GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale

Anonymous Author(s)
Affiliation
Address
email

Abstract

Large language models have been widely adopted but require significant GPU memory for inference and finetuning. We develop methods for Int8 matrix multiplication for transformer multi-layer perceptron (MLP) and attention projection layers, which cut the required memory for inference by half while retaining full precision performance. With our method, a 16/32-bit checkpoint can be loaded, converted to Int8, and used immediately without performance degradation – no post-quantization training is required. The key challenge, which we empirically show for the first time, is that existing quantization methods perform poorly at scale due to emergent outlier feature dimensions. We find that standard quantization techniques for matrix multiplication fail beyond 1.3B parameters. To overcome this barrier, we develop vector-wise quantization, which keeps separate normalization constants for each inner product in the matrix multiplication. Additionally, we identify layer and input invariant feature dimensions in the hidden states, which heavily influence attention and disrupt quantization methods starting at 13B parameters. To scale to 13B, we develop a new mixed-precision matrix decomposition scheme, which allows scaling without performance degradation to at least 13B parameters. This result makes large transformers more accessible, for example, by enabling inference with GPT-J and T5-11B on a single free cloud GPU, GPT-NeoX-20B on a single gaming-grade GPU, and OPT-30B on a single data-center-grade GPU. We open source our software.

Large pretrained language models are widely adopted in NLP (Vaswani et al., 2017; Radford et al., 2019; Zhang et al., 2022) but require significant memory for inference and finetuning. To improve accessibility, 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 significantly reduce the memory footprint, they can also degrade performance, usually require post-quantization training, and have only been studied for small models with less than 350M parameters. Currently, no 8-bit quantization methods exist that make multi-billion parameter transformer models more accessible on common accelerators such as GPUs and TPUs.

For such large-scale transformers, the multi-layer perceptron and attention projection layers make up more than 95% of weights; nearly the entire model is in these large matrices, which must be multiplied during inference. GPUs support Int8 tensor cores, which can accelerate these multiplications while halving the memory compared to 16-bit representations. However, these gains can be challenging to achieve in practice since Int8 data types provide poor quantization performance for hidden states and weights, which are usually normally distributed (Dettmers, 2016). As such, we require new high-precision quantization techniques to avoid performance degradation.

In this paper, we present the first multi-billion-scale Int8 quantization methods for transformers that do not incur any performance degradation. Our methods make it possible to load a 13B parameter transformer with 16/32-bit weights, convert the multi-layer perceptron and attention projection layers

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

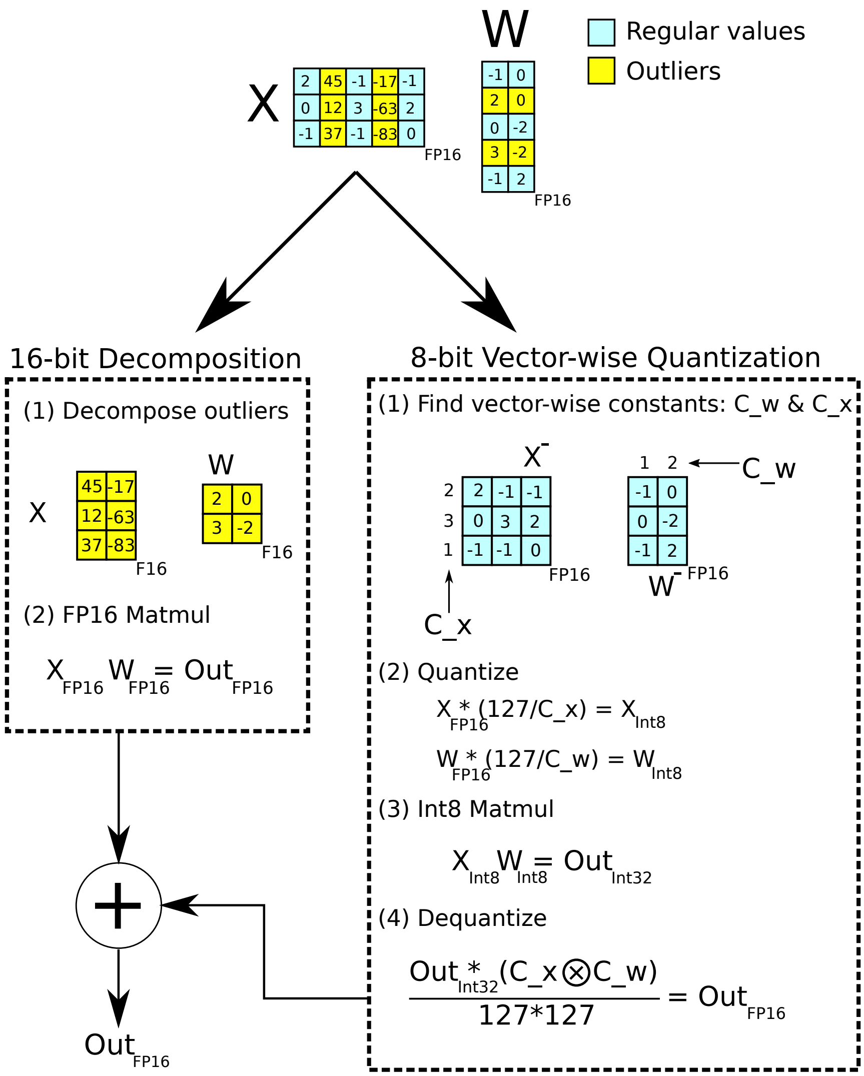

Figure 1: Schematic of Vector-wise 8-bit matrix multiplication with mixed precision decomposition. Given 16-bit floating-point inputs $\mathbf{X}_{f16}$and weights $\mathbf{W}_{f16}$, the outliers are decomposed first sub-matrices of outliers with the corresponding weights which are matrix multiplied in a 16-bit. All other values are matrix multiplied in 8-bit. The 8-bit multiplication is done by finding the row and column-wise absolute maximum of $\mathbf{C}_x$ and $\mathbf{C}_w$. Then matrix multiplication is performed in 8-bit. Afterwards the Int32 outputs are dequantization by the outer product of the normalization constants $\mathbf{C}_{\otimes}\mathbf{C}_w$. Finally, both results are accumulated in 16-bit floating point outputs.

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 moderate scales and the need to explicitly represent the sparse but systematic outliers that destroy quantization when they emerge at scales of 6.7B parameters and beyond.

We develop the high precision quantization method vector-wise quantization to retain performance at moderate scales. Consider two matrices $\mathbf{X} \in \mathbb{R}^{s \times h}$ and $\mathbf{W} \in \mathbb{R}^{h \times o}$ (e.g. within a Transformer). Although they are typically stored in 16-bit floating representations, $\mathbf{X}_{f16}$ and $\mathbf{W}_{f16}$, we aim to convert them to Int8 to save memory on GPUs and to perform fast Int8 matrix multiplication with Int32 accumulation $\mathbf{X}_{i8}\mathbf{W}_{i8} = \mathbf{O}_{i32}$. To recover the 16-bit floating-point representation for the next layer, we convert the Int32 output, $\mathbf{O}_{i32}$ back to $\mathbf{O}_{f16}$ before we perform the next operation.

Unlike other quantization techniques that use a single quantization normalization constant for the weight matrix $\mathbf{W}_{f16}$, vector-wise quantization keeps a quantization normalization constant $c_{x_{f16}}$

for each row of $\mathbf{X}_{f16}$ and $c_{w_{f16}}$ each column of $\mathbf{W}_{f16}$. Dequantization to 16/32-bit floats is done by denormalization through the outer product of these constants vectors $\mathbf{O}_{f16} = \frac{\mathbf{O}_{i32}}{\mathbf{c}_{X_{f16}} \otimes \mathbf{c}_{W_{f16}}}$ which can be implemented without additional overhead through operator fusion. In conjuction with zeropoint quantization, vector-wise quantization allows near lossless inference of transformer models with up to 6.7B parameters.

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 the first descriptive analysis of emergence at scale as observed in feature space. We show that outliers with magnitudes up to 20x larger than the mean maximum feature magnitude first emerge in attention projections inputs and then spread to other layers as we scale transformers to 6B parameters. At 6.7B scale and beyond, these outliers occur in almost all other layers and 75% of all sequence dimensions in the same feature dimension across the entire transformer. They are also critical for effective attention, as they decrease top-1 attention softmax probability mass by more than 20% despite only making up about 0.1% of all input features.

To handle these outliers, we develop mixed-precision matrix decomposition where we perform 16-bit matrix multiplication for outlier dimensions and 8-bit matrix multiplication for other dimensions. We show that we can use 13B parameter 8-bit transformers without any performance degradation by combining mixed-precision matrix decomposition and vector-wise quantization. We open source our software.

1 Background

1.1 8-bit Data Types and Quantization

8-bit Transformers at Scale: Where is the memory and compute? Work on 8-bit transformers at scale requires a slightly different perspective than previous quantization work that focused on convolutional networks (CNNs) or sub-billion parameter transformers. Rather than focusing on mobile devices and integer-only accelerators, the main goal of developing 8-bit transformers at scale is to make large transformers accessible so that fewer or cheaper accelerators, such as GPUs, are required to run them. These goals can take two forms: (1) reduce the memory footprint of such models and (2) reduce the runtime of these models on GPUs. In this work, we focus on reducing the memory footprint, although we hope our methods will be helpful for creating custom kernels in the future that also achieve significant runtime gains.

For the case of inference, almost the entire memory footprint of inference comes from transformer MLP and attention projection layers. For example, for a 13B parameter model, more than 97% of parameters come from these layers. These layers also make up most of the compute with MLP layers using 40-60%, and attention projections about 25% of runtime, with the rest used by attention (Ilharco et al., 2020). For this reason, we focus solely on efficient large matrix multiplication, the key computation for the MLP, and attention projection layers. While we do not focus on accelerated inference, we provide more details in the Appendix, where we do show preliminary results where our method accelerates inference for large models due to the overhead of multiplying huge matrices within the models. Future work could focus on achieving such gains for models of all sizes.

Absolute maximum vs. zeropoint quantization To quantize floating-point inputs into the Int8 range, we need to scale the inputs in the range $[-128, 127]$. The range $[-127, 127]$ is usually chosen for practical purposes instead. The most common way to scale the inputs into this range is to perform absolute maximum (absmax) or zeropoint quantization, as defined in the following two paragraphs. For symmetric distributions, like the normal distribution, absmax and zeropoint quantization have the same quantization precision. However, zeropoint quantization is superior to absmax quantization in the case of asymmetric distributions such as tensor outputs from ReLU non-linearities because it scales any input distribution to the full $[-127, 127]$ range. The downside of zeropoint quantization is that it needs a special instruction that combines 8-bit multiplication with 16-bit addition of the zeropoint offset to run efficiently.[1] This can make zeropoint quantization slow on devices such as CPUs, TPUs, and GPUs that do not support this instruction (Jacob et al., 2017).

[1] https://www.felixcloutier.com/x86/pmaddubsw

Absmax quantization scales inputs into the range $[-127, 127]$ by dividing by the absolute maximum of the entire tensor and multiplying by 127. This is equivalent of the diving 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:

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

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

Zeropoint quantization shifts the 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. It is an essential detail that for zeropoint quantization, the zeropoint falls onto an exact integer to represent padding constants and other 0-valued entries such as ReLU outputs with full precision. Zeropoint quantization is given by the following equations:

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

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

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

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 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 special 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 $nd_{b_{f16}}$.

1.2 Int8 Matrix Multiplication with 16-bit Float Inputs and Outputs

Given hidden states $\mathbf{X}_{f16} \in \mathbb{R}^{s \times h}$ and weight matrix $\mathbf{W}_{f16} \in \mathbb{R}^{h \times o}$ with sequence dimension s, hidden 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 scaling constants s_x and s_w for absmax or nd_x ande nd_w for zeropoint quantization.

2 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 develop vector-wise quantization to improve upon row-wise quantization(Khudia et al., 2021), as described in more detail below.

Furthermore, if we have an outlier in each row of the input matrix or each column in the weight matrix, additional quantization techniques are required for high-precision quantization. For this purpose, we develop mixed-precision matrix decomposition, where a small number of entries ($\approx$0.1%) can be represented in higher precision and computed efficiently. However, most entries are still represented in low-precision, thus retaining roughly 50% memory reductions compared to 16-bit representations.

Both methods, vector-wise quantization and mixed-precision matrix decomposition, are depicted in Figure 1.

2.1 Vector-wise Quantization

To maximize the number of scaling constants for matrix multiplication, we want to define blocks of constants in a way such that we can dequantize the matrix multiplication output by simple scaling by a tensor without additional operations to recover the dequantized matrix multiplication output, for example, as is required for zeropoint quantization if 16-bit multiplication with 8-bit inputs is unavailable.

One way to achieve this 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.

2.2 Mixed-precision Matrix Decomposition

In our analysis, we demonstrate that a significant problem for billion-scale 8-bit transformers is that they have outliers 20x larger than the average outlier in almost every row (sequence dimension), which makes even our best quantization technique – vector-wise quantization – ineffective. At the same time, transformer performance degrades significantly even for small errors of these large outliers. As such, we require high precision matrix multiplication for these outliers. Luckily, we see that these outliers are incredibly sparse in practice, allowing us to develop new matrix decomposition techniques.

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 hidden dimensions h. As such, we propose *mixed-precision matrix decomposition* for matrix multiplication where we separate outlier hidden 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 use $\alpha = 6.0$. Using Einstein notation where all indices are superscripts, given the weight matrix $\mathbf{W}_{f16} \in \mathbb{R}^{h \times o}$, mixed-precision matrix decomposition for matrix multiplication is defined as follows:

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

where $\mathbf{S}_{f16}$ is the denormalization term for the Int8 quantized inputs and weight matrices $\mathbf{X}_{i8}$ and $\mathbf{W}_{i8}$. Since $|O| \leq 7$ for transformers up to 13B parameters, this decomposition operation only consumes about 0.1% additional memory.

2.3 Experimental Setup

We measure the robustness of quantization methods as we scale the size of several publicly available pretrained language models up to 13B 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 the model size.

As such, 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.,

Table 1: C4 validation perplexities of different quantization methods for different transformer sizes ranging from 125M to 13B parameters. We see that common absmax, absmax row-wise, zeropoint, as well as vector-wise quantization methods lead to significant performance degradation, particularly at the 13B mark where 8-bit 13B perplexity is worse than 8-bit 6.7B perplexity. On the other hand, vector-wise quantization with mixed-precision decomposition is able to recover almost baseline perplexity for all model sizes. Looking at the trends, we see that as we scale the model size, the quantization error improves if we use vector-wise quantization in conjunction with mixed-precision matrix decomposition while all other methods degrade in performance. Zeropoint quantization is no longer advantageous when used with mixed-precision matrix 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 Absmax Row-wise	30.93	17.08	15.24	14.13	16.49
Int8 Zeropoint	56.66	16.24	14.76	13.49	13.94
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
Int8 Absmax Vector-wise + decomposition	25.83	15.93	14.44	**13.24**	**12.45**
Int8 Zeropoint Vector-wise + decomposition	**25.69**	**15.92**	**14.43**	**13.24**	**12.45**

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 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.[2] We use NVIDIA A40 GPUs for this evaluation.

3 Main Results

The main results on the 125M to 13B Int8 models evaluated on the C4 corpus can be seen in Table 1. Existing quantization techniques such as absmax, row-wise, and zeropoint quantization see considerable performance degradation that grows with scale. Using these methods with the 13B parameter model is worse than the 6.7B model. While vector-wise quantization improves the scaling trend, 13B performance is still worse than 6.7B. If we add mixed-precision decomposition, we can recover the full-precision performance for the larger models. Zeropoint quantization has almost no advantage over absmax quantization when used with mixed-precision decomposition. However, since zeropoint quantization can be slow on devices that do not support the required instructions, absmax quantization has an inference speed advantage at similar predictive performance when used with mixed-precision matrix decomposition.

4 Analysis: Emergent Outliers in Transformers at Scale

We developed mixed-precision matrix decomposition after gaining insights into how extreme outliers emerge in particular feature dimensions in the hidden states of transformers as we scale. These insights were critical to developing a simple method with a low computational overhead that preserves predictive performance. The data from the following outlier analysis explains the performance degradation of previous quantization methods, which we saw empirically in our experiments.

Quantitative Setup 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 or sequence length and h the hidden dimension, we define a feature to be a particular dimension h in any of the hidden states $\mathbf{X}_l$. Since a transformer has thousands of feature dimensions and dozens of layers, and each feature is activated differently for different inputs,

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

Table 2: Summary statistics of outliers with a magnitude of at least 6 that occur in at least 20% of all layers and at least 5% 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%

aggregation of all descriptive feature statistics would be incomprehensible. As such, we limit our analysis to a subset of these features.

We limit the number of features by only aggregating the statistics of features that breach certain thresholds – this makes these features accessible to analysis. To set these thresholds, we first observe rough general patterns.

The main patterns that we find are as follows: As we increase the scale of our models, we find that outliers in the feature dimension h has increasingly large magnitudes, which occur in more layers l and in more sequence dimensions s as we scale. On the other hand, outliers do not occur in the second MLP layer and are scattered in complex patterns in the attention inputs, which are difficult to analyze. As such, we limit our analysis to the hidden input states $\mathbf{X}_l$ of the first MLP layer and the attention key-query-value and output projection layers.

From these observations, we set the following thresholds for features: We track dimensions $h_i, 0 \leq i \leq h$, which have at least one outlier with a magnitude of $\alpha \geq 6$ and we only collect statistics if these outliers occur in the same hidden dimension h_i in at least 20% of transformer layers $0...L$ and appear in at least 5% of all sequence dimensions s across all hidden states $\mathbf{X}_l$.

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: GPT-2 models use OpenAI software, FSEQ models use fairseq(Ott et al., 2019), and GPT-J uses Tensorflow-Mesh (Shazeer et al., 2018). We also perform our analysis in two different inference software frameworks: fairseq (Ott et al., 2019) and Hugging Face Transformers (Wolf et al., 2019).

To demonstrate that the outlier features are essential for attention, we remove the outliers before feeding the hidden states $\mathbf{X}_l$ into the attention projection layers and then compare the largest softmax probability for each sequence dimension with the softmax probability if we do not remove the outliers.

Quantitative Results Our quantitative results are summarized in Table 2. We can see that the number of hidden state outlier feature dimensions in h_i increases from 1 in 125M parameter transformers to 7 in 13B models – roughly proportional to the C4 validation perplexity. We also see that the frequency that the same outlier dimension h_i is active in all hidden states across layers $\mathbf{X}_l^i$ increases from 25% for 125M models to 100% in 6.7B/13B models. This means, the same outlier feature dimension h_i, say, dimension $i = 888$ for $h = 1024$, has large outliers in all hidden states $\mathbf{X}_l^i$ of the key-query-value, attention output and first MLP projection layers. For all data in C4 validation data, 6% of all sequence dimensions s have outliers in the same feature dimension h_i for the 125M parameter model, which increases to 73-75% of all sequence dimensions for the 6.7B/13B models. This means at the 6.7B scale, and beyond, if we have, say, a sequence length of $s = 2048$ and 10 transformer blocks, we have roughly $2048 \times 10 \times 0.75 = 15360$ outliers per sequence for the entire model for each of the attention key-query-value projection, first MLP and attention output projection layers, so in total $L \times 15360 = 3 \times 10 \times 15360 = 46080$ outliers per sequence on average across the C4 validation set.

If the outliers are removed, the mean top-1 softmax probability across all sequences is reduced from about 45% to about 15% even though the outlier dimensions h_i make up only 0.1% of all input feature dimensions h. This indicates that even minor quantization errors of the dimensions h_i might significantly affect overall model performance.

Interpretation of Quantization Performance Our analysis shows that these outliers are ubiquitous, and one can expect that quantization methods that cannot handle these large magnitude outliers well will significantly degrade model performance as they accumulate large errors throughout the network. Furthermore, since these outliers occur in each sequence dimension for large models, row-wise and vector-wise quantization do not have a significant advantage over absmax quantization since these methods have a normalization constant for each sequence s_i for each hidden state $\mathbf{X}_l$ 75% of which have outliers in large models.

From the quartiles, we can also see that most outliers have one-sided asymmetric distributions, meaning they do not cross zero and have either solely positive or negative values. This makes zeropoint quantization particularly effective for these outliers as zeropoint quantization 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.

If we perform mixed-precision matrix decomposition, the advantage of zeropoint quantization disappears for large models indicating that the remaining decomposed features are essentially symmetric for these models. However, vector-wise quantization 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

We can separate the related work into general low-bitwidth quantization methods, 8-bit methods for CNNs, and transformers.

Low-bitwidth and Convolutional Network Quantization While our work studies quantization techniques surrounding the Int8 data type, 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) provides an excellent analysis of when certain fixed point exponent/fraction bit widths are optimal for inputs with a particular standard deviation. We believe the insights provided by their work are critical for developing robust FP8 transformers.

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 be accelerated through Int8 tensor cores.

Quantization of Transformers A prime target for quantization of transformers has been BERT(Devlin et al., 2018) and RoBERTa(Liu et al., 2019) models. 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). All these models require quantization-aware training to make the BERT model usable in low-precision, while with our methods, the model can be used directly without performance degradation.

The only other work that we are aware of that quantizes transformers other than BERT is Chen et al. (2020) which uses quantization-aware training with zeropoint quantization in the forward pass and zeropoint-row-wise quantization in the backward pass. We compare with both zeropoint and row-wise quantization in our evaluations and do not require quantization-aware training.

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 outliers in hidden state feature dimensions at scale to develop mixed-precision matrix decomposition to isolate outliers in a separate 16-bit matrix multiplication. Furthermore, to recover full precision performance, we develop vector-wise quantization.

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 won in our work for Int8 data types directly translate to FP8 data types.

Another limitation is that we only study models up to the size of 13B parameters. While we quantize a 13B 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 batched Int8 matrix multiplication for the attention layers. 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 fourth limitation is that we focus on inference but do not study training or finetuning. We provide an initial analysis of Int8 training at scale in the appendix. Int8 training requires complex trade-offs between quantization precision, training speed, and engineering complexity, which we again leave to future work.

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. Additionally, resource-rich organizations with many GPUs can now serve the same number of models on fewer GPUs.

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.

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.

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. (2016). 8-bit approximations for parallelism in deep learning. *International Conference on Learning Representations (ICLR)*.

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*.

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*.

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.

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.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. (2017). Quantization and training of neural networks for efficient integer-arithmetic-only inference. arxiv e-prints, art. *arXiv preprint arXiv:1712.05877*.

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*.

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*.

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*.

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.

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.

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.

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*.

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.

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*.

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]