IM-Unpack: Training and Inference with Arbitrarily Low Precision Integers

Zhanpeng Zeng [1] **Karthikeyan Sankaralingam** [1 2] **Vikas Singh** [1]

Abstract

GEneral Matrix Multiply (GEMM) is a central operation in deep learning and corresponds to a large chunk of the compute footprint. Therefore, improving its efficiency is an active topic of research. A popular strategy is the use of low bit-width integers to approximate the original matrix entries. This allows efficiency gains, but often requires sophisticated techniques to control the rounding error. In this work, we first verify that when the *low bit-width* restriction is removed, for a variety of Transformer-based models, integers are, in fact, sufficient for all GEMMs need – for *both* training and inference stages, and achieve parity (with floating point). No sophisticated techniques are needed. We find that while a large majority of entries in matrices (encountered in such models) can be easily represented by *low* bit-width integers, the existence of a few heavy hitter entries make it difficult to achieve efficiency gains via the exclusive use of low bit-width GEMMs alone. To address this issue, we develop a simple algorithm, Integer Matrix Unpacking (IM-Unpack), to *unpack* a matrix with large integer entries into a *larger* matrix whose entries all lie within the representable range of arbitrarily low bit-width integers. This allows *equivalence* with the original GEMM, i.e., the exact result can be obtained using purely low bit-width integer GEMMs. This comes at the cost of additional operations – we show that for many popular models, this overhead is quite small. Code is available at `https://github.com/vsingh-group/im-unpack`.

1. Introduction

Calculating the product of two matrices using GEneral Matrix Multiply (GEMM) is one of the most widely used operations in modern machine learning. Given matrices $\mathbf{A}$ and $\mathbf{B}$ of size $n \times d$ and $h \times d$ respectively, the output of a GEMM is calculated as

$$\mathbf{C} = \mathbf{A}\mathbf{B}^\top$$

Choosing the appropriate numerical precision or data type (FP32, FP16, or BF16) for GEMM is often important, and hinges on several factors including the specific application, characteristics of the data, model architecture, as well as numerical behavior such as convergence. This choice affects compute and memory efficiency most directly, since a disproportionately large chunk of the compute footprint of a model involves the GEMM operator. A good example is the large improvement in latency and memory achieved via low bit-width GEMM, and made possible due to extensive ongoing work on quantization (to low bit-width data types) and low-precision training (Banner et al., 2019; Nagel et al., 2019; Kim et al., 2021; Dettmers et al., 2022; Li & Gu, 2023; Xiao et al., 2023; Dettmers et al., 2022; Liu et al., 2023b;a; Lin et al., 2022; Li & Gu, 2023; Yuan et al., 2022; Ding et al., 2022; Li et al., 2023; Wang et al., 2018; Wu et al., 2018; Zhu et al., 2020; Wortsman et al., 2023). Integer quantization is being actively pursued for inference efficiency, and the use of *low bit-width* integers is universal to deliver the efficiency gains. However, this strategy often incurs large rounding errors when representing all matrix entries as low bit-width integers, and explains the drop in performance and thereby, a need for error correction techniques (Frantar et al., 2023; Xiao et al., 2023; Chee et al., 2023; Adepu et al., 2024). So how much of the performance degradation is due to (a) rounding to integers versus (b) restricting to low bit-width integers? To answer this question, it appears worthwhile to check whether integer GEMMs will achieve parity without sophisticated techniques (for the inference stage, and more aspirationally, for training) for popular models if we do *not* restrict to low bit-width integers.

Overview. The starting point of our work is to first experimentally verify that the aforementioned hypothesis – that integer GEMM may work – is true (see §2). But by itself, this finding offers no value proposition for efficiency. Nonetheless, this experiment is useful for the following reason. For a particular class of models (e.g., Transformers), we can easily contrast the corresponding input matrices $\mathbf{A}$ and $\mathbf{B}$ between (a) integer GEMM and (b) low bit-width integer GEMM and probe if any meaningful structure can

[1]University of Wisconsin–Madison [2]NVIDIA Research. Correspondence to: Zhanpeng Zeng <zzeng38@wisc.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

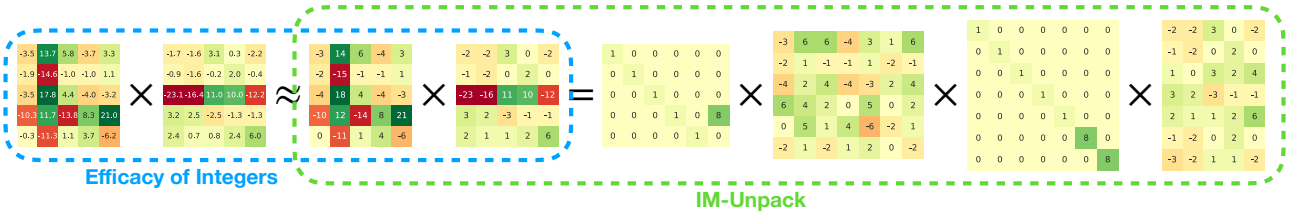

Figure 1. Overall Illustration. We verify the **Efficacy of Integers (Contribution 1)** in §2, but note that the integer matrices contain heavy hitters (§3). Then, we describe our proposed algorithm, **IM-Unpack (Contribution 2)**, to resolve these heavy hitters in §4.

be exploited. While there *is* a clear difference in the *outputs* of (a) integer GEMM versus (b) low bit-width integer GEMM, we find that *a large majority* of entries of $\mathbf{A}$ and $\mathbf{B}$ can be represented using low bit-width integers – and the difference in the outputs can be attributed to a few *heavy hitter* entries in $\mathbf{A}$ and $\mathbf{B}$, that cannot be represented using low bit-width integers. Other works have also run into this issue of "outliers" and use high precision (Dettmers et al., 2022) or a separate quantization for these entries (Yuan et al., 2022; Xiao et al., 2023). Based on the observation that we can represent a large integer by a series of smaller integers, our algorithm, Integer Matrix Unpack (IM-Unpack), enables unpacking any integer into a series of low-bit integers. The benefit is that the calculation can be carried out entirely using low bit-width integer arithmetic and thus unifies calculations needed for heavy hitters and the other entries (already amenable to low-bit integer arithmetic). Specifically, IM-Unpack unpacks an integer matrix such that all values of the unpacked matrices always stay within the representable range of low bit-width integers (bit-width can be chosen arbitrarily, as low as two). We obtain the exact result of the original integer GEMM using purely low bit-width integer GEMMs. Since the bit-width of integer arithmetic is independent of the actual range of the original matrices, the construction will simplify the hardware/compiler support by only needing support for *one* specific bit-width. The structure/contributions of this paper is shown in Fig. 1.

Notations. To simplify the presentation, we will narrow the scope of our discussion exclusively to Transformers. We first define notations for all relevant GEMMs. For the linear layer, let the input activation and parameter matrix be $\mathbf{X}$ and $\mathbf{W}$. Let the query, key, value matrices needed in self-attention be $\mathbf{Q}, \mathbf{K}, \mathbf{V}$. Below, we itemize all GEMMs:

$$\mathbf{Y} = \mathbf{X}\mathbf{W}^\top \quad \mathbf{P} = \mathbf{Q}\mathbf{K}^\top \quad \mathbf{O} = \mathbf{M}\mathbf{V}$$

where $\mathbf{M}$ is the attention score between $\mathbf{Q}$ and $\mathbf{K}$ defined as $\mathbf{M} = \text{softmax}(\mathbf{P})$ (omitting scaling factors). Now, given the gradient for $\mathbf{Y}, \mathbf{P}, \mathbf{O}$ denoted as $\nabla_\mathbf{Y}, \nabla_\mathbf{P}, \nabla_\mathbf{O}$, the other gradients are calculated via GEMMs as well:

$$\nabla_\mathbf{X} = \nabla_\mathbf{Y}\mathbf{W} \quad \nabla_\mathbf{Q} = \nabla_\mathbf{P}\mathbf{K} \quad \nabla_\mathbf{M} = \nabla_\mathbf{O}\mathbf{V}^\top$$
$$\nabla_\mathbf{W} = \nabla_\mathbf{Y}^\top\mathbf{X} \quad \nabla_\mathbf{K} = \nabla_\mathbf{P}^\top\mathbf{Q} \quad \nabla_\mathbf{V} = \mathbf{M}^\top\nabla_\mathbf{O}$$

These notations will help refer to each type of GEMM later.

2. Round to Nearest: What do we lose?

Let us start by using the simplest Rounding To Nearest (RTN) to map FP to integers, and check the extent to which integer GEMMs work satisfactorily for both training and inference, if we do *not* restrict to low bit-width integers. Specifically, for matrix $\mathbf{A}$, **all** entries of $\mathbf{A}$ are quantized via

$$\mathbf{A}_q = \text{round}(0.5\beta/\alpha_p(\mathbf{A})\mathbf{A}) \tag{1}$$

where $\alpha_p(\mathbf{A})$ gives the p-th percentile (see §A.1 for reasons behind using percentile) based on the magnitude of entries in $\mathbf{A}$, i.e., $p\%$ of entries in $\mathbf{A}$ fall in the interval $[-\alpha_p(\mathbf{A}), \alpha_p(\mathbf{A})]$. We only need $\alpha_p(\mathbf{A})$ as a meaningful estimate of the approximate range of values, and so we set $p = 95\%$ for all experiments except a few cases noted explicitly. The hyperparameter β is the number of distinct integers that we want to use to encode values that are within $[-\alpha_p(\mathbf{A}), \alpha_p(\mathbf{A})]$. Then, after quantization, the GEMM for the original matrices can be approximated (because we incur a rounding error) in the quantized domain using integer GEMMs. The approximated GEMM is computed using the quantized $\mathbf{A}$ and $\mathbf{B}$:

$$\mathbf{C} \approx \frac{\alpha_p(\mathbf{A})\alpha_p(\mathbf{B})}{(0.5\beta)^2}\mathbf{A}_q\mathbf{B}_q^\top \tag{2}$$

The scaling factor in (2) is used to undo the scaling in (1). Here, $\mathbf{A}_q\mathbf{B}_q^\top$ is an integer GEMM, as desired. For notational simplicity, if clear from context, we will drop the q subscript from $\mathbf{A}$ and $\mathbf{B}$.

In §2.1–§2.2, we evaluate the efficacy of integer GEMMs as a replacement to FP GEMMs by evaluating how well simple RTN works for inference and training (error analysis of RTN is discussed in §A.2). We do not strictly constrain the quantized integers to be within the representable range of certain bit-widths (the maximal value of quantized integers can be very large, up to maximum of INT32), so we use INT without specifying bit-width to denote the data type used in RTN in these subsections. The use of integers with specific bit-widths will be discussed later in §3–§4.

2.1. Efficacy of Integers: Inference

A majority of the literature on quantized low precision calculations focuses on inference efficiency (Frantar et al., 2023;

Table 1. Inference: Comparison on LLaMA-7B zero-shot performance and ViT ImageNet classification when quantize parameter only. $\overline{\text{Bits}}$ is the average bit required to store a weight parameter. HS: HellaSwag, WG: WinoGrande.

	Method	β	$\overline{\text{Bits}}$	ARC-c	ARC-e	BoolQ	HS	PIQA	WG
LLaMA-7B	Full-Precision	-	16	43.1	76.3	77.8	57.2	78.0	68.8
	GPTQ	-	4	37.4	72.7	73.3	54.9	77.9	67.9
	LLM-FP4	-	4	40.4	74.9	74.2	55.8	77.8	69.9
	QuIP	-	2	22.3	42.8	50.3	34.0	61.8	52.6
	RTN+HE	5	2.5	39.3	72.8	69.9	53.4	74.9	66.4
		7	2.9	42.6	73.9	72.3	55.9	77.0	67.4
		11	3.5	43.9	76.1	77.3	56.3	77.3	69.3
		15	4.0	43.0	75.7	77.5	57.0	78.0	69.2
		31	5.0	42.7	76.1	76.1	57.3	77.3	69.3

	Method	β	$\overline{\text{Bits}}$	Tiny	Small	Base	Large	Huge
ViT	Full-Precision	-	32	75.5	81.4	85.1	85.8	87.6
	PTQ4ViT	-	3	18.3	36.2	21.4	81.3	78.9
	RTN+HE	3	1.8	0.5	8.3	63.6	81.9	83.3
		5	2.4	38.2	69.0	81.1	84.9	86.7
		7	2.9	63.6	76.7	83.6	85.4	87.2
		15	4.0	73.4	80.5	84.8	85.7	87.6

Chee et al., 2023; Liu et al., 2023a; Yuan et al., 2022; Frantar et al., 2023; Chee et al., 2023; Liu et al., 2023a; Yuan et al., 2022; Lin et al., 2022; Li & Gu, 2023; Yuan et al., 2022; Ding et al., 2022; Li et al., 2023). Here, given a trained model, quantization seeks to reduce the precision of parameters and input activations to low precision. This allows faster low precision arithmetic for compute efficiency while maintaining model performance. So, we first evaluate how well RTN preserves model performance compared to baselines in this inference regime using downstream tasks: ARC (Clark et al., 2018), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), and WinoGrande (Sakaguchi et al., 2021). (§A.3 includes more experiments). Most quantization schemes for LLMs focus on quantizing GEMMs in the Linear layers, while quantization methods for Vision Transformers are more ambitious and quantize *all* GEMMs in a Transformer. We follow this convention for baselines, but present all variants for RTN.

Quantize Parameters Only. One direction of quantization research focus on quantizing the parameters for better storage and memory usage (Frantar et al., 2023; Chee et al., 2023). We also evaluate how well RTN works for storage and memory efficiency. After quantization, the quantized $\mathbf{W}_q$ usually contains a few hundreds of distinct integers. Simply representing $\mathbf{W}_q$ in plain integer format would not be efficient and usually requires larger than 8 bits per value for memory. By inspecting the value distribution of $\mathbf{W}_q$, we fing that a few values occur much more frequently than others, which create a clear opportunity for compression. We simply apply Huffman Encoding (HE), which was also used in (Han et al., 2016) to compress models for memory efficiency, to use shorter encoding for more frequent values. As shown in Table 1, with RTN and HE, we are able to significantly reduce the average bits per value with small

Table 2. Inference: Comparison on LLaMA-7B zero-shot performance and ViT ImageNet classification when quantize computation in all linear layers. The super-script ‡ indicates that LLM.int8() uses mixed-precision (INT8+FP16) to process outliers using FP16.

	Method	β	Type	ARC-c	ARC-e	BoolQ	HS	PIQA	WG
LLaMA-7B	Full-Precision	-	BF16	43.1	76.3	77.8	57.2	78.0	68.8
	LLM.int8()	-	INT8‡	43.8	75.5	77.8	57.4	77.6	68.7
	SmoothQuant	-	INT8	37.4	74.4	74.0	55.0	77.5	69.6
	LLM-QAT	-	INT4	30.2	50.3	63.5	55.6	64.3	52.9
	LLM-FP4	-	FP4	33.6	65.9	64.2	47.8	73.5	63.7
	RTN	5	INT	39.3	72.8	69.9	53.4	74.9	66.4
		7	INT	42.6	73.9	72.3	55.9	77.0	67.4
		11	INT	43.9	76.1	77.3	56.3	77.3	69.3
		15	INT	43.0	75.7	77.5	57.0	78.0	69.2
		31	INT	42.7	76.1	76.1	57.3	77.3	69.3

	Method	β	Type	Tiny	Small	Base	Large	Huge
ViT	Full-Precision	-	FP32	75.5	81.4	85.1	85.8	87.6
	RTN	5	INT	3.9	36.9	78.7	83.6	85.3
		7	INT	41.0	70.9	82.8	84.9	86.7
		15	INT	71.4	79.8	84.6	85.6	87.5

Table 3. Inference: Comparison on LLaMA-7B and ViT when quantize computation in all GEMMs. *: PTQ4ViT uses a twin uniform quantization so GEMMs cannot be performed on INT6 directly and requires some modifications.

	Method	β	Type	ARC-c	ARC-e	BoolQ	HS	PIQA	WG
LLaMA-7B	Full-Precision	-	BF16	43.1	76.3	77.8	57.2	78.0	68.8
	RTN	5	INT	23.5	34.3	54.8	32.5	57.6	49.7
		7	INT	34.2	64.0	64.6	50.1	70.3	61.2
		11	INT	41.6	72.4	68.7	55.1	75.4	65.1
		15	INT	44.0	75.0	74.6	56.4	77.0	66.3
		31	INT	43.4	75.8	76.8	57.5	77.4	68.4

	Method	β	Type	Tiny	Small	Base	Large	Huge
ViT	Full-Precision	-	FP32	75.5	81.4	85.1	85.8	87.6
	FQ-ViT	-	INT8	-	-	83.3	85.0	-
	I-ViT	-	INT8	-	81.3	84.8	-	-
	PTQ4ViT	-	INT6*	66.7	78.3	82.9	84.9	86.6
	APQ-ViT	-	INT4	17.6	48.0	41.4	-	-
	RepQ-ViT	-	INT4	-	65.1	68.5	-	-
	RTN	5	INT	3.5	28.5	76.9	83.2	84.9
		7	INT	39.0	69.9	82.1	84.7	86.5
		15	INT	71.1	79.8	84.5	85.6	87.5

or no performance degradation and result in significantly better efficiency compared to baselines (Frantar et al., 2023; Chee et al., 2023; Liu et al., 2023a; Yuan et al., 2022) for both Transformer based LLMs and Vision Transformers.

Quantize GEMMs in Linear layers. It is common (Xiao et al., 2023; Liu et al., 2023a) to try and quantize the weight and input activation of *linear layers* to low precision for compute efficiency. We summarize our comparisons in Tab. 2. Here, we compare RTN to (Xiao et al., 2023; Dettmers et al., 2022; Liu et al., 2023b;a). As shown in Tab. 2, a simple RTN works remarkably well compared to other baselines. We use INT as a data type for RTN here; in §4, we show that we can compute integer GEMMs of any bit-widths using arbitrarily low bit-width GEMMs.

Quantize all GEMMs. A more ambitious goal is to quan-

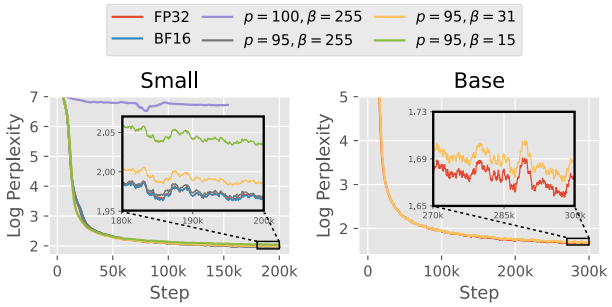

Figure 2. Training: Comparison of RoBERTa loss curves.

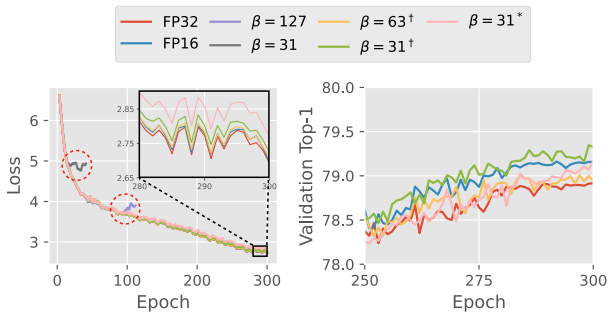

Figure 3. Training: Comparison of ViT-Small. † and *: we set $\beta = 16383$ and $\beta = 1023$, respectively, for the set $\{\nabla_{\mathbf{Y}}, \nabla_{\mathbf{P}}, \nabla_{\mathbf{O}}\}$.

Table 4. Training: Validation log perplexity of RoBERTa.

Size	FP32	BF16	$\beta = 255$	$\beta = 31$	$\beta = 15$
Small	1.869	1.868	1.823	1.840	1.891
Base	1.611	-	-	1.601	-

Table 5. Training: Validation top-1 accuracy of ViT-Small.

FP32	FP16	$\beta = 63^{\dagger}$	$\beta = 31^{\dagger}$	$\beta = 31^{*}$
78.91	79.16	78.94	79.33	79.17

tize *every* GEMM in a Transformer model for higher efficiency. The comparison results with (Lin et al., 2022; Li & Gu, 2023; Yuan et al., 2022; Ding et al., 2022; Li et al., 2023) are summarized in Tab. 3. We can draw a similar conclusion that a simple RTN offers strong performance.

2.2. Efficacy of Integers: Training

The transition from FP32 to FP16 and BF16 for GEMMs has doubled the compute efficiency of modern deep learning models. However, far fewer efforts have focused on low precision *training* (relative to inference) and this usually requires more sophisticated modifications (Wang et al., 2018; Wu et al., 2018; Zhu et al., 2020; Wortsman et al., 2023). In this subsection, we evaluate how well quantizing *all* GEMMs (both forward and backward pass) using RTN works for training Transformer models. To ensure that the updates can be properly accumulated for the parameters, we use FP32 for storing the parameters and use the quantized version for GEMMs. To limit the amount of compute but still gather useful feedback, we evaluate RTN on RoBERTa (Liu et al., 2019) pretraining using masked language modeling (Devlin et al., 2019) on the English Wikipedia corpus (Foundation) and ImageNet classification (Deng et al., 2009) using ViT (Dosovitskiy et al., 2021) (and see T5-Large (Raffel et al., 2020) finetuning in §A.3). All hyperparameters (including random seed) are the same for full-precision and RTN quantized training. See §A.3 for more details of training configurations.

RoBERTa. As shown in Fig. 2, when $p = 95\%$, for both Small and Base models, the RTN quantized training gives an almost identical log perplexity (loss) curves as FP32 training for $\beta \in \{15, 31, 255\}$. For larger β, the the curve is even closer to the FP32 training curve. We see that $\beta = 31$ already gives a remarkably good result. Surprisingly, despite a marginally higher training log perplexity when using RTN, the validation log perplexity of RTN ($\beta = 31$ and $\beta = 255$) is marginally lower than FP32 and BF16, see Tab. 4.

ViT. For ViT, compared to RoBERTa pretraining, we found that it may be necessary to allow the gradients $\nabla_{\mathbf{Y}}, \nabla_{\mathbf{P}}, \nabla_{\mathbf{O}}$ of the model to have higher bit-widths. As shown in Fig. 3, when β is the same ($\beta = 31$ and $\beta = 127$ for the set $\{\mathbf{X}, \mathbf{W}, \mathbf{Q}, \mathbf{K}, \mathbf{M}, \mathbf{V}\}$ and $\{\nabla_{\mathbf{Y}}, \nabla_{\mathbf{P}}, \nabla_{\mathbf{O}}\}$, we see divergence in the middle of training. Alternatively, when using a larger β for only the set $\{\nabla_{\mathbf{Y}}, \nabla_{\mathbf{P}}, \nabla_{\mathbf{O}}\}$, the loss curve of RTN quantized training is almost identical to FP32 training. Surprisingly, we observed similar results as RoBERTa training: despite marginally higher training loss when using RTN, the validation top-1 accuracy of RTN is higher than FP32 as shown in Fig. 3 and Tab. 5.

3. What happens with Low Bit-Width?

Converting floating point to integers alone will not provide efficiency benefits. Rather, we want to use a representation that can be efficiently computed (and this is why low bit-width integers are common in integer quantization). Notice that as a direct consequence of RTN, by (1), 95% of values can be represented using β distinct numbers, which requires only $\log_2(\beta + 1)$ bits. For example, if $\beta = 15$, then we can represent these 95% of values with 4-bit signed integers, which is already low bit-width. So, is there still a problem?

It turns out that the challenge involves dealing with the *remaining* 5% of entries. To get a sense of how large these values are, we calculate the ratio $\alpha_{100}(\cdot)/\alpha_{95}(\cdot)$ between the maximum and 95$^{\text{th}}$-percentile of the magnitude of each matrix in GEMMs when performing (a) inference (forward pass) of LLaMA-7B and ViT-Large and (b) training (forward pass and backward pass) of RoBERTa-Small at different training phases. We can check the ratios in Tab. 6 and Tab. 7, respectively. We see extremely large values across both training and inference and across the entire duration of training, so simply increasing the representation bit width of low precision integers by a few more bits will *not* be

Table 6. Maximal ratios between the maximum and 95-percentile of magnitudes of each matrix involved in GEMMs.

Model	$\mathbf{X}$	$\mathbf{W}$	$\mathbf{Q}$	$\mathbf{K}$	$\mathbf{M}$	$\mathbf{V}$
LLaMA-7B	141312.0	47.8	8.4	8.1	4448.0	36.2
ViT-Large	284402.4	34.8	4.3	4.3	120.0	8.9

Table 7. Maximal ratios between the maximum and 95-percentile of magnitudes of each matrix involved in GEMMs during the training of RoBERTa-Small.

Progress	$\mathbf{X}$	$\mathbf{W}$	$\nabla_{\mathbf{Y}}$	$\mathbf{Q}$	$\mathbf{K}$	$\nabla_{\mathbf{P}}$	$\mathbf{M}$	$\mathbf{V}$	$\nabla_{\mathbf{O}}$
1/3	28.7	7.1	292.5	3.7	3.0	309365.2	3924.6	3.1	25.8
2/3	25.7	13.8	235.4	4.2	2.7	283742.8	2283.3	3.3	32.4
3/3	22.0	16.0	290.3	4.0	3.0	218376.0	2018.6	3.4	28.9

Table 8. Catastrophic performance degradation when restricting outliers to a representable range of quantized domain or clipping the outliers on zero-shot inference of LLaMA-7B and ImageNet classification of quantized ViT models. $p = 100$ means we keep outliers within representable range of β distinct integers. $\beta = \infty$ means that we do not quantize the values. Clip means we clip the values that are larger than p-percentile.

	p	β	Clip	ARC-c	ARC-e	BoolQ	HS	PIQA	WG
LLaMA-7B	Full-Precision			43.1	76.3	77.8	57.2	78.0	68.8
	100	255	No	35.8	66.2	57.8	47.4	71.3	63.9
	99.5	∞	Yes	21.4	25.5	60.2	25.8	53.5	49.9
	95	31	No	43.4	75.8	76.8	57.5	77.4	68.4

	p	β	Clip	Tiny	Small	Base	Large	Huge
ViT	Full-Precision			75.5	81.4	85.1	85.8	87.6
	100	127	No	53.9	69.1	72.0	81.6	83.6
	99.5	∞	Yes	11.3	24.1	9.0	15.8	0.6
	95	15	No	71.1	79.8	84.5	85.6	87.5

sufficient to represent these heavy hitters.

We performed experiments studying different ways of handling these heavy hitters when quantizing all GEMMs (linear layers and self-attention computation) in Transformer models. Unless β is inordinately large (based on Tab. 6 and Tab. 7, more than 10^5 times larger than our choice of β for $p = 95\%$), simply ensuring that the heavy hitters lie within the representable range of β for $\beta = 255$ or $\beta = 127$ results in a huge performance drop as shown in Tab. 8. On the other hand, clipping the extreme heavy hitters (at the 99.5-percentile) also fails as shown in Tab. 8. Our observations for training are similar – we can see the loss curves for $p = 100\%, \beta = 255$ and $p = 95\%, \beta = 31$ in Fig. 2.

As briefly mentioned earlier, some ideas have been proposed to process these so-called outliers. The approach in Dettmers et al. (2022) exploits the location structure of where these outliers occur and moves the columns or rows of each matrix (with these outliers) into a different matrix, then GEMM is performed using FP16. The authors in Xiao et al. (2023) propose to smooth the outliers in activation and mitigate the quantization difficulty to parameters via a transformation. These strategy requires GEMM hardware

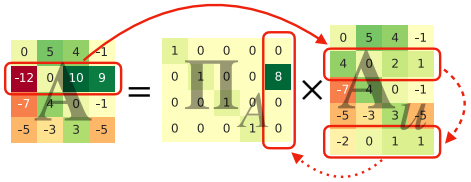

Figure 4. Illustration of unpacking row vectors. The solid, dashed, and dotted arrows correspond to lines 5, 4, and 6 in Algo. 1

support for different precisions or may lower the performance as shown in our baseline comparisons in §2.1.

Goals. We desire an approach that does not alter the results of integer GEMMs; in other words, all results in §2.1 and §2.2 must remain exactly the same, yet we should not need calculations using different precisions. This may appear unrealistic but our simple procedure, IM-Unpack, allow representing heavy hitters using low bit-width integers. Calculations are carried out using low bit-width integer arithmetic. Specifically, IM-Unpack unpacks a matrix containing heavy hitters into a *larger* unpacked matrix (we study how large the expansion will be in §4.2) whose values are all representable by low bit-width integers. IM-Unpack obtains the exact output of the original GEMM using purely low bit-width integer GEMMs on these unpacked matrices.

4. IM-Unpack: Integer Matrix Unpacking

Our approach starts with a simple observation that, for example, a 32-bit integer v can be represented as

$$v = v_0 + 128v_1 + 128^2v_2 + 128^3v_3 + 128^4v_4$$

where v_i are 8-bit integers. Multiplication/addition of two 32-bit integers can be performed on these decomposed 8-bit integers followed by some post-processing steps (scaling via bit shifting and accumulation). Note that while this unpacking enables performing high bit-width arithmetic using lower bit-width, it *does* need more operations. For example, one 32-bit addition now becomes five 8-bit additions with some follow up processing, and one 32-bit multiplication becomes twenty five 8-bit multiplications (distributive law).

Remark 4.1. The reason why this unpacking is still useful is because the additional work depends on the number and spatial distribution of the heavy hitters/outliers. We harvest gains because outliers account for a very small portion of the matrices that appear in practice in training/inference stages of Transformer models. Exploitation of the sparsity of outliers in quantization can also be found in Dettmers et al. (2022); Xiao et al. (2023).

Let b be the target bit-width of low bit-width integers and $s = 2^{b-1}$ be the representable range of bit-width b: b-bit integers can represent a set $\{-s+1, \cdots, 0, \cdots, s-1\}$. We refer to any integers *inside* of this set as In-Bound (IB) values and

Algorithm 1 UnpackRow($\mathbf{A}, b$)

Let $\mathbf{\Pi} \leftarrow \mathbf{I}$ and $s \leftarrow 2^{b-1}$ and $i \leftarrow 0$
while $\mathbf{A}[i,:]$ exists **do**
  if $\mathbf{A}[i,:]$ contains OB entries **then**
    Append floor($\mathbf{A}[i,:]/s$) as a new row to $\mathbf{A}$
    $\mathbf{A}[i,:] \leftarrow \mathbf{A}[i,:] \bmod s$
    Append $s\mathbf{\Pi}[:,i]$ as a new column to $\mathbf{\Pi}$
  end if
  $i \leftarrow i+1$
end while
return $\mathbf{A}, \mathbf{\Pi}$

any integers *outside* of this set as Out-of-Bound (OB) values, which will be used in later discussion to refer to the values that need to be unpacked. We will first show how to unpack a vector to multiple low bit-width vectors. Then, we will discuss how to unpack a matrix using different strategies to achieve better results in different cases in §4.1. Lastly, we will evaluate how well IM-Unpack works in §4.2, and provide an end to end quantization baseline comparison and discuss model speedup in §A.3 when employing our IM-Unpack as supplement to §2.

Unpacking an integer vector. Let $\mathbf{v}$ be an integer vector and define a function:

$$m(\mathbf{v}, s, i) = \text{floor}(\mathbf{v}/s^i) \bmod s \tag{3}$$

such that for all i, all entries of $m(\mathbf{v}, s, i)$ are bounded (IB), i.e., lie in the interval $[-s+1, s-1]$. When s is clear from the context, we shorten the LHS of (3) to just $m(\mathbf{v}, i)$. Then,

$$\mathbf{v} = \sum_{i=0}^{\infty} s^i m(\mathbf{v}, i) \tag{4}$$

Note that $\mathbf{v}/s^i$ decreases to 0 exponentially fast, so we are able to unpack a vector with just a few low bit-width vectors.

4.1. Variants of Matrix Unpacking

In this subsection, we discuss different strategies of matrix unpacking for different structure-types of matrices. First, we discuss the case where $\mathbf{A}$ is the matrix containing OB values to be unpacked and $\mathbf{B}$ is a matrix whose values are all IB. Next, we discuss how unpacking works when both $\mathbf{A}$ and $\mathbf{B}$ contains OB values.

Unpacking row vectors. We start with the simplest means of unpacking a matrix: unpacking the row vectors. Given a matrix $\mathbf{A}$, if one row of $\mathbf{A}$ contains OB values, we can unpack the row to multiple rows whose entries are all bounded. The exact procedure is described in Alg. 1 and illustrated in Fig. 4. In Fig. 4, when the second row in $\mathbf{A}$ contains OB values, we can unpack it to two row vectors (the second and fifth row) and the post-processing step takes the form of applying $\mathbf{\Pi}_A$ to the unpacked matrix $\mathbf{A}_u$.

Reconstructing A. $\mathbf{A}$ can be reconstructed using the unpacked matrix $\mathbf{A}_u$ whose entries are IB and a sparse matrix

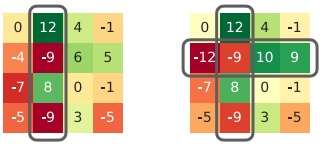

Figure 5. Left: Failure case for unpacking rows. Right: Failure case for unpacking rows or columns alone.

$\mathbf{\Pi}$ whose column contains *only one* non-zero:

$$\mathbf{A}_u, \mathbf{\Pi}_A = \text{UnpackRow}(\mathbf{A}, b)$$
$$\mathbf{A} = \mathbf{\Pi}_A \mathbf{A}_u$$

Here, applying $\mathbf{\Pi}_A$ to $\mathbf{A}_u$ can be efficiently computed easily (for example, via `torch.index_add`).

Are we done? If we do not care about maximizing efficiency, then the above scheme already provides a way to perform high bit-width GEMM using low bit-width GEMM. However, this might not be the optimal unpacking strategy for some matrices. For example, consider the left matrix shown in Fig. 5. Since every row of this matrix contains OB values, every row needs to be unpacked, resulting in a much larger matrix. In this case, it might be better to try and unpack the column vectors. Let us apply a similar idea of unpacking row vectors to unpack column vectors of $\mathbf{A}$:

$$\mathbf{A} = \mathbf{A}'_u \mathbf{\Pi}'_A$$
$$\mathbf{A}\mathbf{B}^\top = \mathbf{A}'_u \mathbf{\Pi}'_A \mathbf{B}^\top \tag{5}$$

While unpacking column vectors is reasonable, the sparse matrix $\mathbf{\Pi}'_A$ creates an issue when performing a GEMM of two lower bit-width matrices: $\mathbf{\Pi}'_A$ has to be applied to $\mathbf{A}'_u$ or $\mathbf{B}^\top$ before GEMM, but the result/output may contain OB entries after application, disabling low bit-width integer GEMM. This problem is similar to the per-channel quantization. It is not simple to handle and becomes more involved when $\mathbf{B}$ also need to be unpacked.

Unpacking column vectors. Alternatively, let us look at how $\mathbf{A}\mathbf{B}^\top$ is computed via outer product of column vectors:

$$\mathbf{C} = \mathbf{A}\mathbf{B}^\top = \sum_{i=1}^{d} \mathbf{A}[:,i]\mathbf{B}[:,i]^\top$$

Let us look at the i-th outer product. Let us try unpacking $\mathbf{A}[:,i]$ using (4), then we have

$$\mathbf{A}[:,i]\mathbf{B}[:,i]^\top = \sum_{j=0}^{\infty} s^j m(\mathbf{A}[:,i], j)\mathbf{B}[:,i]^\top$$

Suppose that $m(\mathbf{A}[:,i], j) = 0$ for $j \geq k$, then we can unpack one outer product to k outer products. This is equivalent to appending $m(\mathbf{A}[:,i], j)$ for $0 \leq j < k$ to the columns of $\mathbf{A}$, appending k identical $\mathbf{B}[:,i]$ to the columns of $\mathbf{B}$, and maintaining a diagonal matrix to keep track of the scaling factor s^j. The exact procedure is described in Alg. 2, and

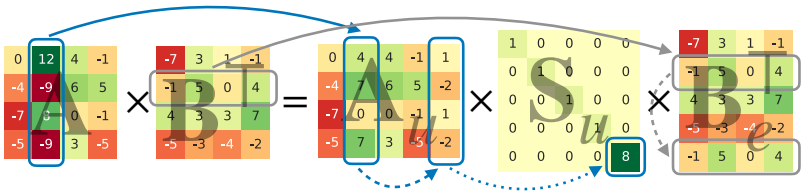

Figure 6. Illustration of unpacking column vectors. The blue solid, dashed, and dotted arrows correspond to lines 5, 4, and 7 in Algo. 1, and the gray dashed arrow corresponds to line 6 in Algo. 1.

Algorithm 2 UnpackColumn$(\mathbf{A}, \mathbf{B}, \mathbf{S}, b)$

Let $s \leftarrow 2^{b-1}$ and $i \leftarrow 0$
while $\mathbf{A}[:, i]$ exists **do**
  if $\mathbf{A}[:, i]$ contains OB entries **then**
    Append floor$(\mathbf{A}[:, i]/s)$ as a new column to $\mathbf{A}$
    $\mathbf{A}[:, i] \leftarrow \mathbf{A}[:, i]$ mod s
    Append $\mathbf{B}[:, i]$ as a new column to $\mathbf{B}$
    Append $s\mathbf{S}[i, i]$ as a new diagonal entry to $\mathbf{S}$
  end if
  $i \leftarrow i + 1$
end while
return $\mathbf{A}, \mathbf{B}, \mathbf{S}$

Algorithm 3 ScaledMatMul$(\mathbf{A}, \mathbf{B}, \mathbf{S})$

Let $\mathbf{C} \leftarrow 0$
for all distinct diagonal entry s^i in $\mathbf{S}$ **do**
  Let $\mathcal{I}$ be the index set where $\mathbf{S}[j, j] = s^i$ for $j \in \mathcal{I}$
  $\mathbf{C} \leftarrow \mathbf{C} + s^i \mathbf{A}[:, \mathcal{I}]\mathbf{B}[:, \mathcal{I}]^\top$
end for
return $\mathbf{C}$

Fig. 6 shows a visualization of unpacking columns. Using column unpacking, we have

$$\mathbf{A}_u, \mathbf{B}_e, \mathbf{S}_u = \text{UnpackColumn}(\mathbf{A}, \mathbf{B}, \mathbf{I}, b)$$
$$\mathbf{A}\mathbf{B}^\top = \mathbf{A}_u \mathbf{S}_u \mathbf{B}_e^\top$$

Naively, this still suffers from the same problem as discussed in (5) in that there is a diagonal scaling matrix between two low bit-width matrices making low bit-width GEMMs difficult. However, since $\mathbf{S}_u$ is a diagonal matrix whose diagonal entries consist of a few distinct factors in $\{1, s, s^2, ...\}$, we can easily compute one GEMM for each distinct diagonal entry as shown in Alg. 3.

$$\mathbf{A}\mathbf{B}^\top = \text{ScaledMatMul}(\mathbf{A}_u, \mathbf{B}_e, \mathbf{S}_u)$$

Further, since s is a power of 2, the scaling can be efficiently implemented via bit shifting.

Are we done yet? Unpacking columns is efficient for the left matrix shown in Fig. 5. However, neither unpacking rows nor unpacking columns will be efficient for unpacking the right matrix shown in Fig. 5. All rows and columns contains OB values. Unpacking rows or columns alone will not be ideal. For the right matrix in Fig. 5, a better strategy is to unpack the second row and the second column simultaneously.

Unpacking both rows and columns simultaneously. Our final strategy combines row and column unpacking together and selectively performs row unpack or column unpack based on the number of OB values that can be eliminated. The procedure is described in Alg. 4, and we provide an illustration of unpacking both dimensions in Fig. 7. With this procedure, we can obtain the output of high bit-width GEMM using low bit-width as:

Algorithm 4 UnpackBoth$(\mathbf{A}, \mathbf{B}, \mathbf{S}, b)$

Let $s \leftarrow 2^{b-1}$ and
while True **do**
  Let $(c_0, i), (c_1, j)$ be the tuples of top OB count in row/column vectors and corresponding index
  if $c_0 = 0$ and $c_1 = 0$ **then**
    break
  else if $c_0 \geq c_1$ **then**
    Append floor$(\mathbf{A}[i, :]/s)$ as a new row to $\mathbf{A}$
    $\mathbf{A}[i, :] \leftarrow \mathbf{A}[i, :]$ mod s
    Append $s\mathbf{\Pi}[:, i]$ as a new column to $\mathbf{\Pi}$
  else
    Append floor$(\mathbf{A}[:, j]/s)$ as a new column to $\mathbf{A}$
    $\mathbf{A}[:, j] \leftarrow \mathbf{A}[:, j]$ mod s
    Append $\mathbf{B}[:, j]$ as a new column to $\mathbf{B}$
    Append $s\mathbf{S}[j, j]$ as a new diagonal entry to $\mathbf{S}$
  end if
end while
return $\mathbf{A}, \mathbf{B}, \mathbf{S}, \mathbf{\Pi}$

$$\mathbf{A}_u, \mathbf{B}_e, \mathbf{S}_u, \mathbf{\Pi}_A = \text{UnpackBoth}(\mathbf{A}, \mathbf{B}, \mathbf{I}, b)$$
$$\mathbf{A}\mathbf{B}^\top = \mathbf{\Pi}_A \mathbf{A}_u \mathbf{S}_u \mathbf{B}_e^\top \tag{6}$$

Here, $\mathbf{A}_u \mathbf{S}_u \mathbf{B}^\top$ can be calculated via Alg. 3, and applying $\mathbf{\Pi}_A$ can be carried out efficiently as discussed.

Combining everything. Since we have different strategies for unpacking, let us first define a unified interface in Alg. 5. One can verify that for any strategies s_A:

$$\mathbf{A}_u, \mathbf{B}_e, \mathbf{S}_u, \mathbf{\Pi}_A = \text{Unpack}(\mathbf{A}, \mathbf{B}, \mathbf{I}, b, s_A)$$
$$\mathbf{A}\mathbf{B}^\top = \mathbf{\Pi}_A \mathbf{A}_u \mathbf{S}_u \mathbf{B}_e^\top \tag{7}$$

In the previous discussion, $\mathbf{B}$ was assumed to have all values IB. When $\mathbf{B}$ contains OB values, we note that $\mathbf{B}$ can be unpacked in a similar manner, and the choice of unpacking strategies for $\mathbf{B}$ is independent of the unpacking strategy for $\mathbf{A}$. For example, $\mathbf{A}$ can be unpacked row-wise, while $\mathbf{B}$ is unpacked column-wise. By taking the unpacked $\mathbf{A}_u, \mathbf{B}_e, \mathbf{S}_u, \mathbf{\Pi}_A$ from (7), we can further unpack $\mathbf{B}$ using

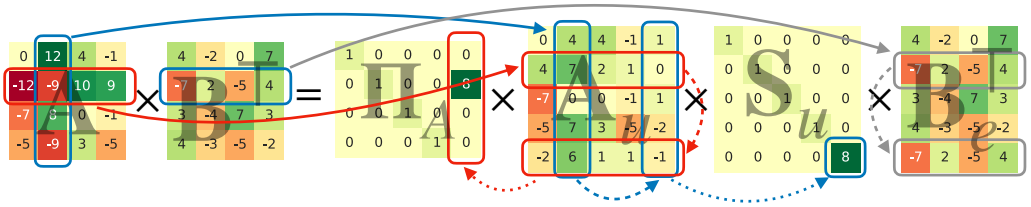

Figure 7. Illustration of unpacking both rows and columns based on the OOB counts. The red solid, dashed, and dotted arrows correspond to lines 8, 7, and 9 in Algo. 4. The blue solid, dashed, and dotted arrows correspond to lines 12, 11, and 14 in Algo. 4, and the gray dashed arrow corresponds to line 13 in Algo. 4.

Algorithm 5 Unpack$(\mathbf{A}, \mathbf{B}, \mathbf{S}, b,$ strategy$)$

if strategy is UnpackRow **then**
  $\mathbf{A}_u, \mathbf{\Pi}_A \leftarrow \text{UnpackRow}(\mathbf{A}, b)$
  $\mathbf{S}_u, \mathbf{B}_e \leftarrow \mathbf{S}, \mathbf{B}$
else if strategy is UnpackColumn **then**
  $\mathbf{A}_u, \mathbf{B}_e, \mathbf{S}_u \leftarrow \text{UnpackColumn}(\mathbf{A}, \mathbf{B}, \mathbf{S}, b)$
  $\mathbf{\Pi}_A \leftarrow \mathbf{I}$
else
  $\mathbf{A}_u, \mathbf{B}_e, \mathbf{S}_u, \mathbf{\Pi}_A \leftarrow \text{UnpackBoth}(\mathbf{A}, \mathbf{B}, \mathbf{S}, b)$
end if
return $\mathbf{A}_u, \mathbf{B}_e, \mathbf{S}_u, \mathbf{\Pi}_A$

strategy s_B:

$$\mathbf{B}_{eu}, \mathbf{A}_{ue}, \mathbf{S}_{uu}, \mathbf{\Pi}_B = \text{Unpack}(\mathbf{B}_e, \mathbf{A}_u, \mathbf{S}_u, b, s_B)$$
$$\mathbf{A}\mathbf{B}^\top = \mathbf{\Pi}_A \mathbf{A}_{ue} \mathbf{S}_{uu} \mathbf{B}_{eu}^\top \mathbf{\Pi}_B^\top$$

Here, values in both $\mathbf{A}_{ue}$ and $\mathbf{B}_{eu}$ are IB, and the result can be obtained similar to discussion in Eq. (6).

As we noted during the discussion, $\mathbf{\Pi}_A, \mathbf{\Pi}_B, \mathbf{S}_{uu}$ are special sparse matrices. $\mathbf{\Pi}_A$ and $\mathbf{\Pi}_B$ are sparse matrices whose column contains only one non-zero, and $\mathbf{S}_{uu}$ are diagonal matrices whose diagonal entries consist of a few distinct factors. As a result, $\mathbf{A}_{ue}\mathbf{S}_{uu}\mathbf{B}_{eu}^\top$ is computed via Alg. 3 using GEMMs on $\mathbf{A}_{ue}$ and $\mathbf{B}_{eu}$, and applying Π_A and Π_B is possible simply via `torch.index_add`.

Summary. We introduced three strategies to unpack a matrix to low bit-width integer matrices for different structures of OB values in a matrix. While these strategies work for arbitrary matrices, we can clearly see that these unpacking strategies are most efficient when the OB values concentrate in a few columns and rows. Luckily, the matrices of interest in Transformer models indeed have this property, which is studied and exploited in several works (Dettmers et al., 2022; Xiao et al., 2023).

4.2. Evaluating Unpacking Overhead

The idea of IM-Unpack is to use more low bit-width arithmetic operations to compute a high bit-width operation. As we see in the description of IM-Unpack algorithm, the number of row and column vectors will increase, so the unpacked matrices $\mathbf{A}_{ue}$ and $\mathbf{B}_{eu}$ is larger in terms of size compared to $\mathbf{A}$ and $\mathbf{B}$, which obviously increases the computational cost of low bit-width GEMMs. In this subsection, we evaluate how much this cost will increase. For two matrices $\mathbf{A}$ and $\mathbf{B}$, the complexity of a GEMM is $\mathcal{O}(ndh)$. Similarly, let n', d' be the size of $\mathbf{A}_{ue}$ and h' be the number of rows of $\mathbf{B}_{eu}$. The cost of $\mathbf{A}_{ue}\mathbf{S}_{uu}\mathbf{B}_{eu}^\top$ is $\mathcal{O}(n'd'h')$, we can directly measure the unpack ratio

$$r = \frac{n'd'h'}{ndh}$$

Table 9. Averaged unpack ratios of each type of GEMMs in LLaMA-7B: linear layers (computing $\mathbf{Y}$), attention score (computing $\mathbf{P}$), and attention output (computing $\mathbf{O}$) when using different unpack strategies and integer bit-width b under quantization β settings. AS: Attention Score, AO: Attention Output.

	β			5			15			31		
	Integer Bits b			3	4	5	4	5	6	5	6	7
Linear ($\mathbf{Y}$)	$\mathbf{X}$	$\mathbf{W}$	Row	Row	2.67	1.93	1.57	2.47	2.02	1.73	2.12	2.00	1.74
			Row	Col	10.76	2.35	1.61	9.91	5.36	1.84	8.44	5.62	1.86
			Row	Both	5.46	1.95	1.57	5.15	2.20	1.73	4.71	2.24	1.75
			Col	Row	3.80	1.32	1.06	3.98	1.64	1.16	3.93	1.68	1.17
			Col	Col	15.40	1.62	1.09	16.00	4.01	1.25	15.69	4.33	1.27
			Col	Both	5.21	1.34	1.06	6.04	1.76	1.16	5.98	1.82	1.17
			Mix		2.6	1.27	1.06	2.44	1.4	1.15	2.1	1.42	1.16
AS ($\mathbf{P}$)	$\mathbf{Q}$	$\mathbf{K}$	Row	Row	1.97	1.60	1.0	2.00	1.87	1.15	2.00	1.87	1.18
			Row	Col	3.22	1.64	1.0	5.35	2.07	1.17	5.36	2.09	1.20
			Col	Row	1.81	1.04	1.0	2.91	1.14	1.01	2.91	1.15	1.01
			Col	Col	3.36	1.08	1.0	8.66	1.32	1.03	8.67	1.35	1.03
			Mix		1.72	1.03	1.0	1.95	1.13	1.01	1.95	1.14	1.01
AO ($\mathbf{O}$)	$\mathbf{M}$	$\mathbf{V}$	Row	Row	6.02	4.18	3.27	4.72	3.65	3.02	3.93	3.24	2.81
			Row	Col	15.10	4.53	3.35	18.21	4.64	3.16	15.07	4.21	2.95
			Col	Row	16.29	8.14	5.12	11.28	6.98	4.91	8.41	5.84	4.42
			Col	Col	42.21	8.76	5.21	43.57	9.11	5.09	32.31	7.74	4.61
			Mix		5.98	4.11	3.16	4.7	3.62	2.97	3.92	3.22	2.77

Table 10. Averaged unpack ratios of each type of quantized GEMMs in both forward and backward of a RoBERTa-Small when using different integer bit length b at different training phrases of the $\beta = 31$ experiment in Fig. 2. The optimal strategies (Mix as in Tab. 9) for each GEMM is used.

Progress		1/3			2/3			3/3		
Integer Bits b		5	6	7	5	6	7	5	6	7
Linear	$\mathbf{Y}$	2.00	1.31	1.08	2.00	1.32	1.07	2.00	1.32	1.05
	$\nabla_{\mathbf{X}}$	1.50	1.31	1.15	1.50	1.30	1.16	1.50	1.30	1.15
	$\nabla_{\mathbf{W}}$	1.98	1.25	1.04	1.98	1.25	1.03	1.98	1.25	1.03
AS	$\mathbf{P}$	1.66	1.04	1.00	1.42	1.05	1.0	1.40	1.04	1.00
	$\nabla_{\mathbf{Q}}$	2.22	1.90	1.71	2.22	1.91	1.7	2.24	1.92	1.71
	$\nabla_{\mathbf{K}}$	1.79	1.06	1.00	1.49	1.07	1.0	1.45	1.07	1.00
AO	$\mathbf{O}$	3.11	2.71	2.30	3.10	2.68	2.24	3.10	2.62	2.22
	$\nabla_{\mathbf{M}}$	1.21	1.10	1.04	1.21	1.10	1.04	1.21	1.10	1.04
	$\nabla_{\mathbf{V}}$	2.88	2.52	2.12	2.87	2.48	2.10	2.86	2.41	2.09

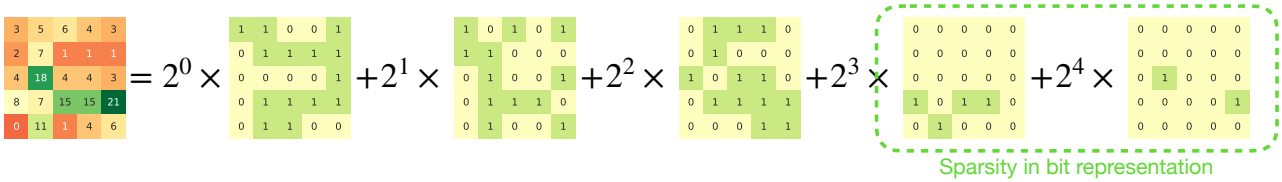

Figure 8. Illustration of the bit representation of a matrix. Heavy-hitters have higher order non-zero bits. When a matrix contains heavy-hitters, its bit representation has a sparsity structure in the higher order bits as illustrated.

Table 11. Averaged ratios of quantized GEMMs ($\beta = 15$) in linear layers on ViT-Large when using different strategies and a range of integer bit-widths b to the lowest bit-width possible.

	Integer Bits b			2	3	4	5	6	7
$\mathbf{X}$	Row	$\mathbf{W}$	Row	7.24	3.80	2.63	2.22	1.54	1.43
	Row		Col	194.89	27.52	10.46	4.31	1.62	1.43
	Row		Both	85.92	13.80	6.22	2.76	1.56	1.43
	Col		Row	19.27	4.85	3.06	1.46	1.25	1.12
	Col		Col	526.31	35.86	12.22	2.81	1.32	1.13
	Col		Both	27.06	13.31	7.59	1.78	1.26	1.13
	Both		Row	7.62	3.39	2.58	1.64	1.42	1.33
	Both		Col	206.32	24.45	10.27	3.15	1.49	1.34
	Both		Both	79.19	11.16	6.09	2.01	1.43	1.34
	Mix			6.29	2.98	2.24	1.40	1.23	1.11

to understand by how much the cost for low bit-width GEMMs increases. We uses LLaMA-7B to study the unpack ratio r when using different unpacking strategies (Tab. 9). Note that since unpacking both requires keeping track of the OB count in each row and column vector which is not as fast as the other two strategies, we only use it for unpacking parameters $\mathbf{W}$ for inference since it can be performed once when loading the model. The Mix in Tab. 9 means that for each GEMM, we compare different strategies and choose the optimal strategy that results in the smallest unpack ratio. We note that the unpack ratios of computing $\mathbf{Y}$ and $\mathbf{P}$ are quite reasonable, but the ratios of computing $\mathbf{O}$ is larger. This is expected since the large outliers of the self-attention matrix $\mathbf{M}$ mainly concentrate in the diagonal (Beltagy et al., 2020). We also study the unpack ratios of each type of quantized GEMMs at different training phases, and show the results of Mix strategy in Tab. 10. The ratios stay relatively unchanged as training progresses. Also, we can observe similar high unpack ratio when computing $\mathbf{O}$ and $\nabla_{\mathbf{V}}$ since these GEMMs involve the self-attention matrix $\mathbf{M}$. Lastly, we verify that we can unpack matrices to arbitrarily low integer matrices (Tab. 11). The 2-bit setting is the lowest bit width that can be used for symmetric signed integers ($\{-1, 0, 1\}$). In §A.1, we also describe a small adjustment to reduce this 2-bit setting to 1-bit arithmetic.

4.3. Limitations

To simplify the presentation, we used the simplest RTN quantization, which might not deliver the optimal performance. More sophisticated techniques are likely to further improve the results. For example, we may be able to remove the demands of large β for the set $\{\nabla_{\mathbf{Y}}, \nabla_{\mathbf{P}}, \nabla_{\mathbf{O}}\}$ for ViT training. The current unpacking strategies cannot handle the self-attention matrix $\mathbf{M}$ efficiently since the outliers mainly concentrate on the diagonal region rather than rows or columns; this needs further investigation.

5. Conclusions

In this paper, we verify the efficacy of integer GEMMs in both training and inference for Transformer-based models in language modeling and vision. A simple RTN quantization strategy works well compared to baselines. But the presence of large outliers/heavy-hitters makes it difficult to make use of efficient low bit-width integer GEMMs since these outliers are much larger than the representable range of low bit-width integers. We take a "multi-resolution" view (Zeng et al., 2022) in how we extract a spectrum of bit-width tradeoffs. This is loosely similar to sparsity but here, instead of making a zero versus non-zero distinction between the entries, our heavy-hitters (which need higher bit-width representations) are analogous to "non-sparse" entries (as illustrated in Fig. 8). To handle high bit-width heavy-hitters, we develop an algorithm to unpack integer matrices that contains arbitrarily large values to slightly larger matrices with the property that all values lie within the representable range of low bit-width integers and a procedure to obtain the GEMM output of original matrices using only low bit-width integer GEMMs on the unpacked matrices followed by some scaling (using bit shifting) and accumulation. Our algorithm can greatly simplify the design of hardware and improve the power efficiency by only supporting low bit-width integer GEMMs for both training and inference.

Acknowledgments

The authors are grateful to the doctors, nurses, and staff at the UW Neuro ICU for their exceptional care and support in aiding one of the author's recovery during the preparation of this paper. This work was supported in part by funding from the Vilas Board of Trustees and UW–Madison Office of the Vice Chancellor for Research and Graduate Education. The authors thank Michael Davies for many helpful discussions.

Impact Statement

This paper presents an approach to use low precision computing for training and inference. The goal is to accelerate computing speed and reduce energy consumption and thus promote more sustainable computational practices. We do not identify any specific societal implications or ethical considerations that must be highlighted.

References

Adepu, H., Zeng, Z., Zhang, L., and Singh, V. Framequant: Flexible low-bit quantization for transformers. In *International Conference on Machine Learning*, 2024.

Banner, R., Nahshan, Y., and Soudry, D. Post training 4-bit quantization of convolutional networks for rapid-deployment. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/c0a62e133894cdce435bcb4a5df1db2d-Paper.pdf.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. URL https://arxiv.org/abs/2004.05150.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, 2020.

Chee, J., Cai, Y., Kuleshov, V., and Sa, C. D. QuIP: 2-bit quantization of large language models with guarantees. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=xrk9g5vcXR.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

De Sa, C., Leszczynski, M., Zhang, J., Marzoev, A., Aberger, C. R., Olukotun, K., and Ré, C. High-accuracy low-precision training. *arXiv preprint arXiv:1803.03383*, 2018. URL https://arxiv.org/abs/1803.03383.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *Conference on Computer Vision and Pattern Recognition*, 2009.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=dXiGWqBoxaD.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Ding, Y., Qin, H., Yan, Q., Chai, Z., Liu, J., Wei, X., and Liu, X. Towards accurate post-training quantization for vision transformer. In *Proceedings of the 30th ACM International Conference on Multimedia*, MM '22, pp. 5380–5388, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392037. doi: 10.1145/3503161.3547826. URL https://doi.org/10.1145/3503161.3547826.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Foundation, W. Wikimedia downloads. URL https://dumps.wikimedia.org.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. OPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=tcbBPnfwxS.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1510.00149.

Javaheripi, M., Bubeck, S., Abdin, M., Aneja, J., Bubeck, S., Mendes, C. C. T., Chen, W., Del Giorno, A., Eldan, R., Gopi, S., et al. Phi-2: The surprising power of small language models. *Microsoft Research Blog*, 2023.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Kim, S., Gholami, A., Yao, Z., Mahoney, M. W., and Keutzer, K. I-bert: Integer-only bert quantization. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5506–5518. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/kim21d.html.

Li, Z. and Gu, Q. I-vit: Integer-only quantization for efficient vision transformer inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17065–17075, 2023.

Li, Z., Xiao, J., Yang, L., and Gu, Q. Repq-vit: Scale reparameterization for post-training quantization of vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17227–17236, 2023.

Lin, Y., Zhang, T., Sun, P., Li, Z., and Zhou, S. Fq-vit: Post-training quantization for fully quantized vision transformer. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 1173–1179, 2022.

Liu, S.-y., Liu, Z., Huang, X., Dong, P., and Cheng, K.-T. LLM-FP4: 4-bit floating-point quantized transformers. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 592–605, Singapore, December 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.39. URL https://aclanthology.org/2023.emnlp-main.39.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach, 2019. URL https://arxiv.org/abs/1907.11692.

Liu, Z., Oguz, B., Zhao, C., Chang, E., Stock, P., Mehdad, Y., Shi, Y., Krishnamoorthi, R., and Chandra, V. Llm-qat: Data-free quantization aware training for large language models, 2023b. URL https://arxiv.org/abs/2305.17888.

Nagel, M., Baalen, M. v., Blankevoort, T., and Welling, M. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1325–1334, 2019.

Narayan, S., Cohen, S. B., and Lapata, M. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In Riloff, E., Chiang, D., Hockenmaier, J., and Tsujii, J. (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1797–1807, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1206. URL https://aclanthology.org/D18-1206.

NVIDIA. Nvidia a100. URL https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21 (140):1–67, 2020.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Wang, N., Choi, J., Brand, D., Chen, C.-Y., and Gopalakrishnan, K. Training deep neural networks with 8-bit floating point numbers. *Advances in neural information processing systems*, 31, 2018.

Wortsman, M., Dettmers, T., Zettlemoyer, L., Morcos, A. S., Farhadi, A., and Schmidt, L. Stable and low-precision training for large-scale vision-language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=sqqASmpA2R.

Wu, S., Li, G., Chen, F., and Shi, L. Training and inference with integers in deep neural networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJGXzmspb.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Yuan, Z., Xue, C., Chen, Y., Wu, Q., and Sun, G. Ptq4vit: Post-training quantization for vision transformers with twin uniform quantization. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XII*, pp. 191–207, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-19774-1. doi: 10.1007/978-3-031-19775-8_12. URL https://doi.org/10.1007/978-3-031-19775-8_12.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zeng, Z., Pal, S., Kline, J., Fung, G. M., and Singh, V. Multi resolution analysis (MRA) for approximate self-attention. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 25955–25972. PMLR, 17–23 Jul 2022.

Zhu, F., Gong, R., Yu, F., Liu, X., Wang, Y., Li, Z., Yang, X., and Yan, J. Towards unified int8 training for convolutional neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1969–1979, 2020.

Table 12. Standard deviation vs percentile when removing largest outliers. $\mathbf{X}$ has 2.25×10^7 entries and $\mathbf{W}$ has 1.68×10^7 entries.

Number of Largest Outliers Removed		0	10	10^2	10^3
$\mathbf{W}$	Standard Deviation	0.0082	0.0082	0.0082	0.0082
	95-Percentile	0.0177	0.0177	0.0177	0.0177
$\mathbf{X}$	Standard Deviation	0.0330	0.0327	0.0320	0.0214
	95-Percentile	0.0280	0.0280	0.0280	0.0278

A. Appendix

We provide more details about design choices and experimental setup as well as additional experiments that were omitted from the main paper due to space.

A.1. Rationale behind certain design choices

Why Use Percentiles? We need a way of mapping the actual range of values in a floating point matrix to an integer range. In this process, we should ensure that most values fall within the desired range and fill up the representable range as much as possible, so we need a statistic to estimate the range of values in a FP matrix. We compared percentile and standard deviation and inspected different parameter matrices $\mathbf{W}$ and the corresponding inputs $\mathbf{X}$ in the LLaMA-7B model (Touvron et al., 2023). The outlier problem in $\mathbf{W}$ is moderate: we can see in Tab. 12, that both standard deviation and percentile estimates are stable across columns. On the other hand, the outliers in $\mathbf{X}$ is a bit more problematic and includes a few entries that are much larger than the non-outliers. The estimation of standard deviation varies much more as shown in Tab. 12: even removing an extremely small subset of the largest outliers can sizably alter the estimates. In contrast, the percentile is more stable. As a result, we choose percentile as the estimation of value range.

Can We Use as Low as 1-bit Encoding? As discussed in §4.2, we use 2 bits to encode $\{-1, 0, +1\}$ for symmetric encoding, so the 2 bit setting is the lowest bit width that our method supports. However, with small adjustments, it is possible in principle to derive a 1-bit encoding scheme for this 2-bit encoding. Given $\mathbf{A}$ and $\mathbf{B}$ matrices whose values are $\{-1, 0, +1\}$, we can easily decompose

$$\mathbf{A} = \mathbf{A}_p - \mathbf{A}_n$$
$$\mathbf{B} = \mathbf{B}_p - \mathbf{B}_n$$

where $\mathbf{A}_p, \mathbf{A}_n, \mathbf{B}_p, \mathbf{B}_n$ consist of values $\{0, 1\}$. Then,

$$\mathbf{A}\mathbf{B}^\top = \mathbf{A}_p\mathbf{B}_p^\top - \mathbf{A}_n\mathbf{B}_p^\top - \mathbf{A}_n\mathbf{B}_p^\top + \mathbf{A}_n\mathbf{B}_n^\top$$

becomes four 1-bit GEMMs, so 1-bit encoding is feasible in this sense.

One benefit of 1 bit quantization is that multiplication in GEMM becomes logic AND and accumulation becomes counting the number of 1's. Notice that similar benefits can be exploited in the 2-bit encoding for $-1, 0, +1$. The mapping between 2-bit binary representation and decimal numbers is $00_b = 0, 01_b = 1, 10_b = 0, 11_b = -1$. We call the left bit a sign bit and right bit a value bit. We note that the multiplication becomes logical XOR in the sign bit and logical AND in the value bit, and accumulation becomes counting the number of 1's in value bit and then subtracting the number of 11_b. We have not performed experiments evaluating this approach yet.

A.2. Analyzing the Error of Rounding to Nearest

Our method consists of two main steps: 1. Rounding to Nearest (RTN) integer, which maps floating-point entries in the matrices to integer entries; and 2. Our main algorithm IM-Unpack, which unpacks integer matrices to the desired bit-width. As described at the end of §3, Step 2 does not incur any loss and so, can be ignored in the discussion below.

We can perform a simple technical analysis for Step 1. Since the scaling in (1) and (2) will not affect the analysis, we can assume, without loss of generality, that the scaling is simply set to 1. We are interested in the error

$$\mathbf{E} = \mathbf{A}\mathbf{B}^\top - \mathbf{A}_q\mathbf{B}_q^\mathbf{T}$$

We can examine a specific entry in the product $\mathbf{C}$. Let $\mathbf{u}$ and $\mathbf{v}$ be the rows of $\mathbf{A}$ and $\mathbf{B}$, respectively, whose inner product $\sum_i^d \mathbf{u}[i]\mathbf{v}[i]$ is a specific entry in $\mathbf{C}$. In Step 1, each $\mathbf{u}[i]$ and $\mathbf{v}[i]$ is rounded to the nearest integer, $\hat{\mathbf{u}}[i]$ and $\hat{\mathbf{v}}[i]$, resulting in a deviation of at most ± 0.5 from the original entries.

Let the error incurred in this i-th term due to rounding be denoted as

$$\epsilon[i] = \hat{\mathbf{u}}[i]\hat{\mathbf{v}}[i] - \mathbf{u}[i]\mathbf{v}[i]$$

After some calculations, we see that the product $\hat{\mathbf{u}}[i]\hat{\mathbf{v}}[i]$ of the rounded values will deviate by at most $\pm 0.5(|\mathbf{u}[i]| + |\mathbf{v}[i]| + 0.5)$ from the product $\mathbf{u}[i]\mathbf{v}[i]$ of the original values. Let $\mathbf{c}[i] = 0.5(|\mathbf{u}[i]| + |\mathbf{v}[i]| + 0.5)$, and let c be a vector whose i-th entry is $\mathbf{c}[i]$. Then, it follows that $-\mathbf{c}[i] \leq \epsilon[i] \leq \mathbf{c}[i]$. We can now examine the total error in the inner product $\sum_i \mathbf{u}[i]\mathbf{v}[i]$.

When $\epsilon[i] > 0$, this term over-contributes, and when $\epsilon[i] < 0$, it under-contributes. Assuming that these two scenarios are equally likely, the cumulative error in $\sum_i \mathbf{u}[i]\mathbf{v}[i]$ can be represented as a sum involving a Rademacher-distributed variable X (half-half chance of being $+1$ or -1), modulated by $\epsilon[i]$ as coefficients. Our interest is in the error $\sum_i \mathbf{x}[i]\epsilon[i]$, where $\mathbf{x}[i]$ is a set of random variables following a Rademacher distribution. We want to check whether the probability of a bad event, where this sum (error) exceeds a suitably high threshold, decays quickly as the threshold increases. This is related to Tomaszewski's conjecture, and in particular, we now know that

$$\mathcal{P}(\sum_i \mathbf{x}[i]\epsilon[i] > t\,\|\epsilon\|_2) < \exp(-t^2/2) \quad \text{and} \quad \mathcal{P}(|\sum_i \mathbf{x}[i]\epsilon[i]| > t\,\|\epsilon\|_2) < 2\exp(-t^2/2)$$

implying an exponential decay in the probability of the error exceeding a threshold as desired. By incorporating the maximal possible error, denoted as $\mathbf{c}[i]$, we obtain:

$$\mathcal{P}(|\sum_i \mathbf{x}[i]\epsilon[i]| > t\,\|\mathbf{c}\|_2) < 2\exp(-t^2/2)$$

Since $-\mathbf{c}[i] \leq \epsilon[i] \leq \mathbf{c}[i]$, we can also use Hoeffding's inequality, which is less tight but has a similar form of dependency.

However, the concentration bounds do not adequately explain the strong empirical behavior. To assess impact on the impact on training, if desired, we can use results such as the one in (De Sa et al., 2018) for convergence analysis but with some modifications to the optimization. For example, if we use LP-SVRG in (De Sa et al., 2018) and just use stochastic rounding instead of deterministic RTN (fixed point arithmetic in (De Sa et al., 2018) is the same as the integer arithmetic with scaling that we use in (2)), then under the assumption that the objective function is μ-strongly convex with respect to parameters (and an additional L-Lipschitz requirement), with an additional cost of variance reduction, we can get a linear convergence rate.

A.3. Experimental Details and Additional Results

Additional details of training experiments. We run all of our experiments on NVIDIA RTX 3090. Below we provide the training hyperparameters. RoBERTa-Small is a 4-layer Transformer encoder whose model dimension is 512, hidden dimension is 2048, and number of heads is 8. For RoBERTa-Small models, we train each model for 200K steps with batches of 256 512-length sequences. We use an AdamW optimizer with 1e-4 learning rate, 10,000 warm-up steps, 0.01 weight decay, and linear decay. For RoBERTa-Base models, we train each model for 300K steps with batches of 128 512-length sequences. We use an AdamW optimizer with 5e-5 learning rate, 10,000 warm-up steps, 0.01 weight decay, and linear decay. We use timm to train our ViT-Small models. The hyperparameters of all experiments are the same: batch size 1024, optimizer AdamW, learning rate 0.001, weight decay 0.05, augmentation rand-m9-mstd0.5-inc1, mixup 0.8, cutmix 1.0.

Unpack Ratios of ViT-Large. Similar to Tab. 9 in the main paper, we also evaluate the unpack ratios of ViT-Large, which are shown in Tab. 13. The overall results are similar to what was observed in unpack ratios of LLaMA-7B (Tab. 9).

More Empirical Results on LLM Quantization. To evaluate how well RTN works for inference of different models and different model sizes, beside the experiments shown in the main text, we also run experiments on LLaMA-13B (Touvron et al., 2023), Mistral-7B (Jiang et al., 2023), and Phi-2 (Javaheripi et al., 2023). The results are summarized in Tab. 14, Tab. 15, and Tab. 16. To minimize code change, we only evaluate the quantization of linear layers, consistent with quantization-focused papers for Mistral-7B and and Phi-2.

More Empirical Results on Training. We wanted to understand how well RTN works for training of larger models but wanted to avoid allocating a significant compute budget to perform such an experiment. So, we resorted to finetuning a T5-Large model on the first 50K instance of the XSum summarization dataset (Narayan et al., 2018) using BF16 and RTN, and show the results in Fig. 9. The validation metrics are shown in Tab. 17. We can draw a similar conclusion as in the main paper that RTN quantized training gives similar results to BF16 training.

Table 13. Averaged unpack ratios of each type of GEMMs in ViT-Large: linear layers (computing $\mathbf{Y}$), attention score (computing $\mathbf{P}$), and attention output (computing $\mathbf{O}$) when using different unpack strategies and integer bit length b under quantization β settings. AS: Attention Score, AO: Attention Output.

	β				5			7			15		
	Integer Bits b				3	4	5	3	4	5	4	5	6
Linear ($\mathbf{Y}$)	$\mathbf{X}$	Row	$\mathbf{W}$	Row	2.90	2.00	1.55	3.01	2.38	1.59	2.63	2.22	1.54
		Row		Col	10.97	2.32	1.56	12.34	4.12	1.65	10.46	4.31	1.62
		Row		Both	6.24	2.08	1.55	6.84	2.82	1.60	6.22	2.76	1.56
		Col		Row	2.33	1.39	1.20	3.38	1.51	1.26	3.06	1.46	1.25
		Col		Col	8.89	1.64	1.22	13.97	2.63	1.32	12.22	2.81	1.32
		Col		Both	4.99	1.44	1.20	7.99	1.76	1.27	7.59	1.78	1.26
	Mix				2.60	1.27	1.06	2.44	1.40	1.15	2.10	1.42	1.16
AS ($\mathbf{P}$)	$\mathbf{Q}$	Row	$\mathbf{K}$	Row	1.84	1.07	1.00	2.01	1.35	1.00	1.99	1.40	1.00
		Row		Col	3.06	1.07	1.00	6.38	1.39	1.00	6.34	1.46	1.00
		Col		Row	1.34	1.01	1.00	2.50	1.04	1.00	2.49	1.05	1.00
		Col		Col	2.39	1.01	1.00	8.25	1.08	1.00	8.24	1.10	1.00
	Mix				1.33	1.01	1.00	1.91	1.04	1.00	1.90	1.04	1.00
AO ($\mathbf{O}$)	$\mathbf{M}$	Row	$\mathbf{V}$	Row	2.84	2.07	1.65	3.07	2.24	1.80	2.56	2.11	1.78
		Row		Col	5.78	2.12	1.65	11.12	2.47	1.81	9.22	2.35	1.79
		Col		Row	3.98	2.26	1.64	4.69	2.57	1.81	3.58	2.33	1.77
		Col		Col	8.42	2.29	1.64	16.97	2.83	1.81	12.92	2.60	1.77
	Mix				2.25	1.61	1.32	2.55	1.77	1.42	2.22	1.70	1.42

Table 14. Baseline comparison on LLaMA-13B when quantize computation in all linear layers.

Method	β	Type	ARC-c	ARC-e	BoolQ	HellaSwag	PIQA	WinoGrande
Full-Precision	-	BF16	48.0	79.5	80.6	60.0	79.2	72.1
SmoothQuant	-	INT8	45.5	76.3	76.5	58.0	78.0	72.1
	-	INT4	25.1	49.9	57.6	56.0	61.3	52.6
LLM-FP4	-	FP4	39.9	71.7	71.9	53.3	74.8	66.7
RTN	5	INT	37.6	70.0	69.1	51.9	72.4	64.6
	7	INT	44.1	76.1	73.5	57.3	76.7	67.6
	15	INT	46.9	78.8	79.4	59.0	78.2	72.5
	31	INT	48.0	79.7	80.2	59.9	78.0	71.3

Table 15. Baseline comparison on LLaMA-13B when quantize all GEMMs in a Transformer.

Method	β	Type	ARC-c	ARC-e	BoolQ	HellaSwag	PIQA	WinoGrande
Full-Precision	-	BF16	48.0	79.5	80.6	60.0	79.2	72.1
RTN	5	INT	25.1	44.4	54.8	37.7	57.9	52.0
	7	INT	38.0	66.9	70.1	53.3	72.5	64.2
	15	INT	45.9	77.6	80.0	59.5	77.5	71.5
	31	INT	47.9	79.3	80.0	60.5	78.6	70.9

Table 16. RTN performance on Mistral-7B and Phi-2 when quantize computation in all linear layers.

	Method	β	ARC-c	ARC-e	BoolQ	HellaSwag	PIQA	WinoGrande
Mistral-7B	Full-Precision	-	50.3	80.9	83.6	61.3	80.7	73.8
	RTN	5	38.1	70.5	69.9	53.9	73.3	61.4
		7	44.9	75.0	76.0	58.7	77.8	68.6
		15	48.8	79.7	80.3	60.8	79.6	73.2
		31	50.3	80.1	83.5	61.5	80.7	74.4
Phi-2	Full-Precision	-	20.6	26.1	41.3	25.8	54.3	49.3
	RTN	5	22.1	26.7	41.5	25.6	52.3	48.1
		7	21.3	25.8	40.9	25.8	53.9	49.5
		15	21.3	27.3	45.4	25.8	53.4	48.8
		31	21.0	25.8	40.8	25.7	53.0	50.7

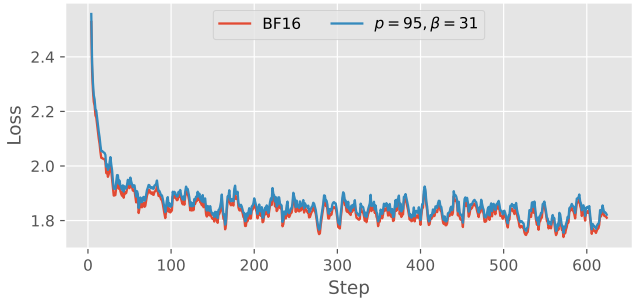

Figure 9. Loss curves of T5-Large finetuning on 1/4 of XSum dataset for 1 epoch.

Table 17. Validation metrics of T5-Large finetuning on 1/4 of XSum dataset for 1 epoch.

Method	β	Type	Loss	Rouge1	Rouge2	Rougel	Rougelsum
Full-Precision	-	BF16	1.65	36.12	13.00	29.21	29.20
RTN	31	INT	1.66	36.03	13.83	29.03	29.04

Table 18. Inference: end to end baseline comparison combining information from Tab. 2 and Tab. 9 for LLaMA-7B.

Method	β	Type	r	bits × $r/16$	ARC-c	ARC-e	BoolQ	HS	PIQA	WG
Full-Precision	-	BF16	1	1	43.1	76.3	77.8	57.2	78.0	68.8
LLM.int8()	-	INT8+FP16	1	> 0.5	43.8	75.5	77.8	57.4	77.6	68.7
SmoothQuant	-	INT8	1	0.5	37.4	74.4	74.0	55.0	77.5	69.6
LLM-QAT	-	INT4	1	0.25	30.2	50.3	63.5	55.6	64.3	52.9
LLM-FP4	-	FP4	1	0.25	33.6	65.9	64.2	47.8	73.5	63.7
RTN+IMUnpack	5	INT4	1.27	0.318	39.3	72.8	69.9	53.4	74.9	66.4
	5	INT5	1.06	0.331	39.3	72.8	69.9	53.4	74.9	66.4
	7	INT4	1.41	0.352	42.6	73.9	72.3	55.9	77.0	67.4
	7	INT5	1.14	0.356	42.6	73.9	72.3	55.9	77.0	67.4
	11	INT5	1.27	0.397	43.9	76.1	77.3	56.3	77.3	69.3
	11	INT6	1.07	0.401	43.9	76.1	77.3	56.3	77.3	69.3
	15	INT5	1.40	0.438	43.0	75.7	77.5	57.0	78.0	69.2
	15	INT6	1.15	0.431	43.0	75.7	77.5	57.0	78.0	69.2
	31	INT6	1.42	0.532	42.7	76.1	76.1	57.3	77.3	69.3
	31	INT7	1.16	0.507	42.7	76.1	76.1	57.3	77.3	69.3

End-to-End Quantization Baseline Comparison. An end-to-end baseline comparison requires combining information from Tab. 1, Tab. 2, and Tab. 3 with Tab. 9 and Tab. 13. As we change the bit-width of integers that we will use, the unpack ratio r can be determined from Tab. 9 and 13. Since after unpacking, the calculation is similar or the same as standard GEMMs (except that the input sizes are different), we could first estimate the runtime of GEMMs, and discuss the runtime overhead of unpacking later.

Note that INT2, INT3, INT5, INT6, INT7 GEMM implementations are not yet publicly available. Although INT4 and INT8 are now available in NVIDIA's CUTLASS, their integration (e.g., in PyTorch) is ongoing (more discussion below). As a result, we can only estimate the runtime of these GEMMs based on publicly available information. According to NVIDIA, INT4 offers a 4× performance bump compared to FP16, and INT8 is 2× performance compared to FP16. We measured the runtime of INT4 and INT8 GEMMs in CUTLASS standalone, and the performance is indeed 4× and 2×, respectively, compared to FP16. We can reasonably assume that the performance of x-bit INT GEMMs can approach $\frac{16}{x}$ performance compared to FP16, and the runtime approaches $\frac{x}{16}$ of the runtime of FP16. Further, the runtime of GEMMs is linearly proportional to $n \times d \times h$ for matrices of size $n \times d$ and $h \times d$. When comparing runtime of GEMMs for different sizes, the ratio $r = \frac{n \times d \times h}{n' \times d' \times h'}$ can give an accurate estimate of the runtime comparison when these GEMMs finally become available (we verified this linear relationship in FP16 GEMMs for a contiguous range of matrix sizes, not just a power of 2). As a result, we can use $\frac{xr}{16}$ as a proxy for the runtime of our GEMMs. In Tab. 18, we provided $\frac{xr}{16}$ values (bits × $r/16$ column) for comparing across different baselines and the corresponding model accuracy. We see that our method has better model accuracy with faster runtimes under the proposed proxy when not accounting for the runtime overhead of quantization.

The only remaining task is to check the runtime overhead of quantization and unpacking. Since the goal is to speed up

Table 19. Runtime overhead of UnpackColumn and UnpackRow compared to GEMM and Clone for different bit-widths, unpack ratios r, and input matrix shapes.

Shape $n \times d \times h$	GEMM FP32	GEMM FP16	Clone FP32	UnpackCol FP32						UnpackRow FP32					
Bit Width	-	-	-	2		4		8		2		4		8	
Unpack Ratio	-	-	-	1.21	1.44	1.21	1.44	1.21	1.44	1.21	1.44	1.21	1.44	1.21	1.44
$2^{13} \times 2^{13} \times 2^{13}$	63.4	5.0	0.8	1.0	0.9	1.0	1.0	1.1	1.1	1.1	1.1	1.0	1.0	1.0	1.0
$2^{13} \times 2^{13} \times 2^{14}$	116.9	10.4	1.2	1.4	1.3	1.4	1.4	1.5	1.5	1.5	1.4	1.4	1.4	1.5	1.5
$2^{13} \times 2^{13} \times 2^{15}$	252.7	25.1	2.0	2.2	2.2	2.2	2.2	2.4	2.5	2.5	2.2	2.3	2.3	2.4	2.4
$2^{14} \times 2^{13} \times 2^{13}$	126.3	10.5	1.2	1.4	1.4	1.4	1.4	1.5	1.5	1.5	1.4	1.4	1.4	1.5	1.5
$2^{14} \times 2^{13} \times 2^{14}$	253.8	20.9	1.6	1.8	1.8	1.8	1.8	2.0	2.0	2.0	1.8	1.8	1.9	1.9	2.0
$2^{14} \times 2^{13} \times 2^{15}$	510.8	60.1	2.4	2.6	2.6	2.6	2.7	2.9	3.0	3.0	2.6	2.7	2.7	2.9	2.9
$2^{15} \times 2^{13} \times 2^{13}$	255.5	20.9	2.0	2.2	2.2	2.2	2.2	2.4	2.5	2.4	2.2	2.3	2.3	2.4	2.5
$2^{15} \times 2^{13} \times 2^{14}$	514.3	60.0	2.4	2.6	2.6	2.6	2.7	2.9	2.9	3.0	2.6	2.7	2.7	2.9	2.9
$2^{15} \times 2^{13} \times 2^{15}$	957.7	121.5	3.2	3.4	3.4	3.5	3.5	3.8	3.9	3.9	3.5	3.6	3.6	3.8	3.9

Table 20. Inference: estimated speedup of the entire model for LLaMA-7B.

Method	β	Type	r	bits $\times\, r/16$	Speedup	ARC-c	ARC-e	BoolQ	HS	PIQA	WG
Full-Precision	-	BF16	1	1	-	43.1	76.3	77.8	57.2	78.0	68.8
RTN+IMUnpack	5	INT4	1.27	0.318	82%	39.3	72.8	69.9	53.4	74.9	66.4
	7	INT4	1.41	0.352	74%	42.6	73.9	72.3	55.9	77.0	67.4
	11	INT5	1.27	0.397	64%	43.9	76.1	77.3	56.3	77.3	69.3
	15	INT6	1.15	0.431	57%	43.0	75.7	77.5	57.0	78.0	69.2
	31	INT7	1.16	0.507	44%	42.7	76.1	76.1	57.3	77.3	69.3

GEMMs, the quantization overhead must be small or else, the overhead will eliminate the gain of faster GEMMs. As a result, the code needs to be implemented as custom CUDA kernels to minimize the overhead. We implemented UnpackRow and UnpackCol cuda kernels for FP32 matrices A and B. The supported unpack bit widths are 2, 4, 8 by packing the bit representation of 16 INT2 or 8 INT4 or 4 INT8 into an INT32. For other bit-widths, an efficient implementation would need additional hardware support. We expect the runtime of these kernels would be halved if the input date type is FP16.

We compare these kernels to the tensor.clone() operation. The clone operation is the simplest pytorch operation that copies the values of a memory chunk (tensor) to another memory chunk (new tensor). The scaling and rounding operations in common quantization methods share a similar runtime as clone operation. We profiled UnpackRow and UnpackCol kernels (scaling and rounding are also computed within the kernel) for matrices of different sizes, bit-widths, and unpack ratio r. The runtime overhead of quantization and unpacking together is only between 1× to 1.5× of the runtime of clone operation applied to the matrices of the same size. The runtime ratio approaches 1 as the matrix size increases! We note that the kernels are still not fully optimized, and therefore, further code optimization might further lower the runtime. The overhead is very small relative to clone operation, so any other quantization scheme is unlikely to be much faster than our method in terms of overhead. For estimating the overhead compared to GEMMs, we also provide runtime for FP32 GEMM and FP16 GEMM for these matrix sizes alongside the runtimes for clone, UnpackRow, and UnpackCol as shown in Tab. 19.

Estimated Overall Speedup of Entire Models. We have carefully calculated the speedup of the entire model runtime. We measure the breakdowns of different operators in LLaMA-7B and LLaMA-70B for an input of shape [16, 512] where 16 is the batch size and 512 is the sequence length. The overall model runtime of LLaMA-7B and LLaMA-70B is 1.84s and 18.92s, respectively. The overall runtime of GEMMs for Linear layers accounts for 77.32% and 88.96% of the overall model runtime, respectively. We exclude GEMMs in attention computation for simplicity which are accounted for in the much smaller 22.68% and 11.04% of the model's runtime (this percentage includes GEMMs in attention computation, activation functions, layer norm, residual connection, etc.). Let p be the percentage spent of GEMMs for Linear layers and h be the runtime ratio of the runtime overhead of IM-Unpack compared to the runtime of GEMMs, if we use $\frac{x}{16}$ as runtime of INT-x GEMM compared to FP16 GEMMs, we estimate the efficiency gain of using INT-x for model computation. The speedup of the overall model runtime can be estimated via $\frac{1}{(\frac{xr}{16}+h)p+(1-p)}$ using Amdahl's law. We know from Tab. 19 that $h < 0.1$, so we can populate the speedup of the overall model runtime in Tab. 20 using the worst possible overhead $h = 0.1$. Since hardware and software support for different bit-widths is not publicly available, this estimate is based on calculations using actual measurements and profiling.