Extremely Low Bit Transformer Quantization for On-Device Neural Machine Translation

Insoo Chung[*] **Byeongwook Kim**[*] **Yoonjung Choi** **Se Jung Kwon**
Yongkweon Jeon **Baeseong Park** **Sangha Kim** **Dongsoo Lee**
Samsung Research, Seoul, Republic of Korea
{insooo.chung, byeonguk.kim, yj0807.choi, sejung0.kwon, dragwon.jeon, bpbs.park, sangha01.kim, dongsoo3.lee}@samsung.com

Abstract

The deployment of widely used Transformer architecture is challenging because of heavy computation load and memory overhead during inference, especially when the target device is limited in computational resources such as mobile or edge devices. Quantization is an effective technique to address such challenges. Our analysis shows that for a given number of quantization bits, each block of Transformer contributes to translation quality and inference computations in different manners. Moreover, even inside an embedding block, each word presents vastly different contributions. Correspondingly, we propose a mixed precision quantization strategy to represent Transformer weights by an extremely low number of bits (e.g., under 3 bits). For example, for each word in an embedding block, we assign different quantization bits based on statistical property. Our quantized Transformer model achieves 11.8× smaller model size than the baseline model, with less than -0.5 BLEU. We achieve 8.3× reduction in run-time memory footprints and 3.5× speed up (Galaxy N10+) such that our proposed compression strategy enables efficient implementation for on-device NMT.

1 Introduction

Transformer (Vaswani et al., 2017) is one of the state-of-the-art approaches for Neural Machine Translation (NMT), and hence, being widely accepted. For example, in WMT19 machine translation tasks, it is reported that 80% of submitted systems have adopted the Transformer architecture (Barrault et al., 2019). Note that high translation quality of Transformer models entails a large number of parameters. Moreover, the Transformer model is inherently much slower than conventional machine translation approaches (e.g., statistical approaches) mainly due to the auto-regressive inference scheme (Graves, 2013) incrementally generating each token. As a result, deploying the Transformer model to mobile devices with limited resources involves numerous practical implementation issues.

To address such implementation challenges with little degradation in translation quality, we study a low-bit quantization strategy for Transformer to accomplish high-performance on-device NMT. We note that most previous studies to compress Transformer models utilize uniform quantization (e.g. INT8 or INT4). While uniform quantization may be effective for memory footprint savings, it would face various issues to improve inference time and to maintain reasonable BLEU score. For example, even integer arithmetic units for inference operations present limited speed up (Bhandare et al., 2019) and resulting BLEU score of quantized Transformer can be substantially degraded with low-bit quantization such as INT4 (Prato et al., 2019).

While determining the number of quantization bits for Transformer, it is crucial to consider that each component of Transformer may exhibit varied sensitivity of quantization error toward degradation in translation quality (Wang and Zhang, 2020). Accordingly, a mixed precision quantization can be suggested as an effort to assign different numbers of quantization bits depending on how each component after quantization is sensitive to the loss function. In addition, as we illustrate later, even assigning different quantization bits for each row of an embedding block can further reduce the overall number of quantization bits of the entire Transformer model. Our proposed quantization strategy, thus, provides a finer-grained mixed precision approach compared to previous methods, such as (Dong et al., 2019; Wu et al., 2018; Zhou et al.,

[*] Equal Contribution.

2017; Wang and Zhang, 2020) that consider layer-wise or matrix-wise mixed precision.

Accommodating distinguished implementation properties (e.g., latency and translation quality drop) of each component in Transformer, we propose the following methodologies to decide precision of a block: 1) in the case of embedding block, statistical importance of each word is taken into account and 2) for encoder and decoder blocks, sensitivity of each quantized sub-layer is considered. The main contributions of this paper are as follows:

- We propose a mixed precision quantization strategy while embedding block allows another level of mixed precision in word level according to statistical properties of natural language.
- Our proposed quantization scheme allows the number of quantization bits to be as low as under 3 bits for the Transformer with little BLEU score degradation (under -0.5 BLEU).
- We demonstrate that our quantization technique reduces a significant amount of run-time memory and enhances inference speed so as to enable fast on-device machine translation by large Transformer models.

2 Background

2.1 Transformer

Transformer adopts an an encoder-decoder architecture (Cho et al., 2014) composed of three different blocks: encoder, decoder and embedding that account for 31.0%, 41.4%, and 27.6%, respectively, in terms of the number of parameters in a Transformer base model. An embedding block is a single weight matrix that serves multiple purposes in the Transformer. For example, each row in the embedding block represents a word in a bi-lingual vocabulary. Another purpose of the embedding block is to serve as a linear transformation layer which converts decoder outputs to next token probabilities as suggested in Press and Wolf (2017). Encoder and decoder blocks are composed of multiple layers while each layer employs attention and feed-forward sub-layers.

Due to auto-regressive operations during inference of Transformer (Graves, 2013), the correlation between the number of operations and the number of parameters can be vastly different for each component. Based on such different correlations, Transformer's inference scheme can be divided into encoding steps of high parallelism and decoding steps of low parallelism. As for encoding steps, given a sequence in the source language, a single forward propagation of the encoder produces a sequence of hidden representations for all words in a given sequence. In each decoding step, decoder and embedding blocks produce a probability distribution of possible words, *one word at a time*. Unlike encoding steps, the computation of decoding steps is not parallelizable because each decoding step depends on outputs of all prior decoding steps.

Note that such lack of parallelism during decoding steps potentially induces the memory wall problem in practice with commodity hardware; parameters of decoder and embedding blocks are required to be loaded to cache and unloaded from the cache repeatedly throughout decoding steps. Furthermore, an embedding block is usually represented by a significantly large matrix that also incurs the memory wall problem (Jeon et al., 2020).

2.2 Non-uniform Quantization Based on Binary-codes

Quantization approximates full precision parameters in neural networks by using a small number of bits (Gong et al., 2014; Rastegari et al., 2016; Guo et al., 2017; Jacob et al., 2018). One of widely adopted quantization methods is uniform quantization. Uniform quantization performs mapping of full precision parameters into one of 2^q values ranging from 0 to 2^q-1 that correspond to a range between the minimum and the maximum full precision parameters, where q denotes the number of quantization bits. Lower precision can reduce the computation cost of arithmetic operation such as multiplication and addition only if all inputs to arithmetic operations (i.e., activations) are also quantized (Jacob et al., 2018). Furthermore, high quantization error may occur when a parameter distribution involves extreme outliers (Zhao et al., 2019).

As such, non-uniform quantization methods are being actively studied to better preserve expected value of parameters which is critical to maintaining model accuracy (Courbariaux et al., 2015). By large, non-uniform quantization methods include codebook-based quantization and binary-code based quantization. Even though codebook-based quantization reduces off-chip memory footprint, computational complexity is not reduced

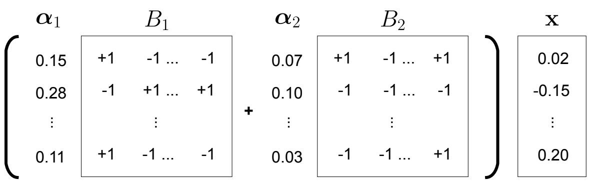

Figure 1: In 2-bit binary-code based quantization, each row of W is approximated to 2 sets of binary code weights ($\{B_1, B_2\}$) and 2 vectors of full precision scales ($\{\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2\}$).

at all because of mandatory dequantization procedure during inference (Stock et al., 2020; Guo, 2018). On the other hand, quantization based on binary-code ($\in\{-1, +1\}$) can achieve both high compression ratio and efficient computation (Rastegari et al., 2016; Guo et al., 2017; Xu et al., 2018; Jeon et al., 2020).

In this paper, we adopt non-uniform binary-code based quantization as our method of quantization. Non-uniform quantization based on binary-code maps a full precision vector $\mathbf{w}\in\mathbb{R}^p$ to a scaling factor $\alpha_i\in\mathbb{R}$, and a binary vector $\mathbf{b}_i\in\{-1, +1\}^p$, where $(1\leq i\leq q)$. Note that p is the length of a vector and q denotes the number of quantization bits. Then, $\mathbf{w}$ is approximated as $\sum_{i=1}^{q} \alpha_i \mathbf{b}_i$. Scaling factors and binary vectors are obtained as follows:

$$arg \min_{\alpha_i, \mathbf{b}_i} \left\| \mathbf{w} - \sum_{i=1}^{q} \alpha_i \mathbf{b}_i \right\|^2 \tag{1}$$

To minimize the quantization error formulated in Eq. 1, heuristic approaches have been proposed (Guo et al., 2017; Xu et al., 2018).

For matrix quantization, the binary-code based quantization can be simply applied to each row or column of a matrix. With a matrix quantized into binary matrices $\{B_1, B_2, ..., B_q\}$ and scaling factor vectors $\{\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, ..., \boldsymbol{\alpha}_q\}$, the matrix multiplication with full precision vector $\mathbf{x}$ produces an output vector $\mathbf{y}$ as follows:

$$\mathbf{y} = \sum_{i=1}^{q} (\boldsymbol{\alpha}_i \circ (B_i \cdot \mathbf{x})), \tag{2}$$

where the operation $\circ$ denotes element-wise multiplication. Figure 1 is an illustration of Eq. 2. Intermediate results of $B_i \cdot \mathbf{x}$ can be pre-computed for further compute-efficiency (Jeon et al., 2020). This allows the efficient matrix multiplication of quantized Transformer weights and full precision activation.

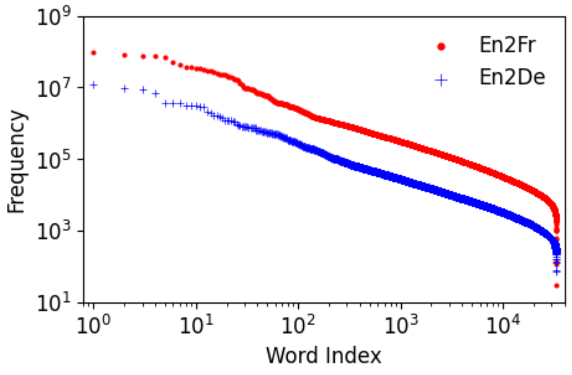

Figure 2: The distributions of word frequency in trainsets. Word indices range from 1 to 32768 where words are sorted in descending order of frequency.

3 Quantization Strategy for Transformer

For Transformer, we suggest the following two techniques to decide the number of quantization bits for each block: 1) in the case of embedding block, frequency of each word is taken into account and 2) for encoder and decoder blocks, we find the minimum number of quantization bits for each type of sub-layers that allows reasonable degradation in BLEU score after quantization.

3.1 Embedding

It has been reported that the word frequency distribution can be approximated as power-law distribution (Chen et al., 2018). Such power-law distribution is illustrated in Figure 2 that presents word frequency distribution in WMT14 datasets. Note that 1% of word vectors account for around 95% of word frequency for both En2Fr and En2De. Intuitively, if word vectors are compressed by the same compression ratio, then word vectors with high frequency in a corpus would result in higher training loss after compression, compared to word vectors with low frequency. Chen et al. (2018) utilizes frequency to provide different compression ratios in different groups of words using low-rank approximation. To the best of our knowledge, word frequency has not yet been considered for Transformer quantization.

We assume that highly skewed word frequency distribution would lead to a wide distribution of the number of quantization bits per word. In such a case, an embedding block may require a substantially high number of quantization bits that would be the maximum in the distribution of the number of quantization bits per word. For example, even though Wang and Zhang (2020) successfully quantized the parameters in attention and feed-forward sub-layers of the BERT architecture (Devlin et al., 2018) into 2-4 bits, 8 bits or higher number of bits were used to represent a parameter in the embedding block.

Algorithm 1: Embedding quantization

```
   Input  : Embedding matrix $E$ of shape
            $[v, d_{model}]$; number of clusters $b$;
            the ratio factor $r$;
   Output : Quantized representation $\hat{E}$
 1 Sort $E$ in descending order of word
     frequency ;
 2 $idx = 0$ ;
 3 for $i = 0...b-1$ do
 4     Compute number of word-vectors in $i$-th
         cluster, $c^i_{size} = \frac{v}{\sum_{k=0}^{b-1} r^k} \cdot r^i$ ;
 5     Compute target bit-precision for $i$-th
         cluster, $c^i_{bit} = b - i$ ;
 6     for $j = 0...c^i_{size}$ do
 7         Initialize $w_{idx}$ = $idx$-th row of $E$ ;
 8         Quantize word vector $w$ to $c^i_{bit}$ bit,
             $\hat{w}_{idx} = quantize(w, c^i_{bit})$ ;
 9         Increment $idx$ by 1 ;
10     end for
11 end for
12 Output: $\hat{E} = \{\hat{w}_0, \hat{w}_1, ..., \hat{w}_{v-1}\}$
```

The underlying principle to quantize embedding blocks is that the number of quantization bits for each word vector is proportional to the frequency in a corpus. To assign a low number of quantization bits to most of the words under such a principle, first, we group word vectors into clusters according to word frequency. r acts as an exponential factor in deciding the number of word vectors in each cluster as in line 4 of Algorithm 1. b denotes the number of clusters and acts as a variable for quantization bits such as line 5 of Algorithm 1. For example, with b=4 and r=2, word vectors are clustered into clusters of ratio r^0:r^1:r^2:r^3=1:2:4:8, then assigned bits as much as $\{b, b-1, b-2, b-3\} = \{4, 3, 2, 1\}$. We empirically set b=4 for all of our embedding quantization experiments.

Figure 3 shows our experimental results with $r \in \{2, 4, 8\}$. For r=2 , the average number of quantization bits in the embedding block is 1.73, and for r=4, it becomes 1.32. With our embedding quantization method, higher translation quality in

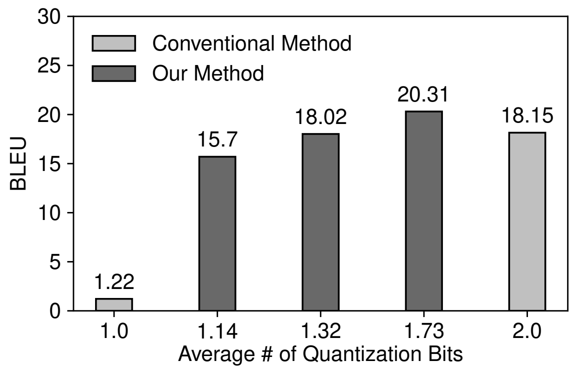

Figure 3: Detokenized-BLEU (beam=4) on newstest2013 after quantizing the embedding block without retraining.

terms of BLEU score can be achieved with lower number of quantization bits as compared to the conventional quantization methods that assign the same number of quantization bits to all word vectors. For example, the Transformer model with 1.73-bit quantized embedding produces more accurate translations than the model with conventional (fixed) 2-bit quantized embedding block.

Algorithm 1 assigns 1-bit to the largest cluster. For example, using b=4 and r=8, 87.5% of word vectors in the embedding block are quantized to 1-bit. We benefit from 1-bit word vectors in terms of inference speed because memory overhead at matrix multiplications of embedding blocks is potentially minimized. One concern is that 1-bit word vectors may degrade translation performance in a way that is not shown with BLEU score. We address such concerns in Section 4.4 and demonstrate that 1-bit word vectors do not limit the quantized model's abilities to predict the subsequent tokens.

3.2 Encoder and Decoder

Each type of sub-layers in the Transformer yields a wide range of sensitivity to quantization error, and thus, to translation quality drop. Table 1 lists measured BLEU scores with various types of sub-layers quantized into different numbers of quantization bits[1]. For each type of sub-layers, we carefully select the number of quantization bits such that the model with quantized sub-layers is able to report reasonable degradation in the BLEU score compared to the baseline.

[1] Emb, Enc, and Dec denote the embedding block, the encoder block, and the decoder block, respectively. ee, ed, and dd denote the encoder-encoder(encoder self), encoder-decoder, and decoder-decoder(decoder self) attention, respectively. ffn denotes the feed forward sub-layer.

Layer	# of bits				Avg.
	4	3	2	1	Deg.
$\boldsymbol{Emb}$	22.9	22.4	19.0	1.0	-9.1
$\boldsymbol{Enc}$	24.6	24.0	20.6	1.3	-7.8
Enc_{ee}	25.3	24.8	23.7	13.6	-3.6
Enc_{ffn}	24.9	24.8	23.1	4.3	-6.2
$\boldsymbol{Dec}$	24.7	23.6	11.1	0.1	-10.6
Dec_{dd}	25.2	25.1	24.8	17.8	-2.2
Dec_{ed}	25.1	24.7	20.6	2.0	-7.3
Dec_{ffn}	25.0	24.9	24.4	17.6	-2.5

Table 1: BLEU measurements from applying quantization to each block and to a type of sub-layers in En2De base model (25.4 BLEU in full precision) without retraining. Avg. Deg. denotes the average BLEU degradation from quantizing each block or a type of sub-layers to 4, 3, 2 and 1 bit. Reported scores are measured in detokenized-BLEU (beam=1, newstest2013, sacrebleu) setting explained in Section 4.2.

Within the decoder block, Dec_{ed} sub-layers are more sensitive by quantization than the other sub-layers, which is aligned with reports of Michel et al. (2019). It is interesting that even though the number of parameters in Dec_{ffn} sub-layers is $2\times$ that of Dec_{ed} sub-layers, BLEU score degradation is greater when Dec_{ed} sub-layers are quantized. Among the sub-layers in the encoder block, Enc_{ffn} sub-layers are more sensitive by quantization than Enc_{ee} sub-layers. Based on such sensitivity analysis, we assign a proper number of quantization bits to each sub-layer in the encoder and decoder blocks.

Another vital aspect to consider is the inference efficiency of quantized Transformer models. As mentioned in Section 2, the auto-regressive nature of the Transformer's inference limits the amount of parallelism in the decoder forward propagation and induces a memory wall problem during inference. Therefore, in order to enable fast on-device NMT, we assign a lower number of bits to the decoder block compared to the encoder block.

4 Experiments

4.1 Quantization Details

Before we present our compression results, we describe our quantization method and retraining algorithm in detail.

Methodology To quantize weights in the Transformer with high performance during retraining, we adopt the Greedy approximation algorithm introduced in (Guo et al., 2017) due to its computational simplicity. In our experiments, we first train the base configuration of the Transformer. Next, we retrain the full precision parameters[2] while periodically quantizing model parameters to retain the translation quality. For retraining, we adopt Non-Regularization period (pNR) as a way to control regularization strength while the best period is empirically obtained (Lee et al., 2020). Variable pNR is investigated for our retraining, which denotes the number of mini-batch updates before the quantization is performed. For example for pNR=1000, we first apply quantization to target Transformer weights, and perform 1000 steps of retraining before quantizing the weights again (i.e, the quantization procedure is periodically executed in an interval of 1000 steps during retraining.). The advantage of adopting pNR is reduced retraining time, as computation overheads induced by quantization are divided by pNR.

Retraining Details Our quantization baselines are retrained warm-starting from our full precision baseline. Note that during the retraining, quantization is applied to all layers of the Transformer model every pNR steps where pNR=2000. Quantization baselines are retrained for 400k steps by using 4$\times$V100 GPUs taking around 1.7 days. Our quantized models are retrained over 3 phases in the order of embedding, decoder, and encoder block; each phase warm-starts from the previous phase. Note that in each phase, compressed blocks of previous phases are also targeted for quantization. For each phase, we use pNR=1000. We train our quantized models for 300k steps/phase and full retraining time is around 3.8 days with 4$\times$V100 GPUs. The reasoning behind the choices of the pNR values and the number of retraining steps is further supported in Appendix A.4

Quantized Parameters Our quantization strategy targets weight matrices that incur heavy matrix multiplications. Targeted weight matrices account for 99.9% of the number of parameters in the Transformer architecture and 99.3% of on-device inference latency (Table 4). We quantize each row of W as in Figure 1, assuming matrix multiplication

[2]In our experiments, full precision parameters are represented by the standard IEEE 32-bit floating-point data format.

Average # of Bits Emb, Dec, Enc	Model[2]	BLEU(beam=1) En2De	 En2Fr	 En2Jp	BLEU(beam=4) En2De	 En2Fr	 En2Jp	Model Size(MB)
FP baseline	32.0	26.7	39.1	25.1	27.5	39.5	26.3	237.8(1.0×)
3-bit baseline	3.0	26.0	38.0	25.3	26.9	38.6	25.9	23.7(10.0×)
2-bit baseline	2.0	23.9	35.4	22.8	24.4	36.1	23.7	15.9(15.0×)
2-bit Emb. baseline	23.7	26.0	N/A	N/A	26.8	N/A	N/A	176.1
2.5[1], FP, FP	23.9	26.7	39.1	25.2	27.6	39.5	25.7	177.7
1.3[1], FP, FP	23.5	26.4	38.7	24.5	27.0	39.3	24.9	175.2
1.1[1], FP, FP	23.5	25.7	38.8	24.7	26.9	39.4	25.3	174.8
2.5, 1.8, FP	11.3	25.9	38.4	24.9	27.0	38.8	25.4	85.1
1.3, 1.8, FP	11.0	25.6	38.1	24.5	26.8	38.8	25.1	82.5
1.1, 1.8, FP	11.0	25.1	37.5	24.6	26.3	38.6	24.8	82.2
2.5, 1.8, 3.7[3]	2.6	26.2	38.6	25.3	27.1	39.2	26.1	20.2(11.8×)
1.3, 1.8, 3.7[3]	2.2	25.6	38.3	24.7	25.9	38.9	25.5	17.6(13.5×)
1.1, 1.8, 3.7[3]	2.2	25.3	38.3	24.6	26.0	39.0	25.3	17.2(13.8×)

[1] For 2.5, 1.3 and 1.1-bit embeddings, Algorithm 1 with b=4, r=1, 4, 8 is applied respectively.
[2] Model column lists the average number of bits in each model.
[3] Average scores over 3 retraining runs are reported for the last retraining phases.

Table 2: Tokenized-BLEU (beam∈{1, 4}, newstest2014, multi-bleu) and compression ratio of baseline models and quantized models using proposed quantization strategy. We report BLEU and model size for each retraining phase. All model parameters are included in the reported model size and compression ratio.

is implemented as $W \cdot \mathbf{x}$ where W is a weight matrix of model. We do not quantize bias vectors and layer normalization parameters. These parameters account for only a tiny fraction in terms of the total number of parameters and computation overhead, but it is important to retain these parameters in high precision. It is commonly acknowledged that quantization error in a bias vector will act as an overall bias (Jacob et al., 2018). Also Bhandare et al. (2019) points out that layer normalization operations will result in high error with low precision parameters as it includes calculations like division, square and square root.

4.2 Experimental Settings

Dataset We test our quantization strategy in 3 different translation directions: English-to-German (En2De), English-to-French (En2Fr), and English-to-Japanese (En2Jp). For En2De and En2Fr, we utilize all of the trainset of WMT2014 and use newstest2013 as devset and newstest2014 as testset (Bojar et al., 2014). For En2Jp, we use KFTT (Neubig, 2011), JESC (Pryzant et al., 2018), and WIT[3] (Cettolo et al., 2012) corpus. We combine the respective trainsets and devsets. We utilize KFTT testset as our testset. All En2Jp data are detokenized as suggested by Michel and Neubig (2018). `sentencepiece 0.1.85` (Kudo and Richardson, 2018) is utilized to learn a bi-lingual vocabulary set of size 32768 for each translation direction. For data statistics and download links, refer to Appendix A.1.

Baseline Model We train the base configuration of the Transformer to be utilized as our full precision reference as well as an initial set of model parameters for our quantization experiments. Training hyper-parameters are listed in Appendix A.3.

BLEU We report both tokenized-BLEU and detokenized-BLEU scores. We report detokenized-BLEU on devsets using `sacrebleu` (Post, 2018). While no tokenization is applied to En2De and En2Fr results and devsets, for En2Jp, `mecab` (Kudo, 2005) tokenized results and devsets are utilized. Simple `sacrebleu` command without additional signatures is used to measure detokenized-BLEU. For testsets, tokenized-BLEU scores are reported. Tokenizers employed are `moses` (Koehn et al., 2007) tokenizer[3] for En2De and En2Fr and `mecab` (Kudo, 2005) tokenizer for En2Jp. On the tokenized results and testsets, `multi-bleu.perl` script in `moses` is used to

[3] https://github.com/moses-smt/mosesdecoder

Average # of Bits Emb, Dec, Enc	Peak MEM.(MB)	Avg Lat.(ms)
FP baseline	247.7	708.9
3-bit baseline	34.5	301.0
2-bit baseline	24.5	235.9
2.5, FP, FP	188.3	464.3
2.5, 1.8, FP	94.5	201.4
2.5, 1.8, 3.7	29.8	200.7

Table 3: Inference latency of our quantized En2De model on a Galaxy N10+. Avg. Lat denotes average latency for translation of an input sequence. All measurements are averaged over 3 runs of translating first 300 sequences in newstest2013. Refer to Appendix A for measurement and implementation details.

Block	FLOPs	Latency(ms)
Encoder	0.52G(20.8%)	36.4(4.4%)
Decoder	1.49G(59.2%)	411.1(49.8%)
Embedding	0.50G(20.0%)	372.4(45.1%)
Total	2.52G	825.1

Table 4: FLOPs and on-device latency required for translation. Decoder-side activation caching is used. Latency is averaged over 100 translation runs on a Galaxy N10+. A translation run denotes an example translation with 30 words input and output sequences. Note that 300 sequences of newstest2013 consist of 27.6 words in average.

measure the tokenized-BLEU score. Note that in each experiment, we report testset's BLEU score using the model parameters that describe the highest BLEU score on devset.

4.3 Results

We compare our quantization strategy to our full precision (FP) baseline and quantization baselines in terms of translation quality and inference efficiency. Note that for the 2-bit baselines and 3-bit baselines, we respectively assign quantization bits of 2 and 3 to all Transformer parameters, and as for the 2-bit Emb. baseline, we assign 2 quantization bits to all word vectors in embedding block. Our quantized models are notated as (average # bits in an embedding parameter, average # bits in a decoder parameter, average # bits in an encoder parameter).

Translation Quality In Table 2, we present translation quality in terms of BLEU scores measured at each phase of the proposed quantization strategy. First, we experiment our embedding quantization method with retraining. Experimental results show that Transformer model with 1.1-bit embedding (1.1, FP, FP) exhibits comparable performance as much as 2-bit Emb. baseline. Furthermore, our experiments with 1.3-bit embedding (1.3, FP, FP) and 1.1-bit embedding verify that a substantially large number of word vectors can be quantized into 1-bit within reasonable BLEU score degradation.

We further quantize Transformer by applying quantization to the decoder block. We study how sensitive each sub-layer is by quantization toward translation quality, and we assign the number of bits for each sub-layer accordingly. Each type of sub-layers in the decoder block are assigned 2, 3, and 1 bits to Dec_{dd}, Dec_{ed}, and Dec_{ffn} respectively. In this case, the average of quantization bits for the decoder block is 1.8. For (2.5, 1.8, FP) model, considering that we quantize the embedding and decoder blocks, which account for large number of parameters (69.0%), into the average of under 3-bit, BLEU score degradation is moderate (within -1 BLEU from the FP baseline).

As we mentioned in Section 2.1, computations for encoder can be easily parallelizable, and thus, we assign slightly higher number of bits to the encoder block. We can improve quantization result of encoder block to be 3.7-bits per weight by assigning 3 bits to Enc_{ee} sub-layers and 4 bits to more sensitive Enc_{ffn} sub-layers. It is interesting that (2.5, 1.8, 3.7) models in various directions show higher BLEU score than (2.5, 1.8, FP) models which are of previous retraining phases with higher number of bits to represent the models. Our 2.6-bit Transformer models (2.5, 1.8, 3.7) attain 11.8$\times$ model compression ratio with reasonable -0.5 BLEU or less in 3 different translation directions. Our quantized models outperform the 3-bit baselines in both BLEU score and model compression ratio.

Inference Speed Up Let us discuss implementation issues regarding Transformer inference operations for on-device deployment. Measurements of the inference latency and the peak memory size on a mobile device is presented in Table 3. Our 2.6-bit quantized model (with (2.5, 1.8, 3.7) configuration) achieves 3.5$\times$ speed up compared to the FP baseline. Interestingly, our (2.5, 1.8, FP) model with the average of 11.3-bit outperforms the

Source
Linda Gray, die die Rolle seiner Ehefrau in der Original- und Folgeserie spielte, war bei Hagman, als er im Krankenhaus von Dallas starb, sagte ihr Publizist Jeffrey Lane.
Reference
Linda Gray, who played his wife in the original series and the sequel, was with Hagman when he died in a hospital in Dallas, said her publicist, Jeffrey Lane.
Generated (full-precision model, beam=4)
Linda Gray, who played the role of his wife in the original and subsequent series, was with Hagman when he died at Dallas hospital, said her journalist Jeffrey Lane.
Generated (model with embedding quantized to 1.1 bit, beam=4)
Linda **Gra**y, who played the role of his **wife** in the original and **subsequent** series, was with **Hag**man when he died in **Dal**las **hospital**, said her publicist **Jeff**rey **Lan**e.

Table 5: A De2En translation sample from a FP model and a (1.1, FP, FP) model. Detokenized-BLEU (beam=1, newstest2013, sacrebleu) for each of the models are 30.5 and 30.4. Words with 1-bit quantization are in **bold letters**. One word with 1-bit quantization is followed by an underlined word. For both full-precision model and quantized model, underlined words are identical.

Method	BLEU		Comp.
	En2De	En2Fr	
Vaswani et al.	27.3	38.1	1.0×
Bhandare et al. - 8bit	27.3	-	≤4.0×
Prato et al. - 8bit	27.6	39.9	3.9×
Prato et al. - 4bit	18.3	1.6	7.7×
Ours - 2.6 bit	27.1	38.0	11.8×

Table 6: Comparison of our quantization strategy with other quantization methods. Comp. denotes compression ratio in terms of model size.

2-bit baseline in terms of inference speed. In other words, as for inference speed up, addressing memory wall problems may be of higher priority rather than attaining a low number of quantization bits.

For each block, Table 4 shows the number of FLOPs and on-device inference latency. The decoder block demands higher FLOPs than the encoder block (3×), and therefore, employs even higher ratio of on-device inference latency than the encoder block (11×). Note that while the embedding block requires an amount of FLOPs to be comparable to that of the encoder block, it causes 11× more inference time than the encoder block. This experiment shows that it is essential to address memory inefficiency for fast on-device deployment of the Transformer.

Comparison Finally, in Table 6, we compare our quantization strategy to previous Transformer quantization methods. All listed methods show results on quantized models based on Transformer base configuration with WMT14 trainsets and report tokenized-BLEU on newstest2014 with exception of Bhandare et al. (2019) lacking specific BLEU scoring method. Our work outperforms previous quantization studies in terms of compression ratio and achieves reasonable translation quality in terms of BLEU as compared to reported BLEU of full precision models. Bhandare et al. (2019) reports speed up but it is not directly comparable because of the difference in inference settings (e.g. device used, decoding method, etc.) and other studies do not mention speed up.

4.4 Qualitative Analysis

In our strategy, after a large portion of word vectors are quantized by using 1 bit, translation quality degradation may occur even if BLEU does not capture such degradation. Correspondingly, as an attempt to empirically assess the quality of generated translation results with 1-bit quantized word vectors, we investigate how a decoder block predicts the next word. In Table 5, we present translation examples generated by models with full precision embedding block or with quantized embedding block. Comparing full precision model and quantized model, we observe that for each word with 1-bit quantization, a decoder block generates the same next word (underlined in Table 5). We present more examples in Appendix C. As such, qualitative analysis suggests that our quantization would not noticeably degrade the prediction capability of a decoder even when an input vector is 1-bit quantized.

5 Related Work

Previous researches proposed various model compression techniques to reduce the size of Transformer models. Gale et al. (2019) apply pruning (Han et al., 2015) to eliminate redundant weights of Transformer and report that higher pruning rates lead to greater BLEU score degradation. As for pruning, achieving inference speed up is more challenging because unstructured pruning method is associated with irregular data formats, and hence, low parallelism (Kwon et al., 2019).

Uniform quantization for Transformer is explored within reasonable degradation in BLEU score at INT8, while BLEU score can be severely damaged at low bit-precision such as INT4 (Prato et al., 2019). In order to exploit efficient integer arithmetic units with uniformly quantized models, activations need to be quantized as well (Jacob et al., 2018). Furthermore, probability mapping operations in Transformer, such as layer norm. and softmax, could exhibit significant amount of error in computational results with low precision data type (Bhandare et al., 2019).

6 Conclusion

In this work, we analyze each block and sub-layer of the Transformer and propose an extremely low-bit quantization strategy for Transformer architecture. Our 2.6-bit quantized Transformer model achieves 11.8× model compression ratio with reasonable -0.5 BLEU. We also achieve the compression ratio of 8.3× in memory footprints and 3.5× speed up on a mobile device (Galaxy N10+).

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Loïc Barrault, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, Christof Monz, Mathias Müller, Santanu Pal, Matt Post, and Marcos Zampieri. 2019. Findings of the 2019 conference on machine translation (WMT19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 1–61, Florence, Italy. Association for Computational Linguistics.

Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. 2019. Efficient 8-bit quantization of transformer neural machine language translation model. *CoRR*, abs/1906.00532.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Christof Monz, Matt Post, and Lucia Specia, editors. 2014. *Proceedings of the Ninth Workshop on Statistical Machine Translation*. Association for Computational Linguistics, Baltimore, Maryland, USA.

Mauro Cettolo, Christian Girardi, and Marcello Federico. 2012. Wit^3: Web inventory of transcribed and translated talks. In *Proceedings of the 16th Conference of the European Association for Machine Translation (EAMT)*, pages 261–268, Trento, Italy.

Patrick Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. 2018. Groupreduce: Block-wise low-rank approximation for neural language model shrinking. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10988–10998. Curran Associates, Inc.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: training deep neural networks with binary weights during propagations. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3123–3131. Curran Associates, Inc.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. 2019. HAWQ: hessian aware quantization of neural networks with mixed-precision. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*.

Andreas Eisele and Yu Chen. 2010. MultiUN: A multilingual corpus from united nation documents. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

Trevor Gale, Erich Elsen, and Sara Hooker. 2019. The state of sparsity in deep neural networks.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization.

Alex Graves. 2013. Generating sequences with recurrent neural networks.

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

Yiwen Guo, Anbang Yao, Hao Zhao, and Yurong Chen. 2017. Network sketching: Exploiting binary structure in deep cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yunhui Guo. 2018. A survey on methods and theories of quantized neural networks.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yongkweon Jeon, Baeseong Park, Se Jung Kwon, Byeongwook Kim, Jeongin Yun, and Dongsoo Lee. 2020. BiQGEMM: Matrix multiplication with lookup table for binary-coding-based quantized dnns.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86. Citeseer.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Taku Kudo. 2005. Mecab : Yet another part-of-speech and morphological analyzer.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Se Jung Kwon, Dongsoo Lee, Byeongwook Kim, Parichay Kapoor, Baeseong Park, and Gu-Yeon Wei. 2019. Structured compression by weight encryption for unstructured pruning and quantization.

Dongsoo Lee, Se Jung Kwon, Byeongwook Kim, Yongkweon Jeon, Baeseong Park, Jeongin Yun, and Gu-Yeon Wei. 2020. Decoupling weight regularization from batch size for model compression.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, pages 14014–14024.

Paul Michel and Graham Neubig. 2018. MTNT: A testbed for machine translation of noisy text. *CoRR*, abs/1809.00388.

Graham Neubig. 2011. The Kyoto free translation task. http://www.phontron.com/kftt.

Matt Post. 2018. A call for clarity in reporting bleu scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191.

Gabriele Prato, Ella Charlaix, and Mehdi Rezagholizadeh. 2019. Fully quantized transformer for machine translation.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*.

R. Pryzant, Y. Chung, D. Jurafsky, and D. Britz. 2018. JESC: Japanese-English Subtitle Corpus. *Language Resources and Evaluation Conference (LREC)*.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. *Lecture Notes in Computer Science*, page 525–542.

Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. 2020. And the bit goes down: Revisiting the quantization of neural networks. In *International Conference on Learning Representations*.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2214–2218, Istanbul, Turkey. European Language Resources Association (ELRA).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Chunpei Wang and Xiaowang Zhang. 2020. Q-bert: A bert-based framework for computing sparql similarity in natural language. In *Companion Proceedings of the Web Conference 2020*, pages 65–66.

Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. 2018. Mixed precision quantization of convnets via differentiable neural architecture search.

Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha. 2018. Alternating multi-bit quantization for recurrent neural networks. In *International Conference on Learning Representations*.

Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. 2019. Improving neural network quantization without retraining using outlier channel splitting. In *International Conference on Machine Learning*, pages 7543–7552.

Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. 2017. Adaptive quantization for deep neural network.

A Experiment Details

A.1 Data

Of data we use, WMT2014 data (Bojar et al., 2014) includes: Europarl v7 (Koehn, 2005), Multi-UN corpus (Eisele and Chen, 2010), News commentary corpus Giga French-English corpus provided by OPUS (Tiedemann, 2012), and data provided by CommonCrawl foundation[4]. Statistics of data is represented in Table 7. Data used for En2De and En2Fr can be found at https://www.statmt.org/wmt14/translation-task.html. Data used for En2Jp can be found at: KFTT (http://www.phontron.com/kftt/), WIT^3 (https://wit3.fbk.eu/mt.php?release=2017-01-trnted), and JESC (https://nlp.stanford.edu/projects/jesc/).

Translate	# of Sequences		
Direction	Train	Dev	Test
En2De	4.5M	3000	3003
En2Fr	40.8M	3000	3003
En2Jp	3.9M	4451	1160

Table 7: Statistics of data used for each translation direction.

A.2 Model

All models follow the base configuration of Transformer architecture composed of 60.9 million parameters (Vaswani et al., 2017).

A.3 Training

Our training and retraining implementation is based on `tensor2tensor` 1.12's implementation of Transformer and utilizes `tensorflow 1.14` (Abadi et al., 2015) modules. All training hyperparameters exactly follow `transformer_base` configuration of the code. We use 4×V100 GPUs for all training and retraining, and for each training step, a mini-batch of approximately 8,000 input words and 8,000 target words is used per GPU. Training of a full precision baseline model takes around 1.7 days. Adam optimizer (Kingma and Ba, 2014) with $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-9}$ is used and we adopt Noam learning rate scheme of Vaswani et al. (2017) using same suggested hyperparameters. Baseline models are trained for 400,000 training steps and we select models that have the highest BLEU score on devset to report as our full precision baseline and to warm start from in our retraining for quantization.

[4] https://commoncrawl.org/

A.4 Retraining

For retraining, we experiment with $pNR \in \{1, 10, 100, 500, 1000, 2000, 4000\}$. With $pNR \in \{1, 10, 100\}$, our retraining experiments resulted in divergence. We find that for a retraining phase where we quantize all blocks of Transformer, $pNR = 2000$ is the most effective in attaining a higher BLEU score with quantized model. And for a retraining phase in 3-phase retraining, where we quantize a block in Transformer, $pNR = 1000$ is the most effective. Hence, we set $pNR = 2000$ for retraining of quantization baselines, and for experiments where we quantize and retrain each block in Transformer at a time, we set $pNR = 1000$. While the choice for the value of pNR is made in empirical manner, it should be noted that in our tests, regardless of the number of quantization bits or other design choices, the choice of pNR value between 1000 and 2000 did not result in high variance on translation quality.

Our learning rate (lr) schedule is similar to the Noam schedule suggested in (Vaswani et al., 2017), but replaced the warm-up stage with a constant lr stage as in Eq. 3:

$$lr = c_{lr} \cdot d_{model}^{0.5} \cdot min(step^{-0.5}, steps_{peak}^{-0.5}) \quad (3)$$

$step$ is incremented by 1 with each mini-batch update and reset to 0 at each retraining phase. We use $c_{lr} = 3$ for all retraining. This scheme results in higher overall learning rate than what we use in our full precision baseline training, which follows the heuristics that large enough learning rate is required to find the best local minima with quantization constraint applied.

For single-phase retraining, we train up to 400,000 steps. Based on BLEU score on devset, single-phase retraining seems to reach convergence at around 300,000 steps. As for 3-phase retraining, we train for 300,000 steps respectively. We found 300,000 steps ample for a retraining phase to reach convergence judging from the reported BLEU scores on the validation set. In the 3-phase retraining, we first retrain and quantize embedding then embedding + decoder and finally all blocks of Transformer. For each phase of retraining, we take a model that reports the highest detokenized-BLEU score on devset. Retraining hyperparameters that are not stated follow corresponding hyperparameters of full precision model training Additionally, we attempt another variant of 3-phase retraining where we target only a single Transformer block at each phase and stop gradients on previously targeted Transformer blocks. However, this method of retraining results mostly in moderately lower BLEU score compared to our current 3-phase retraining method.

A.5 On-Device Inference

On-device inference is implemented with Eigen 3.7 (Guennebaud et al., 2010) for full precision computation and BiQGEMM (Jeon et al., 2020) for computation with quantized weights. With BiQGEMM, the value of redundant intermediate computation that occurs in matrix multiplication of quantized weights is pre-computed and stored to be reused, which is promising in reduction of memory overhead. Each B value is represented with a single bit in memory where 0 denotes -1 and 1 denotes +1, and in our implementation bits are packed into 32-bit integer which is directly used at inference. We follow BiQGEMM in our implementation of quantized inference. In our implementation, we implement decoder-side activation caching following `tensor2tensor`'s implementation of Transformer. We measure on-device latency with a `<chrono>` implementation of C++14 and memory usage with `adb`[5]. Unless otherwise specified, both latency and memory usage are measured while translating the first 300 sequences of En2De testset over 3 translation runs. Additional statistics regarding inference latency and memory of quantized models are available in Table 8.

B Validation Score

We report the validation scores (detokenized-BLEU scores on devset) of experimented models in Table 9.

C Sequences Generated with 1-bit Words

In Table 10, we present actual translation results from full precision embedding block and quantized embedding block. In the first example, 2 out of 2 words that follow 1-bit words are equal to their positional equivalents in the output sequence generated with the full precision model. In the second example, 19 out of 21 matches.

[5] `https://developer.android.com/studio/command-line/adb`

Average # of Bits	Latency contribution(%)				Avg.	Peak	Avg. # of Words	
Emb, Enc, Dec	Emb	Enc	Dec	Other	Lat(ms)	MEM(MB)	Input	Output
FP baseline	44.4%	4.9%	50.1%	0.6%	708.0	247.7	27.6	27.9
3-bit baseline	51.3%	8.7%	38.3%	1.8%	301.0	34.5	27.6	28.0
2-bit baseline	45.9%	9.5%	42.2%	2.3%	235.9	24.5	27.6	28.0
2.5, FP, FP	14.4%	7.0%	77.3%	1.2%	464.3	188.3	27.6	28.4
2.5, 1.8, FP	34.6%	16.3%	46.2%	2.8%	201.4	94.5	27.6	28.4
2.5, 1.8, 3.7	34.4%	16.5%	46.5%	2.7%	200.7	29.8	27.6	27.7

Table 8: Additional statistics regarding reported measurements of Table 3.

Average # of Bits		Validation BLEU(beam=1)		
Emb, Dec, Enc	Model	En2De	En2Fr	En2Jp
FP baseline	32.0	25.4	31.4	18.7
3-bit baseline	3.0	25.3	30.1	18.0
2-bit baseline	2.0	23.9	28.6	16.8
2-bit baseline(Emb)	23.7	25.1	N/A	N/A
2.5, FP, FP	23.9	25.6	31.2	18.5
1.3, FP, FP	23.5	25.3	31.0	17.5
1.1, FP, FP	23.5	25.2	31.0	17.9
2.5, 1.8, FP	11.3	25.2	30.6	18.1
1.3, 1.8, FP	11.0	24.6	30.4	17.7
1.3, 1.8, FP	11.0	24.4	30.4	17.2
2.5, 1.8, 3.7	2.6	25.1	30.9	18.4
1.3, 1.8, 3.7	2.2	24.9	30.6	17.7
1.1, 1.8, 3.7	2.2	24.4	30.5	17.6

Table 9: BLEU score on devset of baseline models and quantized models. We report detokenized-BLEU (beam=1, newstest2013, sacrebleu) for En2De, En2Fr as suggested in in Section 4.2. For En2Jp, outputs and references are tokenized with `mecab` then measured with `sacrebleu`.

Source 1 Im vergangenen Jahr gingen beim CTMO mehr als 1,4 Millionen Anträge auf Markenschutz ein, fast ein Drittel mehr als 2010.
Reference 1 In the past year, more than 1.4 million applications for trademark protection were submitted to the CTMO, almost one third more than in 2010.
Generated 1 (full-precision model, beam=4) Last year, more than 1.4 million applications for trademark protection were received at the CTMO, almost one third more than in 2010.
Generated 1 (model with embedding quantized to 1.1 bit, beam=4) Last year CTMO received more than **1.4** million **trademark** protection applications, almost a third more than in **2010.**
Source 2 Der derzeitige Premierminister Israels, der Falke Netanjahu, ist ein typisches Beispiel eines faschismusanfälligen, den internationalen Bankern loyal ergebenen Politikers, der alles dafür tut, um einen Krieg mit dem Iran zu entfachen, welcher sich angesichts der Mitgliedschaft Irans in der Schanghaier Organisation für Zusammenarbeit (China, Indien, Russland, Pakistan...), rasch zu einem globalen Konflikt ausweiten könnte, und bei dem es wegen der Kontrolle Irans über die nur 2 Meilen breite Straße von Hormus, über die 20% der weltweiten Erdöllieferungen laufen, zu einer Zerstörung der Weltwirtschaft kommen könnte.
Reference 2 Israel's current prime minister, Netanyahu 'the hawk', is a typical example of a fascist politician, loyal to the international bankers, who does everything to instigate war with Iran, which would, due to its membership in the Shanghai Cooperation Organisation (China, India, Russia, Pakistan, ...) lead to a greater threat of global conflict, and through its control of the Hormuz Strait, where 20% of the world's oil must sail (the channel is only 2 miles wide), to the destruction of the world's economy.
Generated 2 (full-precision model, beam=4) The current Prime Minister of Israel, the Falk Netanyahu, is a typical example of a fascism-prone politician loyal to international bankers who is doing everything possible to spark a war with Iran, which, given Iran's membership of the Shanghai Cooperation Organisation (China, India, Russia, Pakistan...), could rapidly spread to a global conflict, and could lead to the destruction of the world economy because of Iran's control of the only 2-mile-wide Strait of Hormus, which accounts for 20% of world oil supplies.
Generated 2 (model with embedding quantized to 1.1 bit, beam=4) Israel's current **prime minister**, **Falk**e **Net**anyahu, is a **typical** example of a fa**sc**ism-prone **politician loyal** to international **bankers** who is doing all he can to **trigger** a war with Iran, which, with Iran's **membership** of the **Shanghai Cooperation** Organisation (**China**, India, Russia, Pakistan**...)**, could **rapidly** develop into a global conflict and could lead to the **destruction** of the world economy because of Iran's control of the only 2 **mile-wide Stra**it of **Hor**mus, which **accounts** for **20%** of world oil **supplies**.

Table 10: De2En translation samples from full-precision model and model with embedding block quantized to 1.1-bit ($b = 4, r = 8$) with Algorithm 1 (1.1, FP, FP). Same models as Table 5 is utilized.