). Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*, pages 11875–11886. PMLR.

Zafrir, O., Boudoukh, G., Izsak, P., and Wasserblat, M. (2019). Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 36–39. IEEE.

Zeng, A., Liu, X., Du, Z., Wang, Z., Lai, H., Ding, M., Yang, Z., Xu, Y., Zheng, W., Xia, X., et al. (2022). Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*.

Zhang, D., Yang, J., Ye, D., and Hua, G. (2018). Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 365–382.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. (2022). Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Zhang, W., Hou, L., Yin, Y., Shang, L., Chen, X., Jiang, X., and Liu, Q. (2020). Ternarybert: Distillation-aware ultra-low bit bert. In *EMNLP*.

Zhao, C., Hua, T., Shen, Y., Lou, Q., and Jin, H. (2021). Automatic mixed-precision quantization search of bert. *arXiv preprint arXiv:2112.14938*.

Zhu, C., Han, S., Mao, H., and Dally, W. J. (2017). Trained ternary quantization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.

## Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...

(a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
(b) Did you describe the limitations of your work? [Yes] See the limitation section
(c) Did you discuss any potential negative societal impacts of your work?[Yes] See the Broader Impacts section
(d) Have you read the ethics review guidelines and ensured that your paper conforms to them?[Yes] Yes, we believe our work conforms to these guidelines.

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We will include our code in the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)?[Yes] See the experimental setup section
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Our experiments are deterministic for each model. Instead of running the same model multiple times, we run multiple models at different scales. We are unable to compute error bars for these experiments.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See the exper2imental setup section
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See experimental setup section
   (b) Did you mention the license of the assets? [No] The license is permissible for all the assets that we use. The individual licenses can easily be looked up.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A] We only use existing datasets.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

# A Memory usage compared to 16-bit precision

Table 3 compares the memory footprint of 16-bit inference and LLM.int8() for different open source models. We can see, that LLM.int8() allows to run the largest open source models OPT-175B and BLOOM-176B on a single node equipped with consumer-grade GPUs.

Table 3: Different hardware setups and which methods can be run in 16-bit vs. 8-bit precision. We can see that our 8-bit method makes many models accessible that were not accessible before, in particular, OPT-175B/BLOOM.

| | | | Largest Model that can be run | |
|---|---|---|---|---|
| Class | Hardware | GPU Memory | 8-bit | 16-bit |
| Enterprise | 8x A100 | 80 GB | **OPT-175B / BLOOM** | **OPT-175B / BLOOM** |
| Enterprise | 8x A100 | 40 GB | **OPT-175B / BLOOM** | OPT-66B |
| Academic server | 8x RTX 3090 | 24 GB | **OPT-175B / BLOOM** | OPT-66B |
| Academic desktop | 4x RTX 3090 | 24 GB | **OPT-66B** | OPT-30B |
| Paid Cloud | Colab Pro | 15 GB | **OPT-13B** | GPT-J-6B |
| Free Cloud | Colab | 12 GB | **T0/T5-11B** | GPT-2 1.3B |

# B Additional Related Work

**Quantization of Transformers with fewer than 1B Parameters** Quantization of transformers has been focused on sub-billion parameter masked language model (MLMs), including BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019). Versions of 8-bit BERT/RoBERTa include Q8BERT (Zafrir et al., 2019), QBERT (Shen et al., 2020), product quantization with quantization noise (Fan et al., 2020), TernaryBERT (Zhang et al., 2020), and BinaryBERT (Bai et al., 2021). Work by Zhao et al. (2021) performs both quantization and pruning. All these models require either quantization-aware finetuning or post-training quantization to make the model usable in low-precision. In contrast with our methods, the model can be used directly without performance degradation.

If one views matrix multiplication as 1x1 convolution, vector-wise quantization is equivalent to channel-wise quantization for convolution combined with row quantization (Khudia et al., 2021). For matrix multiplication, this was used by Wu et al. (2020) for BERT-sized transformers (350M parameters), while we are the first to study vector-wise quantization for autoregressive and large-scale models. The only other work that we are aware of that quantizes transformers other than BERT is Chen et al. (2020), which uses post-training quantization with zeropoint quantization in the forward pass and zeropoint-row-wise quantization in the backward pass. However, this work is still for sub-billion parameter transformers. We compare with both zeropoint and row-wise quantization in our evaluations and do not require post-training quantization.

**Low-bitwidth and Convolutional Network Quantization** Work that uses less than 8-bits for data types is usually for convolutional networks (CNNs) to reduce their memory footprint and increase inference speed for mobile devices while minimizing model degradation. Methods for different bit-widths have been studied: 1-bit methods (Courbariaux and Bengio, 2016; Rastegari et al., 2016; Courbariaux et al., 2015), 2 to 3-bit (Zhu et al., 2017; Choi et al., 2019), 4-bits (Li et al., 2019), more bits (Courbariaux et al., 2014), or a variable amount of bits (Gong et al., 2019). For additional related work, please see the survey of Qin et al. (2020). While we believe that lower than 8-bit width with some performance degradation is possible for billion-scale transformers, we focus on 8-bit transformers that *do not* degrade performance and that can benefit from commonly used GPUs that accelerates inference through Int8 tensor cores.

Another line of work that focuses on convolutional network quantization is to learn adjustments to the quantization procedure to improve quantization errors. For example, using Hessian information (Dong et al., 2019), step-size quantization (Esser et al., 2019), soft quantization (Gong et al., 2019), mixed-precision via linear programming optimization (Yao et al., 2021), and other learned quantization methods (Zhang et al., 2018; Gholami et al., 2021).

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<br>Model dimension | Small<br>768 | Medium<br>1024 | Large<br>1536 | XL<br>2048 | 2.7B<br>2560 | 6.7B<br>4096 | 13B<br>5140 | 175B<br>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 |