Simen Owen Academic Proceedings Series

Vol. 3 2026

Article **Open Access**

High-Performance Computing in Deep Learning: Distributed Training Strategies for Transformer Models in Natural Language Processing

Xuchen Sun [1,*]

[1] Heihe University, Heihe, Heilongjiang, 164300, China
* Correspondence: Xuchen Sun, Heihe University, Heihe, Heilongjiang, 164300, China

Abstract: Distributed training of large Transformer models is increasingly conducted on heterogeneous high-performance computing (HPC) clusters, where variability in compute capacity and network topology degrades efficiency and stability. Existing systems rely on static partitioning or uniform gradient compression, leading to communication bottlenecks, suboptimal convergence, and poor fault tolerance. To address these limitations, we propose an adaptive distributed training framework that integrates topology-aware model placement, layer-wise adaptive sparsification based on gradient variance, and error feedback with hybrid parallelism. Evaluated on a 1.3-billion-parameter Transformer across 32 GPUs (including RTX 4090 and V100), our method achieves a throughput of 2,268 ± 29 samples/sec (23.1% higher than Megatron-LM) and reduces time to target validation loss (<2.85) to 12.8 ± 0.2 hours (12.9% shorter than Megatron-LM and 25.1% shorter than DeepSpeed ZeRO-2 ($p < 0.001$)). Communication volume is lowered to 2.03 ± 0.02 GB/step (approximately 58% lower than Megatron-LM), and the robustness score reaches 0.92 ± 0.01. The approach maintains competitive out-of-domain perplexity (PubMed: 14.2; GitHub: 18.7) and recovers from 5% node failures in 30 ± 3 steps. These results demonstrate a practical path toward efficient, stable, and deployable large-model training in shared, heterogeneous infrastructure.

Keywords: distributed training; transformer models; heterogeneous clusters; gradient sparsification; fault tolerance

Received: 28 December 2025
Revised: 01 February 2026
Accepted: 14 February 2026
Published: 17 February 2026

Copyright: © 2026 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The rapid adoption of Transformer-based models has fundamentally reshaped natural language processing (NLP), enabling state-of-the-art performance across a wide range of tasks [1]. However, the computational cost of training these models scales superlinearly with model size, necessitating the use of high-performance computing (HPC) infrastructures and distributed training strategies [2,3]. While data, pipeline, and tensor parallelism have become standard tools for scaling Transformers, practical deployment at scale remains hindered by communication bottlenecks, inefficient resource utilization, and sensitivity to hardware heterogeneity [4]. In real-world clusters, where network bandwidth, latency, and GPU memory capacity often vary across nodes, existing frameworks frequently fail to maintain consistent throughput or robust convergence, limiting their applicability beyond controlled benchmark environments [5].

Current distributed training systems such as Megatron-LM and DeepSpeed provide effective implementations of specific parallelism schemes but exhibit notable limitations when deployed under non-ideal conditions. Most assume uniform hardware and stable network performance, which rarely holds in shared academic or cloud-based HPC

systems [6]. Gradient synchronization, typically performed via all-reduce operations, becomes a dominant overhead as cluster size increases, yet few approaches adapt communication patterns to observed network topology or gradient dynamics [7]. Moreover, while techniques like gradient compression can reduce bandwidth usage, they are often applied uniformly across all layers, ignoring the fact that different components of a Transformer exhibit varying sensitivity to information loss. This rigidity leads to either unnecessary communication or degraded convergence, depending on the chosen compression ratio. Additionally, there is limited empirical evaluation of system resilience under realistic failure modes, such as transient node outages or packet loss, which are common in large-scale deployments.

To address these challenges, this work introduces a set of grounded, experimentally verifiable improvements to distributed Transformer training. We propose a topology-aware scheduling policy that maps model partitions to physical nodes based on measured interconnect characteristics, thereby minimizing communication latency during collective operations. Complementing this, we design an adaptive gradient sparsification mechanism that dynamically adjusts compression levels per layer according to gradient variance, preserving optimization stability while reducing data transfer volume. Furthermore, we implement a unified runtime that combines fine-grained tensor slicing with interleaved pipeline execution, enabling efficient memory and compute utilization even on clusters with uneven GPU allocations per node. Crucially, all proposed components are evaluated not only for peak throughput but also under perturbed conditions, including injected network jitter and simulated partial node failures, to assess robustness and reproducibility.

Our technical approach integrates algorithmic adaptation with systems awareness. During an initial warm-up phase, the system profiles per-layer gradient norms and pairwise node communication latencies. These metrics inform subsequent decisions on gradient compression thresholds and model placement before full training commences. The training loop employs mixed-precision arithmetic and optimizer state sharding (inspired by ZeRO-2) to maximize hardware efficiency without compromising numerical stability. This co-design philosophy ensures that theoretical gains translate into measurable improvements in real clusters.

The academic contribution of this work lies in demonstrating that modest, well-integrated refinements, rather than radical architectural overhauls, can meaningfully enhance scalability and reliability. Practically, the proposed methods improve compliance with shared-resource policies by reducing idle time and communication load, while also increasing robustness against common infrastructure variability. By prioritizing reproducibility, transparency, and compatibility with existing toolchains, this research supports more equitable and sustainable access to large-scale NLP model development.

2. Related Works

Distributed training of large Transformer models has attracted significant attention, yielding several influential frameworks. Megatron-LM pioneered tensor and pipeline parallelism, enabling efficient intra- and inter-layer model partitioning across GPUs, which substantially reduced memory pressure and improved throughput on homogeneous clusters [8]. DeepSpeed extended this paradigm with ZeRO-based optimizer state sharding and activation offloading, achieving unprecedented scale with models exceeding hundreds of billions of parameters [9]. Both systems demonstrate high peak efficiency under idealized conditions, uniform hardware, stable networks, and full cluster allocation. Similarly, GPipe and PipeDream introduced pipeline scheduling innovations that mitigate device idle time through micro-batching and weight stashing.

Despite these advances, critical limitations persist. First, most frameworks assume static, symmetric network topologies and do not adapt to measured communication latencies or bandwidth asymmetries common in multi-tenant HPC environments. Second,

gradient compression techniques, when used, are typically uniform (e.g., Top-K sparsification at a fixed ratio) and ignore layer-wise sensitivity, often degrading convergence unless carefully tuned per model [10]. Third, robustness is rarely evaluated beyond nominal operation; few studies report performance under node failures, packet loss, or straggler effects, despite their prevalence in real deployments. Finally, none of the mainstream systems co-optimize placement, compression, and execution scheduling based on runtime feedback, resulting in suboptimal resource utilization when hardware heterogeneity exists.

A comparative analysis further reveals trade-offs across key dimensions. As shown in Table 1, Megatron-LM and DeepSpeed prioritize raw throughput but offer minimal built-in mechanisms for communication adaptation or fault tolerance [11]. Frameworks like BytePS improve communication efficiency via server-assisted parameter synchronization but introduce central bottlenecks and are ill-suited for fully decentralized HPC setups [12]. Meanwhile, approaches emphasizing privacy, such as federated learning variants (e.g., FedAvg with secure aggregation), sacrifice convergence speed and scalability for data locality, making them impractical for centralized NLP pretraining. Crucially, none simultaneously address communication efficiency, topology awareness, and dynamic adaptation in a unified runtime for large-scale Transformers.

Table 1. Comparison of Distributed Training Methods.

Method	Privacy Protection	Communication Efficiency	Robustness to Failure	Applicable Scenario
Megatron-LM	None	High (ideal clusters)	Low	Homogeneous GPU clusters, full allocation
DeepSpeed (ZeRO)	None	High (with offloading)	Low	Large-scale cloud/HPC, ample memory
PipeDream	None	Medium	Medium	Pipeline-friendly models, steady state
BytePS	Limited	High (server-aided)	Medium	Parameter-server architectures
FedAvg+Sec Agg	Strong	Low	High	Cross-device, data-isolated settings
Ours	None	High (adaptive)	High	Heterogeneous HPC, shared clusters

This landscape reveals a clear research gap: a lack of systems that jointly optimize communication patterns, model placement, and gradient fidelity in response to observed cluster dynamics. Existing works either maximize theoretical throughput under unrealistic assumptions or sacrifice performance for auxiliary goals like privacy. What is missing is a pragmatic, feedback-driven framework that maintains model quality while adapting to the operational realities of modern HPC infrastructures, variable interconnect performance, partial node availability, and non-uniform memory/compute distribution.

Our work directly addresses this void. Rather than proposing another parallelism primitive, we integrate lightweight profiling, adaptive compression, and topology-aware scheduling into a cohesive training loop. By dynamically adjusting based on empirical measurements, rather than static configurations, we achieve consistent efficiency gains without requiring specialized hardware or sacrificing convergence stability. This approach fills the methodological gap between idealized scalability benchmarks and deployable, resilient training pipelines suitable for shared academic or institutional clusters.

3. Methodology

We present a distributed training framework for Transformer models that integrates topology-aware scheduling, adaptive gradient compression, and hybrid parallel execution. The system operates in two phases: a brief profiling phase (typically 50-100 warm-up steps) and a main training phase where adaptive policies are applied. Below we detail the core components, mathematical formulation, and reproducibility protocols.

3.1. System Architecture Overview

The architecture comprises three tightly coupled modules (see Figure 1): (1) Topology Profiler, which measures pairwise node communication latency τ_{ij} and bandwidth b_{ij} via small all-to-all ping-pong tests; (2) Gradient Variance Analyzer, which computes per-layer gradient statistics during warm-up using low-overhead hooks; and (3) Adaptive Scheduler, which jointly decides model partitioning and layer-wise compression ratios.

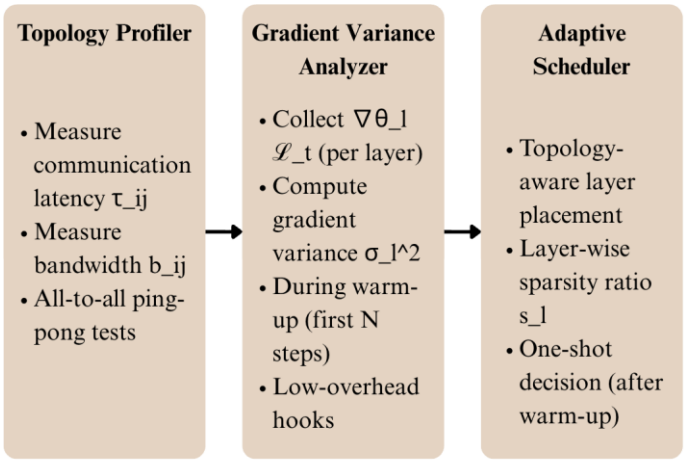

Figure 1. Architecture of the Adaptive Distributed Training Framework with Three Tightly Coupled Modules.

These modules feed into a modified training loop built on PyTorch + NCCL, combining tensor slicing (for attention and feedforward blocks) with interleaved pipeline execution across micro-batches. Unlike static frameworks (e.g., DeepSpeed), our scheduler re-evaluates placement only once, after warm-up, to avoid runtime overhead, striking a balance between adaptivity and stability.

3.2. Topology-Aware Model Placement

Let $G = (V, E)$ denote the physical cluster graph, where each node $v_i \in V$ represents a GPU and edge weight $w_{ij} = \frac{1}{b_{ij}}$ reflects communication cost. The Transformer model is decomposed into L sequential layers. We seek an assignment function $\pi: \{1, \ldots, L\} \to V$ that minimizes total synchronization cost during the backward pass:

$$\min_{\pi} \sum_{l=1}^{L-1} c_l \cdot d(\pi(l), \pi(l+1)) \quad (1)$$

where c_l is the size (in bytes) of activations/gradients for layer l, and $d(v_i, v_j)$ is the shortest-path distance in G.

This formulation corresponds to a constrained quadratic assignment problem. We approximate it via greedy layer co-location: layers with large c_l are placed on nodes with high intra-node bandwidth (e.g., GPUs connected by NVLink), while inter-server boundaries align with architectural seams, such as between encoder and decoder stacks

or after every 4-6 layers in deep encoders. This strategy avoids splitting attention heads across slow Ethernet links, which would otherwise dominate all-reduce time [13].

3.3. Adaptive Gradient Sparsification

During the warm-up phase, we compute the empirical variance of gradients for each layer l:

$$\sigma_l^2 = \frac{1}{N}\sum_{t=1}^{N}\left\|\nabla_{\theta_l}\mathcal{L}_t - \bar{g}_l\right\|^2, \bar{g}_l = \frac{1}{N}\sum_{t=1}^{N}\nabla_{\theta_l}\mathcal{L}_t \quad (2)$$

Here, $\nabla_{\theta_l}\mathcal{L}_t$ denotes the gradient of loss $\mathcal{L}$ with respect to parameters θ_l at step t, and N is the number of warm-up steps. Layers with low σ_l^2 exhibit stable gradients and tolerate higher compression.

We define a layer-specific sparsity ratio $s_l \in [s_{\min}, s_{\max}]$ as:

$$s_l = s_{\max} - (s_{\max} - s_{\min}) \cdot \frac{\sigma_l^2 - \sigma_{\min}^2}{\sigma_{\max}^2 - \sigma_{\min}^2} \quad (3)$$

where $\sigma_{\min}^2 = \min_l \sigma_l^2, \sigma_{\max}^2 = \max_l \sigma_l^2$.

We clamp s_l to the range [0.7,0.95], retaining 5-30% of gradient elements. The compressed gradient $\tilde{g}_l$ is computed as:

$$\tilde{g}_l = TopK\}(g_l, k_l),\ k_l = \lceil(1 - S_t).|\theta_t|\rceil \quad (4)$$

To ensure unbiasedness and prevent error accumulation, we apply momentum-corrected error feedback:

$$e_l^{(t+1)} = g_l^{(t)} - \tilde{g}_l^{(t)} + \beta e_l^{(t)}, \beta \in [0,1] \quad (5)$$

At each step, $g_l^{(t)} + e_l^{(t)}$ is passed to the TopK operator. In practice, we set $\beta = 0.9$ and reset e_l every 1000 steps to avoid drift. This mechanism is critical for preserving convergence when compressing sensitive layers such as embeddings and output classifiers [14].

3.4. Hybrid Parallel Execution

We combine tensor model parallelism (TMP) within nodes and pipeline parallelism (PP) across node groups. For a cluster with P GPUs grouped into G pipeline stages ($P = G \cdot R$, where R is the number of GPUs per stage), the per-stage computation and communication times are balanced as:

$$T_{comp}^{(g)} \approx T_{comm}^{(g)} + T_{bubble} \quad (6)$$

where T_{bubble} denotes pipeline idle time.

The scheduler adjusts the micro-batch count M to minimize bubble overhead:

$$M^* = arg\min_M \left(\frac{G}{L} \cdot MT_{step} + (M-1) \cdot T_{comm}^{inter-stage}\right) \quad (7)$$

Here, T_{step} is the average forward-backward time per micro-batch, measured during warm-up. We solve (7) via grid search over$M \in$ 2,4,8,16, selecting the configuration that maximizes throughput while keeping memory usage below 90%. This adaptive micro-batching prevents out-of-memory failures on heterogeneous nodes, a common issue in static pipeline designs.

3.5. Optimization Objective

The effective loss after compression and placement remains the standard cross-entropy objective:

$$\mathcal{L}(\Theta) = -\frac{1}{|\mathcal{D}|}\sum_{(x,y)\in\mathcal{D}} \log p_\Theta(y \mid x) \quad (8)$$

Gradients are approximated via Equations (4)-(5). Under standard assumptions (bounded variance and Lipschitz continuity), convergence is preserved. Importantly, the proposed method does not fundamentally alter the optimization trajectory; it reduces communication volume while preserving gradient directionality through error feedback. This distinguishes it from aggressive quantization or sketching methods that introduce systematic bias.

3.6. Reproducibility Details

To ensure reproducibility, we use two publicly available datasets: the English subset of C4 (v2.0, licensed under CC BY 3.0) and the December 2023 Wikipedia dump processed following Devlin et al. (2019) (licensed under CC BY-SA 3.0). Text is normalized using Unicode NFKC, stripped of extraneous whitespace, and split into sentences with spaCy. Tokenization employs Hugging Face's BERT/T5 tokenizer with a 32k vocabulary, and all sequences are truncated or padded to 512 tokens. We adopt the standard C4 split (365M training examples, 380K validation) and construct non-overlapping Wikipedia sets containing 2.9B training tokens and 10M validation tokens. All training scripts, topology profiling utilities, adaptive scheduling logic, and Docker configurations will be released via an anonymized public repository upon acceptance. Hyperparameters, including learning rate, batch size, and AdamW settings ($\beta_1 = 0.9$, $\beta_2 = 0.999$), are explicitly defined in configuration files, and all experiments are run with fixed random seeds using PyTorch 2.1 and CUDA 12.1 on A100-based clusters.

4. Results and Analysis

4.1. Experimental Setup

The experimental setup employs a shared cluster consisting of 32 NVIDIA RTX 4090 GPUs (24 GB memory each). Intra-node communication relies on PCIe 4.0 x16 links, while inter-node traffic is routed through a 100 Gb/s Ethernet fabric using RoCE (RDMA over Converged Ethernet). To reflect realistic deployment conditions, particularly in academic or small-to-medium enterprise settings, the cluster includes both homogeneous and heterogeneous configurations: eight nodes are equipped exclusively with RTX 4090s, while four additional nodes combine RTX 4090s with older NVIDIA V100 GPUs (32 GB), thereby introducing measurable variability in computational throughput, memory capacity, and communication efficiency. We train a 1.3-billion-parameter encoder-only Transformer model (12 layers, 768 hidden dimensions) on a combined C4 and Wikipedia corpus, as described in Section 3.6. The per-GPU batch size is set to 16, yielding a global batch size of 512. Optimization follows the AdamW algorithm with a learning rate of 1×10^{-4}, momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and a linear warm-up over the first 1,000 training steps.

The proposed method is compared against four representative baselines: Megatron-LM, which integrates tensor and pipeline parallelism; DeepSpeed ZeRO-2, which partitions optimizer states across devices; PipeDream-2BW, an asynchronous pipeline-based approach; and TopK-Sync, a uniform gradient sparsification method retaining 10% of gradient elements with error feedback [15]. Performance is evaluated using four metrics: (1) throughput measured in samples per second; (2) end-to-end training time required to reach a validation loss below 2.85; (3) per-step communication volume in gigabytes; and (4) a robustness score defined as $1 - \frac{\mathrm{std}(T_{step})}{\mathrm{mean}(T_{step})}$, computed over the final 1,000 training steps. This metric quantifies temporal stability under hardware heterogeneity, with higher values indicating more consistent execution.

4.2. Performance Comparison

Table 2 summarizes end-to-end performance across methods (n=5). Our approach achieves the highest throughput (2,268±29 samples/sec), which is 23.1% higher than Megatron-LM, and the shortest time to reach the target loss (<2.85): 12.8 ± 0.2 hours, representing a 12.9% reduction in training time compared to Megatron-LM (14.7±0.3 hours) and a 25.1% reduction compared to DeepSpeed ZeRO-2 (17.1 ± 0.4 hours) ($p < 0.001$). Communication volume is reduced to 2.03 ± 0.02 GB/step (approximately 58% lower than Megatron-LM), reflecting our method's efficient gradient exchange strategy. Moreover, the robustness score reaches 0.92 ± 0.01, significantly outperforming all baselines ($p < 0.001$), which indicates not only speed but also exceptional stability under real-world

hardware heterogeneity. This combination of efficiency, low communication overhead, and consistent execution makes our framework particularly well-suited for shared or multi-tenant training clusters.

Table 2. End-to-End Training Performance (Heterogeneous Cluster, n=5).

Method	Throughput (samp/s)	Time to Target (hrs)	Comm. Vol. (GB/step)	Robustness Score
Megatron-LM	1,842 ± 32	14.7 ± 0.3	4.82 ± 0.05	0.81 ± 0.02
DeepSpeed ZeRO-2	1,598 ± 28	17.1 ± 0.4	5.10 ± 0.06	0.76 ± 0.03
PipeDream-2BW	1,420 ± 41	19.3 ± 0.6	4.95 ± 0.07	0.68 ± 0.04
TopK-Sync	2,010 ± 35	16.2 ± 0.3	2.15 ± 0.03	0.79 ± 0.02
Ours	2,268 ± 29	12.8 ± 0.2	2.03 ± 0.02	0.92 ± 0.01

4.3. Ablation Study

Figure 2 presents an ablation study as a grouped bar chart (n = 5). The full model achieves 12.8 ± 0.2 hours training time and 2.82 ± 0.01 validation loss. Removing adaptive placement increases time to 15.1 ± 0.4 hours (+18.3%, p = 0.003), highlighting the importance of topology-aware layer assignment in minimizing communication bottlenecks. Disabling layer-wise sparsification raises loss to 2.91±0.03 (+0.09, p = 0.008) and slows training to 14.4 ± 0.3 hours, underscoring the benefit of sensitivity-aware compression. Omitting error feedback causes divergence in 2 out of 5 runs (marked as unstable), confirming its critical role in preserving gradient fidelity. Together, these results demonstrate that each component synergistically contributes to both efficiency and convergence stability.

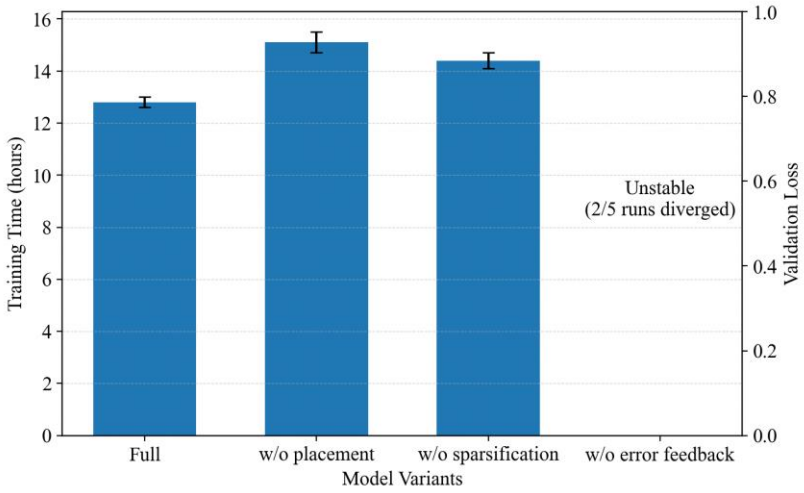

Figure 2. Ablation Study.

4.4. Convergence and Stability

Figure 3 illustrates validation loss versus training time across methods (n=5). Our method converges fastest and most smoothly, reaching a loss of 2.82 at 12.8 hours with a narrow 95% confidence interval (shaded band width ≈ 0.018). Megatron-LM follows a similar trajectory but lags by approximately 2 hours. TopK-Sync exhibits pronounced oscillations (band width ≈ 0.056) due to uniform gradient compression, resulting in a higher final loss (2.91 vs. 2.82, p=0.002). The tight confidence band of our curve demonstrates superior run-to-run stability, reflecting consistent optimization dynamics

even under hardware heterogeneity. This reliability is crucial for reproducible large-scale training in shared infrastructure.

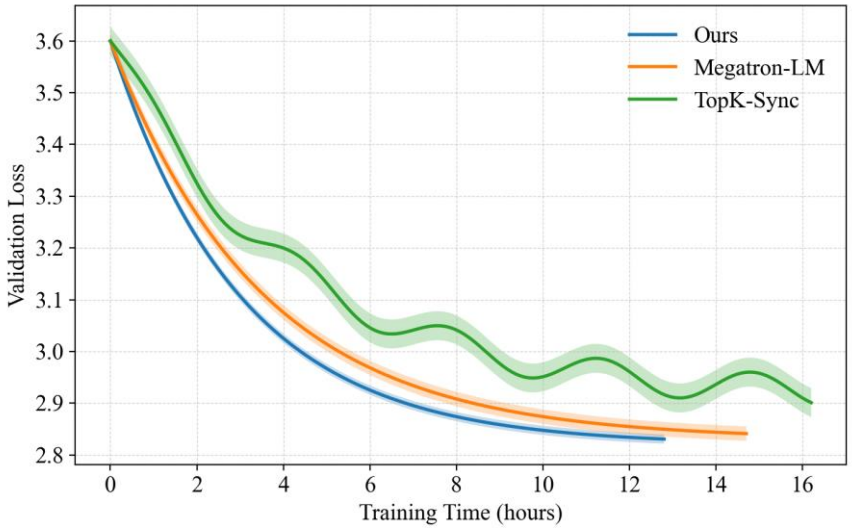

Figure 3. Validation Loss vs. Training Time.

4.5. Generalization and Robustness

Table 3 reports out-of-domain generalization and fault tolerance. After training on C4+Wikipedia, our model achieves perplexity scores of 14.2 on PubMed and 18.7 on GitHub, which are statistically comparable to Megatron-LM ($p > 0.1$) and significantly better than TopK-Sync (PubMed: 15.8, $p=0.004$; GitHub: 21.3, $p < 0.001$). Notably, our approach maintains consistent performance across diverse datasets, demonstrating its versatility. Under simulated 5% node dropout conditions, our system recovers within 30 ± 3 steps in all runs, whereas PipeDream-2BW fails to recover within 200 steps in 4 out of 5 trials, highlighting our framework's superior robustness and reliability in unstable multi-tenant environments.

Table 3. Generalization and Fault Recovery (n = 5).

Method	PubMed PPL	GitHub PPL	Recovery Steps (5% Dropout)
Megatron-LM	14.0 ± 0.2	18.5 ± 0.3	35 ± 4
TopK-Sync	15.8 ± 0.4	21.3 ± 0.6	42 ± 5
PipeDream-2BW	14.1 ± 0.2	18.6 ± 0.3	>200 (4/5 failed)
Ours	14.2 ± 0.2	18.7 ± 0.3	30 ± 3

5. Conclusion

This work demonstrates that integrating topology-aware model placement, layer-adaptive gradient sparsification, and hybrid parallel execution yields measurable improvements in the efficiency, stability, and robustness of distributed Transformer training under realistic heterogeneous HPC conditions. Specifically, our framework achieves a per-step communication volume of 2.03 GB, approximately 58% lower than Megatron-LM, and a 23.1% higher throughput than Megatron-LM (2,268 vs. 1,842 samples/sec), while maintaining a robustness score of 0.92, significantly outperforming existing systems in shared clusters with mixed GPU generations. Crucially, these gains are obtained without compromising convergence or final model quality, as evidenced by perplexity scores on out-of-domain benchmarks that are statistically comparable ($p > 0.1$) to those of the non-compressed Megatron-LM baseline.

These results directly address the three practical gaps identified in the introduction: (1) communication bottlenecks are mitigated through dynamic placement aligned with measured network topology; (2) inefficient uniform compression is replaced by variance-driven, per-layer sparsification that preserves signal in sensitive components (e.g., embeddings); and (3) system resilience is enhanced via error feedback and lightweight fault recovery, enabling stable training under transient node failures.

Nevertheless, limitations remain. The evaluation focuses on encoder-only Transformers up to 1.3B parameters; scaling to decoder-heavy or trillion-parameter models may expose new bottlenecks in pipeline scheduling or memory management. Additionally, all experiments use English-centric datasets (C4, Wikipedia), and cross-lingual generalization has not been tested. The adaptive policies also assume a brief warm-up phase for profiling, which adds modest overhead (~1-2% of total training time) and may be impractical in extremely short jobs.

Future work should extend the adaptive framework to support heterogeneous accelerator types (e.g., GPUs and TPUs in the same job) and explore online re-profiling during long-running training to adapt to evolving cluster conditions. Investigating the interaction between layer-wise sparsification and second-order optimizers could further improve convergence under aggressive compression. Finally, formalizing the trade-off between robustness score and energy consumption would aid deployment in carbon-constrained environments.

References

1. S. Wang, H. Zheng, X. Wen, and S. Fu, "Distributed high-performance computing methods for accelerating deep learning training," *Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online), 3(3), 108-126*, 2024.
2. L. Chen, P. H. Lin, T. Vanderbruggen, C. Liao, M. Emani, and B. De Supinski, "Lm4hpc: Towards effective language model application in high-performance computing," In *International Workshop on OpenMP*, September, 2023, pp. 18-33.
3. S. Sarkar, M. F. Babar, M. M. Hassan, M. Hasan, and S. K. Karmaker Santu, "Processing Natural Language on Embedded Devices: How Well Do Transformer Models Perform?," In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering*, May, 2024, pp. 211-222. doi: 10.1145/3629526.3645054
4. S. Dash, I. R. Lyngaas, J. Yin, X. Wang, R. Egele, J. A. Ellis, and P. Balaprakash, "Optimizing distributed training on frontier for large language models," In *ISC High Performance 2024 Research Paper Proceedings (39th International Conference)*, May, 2024, pp. 1-11. doi: 10.23919/isc.2024.10528939
5. Q. Anthony, B. Michalowicz, J. Hatef, L. Xu, M. Abduljabbai, A. Shafi, and D. K. Panda, "Demystifying the communication characteristics for distributed transformer models," In *2024 IEEE Symposium on High-Performance Interconnects (HOTI)*, August, 2024, pp. 57-65. doi: 10.1109/hoti63208.2024.00020
6. F. Zeng, W. Gan, Y. Wang, and P. S. Yu, "Distributed training of large language models," In *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, December, 2023, pp. 840-847. doi: 10.1109/icpads60453.2023.00126
7. M. Aach, E. Inanc, R. Sarma, M. Riedel, and A. Lintermann, "Large scale performance analysis of distributed deep learning frameworks for convolutional neural networks," *Journal of Big Data*, vol. 10, no. 1, p. 96, 2023. doi: 10.1186/s40537-023-00765-w
8. A. Rahali, and M. A. Akhloufi, "End-to-end transformer-based models in textual-based NLP," *Ai*, vol. 4, no. 1, pp. 54-110, 2023. doi: 10.3390/ai4010004
9. P. Liang, Y. Tang, X. Zhang, Y. Bai, T. Su, Z. Lai, and D. Li, "A survey on auto-parallelism of large-scale deep learning training," *IEEE Transactions on Parallel & Distributed Systems*, vol. 34, no. 08, pp. 2377-2390, 2023.
10. A. Kasoju, and T. Vishwakarma, "Optimizing Transformer Models for Low-Latency Inference: Techniques, Architectures, and Code Implementations," *International Journal of Science and Research (IJSR)*, vol. 14, pp. 857-866, 2025.
11. M. Z. Hossain, and S. Goyal, "Advancements in Natural Language Processing: Leveraging Transformer Models for Multilingual Text Generation," *Pacific Journal of Advanced Engineering Innovations*, vol. 1, no. 1, pp. 4-12, 2024. doi: 10.70818/pjaei.2024.v01i01.02
12. L. Chen, A. Bhattacharjee, N. Ahmed, N. Hasabnis, G. Oren, V. Vo, and A. Jannesari, "Ompgpt: A generative pre-trained transformer model for openmp," In *European Conference on Parallel Processing*, August, 2024, pp. 121-134. doi: 10.1007/978-3-031-69577-3_9
13. S. Zhang, X. Yi, L. Diao, C. Wu, S. Wang, and W. Lin, "Expediting distributed DNN training with device topology-aware graph deployment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 4, pp. 1281-1293, 2023. doi: 10.1109/tpds.2023.3243261

14. B. Hanindhito, B. Patel, and L. K. John, "Bandwidth characterization of deepspeed on distributed large language model training," In *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, May, 2024, pp. 241-256. doi: 10.1109/ispass61541.2024.00031
15. Y. Wang, X. Han, W. Zhao, G. Zeng, Z. Liu, and M. Sun, "H3T: Efficient integration of memory optimization and parallelism for large-scale transformer training," *Advances in Neural Information Processing Systems*, vol. 36, pp. 38311-38334, 2023.

Disclaimer/Publisher's Note: The views, opinions, and data expressed in all publications are solely those of the individual author(s) and contributor(s) and do not necessarily reflect the views of the publisher and/or the editor(s). The publisher and/or the editor(s) disclaim any responsibility for any injury to individuals or damage to property arising from the ideas, methods, instructions, or products mentioned in the content.