Fine-tuning Language Models over Slow Networks using Activation Quantization with Guarantees

Jue Wang\(^1,3\), Binhang Yuan\(^1\), Luka Rimanic\(^1\), Yongjun He\(^1\), Tri Dao\(^2\), Beidi Chen\(^34\), Christopher Ré\(^2\), Ce Zhang\(^1\)

\(^1\)ETH Zürich, Switzerland  \(^2\)Stanford University, USA  \(^3\)Zhejiang University, China  \(^4\)Carnegie Mellon University  \(^5\)Meta AI

\{juewang, binhang.yuan, luka.rimanic, yongjun.he, ce.zhang\}@inf.ethz.ch  
\{beidic, trid, chrismre\}@stanford.edu

Abstract

Communication compression is a crucial technique for modern distributed learning systems to alleviate their communication bottlenecks over slower networks. Despite recent intensive studies of gradient compression for data parallel-style training, compressing the activations for models trained with pipeline parallelism is still an open problem. In this paper, we propose AQ-SGD, a novel activation compression algorithm for communication-efficient pipeline parallelism training over slow networks. Different from previous efforts in activation compression, instead of compressing activation values directly, AQ-SGD compresses the changes of the activations. This allows us to show, to the best of our knowledge for the first time, that one can still achieve \(O(1/\sqrt{T})\) convergence rate for non-convex objectives under activation compression, without making assumptions on gradient unbiasedness that do not hold for deep learning models with non-linear activation functions. We then show that AQ-SGD can be optimized and implemented efficiently, without additional end-to-end runtime overhead. We evaluated AQ-SGD to fine-tune language models with up to 1.5 billion parameters, compressing activation to 2-4 bits. AQ-SGD provides up to 4.3× end-to-end speed-up in slower networks, without sacrificing model quality. Moreover, we also show that AQ-SGD can be combined with state-of-the-art gradient compression algorithms to enable “end-to-end communication compression”. All communications between machines, including model gradients, forward activations, and backward gradients are compressed into lower precision. This provides up to 4.9× end-to-end speed-up, without sacrificing model quality.

1 Introduction

Decentralized or open collaborative training has recently attracted intensive interests \([1] [2] [3] [4]\). Despite their great potential in leveraging geo-distributed powerful GPUs, the computation efficiency is severely hindered by low network bandwidth — typically in the range of 10-400Mbps \([2] [3] [4]\). Recently, efforts in improving communication efficiency have significantly decreased the dependency on fast data center networks — the gradient can be compressed to lower precision or sparsified \([5] [6] [7] [8]\), which speeds up training over low bandwidth networks, whereas the communication topology can be decentralized \([9] [10] [11] [12] [13] [14]\), which speeds up training over high latency networks. Indeed, today’s state-of-the-art training systems, such as Pytorch \([15] [16]\), Horovod \([17]\), Bagua \([18]\), and BytePS \([19]\), already support many of these communication-efficient training paradigms.

\(^*\)Equal contribution.  \(^\dagger\)Now at Google.
The FP32 We then show that 5bit 4bit We propose (b) Activation and delta test perplexity

\[ \text{|delta|} \quad \text{|activation|} \]

12 13 14 15 16 17 18 19 0.7 0.8 0.9 1.0 1.1 2.0 2.5 3.0 3.5 4.0

N/A Assumptions N/A Conv. Rate Unbiased Unbiased O O

zero-shot 0 2 4 6 8

Figure 1: (a) Fine-tune WikiText2 (b) Activation and delta

However, with the rise of large foundation models \(^2\) (e.g., BERT \(^2\), GPT-3 \(^2\), and CLIP \(^2\)), improving communication efficiency via compression becomes more challenging. Current training systems for foundation models such as Megatron \(^4\), Deepspeed \(^5\), and FairScale \(^6\), allocate different layers of the model onto multiple devices and need to communicate — in addition to the gradients on the models — the activations during the forward pass and the gradients on the activations during the backward pass. Compressing these activations leads to a very different behavior compared with compressing the gradient — simply compressing these activations in a stochastically unbiased way will lead to biases in the gradient that cannot be measured easily or expressed in closed form. This either breaks the unbiasedness assumptions made by most gradient compression results \(^5\) \(6\) \(7\) \(8\) or makes error compensation over gradient biases \(^2\) \(27\) \(28\) difficult to adopt.

Previous efforts on activation compression \(^32\) \(33\) \(34\) \(35\) \(36\) illustrate, albeit mostly empirically, that large deep learning models can tolerate some compression errors on these activation values. However, when it comes to the underlying theoretical analysis, these efforts mostly make assumptions that do not apply to neural networks with non-linear activation functions — the only two recent efforts that claim theoretical convergence analysis \(^29\) \(30\) assume that an unbiased compression on activations leads to an unbiased error on the gradient. Not surprisingly, these algorithms lead to suboptimal quality under relatively aggressive compression, illustrated in Figure \(^1\) — in many cases, using activation compression to fine-tune a model might be worse than zero-shot learning without any fine-tuning at all.

In this paper, we focus on the problem of activation compression for training language models over slow networks by asking the following:

• **Q1.** Can we design an algorithm for activation compression with rigorous theoretical guarantees on SGD convergence?

• **Q2.** Can such an algorithm be implemented efficiently without additional runtime overhead and outperform today’s activation compression algorithms without accuracy loss?

Our answers to both questions are Yes. **(Contribution 1)** We propose AQ-SGD, a novel algorithm for activation compression. The idea of AQ-SGD is simple — instead of directly compressing the activations, compress the change of activations for the same training example across epochs. Intuitively, we expect AQ-SGD to outperform simply compressing the activations because it enables an interesting “self-enforcing” dynamics: the more training stabilizes \(\rightarrow\) the smaller the changes of the model across epochs \(\rightarrow\) the smaller the changes of activations for the same training example across epochs \(\rightarrow\) the smaller the compression error using the same #bits \(\rightarrow\) training stabilizes more. **(Contribution 2)** The theoretical analysis of AQ-SGD is non-trivial since we have to analyze the above dynamics and connect it to SGD convergence, which is quite different from most of today’s results on gradient compression and error compensation. Under mild technical conditions and quantization functions with bounded error, we show that AQ-SGD converges with a rate of \(O(1/\sqrt{T})\) for non-convex objectives, the same as vanilla SGD \(^31\) \(37\). To the best of our knowledge, AQ-SGD is the first activation compression algorithm with rigorous theoretical analysis that shows a convergence rate of \(O(1/\sqrt{T})\) (without relying on assumptions of unbiased gradient). **(Contribution 3)** We then show that AQ-SGD can be
optimized and implemented efficiently\(^2\) without adding additional end-to-end runtime overhead over non-compression and other compression schemes (it does require us to utilize more memory and SSD for storage of activations). \textbf{(Contribution 4)} We then conduct extensive experiments on sequence classification and language modeling datasets using DeBERTa-1.5B and GPT2-1.5B models, respectively. We show that AQ-SGD can aggressively quantize activations to 2-4 bits without sacrificing convergence performance, where direct quantization of activations fails to converge; in slow networks, AQ-SGD achieves up to 4.3× end-to-end speedup. \textbf{(Contribution 5)} Last but not least, we also show that AQ-SGD can be combined with state-of-the-art gradient compression algorithms to enable “end-to-end communication compression”: \textit{All data exchanges between machines, including model gradients, forward activations, and backward gradients are quantized into lower precision}. This provides up to 4.9× end-to-end speed-up, without sacrificing model quality.

2 Overview and Problem Formulation

Training large language models over multiple devices is a challenging task. Because of the vast number of parameters of the model and data examples, state-of-the-art systems need to combine different forms of parallelism. Figure\(^2\) illustrates an example in which such a model is distributed over four machines: \textit{(Pipeline Model Parallelism)} The model is partitioned onto Machine 1 and Machine 2 (similarly, Machine 3 and Machine 4), each of which is holding a subset of layers. To compute the gradient over the model using backpropagation, these two machines need to communicate the activations during the forward pass and the gradient on activations during the backward pass. \textit{(Data Parallelism)} Machine 1 and Machine 3 (similarly, Machine 2 and Machine 4) process the same set of layers for different macro-batches. In this case, each of them will hold a replica of the same model. After the gradient over their model parameters are ready, they need to communicate the model gradient, usually by taking an average \[15, 17, 18\].

Communication Compression for Forward Activations and Backward Gradients. In slow networks, the communication among all machines often becomes the bottleneck \[37\]. To improve the speed of training, one can conduct \textit{lossy compression} of the data before they are communicated, illustrated as the C module in Figure\(^2\). When the model fits into a single machine, there have been intensive efforts on compressing the model gradient \[5, 6, 7, 8\]. However, when it comes to pipeline model parallelism, such compression techniques are less studied. In this paper, we focus on designing

\(^2\)Our code is available at: \url{https://github.com/DS3Lab/AC-SGD}
efficient communication compression algorithms to compress both forward activations and backward gradients. As we will show later, both can be compressed significantly with AQ-SGD without hurting the model quality. We also show that it is possible to combine AQ-SGD with state-of-the-art gradient compression techniques to enable the end-to-end compression scheme illustrated in Figure 2.

**Problem Formulation.** In this paper, we focus on the following technical problem. Note that, for the simplicity of notations, we present here the case where the model is partitioned onto $K = 2$ machines. AQ-SGD works for cases with $K > 2$: (1) in experiments, we consider $K = 8$, i.e., a single model is partitioned onto 8 machines; (2) in the supplementary material we provide the theoretical analysis for $K > 2$.

Given a distribution of samples $D$, we consider the following optimization task:

$$
\min_{x \in \mathbb{R}^d} \ f(x) := E_{\xi \sim D} F(b(a(\xi, x^{(a)}), x^{(b)})),
$$

(2.1)

where $F$ is a loss function, $a(-)$ and $b(-)$ correspond to two sets of layers of the model — $a(-)$ has model $x^{(a)}$ and $b(-)$ has model $x^{(b)}$. In Figure 3, Machine 1 would hold $x^{(a)}$ and Machine 2 would hold $x^{(b)}$. In the following, we call the machine that holds $x^{(a)}$ Machine $a$ and the machine that holds $x^{(b)}$ Machine $b$. In the standard backpropagation algorithm, the communication between these two machines are as follows:

- Given a data sample $\xi$, Machine $a$ sends to Machine $b$ the forward activation: $a(\xi, x^{(a)})$
- Machine $b$ sends to Machine $a$ the backward gradient on $a(\xi, x^{(a)})$.

**Difficulties in Direct Quantization.** A natural way at compressing forward activations is to send, instead of $a(\xi, x^{(a)})$, a quantized version $m(\xi, x^{(a)}) = Q(a(\xi, x^{(a)}))$. This is the quantization scheme that state-of-the-art methods such as AC-GC [29] and TinyScript [30] use. Both AC-GC [29] and TinyScript [30] assume that gradient is unbiased when $m(\xi, x^{(a)})$ is an unbiased estimator of $a(\xi, x^{(a)})$. This enables their convergence rates of $O(1/\sqrt{T})$. However, because of the non-linearity of $F$ and $b$ in a deep learning model with non-linear activation functions, an unbiased $m(\xi, x^{(a)})$ will lead to biases on the gradient. In Appendix, we will provide an example showing that such a gradient bias will hurt SGD convergence even for a very simple optimization problem. On the theory side, previous efforts on understanding gradient bias [38] have also shown that even bounded bias on gradient can impact the converges of SGD. As we will show later, empirically, this bias can indeed lead to suboptimal models under aggressive compression.

**Notation.** Throughout the paper we use the following definitions:

- $f^*$ is the optimal value of $f$.
- $N$ is the number of samples.
- $x_t = (x^{(a)}_t, x^{(b)}_t)$ is the full model at iteration $t$.
- $\nabla f(\cdot)$ is the gradient of function $f$.
- $g_{t_1}(x_t) = \nabla F(\xi_t; x_t)$ is the stochastic gradient.
- $Q(\cdot)$ is the quantization function used to compress activations.
- $m(\cdot)$ is the message exchanged between $a$ and $b$ in the feed forward path.
- $\|\cdot\|$ denotes the $L_2$-norm.

### 3 AQ-SGD: Theoretical Analysis and System Implementations

In this section we present the AQ-SGD, with the goal to mitigate the above mentioned difficulties that appear in direct quantization of the activation functions.

#### 3.1 AQ-SGD Algorithm

Algorithm 1 illustrates the AQ-SGD algorithm. The idea behind Algorithm 1 is simple — instead of quantizing the activations directly, quantize the changes of activations for the same training example across epochs. As illustrated in Algorithm 1 for iteration $t$ and the data sample $\xi_t$, if it is the first time
Algorithm 1 AQ-SGD Algorithm

1: Initialize: $x_0$, learning rate $\gamma$, sub-network $a(-)$ weights $x^{(a)}$, sub-network $b(-)$ weights $x^{(b)}$, quantization function $Q$, array of previous messages $m$ initialized to 0
2: for $t = 1, \ldots, T$ do
3: Randomly sample $\xi_t$
4: if $\xi_t$ not seen before then
5: Set $m(\xi_t) = a(\xi_t, x^{(a)}_t)$
6: else
7: Update $m(\xi_t) \leftarrow m(\xi_t) + Q(a(\xi_t, x^{(a)}_t) - m(\xi_t))$
8: end if
9: // Machine $a$ sends $Q(a(\xi_t, x^{(a)}_t) - m(\xi_t))$ to Machine $b$, which knows $m(\xi_t)$ through a local version of $m$
10: Update $x^{(a)}_{t+1} \leftarrow x^{(a)}_t - \gamma \cdot \nabla_{x^{(a)}}(f \circ b)|_m$
11: // Machine $b$ sends $Q(\nabla_a(f \circ b)|_m)$ to Machine $a$
12: Update $x^{(a)}_{t+1} \leftarrow x^{(a)}_t - \gamma \cdot Q(\nabla_a(f \circ b)|_m) \cdot \nabla_{x^{(a)}}a$
13: end for
14: Output: $x = (x^{(a)}_T, x^{(b)}_T)$

that $\xi_t$ is sampled. Machine $a$ communicates the full precision activations without any compression: $m(\xi_t) = a(\xi_t, x^{(a)}_t)$ (Lines 4-5). Both machines will save $m(\xi_t)$ in a local buffer. If $\xi_t$ has been sampled in previous iterations, Machine $a$ communicates a quantized version:

$$Q(a(\xi_t, x^{(a)}_t) - m(\xi_t)),$$

where $m(\xi_t)$ was the previous message, stored in the local buffer. Both machines then update this local buffer:

$$m(\xi_t) \leftarrow m(\xi_t) + Q(a(\xi_t, x^{(a)}_t) - m(\xi_t)).$$

Machine $b$ then use $m(\xi_t)$ as the forward activations, compute backward gradients, and communicate a quantized version of the backward gradient to Machine $a$ (Line 11). We use

$$\delta_\xi = a(\xi_t, x^{(a)}_t) - m(\xi_t)$$

to denote the message error in sending the activations.

Update Rules. The above algorithm corresponds to the following update rules, at iteration $t$ with sample $\xi_t$:

$$x^{(a)}_{t+1} = x^{(a)}_t - \gamma \cdot Q(\nabla_a(f \circ b)|_{q_2(\xi_t, x^{(a)}_t)}) \cdot \nabla_{x^{(a)}}a,$$

$$x^{(b)}_{t+1} = x^{(b)}_t - \gamma \cdot \nabla_{x^{(b)}}(f \circ b)|_m,$$

where $\gamma$ is the learning rate, $\nabla_{x^{(b)}}(f \circ b)|_m$ is the gradient on $x^{(b)}$ using the quantized forward activations $(m(\xi_t, x^{(a)}_t))$, and $Q(\nabla_a(f \circ b)|_m)$ is the quantized backward gradient.

Setting $x_t = (x^{(a)}_t, x^{(b)}_t)$, we can rephrase the update rule as

$$x_{t+1} = x_t - \gamma \cdot (g_\xi(x_t) + \Delta_\xi(x_t)),$$

where $g_\xi(x_t)$ is the stochastic gradient and $\Delta_\xi(x_t)$ is the gradient error introduced by communication compression. We have $\Delta_\xi = (\Delta_{\xi}^{(a)}, \Delta_{\xi}^{(b)})$ given by:

$$\Delta_{\xi}^{(a)}(x_t) = Q(\nabla_a(f \circ b)|_{q_2(\xi_t, x^{(a)}_t)}) \cdot \nabla_{x^{(a)}}a,$$

$$\Delta_{\xi}^{(b)}(x_t) = \nabla_{x^{(b)}}(f \circ b)|_m \cdot \nabla_{x^{(b)}}a,$$

where $\Delta_{\xi}^{(a)}(x_t)$ is the error introduced by the gradient quantization in the backpropagation part, whilst $\Delta_{\xi}^{(a)}(x_t)$ and $\Delta_{\xi}^{(b)}(x_t)$ are the errors that the gradients of $a$ and $b$, respectively, inherit from the bias introduced in the forward pass.
3.2 Theoretical Analysis

We now prove the main theorem which states that, under some standard assumptions that are often used in the literature [31][37], the convergence rate of ΛQ-SGD algorithm is $O(1/\sqrt{T})$ for non-convex objectives, same as vanilla SGD.

**Assumptions.** We make several assumptions on the networks and the quantization. It is important to note that we put no restrictions on either the message error $\delta_\xi$, nor the gradient error $\Delta_\xi$.

- **(A1: Lipschitz assumptions)** We assume that $\nabla f$, $\nabla (f \circ b)$ and $a$ are $L_f$, $L_{fob}$, and $\ell_a$-Lipschitz, respectively, recalling that a function $g$ is $L_g$-Lipschitz if
  \[
  \|g(x) - g(y)\| \leq L_g \|x - y\|, \quad \forall x, \forall y.
  \]
  Furthermore, we assume that $a$ and $f \circ b$ have gradients bounded by $C_a$ and $C_{fob}$, respectively, i.e. $\|\nabla a(x)\| \leq C_a$, and $\|\nabla (f \circ b)(x)\| \leq C_{fob}$.

- **(A2: SGD assumptions)** We assume that the stochastic gradient $g_\xi$ is unbiased, i.e. $\mathbb{E}_\xi [g_\xi(x)] = \nabla f(x)$, for all $x$, and with bounded variance, i.e. $\mathbb{E}_\xi \|g_\xi(x) - \nabla f(x)\|^2 \leq \sigma^2$, for all $x$.

**Theorem 3.1.** Suppose that Assumptions A1, A2 hold, and consider an unbiased quantization function $Q(x)$ which satisfies that there exists $c_Q < \sqrt{1/T}$ such that $\mathbb{E}\|x - Q(x)\| \leq c_Q \|x\|$, for all $x$.[3] Let $\gamma = \frac{1}{3(3L_f+C)\sqrt{T}}$ be the learning rate, where
  \[
  C = \frac{4C_Q\ell_a(1 + C_a)L_{fob}N}{\sqrt{1 - 2\epsilon^2}}.
  \]
  Then after performing $T$ updates one has
  \[
  \frac{1}{T} \sum_{t \in [T]} \mathbb{E}\|\nabla f(x_t)\|^2 \leq \frac{(C + L_f)(f(x_1) - f^*)}{\sqrt{T}} + \frac{\sigma^2 + (c_QC_aC_{fob})^2}{\sqrt{T}}.
  \]

We present the full proof of Theorem 3.1 in Appendix A, whereas here we explain the main intuition. The usual starting point in examining convergence rates is to use the fact that $f$ has $L_f$-Lipschitz gradient. It is well known that this implies
  \[
  \gamma (\nabla f(x_t), g_\xi(x_t)) + f(x_{t+1}) - f(x_t) \leq -\langle \nabla f(x_t), \Delta_\xi(x_t) \rangle + \frac{\gamma^2 L_f}{2} \|g_\xi(x_t) + \Delta_\xi(x_t)\|^2.
  \]
  After taking the expectation over all $\xi_t$ and summing over all $t = 1, \ldots, T$, we easily see that the key quantity to bound is $\sum_{t=1}^T \mathbb{E} \|\Delta_\xi(x_t)\|^2$, where $\Delta_\xi(x_t) = (\Delta^{(a)}_\xi(x_t), \Delta^{(b)}_\xi(x_t))$. On the other hand, the main object that we can control is the message error, $\delta_\xi$. Therefore, we first prove an auxiliary result which shows that $\|\Delta_\xi(x_t)\| \leq (1 + C_a)\ell_a \|\delta_\xi(x_t)\|$, for all $t$. The key arguments for bounding $\delta_\xi$ closely follow the self-improving loop described in the introduction, and can be summarized as follows. Since we are compressing the information in such a way that we compare with all the accumulated differences, this allows us to unwrap the changes which appeared since the last time that we observed the current sample, in an iterative way. However, since these are gradient updates, they are bounded by the learning rate — as long as we have a quantization method that keeps enough information about the signal, we can recursively build enough saving throughout the process. In particular, the more stability we have in the process, the smaller the changes of the model and the compression error gets, further strengthening the stability.

**Tightness.** The bound is tight with respect to quantization — setting $c_Q = 0$ (implying $C = 0$), i.e. quantization does not incur any loss, recovers the original original SGD convergence (cf. [31][37]).

---

[3] Even for a very simple quantization function $Q(x) = \|x\| \cdot \lfloor |x|/\|x\| \rfloor$, where $\lfloor \cdot \rfloor$ denotes rounding to the closest $k/2^n$, stochastically, through a simple bound $c_Q = \sqrt{d}/2^6$; 6 bits suffice in a low-dimensional ($\sim 10^4$), 11 bits in a high-dimensional scenario ($\sim 10^9$), and 16 bits in a super-high-dimensional scenario ($\sim 10^{10}$). In practice, as we show in the experiments, we observe that for 2-4 bits are often enough for fine-tuning GPT-2 style model. This leaves interesting direction for future exploration as we expect a careful analysis of sparsity together with more advanced quantization functions can make this condition much weaker.
Regularization and other optimizers. Assuming A1 and A2 for \( f \), and under further assumptions on \( b \) and \( \nabla f(x) \), one can prove that the \( L_2 \)-regularized loss \( \tilde{f}(x) = f(x) + \frac{\lambda}{2} \| x \|^2 \), which results in weight decay, satisfies Assumptions A1 and A2 with slightly weaker constants. We note that a theoretical analysis for other regularization methods or optimizers such as Adam [39], could be an independent study and represent an interesting line of future research.

3.3 System Implementations and Optimizations.

Additional storage and communication. AQ-SGD requires us to store, for each data example \( \xi \), the quantized activation \( m(\xi) \) in a local buffer in memory or SSD. For example, in GPT2-XL training, a simple calculation shows that we need an approximately extra 1TB storage. When using data parallelism, it reduces to 1TB / \# parallel degree, but also incurs communication overhead if data is shuffled in every epoch. In addition, when the example \( \xi \) is sampled again, \( m(\xi) \) needs to be (1) loaded from this local buffer to the GPU, and (2) updated when a new value for \( m(\xi) \) is ready.

Optimization. It is easy to implement and optimize the system such that this additional loading and updating step do not incur significant overhead on the end-to-end runtime. This is because of the fact that the GPU computation time for a forward pass is usually much longer than the data transfer time to load the activations — for GPT2-XL with 1.5 billion model parameters, a single forward pass on 6 layers require 44 ms, whereas loading \( m(\xi) \) need 0.2 ms from memory and 12 ms from SSD. One can simply pre-fetch \( m(\xi) \) right before the forward pass, and hide it within the forward pass of other data examples. Similarly, updating \( m(\xi) \) can also be hidden in the backward computation. It is also simple to reduce the communication overhead by shuffling data only once or less frequently.

4 Evaluation

We demonstrate that AQ-SGD can significantly speed up fine-tuning large language models in slow networks. Specifically, we show: (1) on four standard benchmark tasks, AQ-SGD can tolerate aggressive quantization on the activations (2-4 bits) and backward gradients (4-8 bits), without hurting convergence and final loss, whereas direct quantization converges to a worse loss or even diverge, (2) in slow networks, AQ-SGD provide an end-to-end speed-up up to \( 4.3 \times \), and (3) AQ-SGD can be combined with state-of-the-art gradient compression methods and achieve an end-to-end speed-up of up to \( 4.9 \times \).

4.1 Experimental Setup

Datasets and Benchmarks. We consider both sequence classification and language modeling tasks with state-of-the-art foundation models. For sequence classification, we fine-tune a 1.5B parameter DeBERTa\(^4\) on two datasets: QNLI and CoLA. For language modeling, we fine-tune the GPT2 model with 1.5B parameters\(^5\) on two datasets: WikiText2 and arXiv abstracts. All datasets are publicly available and do not contain sensitive or offensive content. Detailed setup can be found in Appendix [B].

Distributed Cluster. We conduct our experiments on AWS with 8-32 p3.2xlarge instances, each containing a V100 GPU. For a single pipeline, we partition a model onto 8 machines. When combined with data parallelism, we use 32 instances — data parallel degree is 4 and pipeline parallel degree is 8. By default, instances are interconnected with 10Gbps bandwidth. We simulate slow networks by controlling the communication bandwidth between instances using Linux traffic control.

Baselines. We compare with several strong baselines:

1. FP32: in which all communications are in 32 bit floating point without any compression.
2. DirectQ \([29, 30]\): in which activations and backward gradients are directly quantized.

We use a simple, uniform quantization scheme, which first normalizes a given vector into \([-1, 1]\) and quantize each number into a \( b \)-bit integers by uniforming partitioning the range \([-1, 1]\) into \( 2^b \) intervals \([35]\). Additional details of the configuration can be found in Appendix [C].

---

\(^4\)We use the v2-xxlarge checkpoint: \[https://huggingface.co/microsoft/deberta-v2-xxlarge\].
\(^5\)We use the extra large checkpoint: \[https://huggingface.co/gpt2-xl\].
We note that computation and communication can overlap, so the end-to-end time depends on the communication bandwidth. If communication were to be deployed in a real-world geo-distributed decentralized network, the training throughput would be almost as fast as the performance inside a data center!

As illustrated in Figure 4, AQ-SGD converges fast since it does not introduce any compression errors. DirectQ, under aggressive quantization, can converge to a significantly worse model, or even diverge. This is not surprising, given the biases on model gradients that direct quantization introduced. On the other hand, AQ-SGD converges almost as fast as FP32 in terms of number of training steps.

**Hyperparameter Tuning.** We conduct careful tuning for all methods on all datasets. We perform grid search to choose learning rate from \{2.5e-6, 3e-6, 5e-6, 1e-5\} and macro-batch size from \{32, 64, 96\} for best model performance. We train all models using the Adam optimizer with weight decay.

**4.2 Results**

**Convergence.** We first compare the convergence behavior of different methods. For all compression methods, we try various settings: \(fw \times bw\) means that we use \(x\) bits for forward activation and \(y\) bits for backward gradients. Figure 3 shows the convergence behavior for the sequence classification and language modeling tasks. FP32 converges fast since it does not introduce any compression errors. DirectQ, under aggressive quantization, can converge to a significantly worse model, or even diverge. This is not surprising, given the biases on model gradients that direct quantization introduced. On the other hand, AQ-SGD converges almost as fast as FP32 in terms of number of training steps.

**End-to-End Runtime.** We show the end-to-end runtime of different methods under slow networks. As illustrated in Figure 4, AQ-SGD achieves a 4.3x end-to-end speed-up comparing with that of FP32 (in terms of time to the same loss), illustrating the importance of communication compression in slow networks. Table 2 shows the training throughput and Table 3 shows the breakdown of our algorithm. We note that computation and communication can overlap, so the end-to-end time depends on the larger one of the two. Another interesting observation is that when the network is 100x slower (from 10Gbps to 100Mbps), the training is only about 1.18x slower! This is exciting — if AQ-SGD were to be deployed in a real-world geo-distributed decentralized network, the training throughput would be almost as fast as the performance inside a data center!

Moreover, AQ-SGD does not introduce significant runtime overhead over direct quantization. From Table 2, we see that AQ-SGD is essentially as efficient as direct quantization compression in throughput.
### 4.3 End-to-end Communication Compression: AQ-SGD + QuantizedAdam

AQ-SGD can also be combined with existing methods on gradient compression. This allows us to compress all communications during training. We combine AQ-SGD with QuantizedAdam [40], an error compensation-based gradient compression algorithm for data parallel training.

We quantize the forward activations with 3 bits, the backward gradient with 6 bits, and model gradient with 4 bits. Figure 5 illustrates the convergence and the training throughput under different network configuration. We see that AQ-SGD converges well when combined with QuantizedAdam (Figure 5(a, b)). On the other hand, DirectQ, when combined with gradient compression, converges to a much worse model. In terms of training throughput, with both activation and gradient compression, we can achieve up to $8.5 \times$ throughput improvement compared to the no-compression baseline (Figure 5(c)). We also see that both activation and gradient compression are important in terms of improving end-to-end training throughput — as illustrated in Figure 5(c), disabling any of them will lead to a much lower training throughput.

### 5 Related Work

**Distributed training of foundation models.** Modern distributed training of deep neural networks goes beyond data parallelism [24][41][42] due to the advance of the large-scale foundation models [20], such as BERT [21], GPT-3 [22], and CLIP [23]. Popular systems to support foundation model training include Megatron [24], Deepspeed [25], and Fairscale [26]. To scale out the training of large-scale models, pipeline parallelism (e.g., Gpipe [41], Pipedream [43][44]) is a popular option, where the model is partitioned into multiple stages, different stages are allocated on different GPUs and the exchange of activations and gradients on activations goes through network communication.

**Communication compression of distributed learning.** Communication compression is an effective system relaxation for distributed training, especially in data parallelism [40][45][46][47][48][49][50][51][52][53][54][55]. Popular techniques include quantization [56][57][58], sparsification [59], sketching [60][61] and error compensation [27]. Recently, TinyScript [30] proposes to compress activations and gradients simultaneously.

**Sparse Learning for activation compression.** Sparse learning [30][62][63][64][65][66][67][68][69] has become increasingly popular for training neural networks, as it can significantly reduce the use of computation and memory while preserving the generalization of such models. In particular,
activation compression methods [32, 33, 34, 35, 36, 29] are proposed to reduce the memory footprint by adopting lossless [70, 71, 72] and lossy [73, 29, 74] compression techniques in the training of various deep neural networks (e.g., CNN [66, 75, 76], GNN [77]). These approaches usually compute the activation with full precision in forward propagation, adopt the compression method over the activation, and store the compressed version in DRAM for later use in backward propagation. Thus, compression does not introduce any error in forward propagation in contrast to the scenario of communicating compressed activation in the distributed setting.

**Delta-based compression.** Delta-based compression [78] is a classic solution to various system problems. Recent research has also used the property of spatial proximity within activation in neural network training based on empirical observations [79, 80, 74]. However, to our knowledge, no attempt has been made to consider the proximity of activation through training epochs to enable efficient compression with theoretical guarantee.

6 Conclusion

In this paper, we discuss how to adopt communication compression for activations in distributed learning. We proposed AQ-SGD, a novel activation compression algorithm for communication-efficient pipeline parallelism training over slow networks. AQ-SGD achieves $O(1/\sqrt{T})$ convergence rate for non-convex optimization without making assumptions on gradient unbiasedness. Our empirical study suggests that AQ-SGD can achieve up to 4.3x speedup for pipeline parallelism. When combined with gradient compression in data parallelism, the end-to-end speed-up can be up to 4.9x.

Acknowledgments

CZ and the DS3Lab gratefully acknowledge the support from the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number MB22.00036 (for European Research Council (ERC) Starting Grant TRIDENT 101042665), the Swiss National Science Foundation (Project Number 200021_184628, and 197485), Innosuisse/SNF BRIDGE Discovery (Project Number 40B2-0_187132), European Union Horizon 2020 Research and Innovation Programme (DAPHNE, 957407), Botnar Research Centre for Child Health, Swiss Data Science Center, Alibaba, Cisco, eBay, Google Focused Research Awards, Kuaishou Inc., Oracle Labs, Zurich Insurance, and the Department of Computer Science at ETH Zurich. CR gratefully acknowledges the support of NIH under No. U54EB020405 (Mobilize), NSF under Nos. CCF1763315 (Beyond Sparsity), CCF1563078 (Volume to Velocity), and 1937301 (RTML); ARL under No. W911NF-21-2-0251 (Interactive Human-AI Teaming); ONR under No. N000141712266 (Unifying Weak Supervision); ONR N0001-20-1-2480: Understanding and Applying Non-Euclidean Geometry in Machine Learning: N000142012275 (NEPTUNE); NXP, Xilinx, LETI-CEA, Intel, IBM, Microsoft, NEC, Toshiba, TSMC, ARM, Hitachi, BASF, Accenture, Ericsson, Qualcomm, Analog Devices, Google Cloud, Salesforce, Total, the HAI-GCP Cloud Credits for Research program, the Stanford Data Science Initiative (SDSI), and members of the Stanford DAWN project: Facebook, Google, and VMWare. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, policies, or endorsements, either expressed or implied, of NIH, ONR, or the U.S. This work was done during Jue Wang’s visit to ETH Zürich, which was supported by Key Research and Development Program of Zhejiang Province of China (No. 2021C01009) and Fundamental Research Funds for the Central Universities. The computation required in this work was provided by Together Computer (https://together.xyz/).

References


[14] Pytorch-lightning. [https://www.pytorchlightning.ai/]


A Proof of the Main Theorem

In this section we prove Theorem 3.1. The main idea comes from the “self-enforcing” dynamics described in the introduction of this work: the more training stabilizes → the smaller the changes of the model across epochs → the smaller the changes of activations for the same training example across epochs → the smaller the compression error using the same #bits → training stabilizes more.

We start by providing two auxiliary results. The first one connects the message error and the gradient error.

Lemma A.1. For every sample ξ, one has
\[ \|\Delta(\xi, x)\| \leq C\delta(\xi, x), \]
where \( \Delta(x) = (\Delta(\xi, x), \Delta(\xi, y)) \).

Proof: Note that
\[ \|\Delta(\xi, x)\| = \|\nabla_x a(\xi, x)_{x=\xi}\| \cdot \|\nabla_y (f \circ b)(x, x), x = a(\xi, x)\| \]
\[ \leq C\|\nabla_x a(\xi, x)\|_{x=\xi} \cdot \|\nabla_x (f \circ b)(x, x), x = a(\xi, x)\| \]
\[ \leq C\delta(\xi, x), \]
which together with \( \|\Delta(\xi, x)\| = \|\Delta(\xi, x)\|_\delta + \|\Delta(\xi, y)\|_\delta \) yields the claim. \( \square \)

We now prove that the message error can be efficiently bounded by the true gradient.

Lemma A.2. For \( C' = \frac{16c_0^2l^2N^2}{1-2\epsilon Q} \), one has
\[ \frac{1}{T} \sum_{t \in [T]} E[\|\delta(\xi, x)\|^2] \leq C'^2 \left( \frac{1}{T} \sum_{t \in [T]} E[\|\nabla f(x)\|^2] + \sigma^2 \right) \left( cQ C\delta f \right)^2 \].

Proof: Let ξ be a fixed sample. To simplify the exposition, we abuse the notation slightly by \( a(x) = a(\xi, x), m(x) = m(\xi, x) \). Let \( T(\xi) \) be the number of realizations of ξ before time \( T \). Using the definition of \( \delta(\xi) \) (recalling that \( \delta(\xi, x_0(\xi)) = 0 \), since in the first iteration we send the correct signal), we have
\[ \frac{T(\xi)}{T} \sum_{k=1}^{T(\xi)} \|\delta(\xi, x_{t_k}(\xi))\|^2 = \sum_{k=2}^{T(\xi)} \|a(x_{t_k}(\xi)) - m(x_{t_{k-1}}(\xi))\|^2 \]
\[ = \sum_{k=2}^{T(\xi)} \|a(x_{t_k}(\xi)) - m(x_{t_{k-1}}(\xi)) - Q(a(x_{t_k}(\xi)) - m(x_{t_{k-1}}(\xi)))\|^2 \]
\[ \{\|x - Q(x)\| \leq cQ \|x\|\} \]
\[ \leq c_0^2 \sum_{k=2}^{T(\xi)} \|a(x_{t_k}(\xi)) - a(x_{t_{k-1}}(\xi)) + \delta(\xi, x_{t_{k-1}}(\xi))\|^2 \]
\[ \{\alpha + \beta \leq 2\alpha^2 + 2\beta^2\} \]
\[ \leq 2c_0^2 \sum_{k=2}^{T(\xi)} \|a(x_{t_k}(\xi)) - a(x_{t_{k-1}}(\xi))\|^2 + 2c_0^2 \sum_{k=2}^{T(\xi)} \|\delta(\xi, x_{t_{k-1}}(\xi))\|^2 \]
\[ \{a is \ell_2-\text{Lipschitz}\} \]
\[ \leq 2c_0^2 \sum_{k=2}^{T(\xi)} \|x_{t_k}(\xi) - x_{t_{k-1}}(\xi)\|^2 + 2c_0^2 \sum_{k=2}^{T(\xi)} \|\delta(\xi, x_{t_{k-1}}(\xi))\|^2. \]
Transferring the $\delta_\xi$ part to the LHS, and noting that between every two realizations of $\xi$ at times $t_{k-1}(\xi)$ and $t_k(\xi)$, we can follow updates for $t = t_{k-1}(\xi), \ldots, t_k(\xi) - 1$, we get

\[
(1 - 2c_\omega^2) \sum_{k=1}^{T(\xi)} \|\delta_\xi(x_{t_k(\xi)})\|^2 \leq \gamma^2 \cdot (2c_\omega^2 T)^2 \sum_{k=1}^{T(\xi)} \sum_{t = t_{k-1}(\xi)}^{t_k(\xi)-1} (g_{\xi}(x_t) + \Delta_\xi(x_t))^2 \leq \gamma^2 \cdot (2c_\omega^2 T)^2 \sum_{k=1}^{T(\xi)} \sum_{t = t_{k-1}(\xi)}^{t_k(\xi)-1} \|g_{\xi}(x_t) + \Delta_\xi(x_t)\|^2.
\]

(Cauchy-Schwarz)

\[
\leq \gamma^2 \cdot (2c_\omega^2 T)^2 \sum_{k=1}^{T(\xi)} (t_k(\xi) - t_{k-1}(\xi)) \sum_{t = t_{k-1}(\xi)}^{t_k(\xi)-1} \|g_{\xi}(x_t) + \Delta_\xi(x_t)\|^2
\]

\[
= \gamma^2 \cdot (2c_\omega^2 T)^2 \sum_{t \in [T]} \omega_t(t) \cdot \|g_{\xi}(x_t) + \Delta_\xi(x_t)\|^2,
\]

where $\omega_t: [T] \rightarrow \{0, 1, \ldots\}$ is defined by $\omega_t(t) = t_k(\xi) - t_{k-1}(\xi)$, if $t \in [t_{k-1}(\xi), t_k(\xi) - 1]$, and $\omega_t(t) = 0$, if $t > t_k(\xi)$. We note that for two different samples $\xi$ and $\xi'$, the sums on the LHS are disjoint. Therefore, summing over all samples $\xi$ and taking the expectation over $\xi$ and all $\xi_t$ yields

\[
(1 - 2c_\omega^2) \cdot \frac{1}{T} \sum_{t \in [T]} \mathbb{E}\|\delta_\xi(x_t)\|^2 \leq \gamma^2 \cdot (2c_\omega^2 T)^2 \cdot \frac{1}{T} \sum_{t \in [T]} \mathbb{E}\|g_{\xi}(x_t) + \Delta_\xi(x_t)\|^2 \cdot N \cdot \mathbb{E}[\omega_t(t)]
\]

\[
\leq \gamma^2 \cdot (2c_\omega^2 T)^2 \cdot \frac{1}{T} \sum_{t \in [T]} \mathbb{E}\|g_{\xi}(x_t) + \Delta_\xi(x_t)\|^2,
\]

since $\mathbb{E}[\omega_t(t)] \leq N$. Applying $\|g_{\xi}(x_t) + \Delta_\xi(x_t)\|^2 \leq 3\|g_{\xi}(x_t)\|^2 + 3\|\Delta_\xi^{(Q)}(x_t)\|^2 + 3\|\Delta_\xi(x_t)\|^2$, bounded variance $\mathbb{E}\|g_{\xi}(x) - \nabla f(x)\|^2 \leq \sigma^2$, and Lemma A.1 we get

\[
(1 - 2c_\omega^2 - \gamma^2 \cdot 6c_\omega^2 N^2 (1 + C_a)^2 L_{\text{fob}}^2) \cdot \frac{1}{T} \sum_{t \in [T]} \mathbb{E}\|\delta_\xi(x_t)\|^2
\]

\[
\leq \gamma^2 \cdot (6c_\omega^2 N^2) \left(2\sigma^2 + 2 \cdot \frac{1}{T} \sum_{t \in [T]} \mathbb{E}\|\nabla f(x_t)\|^2 + (c_\omega C_a C_{\text{fob}})^2\right),
\]

which implies

\[
\left(1 - 2c_\omega^2 - \gamma^2 \cdot \frac{3}{8} C^2 \cdot (1 - 2c_\omega^2)\right) \cdot \frac{1}{T} \sum_{t \in [T]} \mathbb{E}\|\delta_\xi(x_t)\|^2
\]

\[
\leq \gamma^2 \cdot (12c_\omega^2 N^2) \left(\sigma^2 + 1 \cdot \frac{1}{T} \sum_{t \in [T]} \mathbb{E}\|\nabla f(x_t)\|^2 + (c_\omega C_a C_{\text{fob}})^2\right),
\]

Recalling the definitions of $C$ and $\gamma$, and the fact that $\gamma \cdot C < \frac{1}{8}$, we can simplify the LHS to get

\[
(1 - 2c_\omega^2) \cdot \frac{1}{T} \sum_{t \in [T]} \mathbb{E}\|\delta_\xi(x_t)\|^2 \leq \gamma^2 \cdot (12 \cdot \frac{24}{23} c_\omega^2 N^2) \left(\sigma^2 + 1 \cdot \frac{1}{T} \sum_{t \in [T]} \mathbb{E}\|\nabla f(x_t)\|^2 + (c_\omega C_a C_{\text{fob}})^2\right),
\]

yielding the claim.

We are ready to prove the main result, with a learning rate of

\[
\gamma = \frac{1}{3(3L_f + C)\sqrt{T}}.
\]

**Proof of Theorem 3.1** Since $f$ has $L_f$-Lipschitz gradient, we know that

\[
f(x_{t+1}) - f(x_t) \leq -\gamma \cdot \langle \nabla f(x_t), g_{\xi}(x_t) + \Delta_\xi(x_t) \rangle + \frac{\gamma^2 L_f}{2} \|g_{\xi}(x_t) + \Delta_\xi(x_t)\|^2.
\]
Since the quantization is unbiased, implying \( \mathbb{E}_Q[\Delta^{(Q)}(x)] = 0 \), taking the expectation with respect to the quantization yields
\[
\mathbb{E}_Q[f(x_{t+1})] - \mathbb{E}_Q[f(x_t)] 
\leq -\gamma \mathbb{E}_Q(\nabla f(x_t), g_{\xi_t}(x_t) + \tilde{\Delta}_{\xi_t}(x_t)) + \frac{3\gamma^2 L_f}{2} \left( \|g_{\xi_t}(x_t)\|^2 + \|\tilde{\Delta}_{\xi_t}(x_t)\|^2 + \mathbb{E}_Q[\Delta^{(Q)}(x)] \right),
\]
where \( \tilde{\Delta}_{\xi_t}(x_t) = (\Delta^{(a)}_{\xi_t}, \Delta^{(b)}_{\xi_t}) \), and we used \( (\alpha + \beta + \rho)^2 \leq 3\alpha^2 + 3\beta^2 + 3\rho^2 \).

Taking the expectation over \( \xi_t \) (simplifying the notation of \( \mathbb{E}_Q \mathbb{E}_\xi \) to simply \( \mathbb{E} \)), and recalling that \( \mathbb{E}[g_{\xi_t}(x_t)] = \nabla f(x_t) \), we can bound the first two terms of the RHS by
\[
-\gamma \mathbb{E}[\nabla f(x_t), g_{\xi_t}(x_t) + \tilde{\Delta}_{\xi_t}(x_t)) + \frac{3\gamma^2 L_f}{2} \mathbb{E}[\|g_{\xi_t}(x_t)\|^2 + \|\tilde{\Delta}_{\xi_t}(x_t)\|^2]
\leq -\gamma \mathbb{E}[\|\nabla f(x_t)\|^2 + \frac{\gamma}{2} \mathbb{E}[\|\tilde{\Delta}_{\xi_t}(x_t)\|^2] + \frac{3}{2} \gamma^2 L_f \mathbb{E}[\|g_{\xi_t}(x_t)\|^2]
\leq \left( -\frac{\gamma}{2} + 3\gamma^2 L_f \right) \mathbb{E}[\|\nabla f(x_t)\|^2] + \left( \frac{\gamma}{2} + \frac{3\gamma^2 L_f}{2} \right) \mathbb{E}[\|\tilde{\Delta}_{\xi_t}(x_t)\|^2] + 3\gamma^2 L_f \sigma^2.
\]
Reorganizing the terms, summing over all \( t \in [T] \) and dividing by \( T \) yields
\[
\gamma \cdot \left( \frac{1}{2} - 3\gamma L_f \right) \cdot \frac{1}{T} \sum_{t \in [T]} \mathbb{E}[\|\nabla f(x_t)\|^2] 
\leq \frac{f(x_1) - \mathbb{E}[f(x_{T+1})]}{T} + \gamma \cdot C'' \cdot \frac{1}{T} \sum_{t \in [T]} \mathbb{E}[\|\tilde{\Delta}_{\xi_t}(x_t)\|^2]
\leq \frac{f(x_1)}{T} + \gamma^2 (C'' + 3\gamma L_f) \cdot (\sigma^2 + (c_{\xi_0} C_{f\xi_0})^2),
\]
where
\[
C'' = \left( \frac{1}{2} + \frac{3}{2} \gamma L_f \right) \left( 1 + C_a^2 \right) L_{f\xi_0}^2 < (1 + C_a^2) L_{f\xi_0}^2,
\]
by the definition of \( \gamma \). Applying Lemma A.2 and regrouping the terms now yields
\[
\gamma \cdot \left( \frac{1}{2} - 3\gamma L_f - \gamma^2 \cdot C'' \cdot C'' \right) \cdot \frac{1}{T} \sum_{t \in [T]} ||\nabla f(x_t)||^2
\leq \frac{f(x_1) - \mathbb{E}[f(x_{T+1})]}{T} + \gamma^2 (C'' + 3\gamma L_f) \cdot (\sigma^2 + (c_{\xi_0} C_{f\xi_0})^2).
\]
Noting that \( C'' \cdot C'' < C^2 \) and recalling that \( \gamma C < 1/3 \) and \( \gamma L_f < 1/9 \), we get
\[
\gamma \cdot \left( \frac{1}{2} - \gamma (3L_f + C) \right) \cdot \frac{1}{T} \sum_{t \in [T]} ||\nabla f_{\xi_t}(x_t)||^2 
\leq \frac{f(x_1) - \mathbb{E}[f(x_{T+1})]}{T} + \gamma^2 (3L_f + C) \cdot (\sigma^2 + (c_{\xi_0} C_{f\xi_0})^2).
\]
Since \( \gamma \cdot (3L_f + C) = \frac{1}{3\sqrt{T}} \leq \frac{1}{3} \), the LHS coefficient is at least \( \gamma/6 \), so dividing by \( \gamma/6 \) now yields the claim by substituting \( \gamma \) with \( \frac{\gamma}{3(3L_f + C)\sqrt{T}} \).

### A.1 Theoretical analysis when \( K > 2 \)

In this section we sketch how one can generalize the already provided theoretical analysis for the \( K = 2 \) case.
Instead of Machines \(a\) and \(b\), we now suppose that we have a stack \(a_1, \ldots, a_K\) of \(K\) Machines such that every pair \(a_i, a_{i+1}\) communicates a compressed message, denoted by \(m_i\). We simplify the notation of the complete model by \(x = (x^{(1)}, \ldots, x^{(K)})\), and, for a sample \(\xi\), further denote

\[
\begin{align*}
\bar{a}^{(i)}(\xi, x_t) &= a_i(a_{i-1}(\ldots(a_1(\xi, x_t^{(1)}), x_t^{(2)}), \ldots, x_t^{(i)})), \\
\bar{m}^{(i)}(\xi, x_t) &= m_i(m_{i-1}(\ldots(m_1(\xi, x_t^{(1)}), x_t^{(2)}), \ldots, x_t^{(i)})),
\end{align*}
\]

for \(i \in [K]\). As in Algorithm [1] for iteration \(t\) and the data sample \(\xi_t\), if it is the first time that \(\xi_t\) is sampled, Machine \(a_t\) communicates to Machine \(a_{i+1}\) the full precision activations without any compression: \(m_i(\xi_t) = \bar{a}^{(i)}(\xi_t, x_t)\). If \(\xi_t\) has been sampled in previous iterations, Machine \(a_t\) communicates a compressed version:

\[
Q(a_i(\bar{m}^{(i-1)}(\xi_t), x_t^{(i)}) - m_i(\xi_t)),
\]

where \(m_i(\xi_t)\) was the previous message, stored in the local buffer. Both machines then update this local buffer:

\[
m_i(\xi_t) \leftarrow m_i(\xi_t) + Q(a_i(\bar{m}^{(i-1)}(\xi_t), x_t^{(i)}) - m_i(\xi_t)).
\]

Machine \(a_{i+1}\) then uses \(m_i(\xi_t)\) as the forward activations. Upon receiving backward gradients from Machine \(a_{i+2}\), it computes backward gradients, and communicates a quantized version of the backward gradient to Machine \(a_i\). We use

\[
\delta^{(i)}_\xi = \bar{a}^{(i)}(\xi, x_t) - \bar{m}^{(i)}(\xi, x_t)
\]

to denote the message error of \(i\)-th machine in sending the activations (accumulated also through messages in previous pairs), and denote the total message error by \(\delta_\xi = (\delta^{(1)}_\xi, \ldots, \delta^{(K-1)}_\xi)\).

**Update Rules.** We can now generalize the update rule for \(a\) and \(b\) to

\[
\begin{align*}
x^{(K)}_{t+1} &= x^{(K)}_t - \gamma \cdot \nabla x^{(K)}(f \circ a_K)\big|_{(\bar{m}^{(K-1)}(\xi_t), x^{(K)}_t)}, \\
x^{(i)}_{t+1} &= x^{(i)}_t - \gamma \cdot Q(\nabla a_i(f \circ a_K \circ \ldots \circ a_{i+1})\big|_{(\bar{m}^{(i)}(\xi_t), x^{(i+1)}_t), \ldots, x^{(K)}_t}) \cdot \nabla x^{(i)} a_i\big|_{(\bar{m}^{(i-1)}(\xi_t), x^{(i)}_t)},
\end{align*}
\]

for \(i = 1, \ldots, K - 1\), where \(\gamma\) is the learning rate. We can rephrase the update rule as

\[
x_{t+1} = x_t - \gamma \cdot (g_\xi(x_t) + \Delta_\xi(x_t)),
\]

where \(g_\xi(x_t)\) is the stochastic gradient and \(\Delta_\xi(x_t)\) is the total gradient error introduced by communication compression. We have \(\Delta_\xi = (\Delta^{(1)}_\xi + \Delta^{(Q,1)}_\xi, \ldots, \Delta^{(K-1)}_\xi + \Delta^{(Q,K-1)}_\xi, \Delta^{(K)}_\xi)\), given by:

\[
\Delta^{(Q,1)}_\xi(x_t) = Q(\nabla a_i(f \circ a_K \circ \ldots \circ a_{i+1})\big|_{(\bar{m}^{(i)}(\xi_t), x^{(i+1)}_t), \ldots, x^{(K)}_t}) \cdot \nabla x^{(i)} a_i\big|_{(\bar{m}^{(i-1)}(\xi_t), x^{(i)}_t)},
\]

\[
\Delta^{(i)}_\xi(x_t) = \nabla a_i(f \circ a_K \circ \ldots \circ a_{i+1})\big|_{(\bar{m}^{(i)}(\xi_t), x^{(i+1)}_t), \ldots, x^{(K)}_t}) \cdot \nabla x^{(i)} a_i\big|_{(\bar{m}^{(i-1)}(\xi_t), x^{(i)}_t)},
\]

\[
\Delta^{(K)}_\xi(x_t) = \nabla x^{(K)}(f \circ a_K)\big|_{(\bar{m}^{(K-1)}(\xi_t), x^{(K)}_t)} - \nabla x^{(K)}(f \circ a_K)\big|_{(\bar{m}^{(K-1)}(\xi_t), x^{(K)}_t)},
\]

for all \(i = 1, \ldots, K - 1\).

**Generalized Assumptions.** In order to state the analogue of Theorem [3.1] for \(K > 2\), we need to define the corresponding assumptions with respect to Lipschitz properties (whereas Assumption GA2 is here only for completeness, being the same as A2).

- **(GA1: Lipschitz assumptions)** We assume that
  - \(f\) had \(L_f\)-Lipschitz gradient,
  - \(f \circ a_K \circ \ldots \circ a_{i+1}\) has \(L_{f \circ a_K \circ \ldots \circ a_{i+1}}\)-Lipschitz gradient, and has gradient bounded by \(C_{f \circ a_K \circ \ldots \circ a_{i+1}}\) for all \(i = 1, \ldots, K - 1\),
  - \(a_i\) is \(\ell_{a_i}\)-Lipschitz, and has gradient bounded by \(C_{a_i}\), for all \(i = 1, \ldots, K\).
• (GA2: SGD assumptions) We assume that the stochastic gradient $g_\xi$ is unbiased, i.e. $\mathbb{E}_\xi [g_\xi(x)] = \nabla f(x)$, for all $x$, and with bounded variance, i.e. $\mathbb{E}_\xi [\|g_\xi(x) - \nabla f(x)\|^2] \leq \sigma^2$, for all $x$.

We have the following analogue of Theorem 3.1.

**Theorem A.3.** Suppose that Assumptions GA1, GA2 hold, and let

$$
\tilde{C} = \sqrt{\sum_{i=1}^{K-1} C_a^2 C_i^2 f_{\text{oaK} \circ \ldots \circ \text{oa}_{i+1}},}
$$

Consider an unbiased quantization function $Q(x)$ which satisfies that there exists $c_Q < \sqrt{1/2}$ such that $\mathbb{E} \|x - Q(x)\| \leq c_Q \|x\|$, for all $x$. Then there exists a constant $C$ that depends only on the constants defined above and on $\sqrt{K}$ and $N$, such that for the learning rate $\gamma$ proportional to $(C + L_I)^{\gamma}$, after performing $T$ updates one has

$$
\frac{1}{T} \sum_{t \in [T]} \mathbb{E} \|\nabla f(x_t)\|^2 \leq \frac{(C + L_I)(f(x_1) - f^*)}{\sqrt{T}} + \sigma^2 + (c_Q \tilde{C})^2.
$$

(A.2)

Instead of performing a tedious job of rewriting the proof of the $K=2$ case with inherently more constant-chasing, we will simply sketch the differences with respect to the proof of Theorem 3.1.

First we note that, having analogous assumptions as in the case when $K=2$, we can easily prove the following analogue of Lemma A.1.

**Lemma A.4.** Let

$$
\Delta_\xi^{(Q)} = (\Delta_\xi^{(Q,1)}, \ldots, \Delta_\xi^{(Q,K-1)}).
$$

For every sample $\xi$, one has

$$
\|\Delta_\xi^{(Q,i)}(x)\| \leq c_Q C_a C_i f_{\text{oaK} \circ \ldots \circ \text{oa}_{i+1}}, \quad i = 1, \ldots, K - 1,
$$

$$
\|\Delta_\xi^{(i)}(x)\| \leq C_a L_{\text{foaK} \circ \ldots \circ \text{oa}_{i+1}} \|\delta_\xi^{(i)}\| + C_i f_{\text{oaK} \circ \ldots \circ \text{oa}_{i+1}} L_a \|\delta_\xi^{(i-1)}\|, \quad i = 1, \ldots, K - 1,
$$

and

$$
\|\Delta_\xi^{(K)}(x)\| \leq L_{\text{foaK}} \|\delta_\xi^{(K-1)}\|,
$$

implying $\|\Delta_\xi^{(Q)}\| \leq \tilde{C} c_Q$.

Comparing this with Lemma A.1, we see that for $\tilde{\Delta}_\xi = (\Delta_\xi^{(1)}, \ldots, \Delta_\xi^{(K)})$, we now have an additional term that depends on $\delta_\xi^{(i-1)}$. However, since in the proof of Theorem 3.1, we only rely on $\|\Delta_\xi\|^2$, we can proceed even with a crude bound

$$
\|\tilde{\Delta}_\xi\|^2 = \sum_{i=1}^{K} \|\Delta_\xi^{(i)}\|^2
$$

$$
\leq (1 + 2 C_a^2) L_{\text{foaK}} \|\delta_\xi^{(K-1)}\|^2 + 2 \sum_{i=1}^{K-2} (C_a^2 L_{\text{foaK} \circ \ldots \circ \text{oa}_{i+1}}^2 + C_i^2 f_{\text{oaK} \circ \ldots \circ \text{oa}_{i+1}} L_a^2) \|\delta_\xi^{(i)}\|^2
$$

$$
\leq 2K \cdot \max \left\{ \left(1 + 2 C_a^2 \right) L_{\text{foaK}}^2, \ max_{i \in [K-2]} \left\{ C_a^2 L_{\text{foaK} \circ \ldots \circ \text{oa}_{i+1}}^2 + C_i^2 f_{\text{oaK} \circ \ldots \circ \text{oa}_{i+1}}^2 L_a^2 \right\} \right\} \cdot \|\delta_\xi\|^2.
$$

Mimicking closely the steps of Lemma A.2, separately for each $i$ and the summing over all $i$ via $\|\ell_a\|^2 = \sum_{i=1}^{K} \|\ell_{a_i}\|^2$, one can straightforwardly yield the following analogue of Lemma A.2.

**Lemma A.5.** For $C' = K \cdot \frac{36 C_a^2 \|\ell_a\|^2 N^2 C_1}{1 - 2 c_Q^2}$, where $\ell_a = (l_{a_1}, \ldots, l_{a_K})$, one has

$$
\frac{1}{T} \sum_{t \in [T]} \mathbb{E} \|\delta_\xi(x_t)\|^2 \leq C' \gamma^2 \cdot \left( \frac{1}{T} \sum_{t \in [T]} \mathbb{E} \|\nabla f(x_t)\|^2 + \sigma^2 + (c_Q \tilde{C})^2 \right).
$$

Theorem A.3 can now be proved by repeating the steps of the proof of Theorem 3.1, carefully substituting Lemma A.1 by Lemma A.4, and Lemma A.2 by Lemma A.5.
<table>
<thead>
<tr>
<th>Dataset</th>
<th># labels</th>
<th># train samples</th>
<th>Task description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QNLI</td>
<td>2</td>
<td>105K question-paragraph pairs</td>
<td>natural language inference</td>
</tr>
<tr>
<td>CoLA</td>
<td>2</td>
<td>8.6K sentences</td>
<td>linguistic acceptability</td>
</tr>
<tr>
<td>WikiText2</td>
<td>-</td>
<td>2M tokens</td>
<td>language modeling</td>
</tr>
<tr>
<td>arXiv</td>
<td>-</td>
<td>7M tokens</td>
<td>language modeling</td>
</tr>
</tbody>
</table>

Table 4: Dataset statistics

B  Benchmark Dataset

The data statistics can be found in Table 4. We fine-tune DeBERTa on QNLI and CoLA datasets. The QNLI task is to determine whether the context sentence contains the answer to the question. The CoLA task aims to detect whether a given sequence of tokens is a grammatical English sentence. In addition, we fine-tune GPT2 on the WikiText2 training set. We also collect 30K arXiv abstracts in 2021 to fine-tune GPT2. Neither corpus is included in the GPT2 pre-training data set.

C  Training Task Setup and Hyper-parameter Tuning

Sequence Classification. We use the AdamW optimizer to fine-tune the model for 10 epochs. Specifically, for QNLI, we set the learning rate to 3.0e-6, learning rate warm-up steps to 1000, max sequence length to 256, the macro-batch size to 64 and micro-batch size to 8; for CoLA, we set the learning rate to 2.5e-6, learning rate warm-up steps to 250, max sequence length to 128, the macro-batch size to 32 and micro-batch size to 8. After the learning rate warm-up stage, we decay the learning rate linearly over the training epochs.

Language Modeling. For both WikiText and arXiv datasets, we use AdamW optimizer with a learning rate of 5.0e-6. We train the model for 10 epochs with a macro-batch size of 32 and a micro-batch size of 1. The max sequence length is set to 1024 for both datasets. After the learning rate warm-up stage, we decay the learning rate linearly over the training epochs.

D  Distributed View of AQ-SGD Algorithm

Algorithm 2 shows a multi-node view of AQ-SGD. For brevity, we omit the first warm-up epoch, where we conduct uncompressed training, and thus we update the previous messages by $m(\xi) \leftarrow a(\xi, x)$.

Algorithm 2 AQ-SGD Algorithm

initialize: $x_0$, learning rate $\gamma$, network $a$’s weights $x^{(a)}$, network $b$’s weights $x^{(b)}$, quantization function $Q$, the arrays of previous messages $m$, where networks $a$ and $b$ each maintain a copy of it.

for $t = 1, \ldots, T$ do

(on network $a$)

Randomly sample $\xi_t$

$\Delta m(\xi_t) \leftarrow Q(a(\xi_t, x_t^{(a)}) - m(\xi_t))$

Update $m(\xi_t) \leftarrow m(\xi_t) + \Delta m(\xi_t)$

Send $\Delta m(\xi_t)$ to network $b$

(on network $b$)

Update $m(\xi_t) \leftarrow m(\xi_t) + \Delta m(\xi_t)$

Update $x_{t+1}^{(b)} \leftarrow x_t^{(b)} + \gamma \cdot \nabla x^{(b)}(f \circ b) |_m$

Send $Q(\nabla a(f \circ b) |_m)$ to network $a$

(on network $a$)

Update $x_{t+1}^{(a)} \leftarrow x_t^{(a)} - \gamma \cdot Q(\nabla a(f \circ b) |_m) \cdot \nabla x^{(a)} a$

end for

Output: $x = (x_T^{(a)}, x_T^{(b)})$
E Decentralized Training over Slow Network

Decentralized training for large foundation models recently attracted intensive interests. Example projects include Learning@home [1], DeDLOC [2], and Training Transformers Together [3]. The goal of these projects is to enable a decentralized open-volunteering paradigm for foundation model training. As many geo-distributed users contribute their GPUs, these GPUs are often connected via slow networks. For example, [2] investigates a heterogeneous with bandwidths of 200Mbps, 100Mbps, and 50Mbps; [3] advocates to train Transformer over the Internet with 10-100Mbps bandwidth; [4] considers network bandwidth less than 400Mbps.

In this setting, communication compression is key to performance. However, when compressing activations, existing methods rely on direct quantization. This inspired our paper, which provides the first activation quantization method with rigorous theoretical guarantee and outperforms direct quantization.

F Discussion on Tensor Parallelism

We here discuss tensor parallelism [24], and the potential adaptation of our algorithm to it. In tensor parallelism, the activations are computed across different machines, and need to be aggregated. Therefore we need to compress activations both before and after allreduce to support tensor parallelism. Specifically, suppose \( N \) machines conduct tensor parallelism, then the output activation is:

\[
A = A_1 + A_2 + ... + A_N, \quad \text{(F.1)}
\]

and we need to compress communication twice:

\[
AQ = Q(Q(A_1) + Q(A_2) + ... + Q(A_N)). \quad \text{(F.2)}
\]

We believe that delta compensation could be applied to all \( Q(-) \), similar to how previous work handles gradient compression (e.g. Eq. 3 and 4 in [27]). However, this requires careful further studies, both empirically and theoretically. We leave activation compression for tensor parallelism as future work.

G Limitation and Potential Future Direction

**Additional Storage.** Our algorithm trades storage for communication. Fortunately, we find this is a reasonable trade-off in our settings. In the following we show that we can offload the activations to SSD and hide it within the GPU computation of other data examples.

We compare the throughput under the bandwidth of 10Gb/s. FP32 achieves a throughput of 3.8 seqs/s, while AQ-SGD, either offloading activations to host memory or SSDs, achieves 4.0 seqs/s. Considering the similar training throughput of the above three settings, we show that the overhead of offloading to SSD can be successfully hided in GPU computation. In particular, our largest experiment in this paper requires 172GB storage per machine, which even with a 10x larger dataset can be easily offloaded to SSDs. For much larger datasets (e.g. 100x), we can use data parallelism to reduce the storage requirement for each machine. For an even larger dataset, our algorithm might not support it well. This then requires further studies.

**Online Learning.** Our proposal relies on iterating over multiple epochs, which is a common setting. We understand our current algorithm has limitations in the settings such as online learning. In the following, we provide a potential solution (to both storage requirement and limitation in online learning) – relaxing AQ-SGD by clustering activations and storing only the centers of the clusters.

Recall that AQ-SGD compresses the “delta” (difference between activations from two epochs) and thereby needs to store activations from the previous epoch. In the relaxed AQ-SGD, we could use algorithms like clustering or locality sensitive hashing to partition the activations and then we only store the “centers” of each partition/cluster. When computing the “delta”, we can first identify which partition/cluster the current activation belongs to and retrieve the corresponding “center”. Then “delta” = activation - “center”. This will potentially help address storage and online learning limitations. We will explore this in future work.
H Additional Results

We provide additional experimental results. Specifically, we show:

- the convergence results with standard deviation;
- the training throughput for different dataset settings;
- the numerical stability of training from scratch;
- the training results under FP16 precision;
- the robustness of AQ-SGD under different hyperparameter settings;
- the effectiveness of AQ-SGD in the split learning scenario.

### H.1 Convergence Results with Standard Deviation

In the main content, we show the convergence performance of different approaches. We repeated each experiment three times to ensure reproducibility. We calculate the moving averages of these convergence curves and then average the results of repeated experiments. We visualize (shaded areas) the moving standard deviation in all repeated experiments in Figure 6. Overall, we observe consistent results for all datasets.

### H.2 Throughput under Different Dataset Settings

We show the training throughput under different dataset settings in Figure 5. In general, the observation is similar to that of the main content: our approach maintains similar throughput even when the network is 100× slower (from 10Gbps to 100Mbps). WikiText2 and arXiv have essentially the same throughput results, since we use the same training settings for them.
H.3 Results of Training from Scratch

We investigate the numerical stability of AQ-SGD by showing the convergence result of training from scratch, where the model parameters are randomly initialized. We train WikiText and arXiv datasets for 20 epochs and use the first 10% of steps as warm-up, respectively. As shown in Figure H.3, we can see that AQ-SGD converges almost as fast as FP32 when training from scratch, which indicates our approach is robust enough even when the model is far from the converged state. In contrast, the curve of DirectQ becomes flatter in the late training stage, showing a clear gap with FP32.

H.4 Results of Training under FP16

To investigate the convergence performance of AQ-SGD under low-precision training, we here show the results of FP16 training, and compare it with FP32 training. Figure 8 compares the results under FP32 and FP16. In general, the convergence curves are consistent with the FP32 case. This confirms the effectiveness of AQ-SGD when the activation is already in low precision.

H.5 Hyper-parameter Sensitivity

Here we demonstrate the robustness of our method in various settings. For fast validation, we focus on evaluating DeBERTa-v3-base on QNL and CoLA datasets. We by default use $K = 4$ devices for pipeline parallel training, 2 bits for forward activation, and 4 bits for backward gradients ($fw2\ bw4$).

Number of Pipeline Stages. We first investigate the influence of the number of pipeline stages on convergence performance. Intuitively, partitioning into more pipeline stages leads to more rounds of data compression and communication, resulting in a larger accumulated compression error. The results of Figures 9a and 9b confirm this intuition. Specifically, the direct quantization method works not bad when $K = 2$, but its performance becomes unsatisfied when we further enlarge $K$. In comparison, our approach can maintain similar convergence performance to FP32.

Number of Bits in Communication. Figures 9c and 9d compare different methods with different numbers of bits in communication. We observe that using more bits can improve the convergence performance but lead to higher communication overheads. In general, our approach achieves better accuracy-efficiency trade-offs.

---

6 https://huggingface.co/microsoft/deberta-v3-base
**Number of Bits for Previous Messages.** We may find that storing all previous messages is space-intensive. To reduce such requirements, we show that previous messages \( m \) can be preserved with low precision. We here perform quantization on \( m \), where \( z \) means that we use \( z \) bits for previous messages. Figures 9e and 9f show the results with different number of bits of the previous messages. When only 2 bits are used for the previous messages, despite the fact that it is slightly worse than our default setting, our approach is still significantly better than DirectQ. And there is no significant performance drop when 8 bits are used for the previous messages.

**Pre-trained Model Sizes.** Figures 9g and 9h show the results of the base and large version of DeBERTa. Surprisingly, larger models seem to be more tolerant of errors from activation compression than smaller models. One possible reason is that larger models usually use much smaller learning rates. So the error of each iteration can be restricted to a smaller range. Here, we use 2.0e-5 for the base model and 7e-6 for the large model, as suggested in the official repository of DeBERTa.

### H.6 Split Learning

Split learning is a scenario of federated learning, where the client trains a shallow part of deep network (known as the cut layer) that accesses the training data, while the rest of the model is trained in a data center. Clients and server need to exchange the activation and its gradients in the cut, where AQ-SGD can be adopted. We evaluated AQ-SGD on a split learning scenario where neither the input data nor its labels are shared with the server—the model is cut twice, one after the first resnet block and one before the last block to generate the prediction. We evaluate AQ-SGD for split learning over Cifar10 and Cifar100 with the ResNet34 model. We set 16 clients and use a Dirichlet distribution with concentration parameter 0.5 to synthesize non-identical datasets. Following the previously established work of split learning, in each communication round, we conduct local training for each client sequentially, and each client will train 3 epochs with its local data. We utilize SGD optimizer with momentum of 0.9, a batch size of 64, and a learning rate of 0.01. We decay the learning rate to its 10% for every 20 communication rounds.

The datasets are augmented with random cropping and flipping. To adapt to random cropping, we do the same cropping operation on the retrieved previous message, and then update its non-cropped part. To adapt to random flipping, we maintain another previous message copy for flipped images, and retrieve and update only the corresponding copy during training.

Figure 10 presents the results of split learning, where \( fw2 \ bx8 [0.2] \) means that, for forward pass, we perform 2-bit quantization, and for backward pass, we keep only the top 20% gradients and then perform 8-bit quantization. We can see that AQ-SGD transfers the activations in 2 bits while maintaining a performance similar to FP32, which indicates the effectiveness of AQ-SGD in improving the communication efficiency in the split learning scenario. Furthermore, compared to DirectQ, AQ-SGD shows advantages in terms of both the convergence and generalization of the trained model.
I  Case Study of Generation Results

We here conduct case study to better understand the generation quality across different methods. All methods are fine-tuned on WikiText with the same seed. We use greedy decoding to generate texts. For DirectQ and AQ-SGD, we use 4-bit in forward pass and 8-bit in backward pass.

We randomly pick documents from the test set of WikiText to construct the prompts and let the model complete the rest. The generation results after fine-tuning are presented in Tables 6 and 7. For instance, in Cases 1 and 2, AQ-SGD produces the same results as FP32; in Case 3, although AQ-SGD does not generate the exact same output as FP32, it still expresses a similar meaning. This indicates our proposed compression scheme has relatively little impact on the learning process. In contrast, DirectQ may generate text that is not relevant to the context. For example, in Case 1, we know that “Matsuo Bashō was strongly influenced by Du Fu”, so we should expect the text below to show the fact how he was influenced, just like FP32 and AQ-SGD did. But DirectQ fails to generate reasonable text based on the context. Other inference results presented in Tables 6 and 7 lead to the same observation.
Table 6: Case study of generation results.

<table>
<thead>
<tr>
<th>Case 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prompt:</strong> Matsuo Bashō, the greatest haiku poet, was also strongly influenced by Du Fu; in <em>Oku no Hosomichi</em>, his masterpiece, he cites the first two lines of <em>A Spring View</em> before a haiku as its introduction and also many of his other haiku have similar wording and themes. It is said that when he died in Osaka during a long travel, a copy of Du Fu’s poetry was found with him as one of a few precious items which he was able to carry around. Gold: his body was placed in a coffin with a Du Fu-like inscription. FP32: his body was placed in a coffin with a Du Fu-like inscription. AQ-SGD: his body was placed in a coffin with a Du Fu-like inscription. DirectQ: his body was covered with a red cloth and his head was shaved.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prompt:</strong> Peppiatt notes that the exhibition showed Bacon no longer needed sensationalist material to make an impact, and was now capable of creating an intense emotional response through more subtle means... Gold: capable of creating an intense emotional response through more subtle means... FP32: able to &quot;make a serious contribution to the art of painting&quot; AQ-SGD: able to &quot;make a serious contribution to the art of painting&quot; DirectQ: able to &quot;make a serious statement about the state of the art&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prompt:</strong> At this point, Dokhturov’s column appeared behind the French line and joined the battle. The French were outnumbered more than three to one, assaulted in the front by Miloradovich’s column, in the middle by Strik’s and in the rear by Dokhturov. Gold: one, assaulted in the front by Miloradovich’s column, in the middle by Strik’s and in the rear by Dokhturov. FP32: one, but they held their ground and forced the Russians to retreat. AQ-SGD: one, but they were able to hold their ground. DirectQ: one, and the Russians were able to hold their ground.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prompt:</strong> Between Jupiter and Fort Pierce, the storm knocked down power and telegraph lines. In the latter city, high waves washed out a portion of the causeway. In the 1980s, an elderly resident recalled that the storm was the worst storm he had ever seen. Gold: worst storm he had ever seen. FP32: worst storm he had ever seen. AQ-SGD: worst storm he had ever seen. DirectQ: worst he had ever experienced.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prompt:</strong> Lesnar was born in Webster, South Dakota, on July 12, 1977. He was raised on a Webster dairy farm owned by his parents, Stephanie and Richard Lesnar. He is of German descent. He has two older brothers named Troy and Chad, and a younger sister named Brandi. At age 17, he joined the National Guard, where he was assigned to an office job after his red-green colorblindness was deemed hazardous to his desire to work with explosives. Gold: served in the South Dakota National Guard. He was a member of the South Dakota National Guard for two years. FP32: served in the South Dakota National Guard. He was a member of the South Dakota National Guard for two years. AQ-SGD: served in the South Dakota National Guard. He was a member of the South Dakota National Guard for two years. DirectQ: served in the South Dakota National Guard for two years. He was discharged in May 1991.</td>
</tr>
</tbody>
</table>
### Table 7: Case study of generation results (more).

<table>
<thead>
<tr>
<th>Case</th>
<th>Prompt</th>
<th>Gold</th>
<th>FP32</th>
<th>AQ-SGD</th>
<th>DirectQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>... Mega placed the game at #7 in their &quot;Top Mega Drive Games of All Time&quot; list. MegaTech magazine praised the game’s graphics, but criticized the game’s easy difficulty level.</td>
<td>explaining simply that &quot;the graphics are excellent, but the game is easy to complete.&quot;</td>
<td>stating that &quot;the game is too easy to be enjoyable&quot;.</td>
<td>stating that it was &quot;too easy&quot;.</td>
<td>which was criticized for being too easy.</td>
</tr>
<tr>
<td>7</td>
<td>... &quot;Obviously, he has great rapport and communication ability with his team.&quot; Yahoo! Sports compared Stevens to legendary coach John Wooden writing &quot;</td>
<td>Brad Stevens is winning at Butler the Wooden way – calm and composed on the sideline.&quot;</td>
<td>He’s a coach who has been around the block, and he’s not afraid to tell you what he thinks.&quot;</td>
<td>Stevens is a coach who has been around the game for a long time, and he has a knack for getting the most out of his players.&quot;</td>
<td>Stevens is a coach who has been able to mold a team into a championship contender. He has a knack for getting the most out of his players, and he has a knack for getting the most out of his players’ personalities.&quot;</td>
</tr>
<tr>
<td>8</td>
<td>... The Gore’s social circle in England revolved around his good friend Rufus King, who was appointed Ambassador to Great Britain in 1796, along with other Massachusetts expatriates.</td>
<td>was a member of the Royal Society.</td>
<td>was a member of the Royal Society.</td>
<td>was a member of the Royal Society.</td>
<td>was a member of the Royal Society.</td>
</tr>
<tr>
<td>9</td>
<td>... The city has 896 barangays that are known by sequential numbers instead of names. These barangays are further group into</td>
<td>100 zones for administrative and municipal purposes.</td>
<td>districts, which are further subdivided into wards.</td>
<td>districts, which are numbered according to the number of barangays in the district.</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>... The National Spiritual Assembly of the Bahá’ís of the Philippines, the governing body of the Filipino Bahá’í community, is headquartered near Manila’s eastern border with Makati.</td>
<td>headquartered near Manila’s eastern border with Makati.</td>
<td>headquartered in the city.</td>
<td>headquartered in the city.</td>
<td>alphabetical order.</td>
</tr>
</tbody>
</table>