A MORE EXPERIMENTAL RESULTS AND TAKEAWAYS

A.1 Experimental setup

System configuration. In order to measure the performance of compression algorithms over different hardware, we conduct our experiments on two different setups. The first setup uses AWS p3.8xlarge machines which have 4 Tesla V100 GPUs with all GPUs connected by NVLink. AWS p3.8xlarge instances have 10 Gbps network bandwidth across instances. Moreover, we also use a local machine which also has 4 Tesla V100 GPUs but does not have NVLink. All the GPUs are connected by a single PCIe bridge. The local server runs Ubuntu 18.04 LTS and the server has 125GB of memory.

Models. The models we use in this section are the same as what we mentioned in Section 4.1.

Experiment parameters. Consistent with the specifications laid out in Section 4.1, our experiments maintain the same settings. We have also expanded the scope of our investigations to study the effect of various hyper-parameters. This includes changing the batch size between $\{8, 32\}$, and adjusting sequence length from $\{128, 512\}$ during fine-tuning. Moreover, we explore the influence of the number of nodes, varying from $\{8, 16, 32, 64\}$, on strong-scaling speedup, while keeping the model size constant.

Roadmap. The ensuing sections are structured as follows: Section A.2 presents the experimental findings on the $\text{BERT}_{\text{BASE}}$ model. In Section A.3, we delve into the impact of model hyper-parameters on throughput and accuracy. Subsequently, Section A.4 scrutinizes the effects of varying compression layers and compression locations on these same metrics. Section A.5 compares the throughput performance between an AWS p3.8xlarge instance equipped with NVLink and a local machine without NVLink, to evaluate the role of hardware. In addition, Section A.6 reports the outcomes of strong scaling experiments. Finally, Section A.7 offers a theoretical analysis of slow network conditions using an analytical cost model, while Section A.8 presents additional evaluation results over A100 GPUs.

A.2 Experimental results over $\text{BERT}_{\text{BASE}}$ model

Takeaway 5 *When the evaluated model is BERT_{BASE}, AE and Quantization can preserve the model's accuracy on fine-tuning tasks over GLUE datasets.*

From Table 9, we can observe that the accuracy loss is within 5% except for the CoLA dataset when using AE and quantization methods for compression. Since CoLA is the smallest dataset among GLUE datasets, slight perturbations can cause drastic changes in the final results.

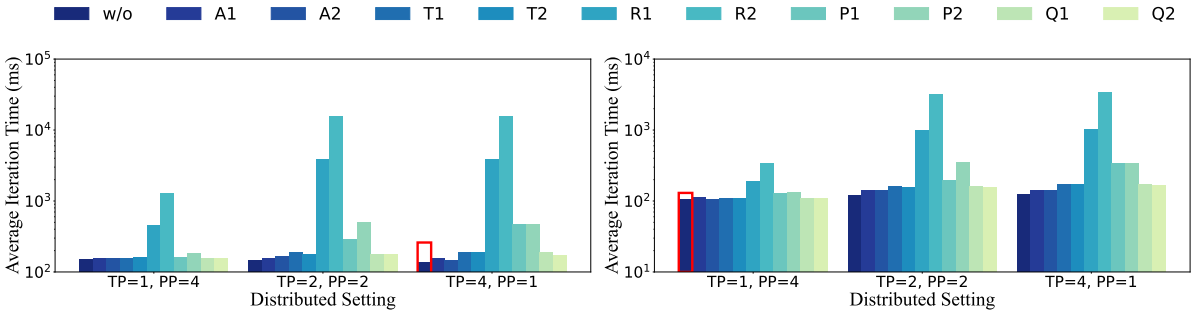
Compression Algorithm	MNLI-(m/mm)	QQP	SST-2	MRPC	CoLA	QNLI	RTE	STS-B	Avg.
w/o	83.45/84.16	90.62	91.40	83.83	57.52	91.31	62.09	84.79	81.02
A1	80.20/81.10	89.75	90.71	80.23	35.74	86.58	61.01	83.03	76.48
A2	80.22/80.83	89.71	90.60	80.64	38.84	86.82	62.82	83.28	77.06
T1	78.83/79.44	88.86	89.91	75.07	23.19	86.66	59.57	80.00	73.50
T2	80.31/80.94	89.20	90.71	74.14	31.60	88.32	62.09	81.61	75.44
P1	74.92/75.39	88.18	87.16	70.20	23.85	84.95	51.99	73.13	69.97
P2	74.04/74.78	87.93	86.93	66.49	0.00	84.88	52.35	71.90	66.59
Q1	82.48/83.04	89.91	91.40	81.33	50.49	89.36	61.01	83.90	79.21
Q2	83.31/84.14	90.50	91.97	83.07	55.22	91.07	61.01	85.15	80.60

Table 9. Fine-tuning results over GLUE dataset on $\text{BERT}_{\text{BASE}}$ model under the setting that the tensor model-parallel size is 2 and pipeline model-parallel size is 2. F1 scores are reported for QQP and MRPC, Matthews correlation coefficients are reported for CoLA, and Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks.

A.3 Impact of model hyper-parameters

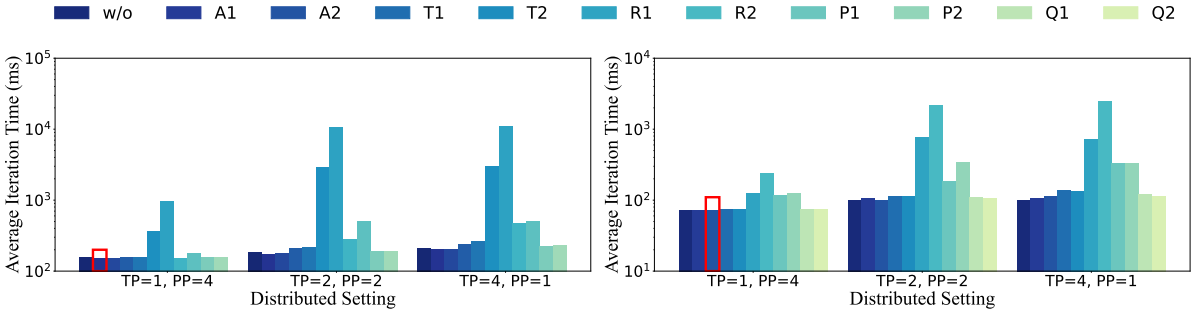
Takeaway 6 *Using a smaller batch size or sequence length for fine-tuning negates the throughput benefits from compression because of the smaller communication cost.*

We vary the batch size from $\{8, 32\}$ and sequence length from $\{128, 512\}$, and report the results in Figure 5(a)-6(b). We notice that when the communication cost over model parallelism is small, the overhead of the compression methods can become the bottleneck. Therefore, we cannot improve system throughput when using compression algorithms with batch size 8 and sequence length 128.



(a) Batch size=32, Sequence length=128 (b) Batch size=8, Sequence length=128

Figure 5. Average iteration time (ms) for fine-tuning with various batch sizes and sequence lengths. The results are collected from the AWS p3.8xlarge instance **with NVLink**. For each setting, we repeat experiments 5 times. Red rectangular boxes highlight the best method.



(a) Batch size=32, Sequence length=128 (b) Batch size=8, Sequence length=128

Figure 6. Average iteration time (ms) for fine-tuning with various batch sizes and sequence lengths. The results are collected from the local machine **without NVLink**. For each setting, we repeat experiments 5 times. Red rectangular boxes highlight the best method.

Takeaway 7 *Using a smaller batch size or sequence length for fine-tuning, AE and Quantization can also preserve the model's accuracy.*

From Table 10 and Table 11, it can be noted that the decrease in accuracy is kept within a 5% range for all but the CoLA and RTE datasets when employing AE and quantization techniques for compression. Furthermore, despite the utilization of fp64 for running the PowerSGD component, precision overflow remains a concern. This issue arises due to the instability of PowerSGD and the activation is not low-rank. In view of this, PowerSGD does not serve as a viable option for activation compression.

A.4 Varying compression layers and location

Takeaway 8 *When the number of compressed layers increases, the model accuracy decreases.*

From Figure 7(a), we can observe that the accuracy for RTE and the matthews correlation coefficient for CoLA decreases as we increase the number of layers compressed. This is because as we increase number of layers compressed, we lose more

Compression Algorithm	MNLI-(m/mm)	QQP	SST-2	MRPC	CoLA	QNLI	RTE	STS-B	Avg.
w/o	86.23/86.07	91.22	91.74	88.17	59.02	92.09	78.70	88.40	84.63
A1	82.49/82.41	89.93	91.85	82.43	43.56	89.84	47.29	87.03	77.43
A2	82.18/82.23	90.45	90.52	83.54	0.00	89.02	62.82	87.66	74.27
T1	49.07/47.96	72.02	83.57	69.33	12.04	83.60	55.60	84.96	62.02
T2	83.99/84.37	35.78	68.30	83.54	47.33	60.52	64.62	86.72	68.35
P1	36.66/37.18	68.28	81.19	67.59	0.00	58.23	55.23	7.26	45.74
P2	32.74/32.95	63.18	50.92	66.72	2.76	56.98	47.29	5.66	39.91
Q1	84.91/85.18	90.54	92.43	85.91	53.25	60.68	57.04	87.91	77.54
Q2	85.66/86.09	90.99	91.74	86.84	53.92	91.31	75.81	88.19	83.39

Table 10. Fintune results over GLUE dataset under the setting using tensor parallelism size 2, pipeline parallelism size 2, batch size 8, and sequence length 128. F1 scores are reported for QQP and MRPC, Matthews correlation coefficient is reported for CoLA, and Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks.

Compression Algorithm	MNLI-(m/mm)	QQP	SST-2	MRPC	CoLA	QNLI	RTE	STS-B	Avg.
w/o	87.87/88.02	91.96	95.18	87.71	59.40	92.99	76.90	88.43	85.38
A1	85.30/85.33	91.28	92.32	84.58	55.18	90.87	59.93	87.92	81.41
A2	85.25/85.19	91.41	93.23	86.72	57.02	90.92	64.26	87.74	82.42
T1	68.76/69.23	64.58	91.40	80.93	0.00	67.34	66.43	69.24	64.21
T2	84.24/85.23	89.17	92.09	81.68	51.54	91.71	63.54	84.80	80.44
P1	32.74/32.95	63.18	49.08	81.68	0.00	50.54	61.73	-7.02	40.54
P2	32.74/32.95	50.27	49.08	78.67	0.00	50.54	44.04	0.00	37.59
Q1	86.85/87.58	91.50	93.58	86.96	59.20	92.24	59.57	86.89	82.71
Q2	87.46/88.02	91.82	94.95	87.48	57.02	93.36	68.95	87.84	84.10

Table 11. Fintune results over GLUE dataset under the setting using tensor parallelism size 2, pipeline parallelism size 2, batch size 32, and sequence length 128. F1 scores are reported for QQP and MRPC, Matthews correlation coefficient is reported for CoLA, and Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks.

information in the activations leading to a loss in accuracy. From Figure 7(a), we observe that compressing activations of the last 8 layers is the best strategy to keep the accuracy loss within 3% for both datasets.

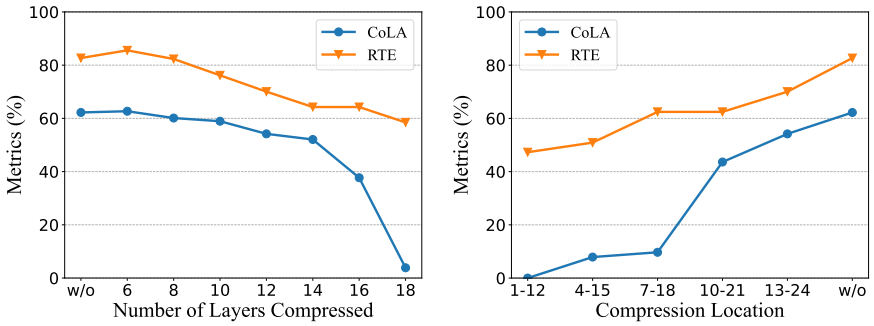
Takeaway 9 *Compressing the activation for the initial layers harms the accuracy of the model.*

We keep the number of layers compressed constant and vary the location where we apply compression (Figure 7(b)). The results indicate that compressing activations of the first few layers of the model significantly harms the model's accuracy. This is because compressing activations generates error and the error in the early layers can be accumulated and propagated to later layers.

A.5 Throughput benefits for fine-tuning on machine without NVLink

Takeaway 10 *Using non-learning-based compression techniques to compress activations only slightly improves system throughput (by 1% or less) due to the large overhead of these methods. However, we see end-to-end speedups of up to 17.8% when using learning-based compression methods on a machine without NVLink.*

When running fine-tune experiments on a p3.8xlarge instance on Amazon EC2, we cannot improve system throughput by using non-learning-based compression algorithms (Figure 3(a)). Comparing Figure 3(a) and Table 12, we can see that the network bandwidth across the GPUs can affect the performance benefits from compression. In other words, we can improve system throughput by at most 17.8% when compressing activation for fine-tuning tasks on a 4-GPU machine



(a) Vary Number of Layers Compressed (b) Vary Compression Location

Figure 7. Fine-tuning results over CoLA and RTE datasets by varying the compression location and number of layers compressed. The above figure shows that model performance vs the number of layers compressed. The below figure shows that model performance versus the compression location. We use tensor model-parallel degree 2, pipeline model-parallel degree 2, batch size 32, and sequence length 512.

without NVLink. That's because, without NVLink, the communication time for model parallelism is much longer. Thus, while the message encoding and decoding time remain unchanged, compression methods can provide more throughput benefits across lower bandwidth links.

Furthermore, from Figure 3(a) and Table 12, we observe that AE outperforms other compression methods. In Table 13, we breakdown the time taken by each algorithm and find that Top-K, Random-K and quantization have large encoding/decoding overheads and thus cannot provide end-to-end throughput improvements. Although AE slightly increases the time taken by the backward step, the $\sim 2\times$ reduction in communication time and the limited encoding/decoding overhead lead to better overall throughput.

With NVLink	w/o	A1	A2
TP=1, PP=4	591.96	591.36	591.47
TP=2, PP=2	440.71	437.98	444.02
TP=4, PP=1	**261.48**	270.22	275.54
Without NVLink	**w/o**	**A1**	**A2**
TP=1, PP=4	633.17	620.10	620.44
TP=2, PP=2	646.14	**586.65**	595.25
TP=4, PP=1	736.01	624.62	636.15

Table 12. The average iteration time (ms) for fine-tuning with/without NVLink. We compare time without compression and with AE on different distributed settings, with batch size 32, and sequence length 512. The best setting on each machine is **bolded**. And the settings, under which we can gain benefits compared with the baseline, are underlined.

Compression Algorithm	Forward	Backward	Optimizer	Waiting & Pipeline Comm.	Total Time	Tensor Enc.	Tensor Dec.	Tensor Comm.
w/o	276.34	354.16	5.80	9.83	646.14	\	\	150.72
A1	213.83	362.61	6.16	4.06	586.65	2.16	3.12	80.88
A2	219.01	366.51	5.67	4.07	595.25	3.12	4.56	84.48
T1	331.70	356.80	5.78	5.00	699.27	72.24	27.36	100.80
T2	376.72	359.19	5.89	6.60	748.41	74.88	45.36	124.56
R1	12,603.79	362.13	6.81	25.28	12,998.01	11,499.12	29.76	139.92
R2	46,968.21	365.36	7.61	22.81	47,363.98	44,038.56	47.52	567.36
P1	450.66	258.23	5.88	2.85	717.62	298.20	14.64	85.66
P2	718.48	282.97	6.85	10.41	1,018.71	556.61	20.13	90.56
Q1	274.03	354.56	5.88	7.98	642.46	20.64	32.16	91.68
Q2	282.64	354.55	5.58	7.58	650.36	19.92	30.24	104.64

Table 13. We breakdown the average iteration time (ms) for fine-tuning with various compression techniques when using TP=2 and PP=2, batch size 32, and sequence length 512. The results are collected from the local machine **without NVLink**. The total time (ms) is divided into following parts: forward step, backward step, optimizer, and waiting & pipeline communication. The last three columns further breakdown the tensor encoder/decoder and communication times which are considered part of the forward step.

A.6 The results of strong scaling experiments

Takeaway 11 *There is a noticeable increase in benefits from model parallelism compression when the number of nodes for a fixed model is adequately incremented.*

As denoted in Table 14, for a static model, we observe an upward trend in the advantages gained from model parallelism compression when the count of nodes escalates from 8 to 64.

hidden size	# layers	# nodes	batch size	speedup
25600	128	8	3072	1.04×
25600	128	16	3072	1.07×
25600	128	32	3072	1.14×
25600	128	64	3072	1.26×

Table 14. Strong-scaling speedup for the Transformer models. The number of tensor model parallelism is 4, and the micro-batch size is $\min\{128, \text{batch size}/\#\text{ nodes}\}$. As for the hidden size, the number of layers, and the batch size, we follow the setting of Table 1 in (Narayanan et al., 2021b).

A.7 Slow network

Previous research (Wang et al., 2022) demonstrates that one application of activation compression is to speed up the fine-tuning process in slow network environments. In this section, we demonstrate that our cost model, as outlined in Eq. 3, enables a greater overall speedup in slow network environments compared to data center networks. The proof is provided below:

Proof 1 *Referring to the analytical cost model described in Section 3.3, we assume that w represents the bandwidth of the data center network and w' represents the bandwidth of a slower network, where $w' < w$. To simplify the notation, we define $A = (\frac{m-1}{n} + 1) \times L \times T$, $B = (\frac{m-1}{n} + 1) \times L \times T_X$, $C = (n-1) \times \frac{Bsh}{w}$, and $D = (n-1) \times \frac{M_c}{w}$.*

It is evident that $T_X > T$ and $M_c < Bsh$. Consequently, this leads us to the conclusion that $\frac{A}{B} < 1 < \frac{C}{D}$. Next, we show

that

$$\frac{A+C}{B+D} < \frac{A+\alpha C}{B+\alpha D}$$

where $\alpha = \frac{w}{w'} > 1$. *The detailed steps are outlined below.*

$$\begin{aligned} \frac{A+C}{B+D} &< \frac{A+\alpha C}{B+\alpha D} \\ \Leftarrow (A+C)(B+\alpha D) &< (B+D)(A+\alpha C) \\ \Leftarrow BC+\alpha AD &< AD+\alpha BC \\ \Leftarrow AD &< BC \\ \Leftarrow \frac{A}{B} &< \frac{C}{D} \end{aligned}$$

This finishes the proof.

A.8 More evaluation over A100 GPUs

In this section, we evaluate OPT-3B (Zhang et al., 2022) over Cloudlab (Duplyakin et al., 2019) d8545 instances where each instance is equipped with 4 A100 GPUs and NVLink in each instance. The results are shown in Table 15. Given that the hidden dimension for OPT-3B is 2560, we set the output dimensions of A1 and A2 to 128 and 256, respectively, to maintain the same compression ratio used in prior experiments. Other compression methods maintain the same compression ratio as the autoencoder. In addition, Q1 and Q2 still reduce the precision to 2 bits and 4 bits respectively.

Distributed Setting	w/o	A1	A2	T1	T2	P1
TP=2, PP=2	**390.35**	412.36	417.22	500.61	516.47	890.64
Distributed Setting	w/o	P2	R1	R2	Q1	Q2
TP=2, PP=2	**390.35**	1,435.42	13,840.75	28,894.11	433.87	470.00

Table 15. The average iteration time (ms) for pre-training OPT-3B with various compression techniques by setting TP=2, PP=2. The results are collected from the Cloudlab d8545 machine **with NVLink** by using batch size 16, and sequence length 512. The best setting is **bolded** in the table. And the settings which see benefits compared with the baseline, are underlined.

B COMPRESSION METHODS IMPLEMENTATION DETAILS

In this section, we present more details about the model parallelism compression implementation.

B.1 Auto-encoders

This implements compression for auto-encoders (AEs).

Algorithm 1 auto-encoders compression

We use `torch.nn.init.xavier_uniform_` to initial two learnable matrices $W_e \in \mathbb{R}^{h \times h_c}$ (the encoder) and $W_d \in \mathbb{R}^{h_c \times h}$ (the decoder). Based on W_e and W_d, we do the compression and decompression operation for each activation $A \in \mathbb{R}^{B \times s \times h}$

Procedure Compress($A \in \mathbb{R}^{B \times s \times h}$)

$A_c \leftarrow AW_e \in \mathbb{R}^{B \times s \times h_c}$

return A_c

Procedure Aggregate+Decompress(worker's compressed activation $A_{c,1}, \ldots, A_{c,n}$)

$\bar{A}_c \leftarrow \sum_{i=1}^{n} A_{c,i}$

$\hat{A} \leftarrow \bar{A}_c W_d \in \mathbb{R}^{B \times s \times h_c}$

return $\hat{A}$

B.2 Top-K

This implements compression of Top-K.

Algorithm 2 Top-K compression

We do the compression and decompression operation for each activation $A \in \mathbb{R}^{B \times s \times h}$

Procedure Compress($A \in \mathbb{R}^{B \times s \times h}$)

We treat A as a vector of length Bsh

Obtain top-k absolute values to get index $I \in \mathbb{R}^k$ and value $V \in \mathbb{R}^k$

return I, V

Procedure Aggregate+Decompress(worker's index $I_1, \ldots, I_n$, value $V_1, \ldots, V_n$)

$\hat{A} \leftarrow \mathbf{0} \in \mathbb{R}^{B \times s \times h}$

$\hat{A} \leftarrow \hat{A} + \sum_{i=1}^{n} \texttt{Decompress}(I_i, V_i)$

return $\hat{A}$

Here, we present more details about the implementation of Top-K compression. As for the `Compress` operation in Algorithm 2, we use `torch.topk` function to get top-k absolute values. For the `Aggregate` step, we use `torch.distributed.all_gather` function to gather index and value from each worker. As for the `Decompress` operation, we utilize `torch.sparse_coo_tensor` to get the sparse matrix first, then we use `to_dense` function to obtain the matrix after the decompression.

B.3 Random-K

This implements compression of Random-K.

Here, we present more details about the implementation of Random-K compression. As for the `Compress` operation in Algorithm 3, we use `random.sample` function to sample a set of k absolute values. For the `Aggregate` step, we use `torch.distributed.all_gather` function to gather index and value from each worker. As for the `Decompress` operation, we utilize `torch.sparse_coo_tensor` to get the sparse matrix first, then we use `to_dense` function to obtain the matrix after the decompression.

Algorithm 3 Random-K compression

We do the compression and decompression operation for each activation $A \in \mathbb{R}^{B\times s\times h}$
Procedure Compress($A \in \mathbb{R}^{B\times s\times h}$)
  We treat A as a vector of length Bsh
  Sample a set of k values without replacement, using the same seed on all workers to get index $I \in \mathbb{R}^k$ and value $V \in \mathbb{R}^k$
  return I, V
Procedure Aggregate+Decompress(worker's index $I_1, \ldots, I_n$, value $V_1, \ldots, V_n$)
  $\hat{A} \leftarrow \mathbf{0} \in \mathbb{R}^{B\times s\times h}$
  $\hat{A} \leftarrow \hat{A} + \sum_{i=1}^{n} \texttt{Decompress}(I_i, V_i)$
  return $\hat{A}$

B.4 PowerSGD

This implements compression of PowerSGD methods.

Algorithm 4 PowerSGD compression

Given a corresponding $Q \in \mathbb{R}^{B\times h\times r}$, we do the compression and decompression operation for each activation $A \in \mathbb{R}^{B\times s\times h}$
Procedure Compress($A \in \mathbb{R}^{B\times s\times h}$, $Q \in \mathbb{R}^{B\times h\times r}$)
  $P \leftarrow AQ \in \mathbb{R}^{B\times s\times r}$
  $P \leftarrow \texttt{All\_Reduce\_Mean}(P)$
  $\hat{P} \leftarrow \texttt{Orthogonalize}(P)$
  $Q \leftarrow M^\top \hat{P}$
  $Q \leftarrow \texttt{All\_Reduce\_Mean}(Q)$
  return $\hat{P}, Q$
Procedure Aggregate+Decompress($\hat{P} \in \mathbb{R}^{B\times s\times r}$, $Q \in \mathbb{R}^{B\times h\times r}$)
  return $\hat{P}Q^\top$

For the `Orthogonalize` step in Algorithm 4, we use the `torch.linalg.qr` function to get the QR decomposition results.

B.5 Quantization-based

This implements compression of Quantization-based methods.

Algorithm 5 Quantization-based compression

We do the compression and decompression operation for each activation $A \in \mathbb{R}^{B\times s\times h}$ by using quantization function Q
Procedure Compress($A \in \mathbb{R}^{B\times s\times h}$)
  $V, S \leftarrow Q(A)$
  return V, S
Procedure Aggregate+Decompress(worker's value $V_1, \ldots, V_n$, scale $S_1, \ldots, S_n$)
  $\hat{A} \leftarrow \mathbf{0} \in \mathbb{R}^{B\times s\times h}$
  $\hat{A} \leftarrow \hat{A} + \sum_{i=1}^{n} \texttt{Decompress}(V_i, S_i)$
  return $\hat{A}$

The implementation of the Quantization function Q is based on the code released by (Wang et al., 2022).

C PRESENT EXPERIMENTAL RESULTS USING TABLES

For the sake of clarity and precision, this section incorporates tables detailing the experimental results previously discussed in the paper. These tabulated results allow readers to pinpoint the exact figures reported. The tables align with the corresponding figures as follows:

- Table 16 corresponds to Figure 3(a).
- Table 17 pairs with Figure 3(b).
- Table 18 matches Figure 5(a).
- Table 19 aligns with Figure 5(b).
- Finally, Table 20 and Table 21 relate to Figure 6(a) and Figure 6(b) respectively.

Distributed Setting	w/o	A1	A2	T1	T2	P1
TP=1, PP=4	591.96	591.36	591.47	599.65	605.05	587.90
TP=2, PP=2	440.71	437.98	444.02	493.16	528.93	750.97
TP=4, PP=1	**261.48**	270.22	275.54	356.57	409.23	848.84
Distributed Setting	w/o	P2	R1	R2	Q1	Q2
TP=1, PP=4	591.96	609.61	1,824.36	5,572.87	595.29	595.45
TP=2, PP=2	440.71	1,044.70	17,117.01	71,058.64	489.27	486.54
TP=4, PP=1	**261.48**	850.11	16,990.88	65,121.79	347.68	350.50

Table 16. The average iteration time (ms) for fine-tuning with various compression techniques by varying the distributed setting. The results are collected from the AWS p3.8xlarge machine **with NVLink** by using batch size 32, and sequence length 512. The best setting is **bolded** in the table. And the settings which see benefits compared with the baseline, are underlined.

Distributed Setting	w/o	A1	A2	T1	T2	P1
TP=2, PP=8	1,625.16	1,550.18	1,579.70	1,593.37	1,682.87	2,311.99
TP=4, PP=4	1,422.40	1,242.97	**1,223.20**	1,410.47	1,721.87	2,840.97
TP=8, PP=2	15,642.30	14,577.29	14,073.45	18,919.92	27,152.07	17,289.43
Distributed Setting	w/o	P2	R1	R2	Q1	Q2
TP=2, PP=8	1,625.16	4,043.56	55,925.28	>100,000	1,759.27	1,752.24
TP=4, PP=4	1,422.40	5,390.51	87,421.46	>100,000	2,435.03	2,594.94
TP=8, PP=2	15,642.30	30,332.80	>100,000	>100,000	16,414.57	16,517.44

Table 17. The average iteration time (ms) for pre-training with various compression techniques by varying the distributed setting. The results are collected from 4 AWS p3.8xlarge machines **with NVLink** by using micro-batch size 128, global batch size 1024, and sequence length 128. The best setting is **bolded** in the table. And the settings, under which we can gain benefits compared with the baseline, are underlined.

Distributed Setting	w/o	A1	A2	T1	T2	P1
TP=1, PP=4	151.82	154.62	155.03	156.84	158.58	162.71
TP=2, PP=2	145.58	157.49	163.63	186.71	178.91	286.59
TP=4, PP=1	**136.66**	155.43	145.97	186.06	190.01	467.49
Distributed Setting	w/o	P2	R1	R2	Q1	Q2
TP=1, PP=4	151.82	182.81	449.70	1,292.15	154.30	153.65
TP=2, PP=2	145.58	506.52	3,915.32	15,732.57	178.09	175.23
TP=4, PP=1	**136.66**	465.98	3,915.52	15,469.87	188.10	168.90

Table 18. The total time (ms) for fine-tuning with various compression techniques by varying the distributed setting. The results are collected from the AWS p3.8xlarge machine **with NVLink** by using batch size 32, and sequence length 128. The best setting is **bolded** in the table. And the settings which see benefits compared with the baseline, are underlined.

Distributed Setting	w/o	A1	A2	T1	T2	P1
TP=1, PP=4	**106.04**	113.67	106.35	109.18	110.57	126.29
TP=2, PP=2	121.26	142.41	154.60	162.00	157.12	194.67
TP=4, PP=1	122.22	142.33	139.47	172.69	170.61	336.59
Distributed Setting	w/o	P2	R1	R2	Q1	Q2
TP=1, PP=4	**106.04**	130.72	187.59	333.61	108.18	109.56
TP=2, PP=2	121.26	347.85	998.51	3,197.42	163.18	155.48
TP=4, PP=1	122.22	336.89	1,007.65	3,406.20	171.06	163.96

Table 19. The total time (ms) for fine-tuning with various compression techniques by varying the distributed setting. The results are collected from the AWS p3.8xlarge machine **with NVLink** by using batch size 8, and sequence length 128. The best setting is **bolded** in the table. And the settings which see benefits compared with the baseline, are underlined.

Distributed Setting	w/o	A1	A2	T1	T2	P1
TP=1, PP=4	154.82	**152.50**	153.47	156.81	158.37	152.83
TP=2, PP=2	184.48	175.29	180.35	207.66	214.30	283.43
TP=4, PP=1	212.76	201.39	200.31	242.62	261.39	467.72
Distributed Setting	w/o	P2	R1	R2	Q1	Q2
TP=1, PP=4	154.82	176.15	368.95	963.62	155.33	154.85
TP=2, PP=2	184.48	501.01	2,900.86	10,499.14	188.82	189.14
TP=4, PP=1	212.76	498.53	2,973.04	10,891.70	225.42	230.69

Table 20. The total time (ms) for fine-tuning with various compression techniques by varying the distributed setting. The results are collected from the local machine **without NVLink** by using batch size 32, and sequence length 128. The best setting is **bolded** in the table. And the settings which see benefits compared with the baseline, are underlined.

Distributed Setting	w/o	A1	A2	T1	T2	P1
TP=1, PP=4	73.19	72.94	**72.58**	73.62	74.86	118.18
TP=2, PP=2	100.86	107.73	100.54	114.86	112.11	183.97
TP=4, PP=1	100.73	107.90	115.18	136.18	133.91	332.64
Distributed Setting	w/o	P1	R1	R2	Q1	Q2
TP=1, PP=4	73.19	124.76	123.78	239.81	73.33	74.41
TP=2, PP=2	100.86	340.98	769.47	2,183.39	111.61	106.75
TP=4, PP=1	100.73	330.98	733.03	2,509.73	120.14	114.73

Table 21. The total time (ms) for fine-tuning with various compression techniques by varying the distributed setting. The results are collected from the local machine **without NVLink** by using batch size 8, and sequence length 128. The best setting is **bolded** in the table. And the settings which see benefits compared with the baseline, are underlined.