A Quantization Functions and Implementations

In this section we introduce more details of quantization functions used in our paper. As mentioned in Section 4.1, we use TWN [26] and LAQ [19] for 2-bit and 4-bit weight quantization, and LSQ [13] for all activation quantization. These functions can be generally included in the form of Equation (1).

TWN Ternary weight network [26] quantizes the full-precision network parameters $\mathbf{x} \in \mathbb{R}^m$ into three distinct values. The quantization function can be written as

$$\hat{x}_i = \begin{cases} s, & \text{if } x_i > \Delta \\ 0, & \text{if } |x_i| < \Delta \\ -s, & \text{if } x_i < -\Delta \end{cases} \tag{5}$$

where Δ is the positive threshold parameter. As discussed in [19], we may also re-write the quantizer equivalently as $\Pi_{\Omega(2)}(\mathbf{x}/s)$ with $\Delta = s/2$ and $\Omega(2) = \{-s, 0, s\}$, which is a special case to Equation (1). Empirically, since the exact solution reuires expensive sorting operation, following [26], Δ is approximately computed as $\Delta^* \approx 0.7 \cdot \sum_i |x_i|$. The step-size in TWN is obtained by $s = \frac{1}{|\mathbf{I}_\Delta|} \sum_{i \in \mathbf{I}_\Delta} |x_i|$, where $\mathbf{I}_\Delta = \{i \big| |x_i| > \Delta\}$ and $|\mathbf{I}_\Delta|$ denotes the number of elements in the set. For the backward pass of the non-differentiable TWN operator, we use straight-through estimator (STE) [4] with identity mapping.

LAQ In LAQ [19], the multi-bit quantization is represented as

$$\hat{\mathbf{x}} = \alpha \mathbf{b}, \text{ where } \alpha > 0, \mathbf{b} \in \mathcal{Q} = \{-1, -\frac{k-1}{k}, ..., -\frac{1}{k}, 0, \frac{1}{k}, ..., \frac{k-1}{k}, 1\}, \tag{6}$$

where α and $\mathbf{b}$ are the full-precision scaling and quantization points respectively. Thus Equation 6 can be equivalently converted to Equation 1 by multiplying k for all quantization points in $\mathcal{Q}$ and divide them back in α accordingly. While analytical solutions for s and $\mathbf{b}$ can be computationally prohibited, following [19], we approximate algorithms to efficiently solve for these variables. Specifically, the approximated algorithm alternates the following updates at each training step:

$$\alpha = \frac{\| \mathbf{b} \odot \mathbf{d} \odot \mathbf{x} \|_1}{\| \mathbf{b} \odot \mathbf{d} \|_1}, \tag{7}$$

$$\mathbf{b} = \Pi_{\mathcal{Q}}(\mathbf{x}/s), \tag{8}$$

where $\mathbf{d}$ represents the approximated diagonal Hessian matrices. In our experiments, we found ten iterations are enough for convergence. Besides, we also find the substitution of $\mathbf{d}$ with $\mathbf{1}$ brings nearly no degradation but accelerates the training process. For the backward pass, we use the same STE rule as TWN.

LSQ In LSQ [13], the quantization function is defined as

$$\bar{\mathbf{x}} = \lfloor \text{clip}(\mathbf{x}/s), -Q_N, Q_P \rceil, \quad \hat{\mathbf{x}} = \bar{\mathbf{x}} \cdot s, \tag{9}$$

where $\lfloor \cdot \rceil$ rounds the element to the nearest integer, and Q_N and Q_P represent the number of negative and positive quantization levels. Thus Equation (9) can be viewed as projecting $\mathbf{x}/s$ onto $\Omega(b) = \{-2^{b-1} + 1, ..., 0, ..., 2^{b-1} - 1\}$. The symmetric uniform quantization also implies that $Q_N = 2^{b-1}$ and $Q_P = 2^{b-1} - 1$. Therefore, Equation (9) can be also included in Equation (1).

For the implementation of LSQ, we strictly follow the original paper [13]. The step-sizes $\mathbf{s}$ of activations $\mathbf{a}$ are initialized to $2\bar{a}/\sqrt{Q_P}$ based on the first batch of calibration data, where $\bar{a}$ represents the mean of the absolute value $|\mathbf{a}|$. In the backward pass of LSQ quantizer, we similarly adopt STE for non-differentiability, and further scale the gradients of $\mathbf{s}$ by $1/\sqrt{N_F Q_P}$ for activations with N_F elements and Q_P quantization levels. The learning rate of $\mathbf{s}$ is set to 1e-4 with no weight decay for all experiments. The activation of each linear layer has a distinct step size $\mathbf{s}$.

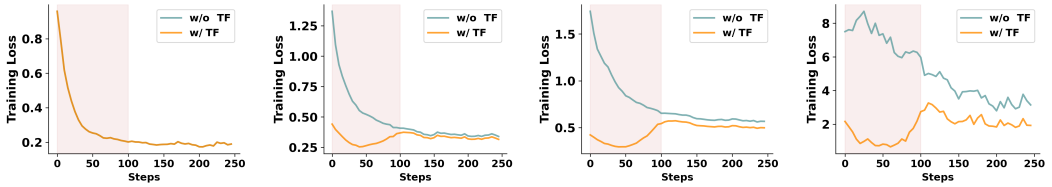
B Additional Experiments

B.1 Baseline Implementation

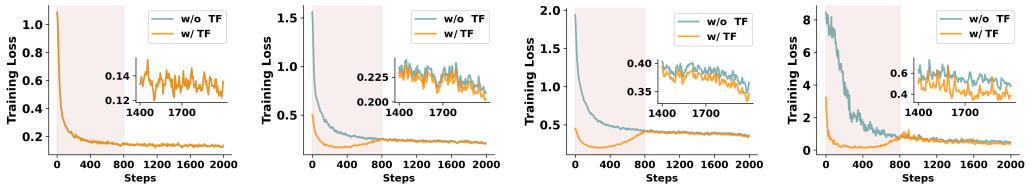
For baselines, we mainly compare with QAT and REM, where the former measures how much PTQ can get close to QAT, and the latter studies the effect of objective granularity in PTQ training. We

Table 6: Results of our proposed MREM-S and MREM-P against QAT and REM on the development set of SQuAD v2.0. "__" denotes results with two gradient accumulation steps under the same total batch size due to memory constraint.

	#Bits (W-E-A)	Quant Method	BERT-base Time (min)↓	Mem (GB)↓	# Data (K)↓	EM (%)↑	F1 (%)↑	BERT-large Time (min)↓	Mem (GB)↓	# Data (K)↓	EM (%)↑	F1 (%)↑
SQuAD v2.0	*full-prec.*	-	255	11.7	130	74.5	77.7	730	30.4	130	77.7	81.0
	4-4-8	QAT	662	18.4	130	74.4	77.5	2,820	28.3	130	77.4	80.5
		REM	60	3.1	4	$53.1_{\pm 0.4}$	$53.6_{\pm 0.4}$	175	7.3	4	$58.2_{\pm 0.2}$	$61.4_{\pm 0.3}$
		MREM-S	76	6.4	4	$73.0_{\pm 0.1}$	$76.3_{\pm 0.1}$	200	14.5	4	$76.4_{\pm 0.1}$	$79.7_{\pm 0.1}$
		MREM-P	19	$5.5_{\times 4}$	4	$72.6_{\pm 0.2}$	$75.9_{\pm 0.2}$	50	$12.3_{\times 4}$	4	$76.3_{\pm 0.1}$	$79.6_{\pm 0.1}$
	2-2-8	QAT	508	17.5	130	73.0	76.2	1,680	28.3	130	76.7	80.0
		REM	60	3.1	4	$51.5_{\pm 0.2}$	$51.8_{\pm 0.2}$	160	7.3	4	$56.3_{\pm 0.2}$	$59.5_{\pm 0.2}$
		MREM-S	60	6.4	4	$71.4_{\pm 0.2}$	$74.8_{\pm 0.2}$	156	14.5	4	$75.4_{\pm 0.2}$	$78.7_{\pm 0.1}$
		MREM-P	15	$5.5_{\times 4}$	4	$70.8_{\pm 0.4}$	$74.3_{\pm 0.4}$	39	$12.3_{\times 4}$	4	$75.3_{\pm 0.3}$	$78.6_{\pm 0.3}$
	2-2-4	QAT	505	17.5	130	71.4	74.6	1,655	28.3	130	75.4	78.9
		REM	60	3.1	4	$39.3_{\pm 1.5}$	$41.4_{\pm 1.3}$	160	7.3	4	$42.9_{\pm 0.8}$	$44.2_{\pm 0.7}$
		MREM-S	60	6.4	4	$67.2_{\pm 0.3}$	$70.6_{\pm 0.2}$	156	14.5	4	$71.3_{\pm 0.3}$	$74.8_{\pm 0.2}$
		MREM-P	15	$5.5_{\times 4}$	4	$66.1_{\pm 0.5}$	$69.8_{\pm 0.5}$	39	$12.3_{\times 4}$	4	$71.5_{\pm 0.3}$	$75.0_{\pm 0.3}$



(a) Module-1 (250 Steps). (b) Module-2 (250 Steps). (c) Module-3 (250 Steps). (d) Module-4 (250 Steps).



(e) Module-1 (2000 Steps). (f) Module-2 (2000 Steps). (g) Module-3 (2000 Steps). (h) Module-4 (2000 Steps).

Figure 6: The training loss curves with and without teacher forcing (TF) in MREM-P. The red area denotes teacher forcing in the first 40% training steps. (a), (b), (c) and (d) in the first row are the four modules trained for 250 steps, and (e), (f), (g) and (h) in the second row are trained for 2,000 steps.

conduct QAT following the state-of-the-art training pipeline [56], i.e., intermediate-layer distillation followed by prediction-layer distillation, which takes 6 training epochs in total. Detailed hyperparameter settings can be found in [56]. In terms of REM, we follow the practice in [33, 23] to minimize the reconstruction error after each matrix multiplication, as introduced in Section 2.1. The learning rate and weight decay of REM are consistent with MREM.

B.2 Results on SQuAD v2.0

In Table 6 we show the results of MREM-S and MREM-P against QAT and REM on the SQuAD v2.0 dataset. The observations are consistent with those discussed in Section 4.2 in general, and our approaches demonstrat superiority in balancing the training time, memory overhead, data accessibility as well as the quantized performance.

B.3 Further Comparison with REM.

Here we provide further discussions with REM on the training efficiency. Note that both REM and MREM-S follow the sequential training procedure, where the output from the previous objective is cached for the next objective. However, as there are many matrix multiplications in each Transformer layer, it can be time-consuming for REM to repeat this procedure recursively. According to Section 4.2, while REM and MREM take roughly the same time, REM is only iterated for 250 steps on MNLI and 500 steps on SQuAD, while MREM takes 2,000 steps and 4,000 steps respectively.

Table 7: Ablation studies of teacher forcing at different training steps over MNLI-m.

#Bits (W-E-A)	Quant Method	# Steps	Time (min)↓	Mem (GB)↓	Acc m(%)↑	Acc mm(%)↑
4-4-8	REM	200	36	2.5	$73.3_{\pm 0.3}$	$74.9_{\pm 0.2}$
	REM	2,000	319	2.5	$81.8_{\pm 0.2}$	$82.5_{\pm 0.1}$
	MREM-S	2,000	36	4.6	$83.5_{\pm 0.1}$	$83.9_{\pm 0.1}$
2-2-8	REM	200	24	2.5	$71.6_{\pm 0.4}$	$73.4_{\pm 0.4}$
	REM	2,000	213	2.5	$78.7_{\pm 0.2}$	$79.2_{\pm 0.2}$
	MREM-S	2,000	24	4.6	$82.7_{\pm 0.2}$	$82.7_{\pm 0.2}$
2-2-4	REM	200	24	2.5	$58.3_{\pm 0.5}$	$60.6_{\pm 0.6}$
	REM	2,000	213	2.5	$73.0_{\pm 0.3}$	$74.4_{\pm 0.4}$
	MREM-S	2,000	24	4.6	$81.1_{\pm 0.2}$	$81.5_{\pm 0.2}$

Table 8: Comparisons between our MREM-P and data-parallel QAT over 4 GPUs on SQuAD v1.1.

#Bits (W-E-A)	Quant Method	Total Batch Size	Mem (GB↓)	# Steps	Time (min↓)	EM(%)↑	F1(%)↑
4-4-8	QAT	$4\ (1\times 4)$	$6.4_{\times 4}$	3000	15.5	$74.7_{\pm 0.2}$	$83.4_{\pm 0.1}$
	QAT	$12\ (3\times 4)$	$9.2_{\times 4}$	2600	15.0	$77.9_{\pm 0.1}$	$85.7_{\pm 0.1}$
	MREM-P	12	$5.5_{\times 4}$	4000	15.0	$79.6_{\pm 0.1}$	$87.3_{\pm 0.1}$
2-2-8	QAT	$4\ (1\times 4)$	$6.4_{\times 4}$	3000	15.5	$71.5_{\pm 0.3}$	$80.9_{\pm 0.2}$
	QAT	$12\ (3\times 4)$	$9.2_{\times 4}$	2600	15.0	$74.4_{\pm 0.3}$	$83.2_{\pm 0.2}$
	MREM-P	12	$5.5_{\times 4}$	4000	15.0	$77.7_{\pm 0.2}$	$85.9_{\pm 0.2}$
2-2-4	QAT	$4\ (1\times 4)$	$6.4_{\times 4}$	3000	15.5	$65.9_{\pm 0.4}$	$76.6_{\pm 0.3}$
	QAT	$12\ (3\times 4)$	$9.2_{\times 4}$	2600	15.0	$69.4_{\pm 0.3}$	$79.4_{\pm 0.2}$
	MREM-P	12	$5.5_{\times 4}$	4000	15.0	$73.0_{\pm 0.3}$	$82.7_{\pm 0.2}$

We also provide results when REM takes the same amount of training steps with MREM-S in Table 7. It can be found that even with $2,000$ iterations, REM is still inferior to MREM-S across all quantization bit-widths. Meanwhile, REM nearly takes around $9\times$ more training time than MREM. Therefore, the module-wise granularity in MREM not only improves the quantization performance with more layer-wise dependencies considered, but also makes the training pipeline efficient with fewer stages to cache intermediate results.

B.4 More Visualizations of Training Curves with Teacher Forcing

We show the training curves of four modules with and without teacher forcing in Figure 6. Note that the first module exhibits the same training curves when trained with and without teacher forcing, since the input is directly from the text instead of the input queues. The rest observations are generally consistent with those discussions in Section 4.4.

C Comparisons with Data Parallelism and Pipeline Parallelism

C.1 Comparisons with Data Parallelism

The proposed parallel strategy (MREM-P) is superior to data parallelism [9, 27] in terms of memory saving and training acceleration when quantizing PTMs. To see that, we implement the data-parallel training for QAT, and see if this common approach can beat our MREM-P. We use the BERT-base model over SQuAD 1.1 on 4 GPUs for illustration, and keep all configurations consistent with MREM-P (e.g., 4K training samples with the learning rate of 5e-5), except for reducing the batch-size and training steps to keep the same memory cost per GPU or training time consumption.

It can be found from Table 8 that: 1) with the same memory cost, QAT requires to load the entire model into the memory, and thus it can only hold 1 sample per GPU to match the memory of MREM-P, which leads to worse performance; 2) even with the same batch size, QAT consumes additional 3.7 GB memory while it still underperforms MREM-P, since the full back-propagation is more time-consuming and thus allows fewer training iterations within 15 minutes. MREM-P is thus still preferred to data parallelism to quantize PTMs when multiple GPUs are available. Nevertheless, we remark that the two parallel mechanisms are orthogonal to each other and can be readily combined for further acceleration.

C.2 Comparisons with Pipeline Parallelism

Notably, our MREM-P with the stale synchronous update is different from the pipeline parallelism, such as G-Pipe [21] and PipeDream [36]. G-Pipe [21], for instance, adopts end-to-end training with synchronous updates between adjacent modules, which gives rise to bubble time on computing devices. While the data batch is divided into M pipelined micro-batches, it still has the bubble time of $O(\frac{N-1}{N+M-1})$ under N partitions. On the one hand, a larger N or smaller M would increase the bubble time. On the other hand, a larger M leads to smaller batches that still cannot fully exploit the computing power, which again affects the acceleration rate. PipeDream [36] optimizes the module partition to minimize the communication cost, but the resulting strategy still highly depends on the model architecture. Differently, our parallel strategy conducts local training with stale synchronous updates of the module. Hence there is negligible bubble time as long as the straggler is faster than the staleness threshold t_0, which can be easily satisfied with balanced module partitions or larger t_0.

D Broader Impact and Limitations

The primary goal of this research is to develop an efficient PTQ method w.r.t. training time, memory overhead and data accessibility to quantize pre-trained language models. These factors are always of high priority in developing modern machine learning algorithms. On the one hand, it is expensive and energy-consuming for the training of huge PTMs. Therefore, it is of necessity to develop environmentally-friendly learning algorithms that are both efficient and lightweight. While the proposed PTQ method significantly reduces the training cost compared with QAT, there are still more computing devices in demand if further training speed-up is desired. On the other hand, data privacy is another important issue in PTQ research. Our approach constructs of the calibration set with thousands of data samples. This may still incur data security issues in some domains, where data privacy is of high priority or only few data samples are available. Thus it is also a promising direction to continually improve the data-efficient PTQ research. Finally, there are also challenges when applying MREM to PLMs at larger scale, when more module partitions are produced and the performance could be sub-optimal due to the blocked gradients between adjacent modules.