NetSenseML: Network-Adaptive Compression for Efficient Distributed Machine Learning

Yisu Wang, Xinjiao Li, Ruilong Wu, Huangxun Chen, and Dirk Kutscher[✉]

The Hong Kong University of Science and Technology (Guangzhou),
Guangzhou, China
{ywang418,xli886,rwu408}@connect.hkust-gz.edu.cn,
{huangxunchen,dku}@hkust-gz.edu.cn

Abstract. Training large-scale distributed machine learning models imposes considerable demands on network infrastructure, often resulting in sudden traffic spikes that lead to congestion, increased latency, and reduced throughput, which would ultimately affect convergence times and overall training performance. While gradient compression techniques are commonly employed to alleviate network load, they frequently compromise model accuracy due to the loss of gradient information. This paper introduces NetSenseML, a novel network adaptive distributed deep learning framework that dynamically adjusts quantization, pruning, and compression strategies in response to real-time network conditions. By actively monitoring network conditions, NetSenseML applies gradient compression only when network congestion negatively impacts convergence speed, thus effectively balancing data payload reduction and model accuracy preservation.

Our approach ensures efficient resource usage by adapting reduction techniques based on current network conditions, leading to shorter convergence times and improved training efficiency. We present the design of the NetSenseML adaptive data reduction function and experimental evaluations show that NetSenseML can improve training throughput by a factor of 1.55 to 9.84× compared to state-of-the-art compression-enabled systems for representative DDL training jobs in bandwidth-constrained conditions.

Keywords: Distributed Systems · Systems for Machine Learning · DNN Training · Gradient Compression

1 Introduction

Training large-scale distributed machine learning models, such as large language models [5,22], imposes considerable demands on the network infrastructure, often resulting in sudden traffic spikes that lead to congestion, increased latency, and reduced throughput, ultimately affecting convergence times and overall training performance.

As distributed machine learning is becoming more widely used, deployments are moving away from proprietary network technologies and isolated workloads toward shared infrastructure and multi-tenant deployments [6]. Distributed Data-Parallel (DDP) training is currently the dominant paradigm for large-scale distributed machine learning [17]. In DDP, gradient communication poses a significant challenge, particularly due to its bursty nature: Periods of computation are followed by sudden bursts of transmission and exchange of the gradient tensor. In such shared environments, unmitigated overload can result in substantial communication volume, potential overload, and packet loss, which can significantly impact the overall training process. Applying traditional congestion control algorithms, such as CUBIC [9] and BBR [7] in protocols such as TCP [3] and QUIC [14], can ensure reliable transmission without overload by adapting sending rates and thus throughput, but it would also decrease training throughput and thus increase convergence times, since machine learning application layers can only adapt their behavior indirectly, as a response to observed transport layer throughput, if at all.

Gradient compression has emerged as a promising solution for mitigating this communication bottleneck by reducing the volume of gradient data exchanged [12]. Techniques such as quantization [16], pruning [10] and sparsification [18] compress gradients to reduce the amount of data transmitted. However, existing gradient compression schemes often fail to accelerate the training process without compromising model accuracy, thereby introducing a trade-off between communication efficiency and training performance.

To address these challenges, we introduce NetSenseML, an adaptive gradient compression framework for distributed machine learning that dynamically monitors network conditions and actively responds to congestion by adjusting compression ratios. NetSenseML aims to reduce Time to Accuracy (TTA), defined as the time required to reach a target model accuracy, effectively balancing transmission efficiency, convergence speed, and overall training performance through techniques such as quantization, pruning [11,15] and sparsification.

In the network sensing phase, NetSenseML begins by carefully monitoring gradient transmission times and estimating the available bandwidth to gauge the current state of the network, thereby adjusting the gradient compression ratio in real time based on these assessments. This adaptive approach allows NetSenseML to tailor gradient compression, in the adaptive compression phase, by appropriately applying quantization, pruning, and sparsification, aligned with the measured compression ratio and density of the gradients. This strategy ensures the optimal adjustment of the compressed gradient size to fit the prevailing network conditions and maximize bandwidth efficiency.

Our experiments demonstrate that in constrained bandwidth environments with multiple workers, NetSenseML significantly enhances the efficiency of data transmission and model training performance. Furthermore, the algorithm maintains robust data transmission and model training speeds, in multitask scenarios and under challenging bandwidth conditions, effectively balancing network utilization and training efficiency.

The main contributions of our research are as follows.

1. To the best of our knowledge, NetSenseML is among the first systems to use transmission times and bandwidth estimates to detect network conditions, thereby driving the dynamic tuning of gradient compression ratios based on perceived network conditions, improving network utilization and training efficiency.
2. NetSenseML ensures the retention of essential gradient information, even in extreme network scenarios by adjusting compression strategies based on the compression ratio and gradient density characteristics, thus minimizing the impacts of compression on model accuracy.
3. Rigorous testing in environments with limited bandwidth and diverse tasks shows that NetSenseML can guarantee efficient gradient transmission, stable and fast model convergence, and maintain high model accuracy, effectively minimizing the trade-offs between key performance metrics.

2   Scenarios for Network-Adaptive DL Training

NetSenseML is designed for use in arbitrary network scenarios within Wide Area Network (WAN) environments. Training a deep neural network (DNN) involves three steps: forward propagation (FP), backward propagation (BP), and the parameter update. Distributed training addresses GPU memory constraints by dividing data across multiple GPUs, each holding a complete model copy. However, this approach requires frequent gradient synchronization across GPUs, leading to intensive communication. As models grow larger, requiring partitioning across GPUs, communication demands increase further. In such cases, the network bandwidth becomes a critical bottleneck.

We consider two WAN training scenarios:

Scenario 1: Multi-cluster communication through a wide area network. When computational resources are insufficient, training tasks often use wide area network connections across data centers to accelerate the process and improve resource efficiency. A typical example of this model is CloudLab [8], which consists of multiple clusters distributed across various locations, providing researchers with full control over computing, networking, and storage resources.

Scenario 2: Offloading to Public Cloud Platforms. Multi-clusters do not always guarantee sufficient resources. For example, CloudLab [8] offers limited allocations per user, often requiring researchers to wait in a queue based on the usage of others. In such cases, public cloud resources serve as ideal complements to local resources. Although public cloud services are often costly, the combined use of lab resources and cloud services proves to be significantly more economical than relying solely on public cloud rentals.

A key challenge in these environments is that the interconnection bandwidth over WANs is significantly lower than the bandwidth available within laboratory clusters or between nodes in public cloud environments. As a result, during the
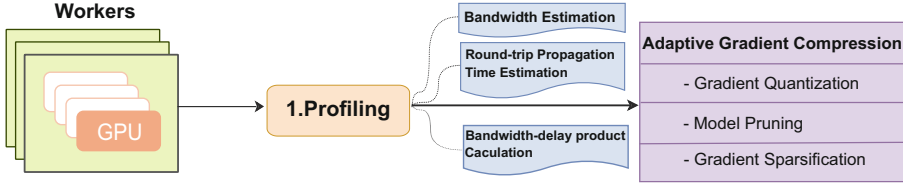
Fig. 1. NetSenseML Overview

BP process, the computed gradients must remain in the GPU/CPU memory of the nodes, waiting until the buffered data are successfully transmitted. This delay can substantially increase the overall training time and reduce GPU utilization.

3   Related Work

Modern distributed machine learning frameworks, such as PyTorch DDP [4], generate a substantial volume of data packets that accumulate in communication queues, slowing down iteration speed. To address this challenge, DC2 [1] leverages a delay monitoring component to track communication latency for each training iteration, enabling dynamic adjustment of gradient compression. Crux [6] infers network congestion by analyzing GPU utilization, allowing it to optimize the data transmission volume. Espresso [24] emphasizes the coordination between different hardware components and communication links during the compression process, aiming to optimize communication time.

Despite existing work that estimates network conditions based on latency, GPU utilization, and bandwidth, there is a lack of effective estimation of actual network capacity.

4   NetSenseML: Network-Aware Gradient Compression for Efficient Distributed Machine Learning

Our goal is to transmit as much gradient information as possible under limited bandwidth, unpredictable network connectivity, and constantly changing network conditions, thereby maintaining model accuracy and convergence speed. As shown in Fig. 1, the architecture includes a profiling module, which continuously monitors network status through bandwidth estimation, round-trip propagation time estimation, and bandwidth-delay product calculation, and an adaptive gradient compression module that dynamically adjusts gradient quantization and model pruning to select the optimal feasible data rate.

4.1   Network Status Sensing and Adaptive Compression Ratio Adjustment

We accurately modeled the available network capacity prior to each gradient aggregation. To do this, we monitor the bottleneck bandwidth (`BtlBw`) and the

---

Algorithm 1: Network Status Sensing and Adaptive Compression Ratio Adjustment

---

Data: Transmission intervals, Data size, RTprop, Compression ratio
**Result**: $ratio$: Ratio used for gradient compression

1 Step 1: start-up:
2   Set ratio $\leftarrow 0.01$;
3   **foreach** *step in start-up phase* **do**
4   Quickly increase the compression ratio ;
5   ratio $\leftarrow \min(1, \text{ratio} + \beta_1)$;
6 Step 2: NetSense:
7   **foreach** *gradient transmission interval i* **do**
8   Measure EBB and RTT for interval $i - 1$;
9 $\text{EBB}_{i-1} \leftarrow \frac{\text{data\_size}_{i-1}}{\text{RTT}_{i-1}}$;
10   Update BtlBw and RTprop;
11 $\text{BtlBw} \leftarrow \text{EBB}_{Max}$;
12 $\text{RTprop} \leftarrow \text{RTT}_{Min}$;
13   Measure BDP;
14 $\text{BDP} \leftarrow \text{BtlBw} \times \text{RTprop}$;
15   Adjust compression ratio ;
16   **if** *data_size $> 0.9 \times BDP$* **then**
17   ratio $\leftarrow \max(0.005, \text{ratio} \times \alpha)$;
18   **else**
19   ratio $\leftarrow \min(1, \text{ratio} + \beta_2)$

---

round-trip propagation time (`RTprop`) during each gradient transmission interval to sense and analyze the state of the network as shown in Fig. 2, and adjust the compression ratio based on the bandwidth delay product (`BDP`). When the transmitted data size is less than the `BDP`, the round-trip time (`RTT`) remains relatively constant at its minimum value, i.e., `RTprop`.

In this phase, the network bandwidth is underutilized. However, as the size of the transmitted data exceeds the `BDP`, the `RTT` increases significantly, indicating a state of network congestion where the data sent surpass the network's transmission capacity, i.e., `BtlBw`. Our goal is to adjust the compression ratio so that the size of the compressed gradients approximates the `BDP` as closely as possible. This approach maximizes bandwidth utilization, enabling higher data throughput, which in turn enhances model accuracy and accelerates convergence speed.
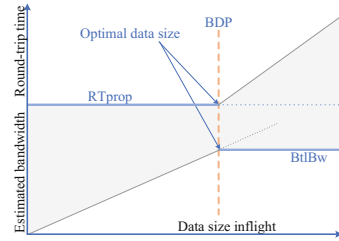
Fig. 2. Network status sensing based on `BtlBw` and `RTprop`.

In BBR [7], the packet sending rate during the startup phase increases rapidly until packet loss or excessive `RTT` occurs, at which point BBR detects the network's capacity limit. In NetSenseML, we initially set the gradient compression ratio to a lower value and rapidly increase this ratio during the first few steps of training until excessive `RTT` is detected.

Existing `RTT`-dependent congestion control mechanisms such as MLT [23] employ reactive strategies that activate only after substantial queue accumulation, typically when `RTT` exceeds twice the propagation delay. This delayed response allows severe congestion to develop, risking packet loss that critically degrades distributed training performance.

NetSenseML introduces proactive congestion prevention through real-time bandwidth-delay product monitoring. As `BDP` defines the network's maximum in-flight data capacity, approaching gradient volumes trigger immediate compression before measurable RTT increases. By maintaining transmitted data below the BDP threshold during parameter synchronization phases, our method eliminates queue buildup at intermediate nodes, sustaining optimal throughput and accelerating model convergence.

Our algorithm is comprised of two specific steps, as shown in Algorithm 1:

Step 1. During the initial start-up phase, we set the compression ratio to 0.01. This ratio is then progressively increased over a limited number of stepsuntil the detection of packet loss or excessive `RTT`.

Step 2. In the *NetSense* phase, we continuously monitor and evaluate the estimated bandwidth (`EBB`) and the `RTT` for each interval. These metrics are crucial for updating the values of `BtlBw` and `RTprop`. The compression ratio is subsequently adjusted based on the `BDP`, which is calculated as the product of `BtlBw` and `RTprop` according Eq. (1) to (3). We determine the `EBB` of the last interval by analyzing the size of the transmitted data and its corresponding `RTT`, using the maximum observed value as the network's `BtlBw`, $\text{BtlBw} = \max(\text{EBB})$, and the minimum recorded `RTT` as the `RTprop`, $\text{RTprop} = \min(\text{RTT})$, under conditions free from network congestion. For data size data_size_i and transmission time RTT_i of interval i, the `EBB` is calculated by Eq. (1),

$$\text{EBB}_i(\text{data\_size}_i, \text{RTT}_i) = \frac{\text{data\_size}_i}{\text{RTT}_i}. \tag{1}$$

After updating the value of `BtlBw` and `RTprop`, `BDP` is then estimated using Eq. (2), based on RTprop and BtlBw prior to each data transmission,

$$\text{BDP} = \text{BtlBw} \times \text{RTprop}. \tag{2}$$

Consequently, the compression ratio is adjusted in accordance with `BDP`, as defined in Eq. (3),

$$\text{ratio} = \begin{cases} \max(0.005, \text{ratio} \times \alpha), & \text{data\_size} > 0.9 \times \text{BDP} \\ \min(1, \text{ratio} + \beta_2), & \text{data\_size} \leq 0.9 \times \text{BDP} \end{cases} \tag{3}$$

In our experiment, α is 0.5, β_2 is 0.01.

Algorithm 2: NetSenseCompression: quantization, pruning, and sparsification

---

Data: Pre-trained model: The initial neural network model,
grads: Gradients for compression,
ratio: Ratio used for gradient compression,
tr_q: Quantization check threshold,
tr_d: Gradient density threshold,
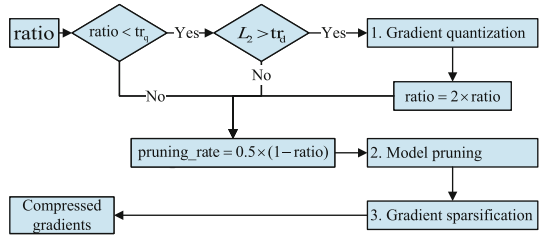**Result**: *Com_grads*: Compressed gradients

1 Step 1: Adaptive Quantization:
2   **if** $ratio < tr_q$ **then**
3 $L_2 \leftarrow$ calculate_L2_norm(*grads*);
4   **if** $L_2 > tr_d$ **then**
5 $grads \leftarrow$ quantize(*grads*);
6   ratio $\leftarrow 2 \times$ ratio;

7 Step 2: Model Pruning:
8   Get pruning rate;
9   pruning_rate $\leftarrow 0.5 \times (1 - \text{ratio})$;
10   pruned_grads $\leftarrow$ prune_weights(pruning_rate);
11   Set pruned weights' gradients to zero;
12   *grads*[pruned_grads] $\leftarrow 0$;
13 Step 3: Gradient Sparsification:
14   *Com_grads* $\leftarrow$ top_k(*grads*, ratio);
15   Transmit the sparsified gradient over the network;

4.2   Adaptive Compression Based on Quantization, Pruning, and Sparsification

To enhance the precision and accelerate the convergence of the model, we have developed an adaptive compression method that incorporates quantization, pruning, and sparsification, as shown in Algorithm 2. This method optimizes gradient transmission under challenging network conditions

Fig. 3. Adaptive quantization based on L_2 norm.

through quantization, boosts model generalizability via pruning, and maintains critical gradient information through sparsification. Our method consists of the following steps:

Step 1. Adaptive quantization. Adaptive quantization is initiated when the compression ratio falls below a pre-defined threshold, tr_q, as shown in Fig. 3. The gradient's L_2 norm is computed as follows:

- If $L_2 >$ tr$_d$, indicating substantial informational content, quantization is applied to preserve non-zero gradient transmission. This is essential for maintaining the gradient's integrity under stringent network conditions.
- To accommodate the halved data size resulting from the reduction of gradient representation from 32-bit to 16-bit floating points, the compression ratio is adjusted to ratio = 2 × ratio. This efficient compression aids in optimizing bandwidth use while ensuring data integrity and model performance.

Step 2: Model Pruning. The pruning rate is set using ratio$_p = 0.5 \times (1 -$ratio$)$, which targets parameters with smaller weights for pruning:

This process involves setting the gradients of the selected parameters to zero, which not only reduces the load during network transmission by eliminating these gradients from the next sparsification step but also minimizes the disproportionate influence of small weights with large gradients on model training. Such selective pruning is crucial for enhancing the model's generalizability and overall performance.

The parameters affected by pruning are not permanently removed but are only excluded from gradient transmission. This allows these pruned parameters the potential to be gradually reactivated in subsequent training iterations based on new data, thereby maintaining the adaptability and depth of the model's learning capability.

Step 3. Gradient sparsification. We employ the compression ratio as the basis for the sparsification ratio, performing TopK sparsification [2] to eliminate gradients with minimal absolute values, and accumulate the local filtered gradients for further aggregation and transmission. This process is calibrated to ensure that the decision to filter gradients is influenced equally by the model's pruning strategy and the inherent magnitude of the gradients.

This structured approach not only streamlines the handling of data in constrained network environments but also strategically manipulates the model parameters to promote faster convergence and enhance the general robustness of the model.

5   Implementation and Evalution

5.1   Setup and Workloads

We built our testbed using the ESXi virtualization platform with a server equipped with 8 A40 GPUs and an Intel Xeon Platinum 8358P processor with 64 CPU cores.

We developed the NetSenseML prototype on top of the PyTorch distributed framework [4], implementing communication using NCCL [19] over TCP for gradient aggregation. In our implementation, we utilized PyTorch's Distributed Data Parallel communication hook to override the default `allreduce` operation, allowing fine-grained control over how gradients are communicated across workers.

Workloads: We evaluated our NetSenseML prototype by training popular computer vision models on an eight-worker testbed. The models and datasets used in our evaluation included ResNet18 [13] and VGG16 [20], both trained on the CIFAR-100 dataset. We set the per-GPU batch size to 32.

Metrics: We define time-to-accuracy (TTA) as the training time required to reach a target validation accuracy. Additionally, we present the training throughput (measured as images per second, referred to as samples per second) across all training tasks under different scenarios and varying bandwidth conditions. We

Fig. 4. Evaluation topology.

define convergence time as the time required for the model's accuracy to stabilize at a target threshold, indicating the model has fully converged.
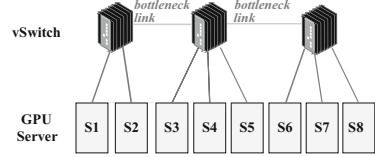
5.2 Evaluation Scenarios

We conducted experiments under three different scenarios using the topology shown in Fig. 4, creating network performance bottlenecks by adjusting the link bandwidth of two connections to the switch.

Scenario 1: Bandwidth constrained but stable network. We ran several tests using different static bottleneck bandwidths, ranging from 200 Mbps to 10 Gbps, by adjusting the link bandwidth between the switch and other nodes. The bandwidth selection was based on the proportion of the maximum bandwidth required by a training model to avoid latency. These tests, which measured time-to-accuracy, convergence time, and training throughput, showed that NetSenseML can fully utilize the available bandwidth (when TopK and AllReduce cannot), leading to faster convergence.

Scenario 2: Degrading Network Conditions. We gradually reduced all communication bandwidths during the experiment to demonstrate the adaptability of our approach. Training throughput was compared to show that as network conditions deteriorate, NetSenseML can achieve higher throughput by reducing gradient transmission, whereas TopK and AllReduce maintain a constant data volume, leading to network congestion and reduced training throughput.

Scenario 3: Fluctuating Bandwidth with Competing Network Traffic. We simulated a cloud service environment where multiple virtual machines share the same physical host's network resources. Specifically, we ran multiple iperf3 [21] processes in parallel between nodes, periodically sending traffic to test the available bandwidth of the current network links. This generated network interference that competed for bandwidth with the training task. By dynamically adjusting the link bandwidth between switches and nodes, we evaluated the robustness of our approach.
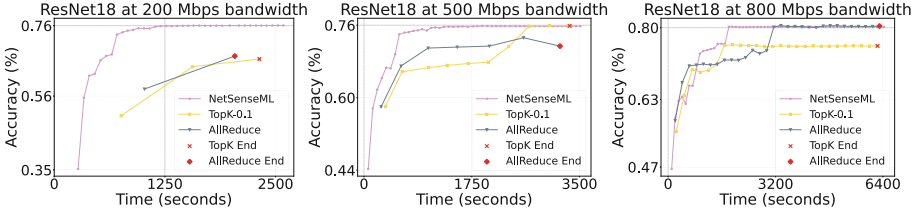
Fig. 5. Time to accuracy (TTA) comparison under different bottleneck bandwidth conditions with ResNet18

5.3   End-to-End Performance

For assessing end-to-end performance, we measured 1) *time-to-accuracy*, 2) *training throughput and communication efficiency*, 3) *dynamic training throughput in degrading network conditions*, and 4) *dynamic training throughput in fluctuating network conditions.*

Time-to-Accuracy. We evaluated NetSenseML against TopK compression (0.1 rate) and AllReduce for ResNet18 and VGG16 models in bandwidth constrained environments (200 Mbps, 500 Mbps, and 800 Mbps). The ResNet18 model size is 46.2MB, making AllReduce impractical without at least 500 Mbps to prevent congestion. Therefore, we tested under bandwidths of 200 Mbps, 500 Mbps, and 800 Mbps.

Under extremely low bandwidth conditions, NetSenseML demonstrates superior convergence speed compared to both AllReduce and TopK. We use the point at which NetSenseML achieves its best test accuracy as the benchmark and terminate the training of AllReduce and TopK at that point.

Figure 5 shows that NetSenseML consistently finds an appropriate compression ratio to maintain stable training and converge effectively, even under bandwidth bottlenecks. At 200 Mbps, NetSenseML significantly improves convergence time and stability compared to TopK and AllReduce, which exhibit both slower convergence and reduced stability. NetSenseML maintains a consistent accuracy curve without major fluctuations, while TopK-0.1 suffers from instability. At 500 Mbps, NetSenseML achieves the target accuracy 5× faster than TopK-0.1. NetSenseML surpasses AllReduce in both convergence speed and stability. Under 800 Mbps, NetSenseML retains its advantage, converging more quickly and steadily to a higher accuracy.

Training Throughput and Communication Efficiency. Table 2 presents a comparison of training throughput and convergence times for NetSenseML, AllReduce, and TopK-0.1 under different bandwidth conditions (Table 1).

The results demonstrate the robustness of NetSenseML in achieving optimal accuracy with significantly reduced convergence time, particularly in challenging low-bandwidth environments.

NetSenseML consistently maintains a specific training throughput across different bandwidth conditions. Taking ResNet18 as an example, when network
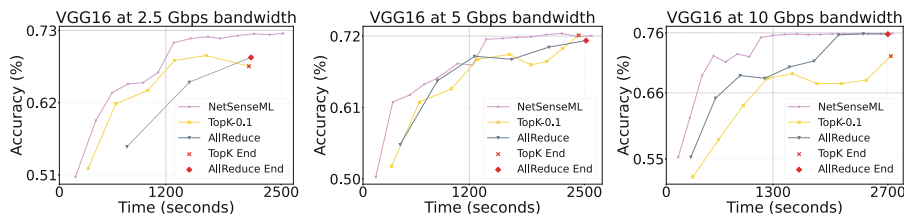
Fig. 6. Time to accuracy (TTA) comparison under different bottleneck bandwidth conditions with VGG16

Table 1. Performance Comparison of ResNet18 under NetSenseML and Other Methods: Training Throughput (samples/second), Convergence Time (seconds), and Best Test Accuracy (%)

Method	Bottleneck Bandwidth	Test Accuracy	Training Throughput	Convergence Time
NetSenseML	200 Mbps	75.79%	642.90	1575
AllReduce	200 Mbps	67.34%	42.20	N/A
TopK-0.1	200 Mbps	66.52%	65.31	N/A
NetSenseML	500 Mbps	76.02%	744.24	1090
AllReduce	500 Mbps	73.24%	179.85	N/A
TopK-0.1	500 Mbps	75.62%	144.27	3252
NetSenseML	800 Mbps	80.18%	824.56	1808
AllReduce	800 Mbps	80.35%	283.60	3150
TopK-0.1	800 Mbps	75.68%	234.77	1920

conditions match the communication scale, avoiding network congestion and preventing increased RTT or packet loss and retransmission, NetSenseML identifies a higher compression rate to avoid compression, optimizing the overall performance. When network limitations become the performance bottleneck for the entire training cluster, such as in cases of network congestion in cloud environments or operational issues in data centers, traditional methods like AllReduce and static compression algorithms like TopK struggle. These methods often slow down the gradient synchronization process, subsequently hindering the entire training process. In contrast, NetSenseML effectively mitigates this issue, adapting dynamically to maintain efficient training.

An interesting observation is that when training ResNet18 under a network bottleneck bandwidth of 200 Mbps, the training throughput of the TopK compression algorithm was higher than that of AllReduce, which aligns with our intuition since TopK has a smaller overall communication volume. However, when the bottleneck bandwidth was increased to 500 Mbps and 800 Mbps, AllReduce achieved higher training throughput compared to TopK-0.1. This can be attributed to the use of the AllGather communication pattern by TopK for gradient synchronization, whereas the NCCL AllReduce operation exhibits a high degree of parallelism, allowing it to efficiently utilize the network link bandwidth.

**Table 2.** Performance Comparison of VGG16 under NetSenseML and Other Methods

Method	Bottleneck Bandwidth	Test Accuracy	Training Throughput	Convergence Time
NetSenseML	2.5 Gbps	72.04%	172.80	2520
AllReduce	2.5 Gbps	66.52%	43.20	N/A
TopK-0.1	2.5 Gbps	65.12%	96.32	N/A
NetSenseML	5 Gbps	72.55%	199.46	2040
AllReduce	5 Gbps	71.31%	91.58	N/A
TopK-0.1	5 Gbps	72.12%	128.61	2436
NetSenseML	10 Gbps	75.90%	340.35	1808
AllReduce	10 Gbps	75.84%	129.87	2454
TopK-0.1	10 Gbps	72.30%	148.61	2736
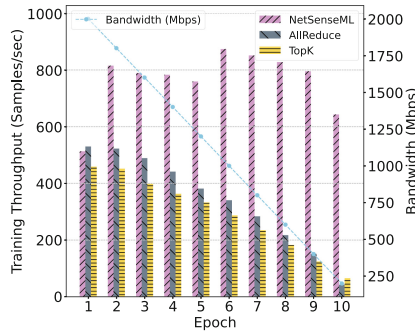
Fig. 7. Impact of Bandwidth Fluctuation on Training Throughput of ResNet18

Dynamic Training Throughput in Degrading Network Conditions. The adaptability of NetSenseML to changing bandwidth conditions is illustrated in Scenario 2, where dynamic training performance is compared between different levels of bandwidth. Using the ResNet18 model, we gradually decreased the bottleneck bandwidth - from 2000 to 200 Mbps in steps of 200 Mbps - and measured training performance at each level. This approach highlights the advantage of NetSenseML over TopK and AllReduce. As network conditions deteriorate, NetSenseML is able to maintain higher throughput by reducing gradient transmission, while TopK and AllReduce maintain constant data volumes, leading to network congestion and reduced throughput.

Figure 7 presents the training throughput of different methods as the initial bottleneck bandwidth of 2000 Mbps gradually decreases to 200 Mbps. In the first epoch, NetSenseML needed time to determine an appropriate compression ratio for the available bandwidth, initially leading to suboptimal throughput. However, as training progressed, NetSenseML adapted effectively, adjusting the compression ratio to accommodate increasingly constrained and adverse network conditions. This adaptability allowed it to maintain consistent training
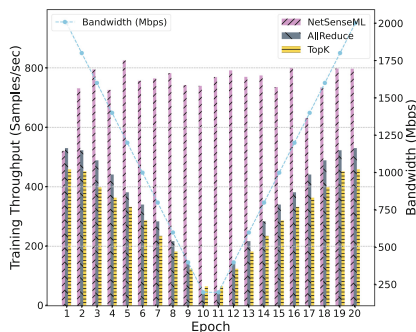
Fig. 8. Impact of Bandwidth Fluctuation on Training Throughput of ResNet18

throughput, which is highly valuable for cloud scenarios where programmers need to quickly observe the final convergence of the model. In contrast, TopK and AllReduce were unable to adapt to the decreasing bandwidth, resulting in lower training throughput as network constraints intensified.

Dynamic Training Throughput in Fluctuating Network Conditions. In Scenario 3, we used the ResNet18 model, and other configurations were kept the same as in Scenario 2. To illustrate the coexistence of our training task with other network applications, we ran iperf3 [21] in a multiprocess setup to compete for the network link, thereby preempting the link between switches and imposing constraints on network traffic during training. Figure 8 shows that regardless of how dynamically competing traffies, NetSenseML is consistently able to proceed with stable training throughput, demonstrating a significantly higher level of stability compared to static collective communication methods such as AllReduce and TopK.

6   Conclusions

Network fluctuations in bandwidth and latency often degrade resource efficiency, slow model convergence, and reduce accuracy, especially in heterogeneous environments. To address this, we propose NetSenseML, which is an adaptive compression algorithm that dynamically adjusts gradient compression ratios based on real-time network performance.

By monitoring network conditions and estimating bandwidth, NetSenseML applies quantization, pruning, and sparsification to optimize both critical gradient retention and model generalizability. NetSenseML performs well in adapting to a variety of network conditions, ensuring robust and efficient gradient transmission. This adaptability significantly improves the performance of model training and convergence rates in various scenarios, effectively mitigating the typical trade-offs between transmission efficiency, convergence speed, and accuracy.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Abdelmoniem, A.M., Canini, M.: DC2: delay-aware compression control for distributed machine learning. In: IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, pp. 1–10 (2021)
2. Aji, A.F., Heafield, K.: Sparse communication for distributed gradient descent. In: Palmer, M., Hwa, R., Riedel, S. (eds.) Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, Copenhagen, Denmark, pp. 440–445. Association for Computational Linguistics (2017)
3. Alizadeh, M., et al.: Data center TCP (DCTCP). SIGCOMM Comput. Commun. Rev. **40**(4), 63–74 (2010)
4. Ansel, J., Yang, E., , et al.: Pytorch 2: faster machine learning through dynamic python bytecode transformation and graph compilation. In: ASPLOS 2024, pp. 929–947. Association for Computing Machinery, New York (2024)
5. Brown, T.B., Mann, B., Ryder, N., et al.: Language models are few-shot learners. CoRR abs/2005.14165 (2020)
6. Cao, J., et al.: Crux: GPU-efficient communication scheduling for deep learning training. In: ACM SIGCOMM 2024, pp. 1–15. Association for Computing Machinery, New York (2024)
7. Cardwell, N., Cheng, Y., Gunn, C.S., Yeganeh, S.H., Jacobson, V.: BBR: congestion-based congestion control: measuring bottleneck bandwidth and round-trip propagation time. Queue **14**(5), 20–53 (2016)
8. Duplyakin, D., et al.: The design and operation of {CloudLab}. In: 2019 USENIX Annual Technical Conference (USENIX ATC 2019), pp. 1–14 (2019)
9. Ha, S., Rhee, I., Xu, L.: Cubic: a new TCP-friendly high-speed TCP variant. ACM SIGOPS Oper. Syst. Rev. **42**(5), 64–74 (2008)
10. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems (NeurIPS), pp. 1135–1143 (2015)
11. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural networks (2015)
12. Han, W., Vargaftik, S., Mitzenmacher, M., Karp, B., Basat, R.B.: Beyond throughput and compression ratios: towards high end-to-end utility of gradient compression (2024)
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015). https://arxiv.org/abs/1512.03385
14. Langley, A., et al.: The quic transport protocol: design and internet-scale deployment. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication. In: SIGCOMM 2017, pp. 183–196. Association for Computing Machinery, New York (2017)

15. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: International Conference on Learning Representations (ICLR) (2016)
16. Li, M., et al.: THC: accelerating distributed deep learning using tensor homomorphic compression (2024)
17. Li, M., et al.: Scaling distributed machine learning with the parameter server. In: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2014), Broomfield, CO, pp. 583–598. USENIX Association (2014)
18. Lin, Y., Han, S., Mao, H., Wang, Y., Dally, W.J.: Deep gradient compression: reducing the communication bandwidth for distributed training. CoRR (2017)
19. NVIDIA: Nvidia collective communications library (NCCL) (2024). https://developer.nvidia.com
20. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2015). https://arxiv.org/abs/1409.1556
21. iPerf Team: iPerf - the ultimate speed test tool for TCP, UDP and SCTP (2024). https://github.com/esnet/iperf
22. Touvron, H., et al.: Llama: open and efficient foundation language models (2023)
23. Wang, H., Tian, H., Chen, J., Wan, X., Xia, J., et al.: Towards domain-specific network transport for distributed DNN training. In: 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 2024), Santa Clara, CA, pp. 1421–1443. USENIX Association (2024)
24. Wang, Z., Lin, H., Zhu, Y., Ng, T.E.: Hi-speed DNN training with espresso: unleashing the full potential of gradient compression with near-optimal usage strategies. In: Proceedings of the Eighteenth European Conference on Computer Systems, pp. 867–882 (2023)