Understanding Communication Characteristics of Distributed Training

Citation: Wenxue Li, Xiangzhou Liu, Yuxuan Li, Yilun Jin, Han Tian, Zhizhen Zhong, Guyue Liu, Ying Zhang, and Kai Chen. 2024. Understanding Communication Characteristics of Distributed Training. In Proceedings of the 8th Asia-Pacific Workshop on Networking (APNet '24). Association for Computing Machinery, New York, NY, USA, 1–8.

As Published: https://doi.org/10.1145/3663408.3663409

Publisher: ACM|The 8th Asia-Pacific Workshop on Networking

Persistent URL: https://hdl.handle.net/1721.1/156676

Version: Final published version: final published article, as it appeared in a journal, conference proceedings, or other formally published context

Understanding Communication Characteristics of Distributed Training

Wenxue Li[1], Xiangzhou Liu[1], Yuxuan Li[1], Yilun Jin[1], Han Tian[2], Zhizhen Zhong[3], Guyue Liu[4], Ying Zhang[5], Kai Chen[1]
[1] *iSING Lab, Hong Kong University of Science and Technology*, [2] *University of Science and Technology of China*,
[3] *MIT*, [4] *Peking Univeristy*, [5] *Meta*

ABSTRACT

Communication is pivotal in distributed training and a thorough understanding of its characteristics is essential for future optimizations. However, prior works are limited, either focusing on customized optimizations or conducting incomplete explorations on communication characteristics. In this work, we systematically analyze the communication characteristics of distributed training, considering two key aspects of communication: *pattern* and *overhead*, and assessing a broad spectrum of determinant factors. In particular, we extensively investigate the features of communication patterns, such as predictability, and comprehensively evaluate the impact of various factors on communication overhead. Additionally, we develop and validate an *analytical formulation* to estimate communication overhead, providing a mathematical understanding of models with predictability.

CCS CONCEPTS

• **Networks** → **Network performance analysis**; • **Computing methodologies** → **Model development and analysis**.

KEYWORDS

Communication Characteristics, Distributed Training

ACM Reference Format:
Wenxue Li, Xiangzhou Liu, Yuxuan Li, Yilun Jin, Han Tian, Zhizhen Zhong, Guyue Liu, Ying Zhang, Kai Chen. 2024. Understanding Communication Characteristics of Distributed Training. In *The 8th Asia-Pacific Workshop on Networking (APNet 2024), August 03–04, 2024, Sydney, Australia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3663408.3663409

1 INTRODUCTION

Deep Neural Networks (DNNs) are increasingly adopted as fundamental building blocks in various modern services, such as language translation and autonomous driving [2, 24]. DNN training is an essential step in producing high-quality deep learning services. Given the limited computational capacity of individual accelerators (e.g., GPUs) and the high requirement, DNN training is typically distributed [12, 17, 18, 26, 28], where each GPU is assigned a partial job and all GPUs collaborate via various communication operations to complete the entire task.

APNet 2024, August 03–04, 2024, Sydney, Australia

ACM ISBN 979-8-4007-1758-1/24/08
https://doi.org/10.1145/3663408.3663409

Communication plays a significant role in distributed training and often becomes a bottleneck. Consequently, several prior works [19, 25, 26, 28, 32, 33] aim to reduce the communication time in training. These works typically concentrate on enhancing performance within certain conditions, such as specific model architectures or hardware platforms, and introducing customized optimizations. Although promising, they do not provide a comprehensive overview of the communication characteristics in distributed training. Such an understanding is essential to develop a systematic and long-term approach to communication optimization.

Prior works [6, 9, 26, 33, 33] attempt to depict the communication characteristics of distributed training. However, they conduct incomplete explorations, either lacking fine-grained analysis or overlooking important factors. Some of them [6, 9, 30] focus on cluster-level metrics, viewing the entire training job as a basic unit and primarily assessing metrics like job completion time and cluster utilization. This approach, however, misses the fine-grained features within individual jobs. Others [10, 26, 33, 34] delve into the within-job characteristics but overlook various key factors. For instance, some studies [26, 33] only focus on data parallelism, ignoring the complexities introduced by model parallelism. Several modeling-related research [10, 34] directly integrate the peak link capacity into formulations, overlooking the impact of network protocols.

In this work, we aim to systematically explore the communication characteristics of distributed training. Our analysis focuses on the individual job scenarios, paying attention to fine-grained within-job features. We analyze communication through two aspects: pattern and overhead. "Pattern" refers to the high-level traffic attributes, such as predictability and regularity. "Overhead" focuses on the metrics of communication time and communication ratio over the entire training process. Our analysis incorporates a multi-dimensional approach, varying key factors to assess their influence and thus obtain a comprehensive understanding.

For pattern analysis, we reveal the coexistence of predictability and semi-predictability in modern DNN training and conduct an in-depth exploration of these features (§3). Regarding overhead, we comprehensively consider various factors, including model architecture, training scale, network protocol, parallelism strategy, and hardware platforms, and evaluate their effect on communication overhead (§4). Furthermore, we devise an analytical formulation to estimate the communication overhead for typical models and validate its accuracy by comparing its predictions against data from our experiments (§5).

This work makes several key findings/conclusions:

- Communication pattern depends on the model architecture and parallelism strategy. For example, densely-activated models demonstrate predictable and regular patterns, while sparsely-activated models exhibit dynamic and semi-predictable patterns.

Platform	RTX3090-PCIe	V100-PCIe	V100-NVLink
GPU specs	RTX3090-24G	V100-32G	V100-32G
Intra-node Network	PCIe3.0x16	PCIe3.0x16	NVLink-V2
Inter-node Network	100 Gbps ConnectX-5	100 Gbps ConnectX-5	100 Gbps ConnectX-5

Table 1: Specifications of three hardware platforms used in this measurement work.

Model architecure	Number of parameters	Batch size	Parallelization strategy	Training scale (no. of GPUs)
ResNet50	24.37M	32	Data Parallel	4~32
ResNet101	42.49M	32	Data Parallel	4~32
VGG16	131.95M	32	Data Parallel	4~32
Bert-base	104.44M	12	Data Parallel	4~32
Bert-large	319.64M	12	Data Parallel	4~32
GPT	1.5B~3B	32	PTD Parallel	4~32

Table 2: DNN models (densely-activated) and their configurations used in this measurement work.

- In the context of predictable models, the predictability extends to both the communication matrix and traffic volume. Moreover, their patterns exhibit a regular "on-off" transmission shape and are recurrent across iterations.
- The factors influencing communication overhead are multifaceted. For example, we observe that the network protocol plays a key role in utilizing link bandwidth and thus significantly influences communication overhead.
- We develop an analytical formulation to estimate the communication overhead for densely-activated models and confirm its accuracy through empirical verification.

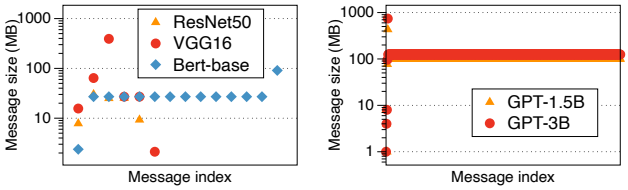
2 METHODOLOGY & OVERVIEW

Our experimental design incorporates a multi-dimensional approach, varying key factors to assess their impact on communication patterns and overhead. The following describes the details of our experimental setup.

Hardware platforms. Regarding hardware, we examine three hardware platforms: RTX3090-PCIe, V100-PCIe, and V100-NVLink. Each platform has its unique specifications, including GPU computation capacity, GPU memory size, and intra-node network capacity, as detailed in Table 1. All platforms are equipped with a Mellanox ConnectX-5 [3] RDMA NIC (RNIC) owning a 100Gbps bandwidth; each machine's GPUs share a single RNIC. The CPU capacity and host memory are similar across all platforms.

DNN workload and training scale. We evaluate four popular DNN models: ResNet [8], VGG16 [23], Bert [4], and GPT [20], representing CV, encoder-only and decoder-only NLP models. The configurations for each model, including parameter numbers, batch size, adopted parallelization strategy, and training scale, are detailed in Table 2. We vary the training scales for each model to assess the impact of the scale of training on communication.

Parallelization strategy and distributed framework. Different from prior works that focus mostly on Data Parallelism (DP) [33], we emphasize the analysis of hybrid parallelism, combining data and model parallelism. Two primary model parallelism approaches are Tensor Parallelism (TP) and Pipeline Parallelism (PP). Megatron-LM [18] terms this hybrid parallelism, consisting of PP, TP, and DP, as *PTD Parallelism* (PTD-P). Another notable model parallelism approach is Sequence Parallelism (SP) [11], which can be considered a variant of TP, exhibiting similar traffic volume and communication time but with lower activation memory consumption. Given our focus on the analysis of communication, we solely measure TP and assume SP to be equivalent to TP.



(a) Pure DP models. **(b) GPT models with PTD-P.**

Figure 1: Message distribution of three DP models using Pytorch DDP and two GPT models using DeepSpeed.

Regarding parallelism strategy selection, we employ data parallelism to train VGG, ResNet, and Bert models, with PyTorch DistributedDataParallel (short as PyTorch DDP) as the framework [13]. For partitioning GPT models across GPUs, we utilize Microsoft's DeepSpeed framework [16] to employ the PTD parallel training.

Network protocol. In each experiment, we assess two prevalent network protocols, TCP with Cubic [7] as congestion control (CC) and RoCEv2 with DCQCN [35] as CC, evaluating their effects on communication overhead.

Topology. Unless otherwise specified, we employ a ToR (Top of Rack) topology, wherein all machines are connected to the same ToR switch, forming a direct-connect topology.

Measurement overview. In this paper, we first focus on the features of communication patterns and analyze the impact of various factors on them (§3). We then shift to the communication overhead, *i.e.*, the ratio of communication over iteration time, and evaluate its determinant factors (§4). Finally, we construct an analytical formulation to estimate the communication overhead, the accuracy of which is verified using realistic results (§5).

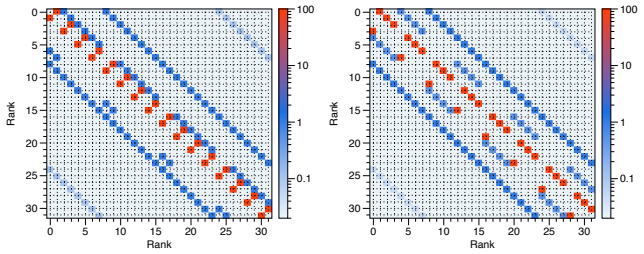
3 PATTERN ANALYSIS

In this section, we investigate the communication patterns of distributed training, beginning with an in-depth exploration of the predictability feature in traditional densely-activated models that utilize DP and PTD-P strategies. We then demonstrate the existence of semi-predictability in sparsely-activated models, such as Mixture of Experts (MoE).

3.1 In-depth Exploration of Predictability

The communication pattern is characterized by two primary elements: communication matrix and traffic volume. For traditional densely-activated models, we observe *both of them are predictable*. Furthermore, we observe that the factors (that determine communication patterns) include model architecture, parallelism strategy, and parallelism mapping from the logical parallelism strategy to the physical hardware platform.

Model architecture dependency. The model's internal architecture is a critical determinant of communication patterns, affecting the total traffic volume and message distribution. For instance, in



(a) Strategy $(p, t, d) = (4, 2, 4)$ **(b) Strategy** $(p, t, d) = (4, 4, 2)$

Figure 2: Traffic heatmaps of a GPT-3B model with two parallelism strategies on 32 GPUs.

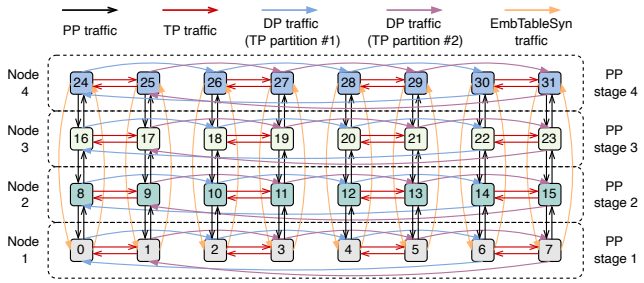


Figure 3: DeepSpeed's default parallelism mapping from $(p, t, d) = (4, 2, 4)$ **to 32 GPUs (4 nodes).**

pure DP training, the total traffic volume equals parameter number × precision. Thus, models listed in Table 2 exhibit varying total traffic volumes. Additionally, since Pytorch DDP transmits gradients layer-wise, the message distribution also depends on the model architecture. For example, we record the message size of ResNet50, VGG16 and Bert-base, showing the results in Fig. 1a. The three models exhibit different distributions. Specifically, several large messages (the largest one is 392MB) in VGG16 results from its fully connected (FC) layers. By contrast, all messages of ResNet50 are relatively small. Pytorch DDP applies a *bucket size* of 25MB to fuse gradients from small layers; that is the reason why many messages are near 25MB.

The message size distribution of two GPT models is further depicted in Fig. 1b, which (together with Fig. 1a) highlights a prevalence of large flows (e.g., 100MB). Due to the numerous parameters in DNN models and the existence of fusion mechanisms, *DNN training is dominated by large messages*, distinguishing from traditional datacenter applications [1, 14, 15, 29].

Parallelism strategy dependency. In models utilizing PTD-P training, communication is influenced by both model architecture and parallelism strategy. We consider a GPT model with 3 billion parameters over a platform of 32 RTX3090 GPUs. We evaluate two parallelism strategies: $(p, t, d) = (4, 2, 4)$ & $(4, 4, 2)$. The corresponding traffic heatmaps are depicted in Fig. 2 (the unit of colorbar is gigabytes (GB)). The heatmaps are distinct, showing two main differences. First, the latter strategy exhibits a larger TP communication range (more red squares). Second, the latter one shows less DP traffic, as it has a larger TP scale which reduces the number of parameters per GPU, leading to less DP traffic.

Communication matrix is predictable. Given the logical parallelism strategy and parallelism mapping from logical strategy to physical platform, we can predict the communication matrix, which specifies the set of GPU pairs that hold traffic. For instance, Fig. 2a with strategy $(p, t, d) = (4, 2, 4)$ adopts the *default mapping* approach of DeepSpeed, whose principle is to allocate TP and DP groups within machines and distribute PP stages across machines. We can depict the communication matrix of this setting before actually running the model, as illustrated in Fig. 3, where communication exists only between GPU pairs connected by arrows.

Traffic type	Volume	Number of messages	Message size
TP	~85 GB	680	125 MB
PP	~1 GB	16	125 MB
DP	741 MB	1	741 MB
EmbTableSyn	96 MB	1	96 MB

Table 3: Traffic volume and composition at $rank_0$ **during a GPT-3B model training over 32 GPUs.**

With the same parallelism strategy, parallelism mapping could make traffic patterns different. As an example, we maintain the same parallelism strategy with Fig. 2a and experiment with a *customized mapping* approach (Fig. 12): allocating TP groups and PP stages within machines and DP groups across machines. This approach exhibits distinct traffic heatmaps, as shown in Fig. 11b. Due to the page limitation, we leave the detailed explanation in Appendix A.1.

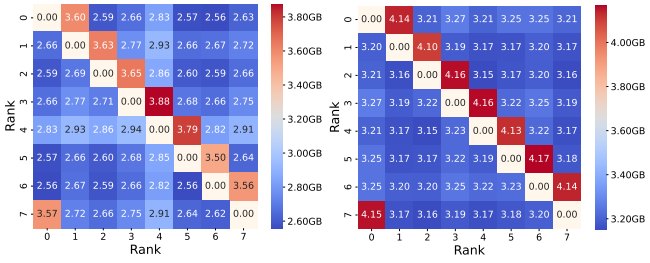
Traffic volume is computable. Given the model architecture and parallelism strategy, the traffic volume is computable. For example, under a GPT-3B model with the configuration shown in Fig. 3, we measure the transmitted traffic volume at $rank_0$ during one iteration. The results are demonstrated in Table 3, with traffic being categorized into four primary types: TP, PP, DP, and Embedding Table Synchronization (EmbTableSyn). The EmbTableSyn traffic occurs between the first and last PP stages for aggregating gradients of embedding tables.

Given the model configuration (l, h, s, gb, b, m) and parallelism strategy (p, t, d) (notations explained in Table 4), we can pre-calculate the traffic volume, including the total volume, number of messages and message size, to precisely match the results in Table 3. The formula for this calculation is detailed in §5. In this case, TP traffic constitutes about 99% of the total volume. This predominance of TP traffic is a consistent observation across all our experiments and is also mentioned by other studies [11, 18].

Regularity. DNN training exhibits distinct regularity in its communication pattern, specifically a consistent "on-off" transmission shape that recurs across iterations. This regularity arises from the dependent and structured communication and computation phases during DNN training, leading to regular transmission cycles. Although this has been previously mentioned in recent works [21, 27], we highlight it here to ensure a comprehensive summary of the characteristics.

3.2 Semi-predictability of Sparse Models

The MoE structure is a popular way to implement sparsely-activated models. Training large MoE models requires Expert Parallelism (EP) to distribute expert layers across multiple GPUs [5, 22]. This approach introduces all-to-all communication and makes MoE training with dynamic communication patterns. Notably, we find its pattern exhibits semi-predictability, *i.e.*, increasing predictability and uniformity as the model converges.



(a) Iteration 10. (b) Iteration 90.

Figure 4: Traffic heatmaps of two iterations during MoE (760M) training.

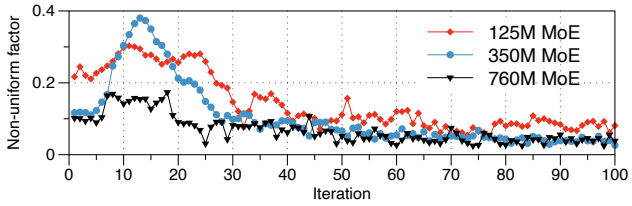


Figure 5: Increasing uniformity during MoE training.

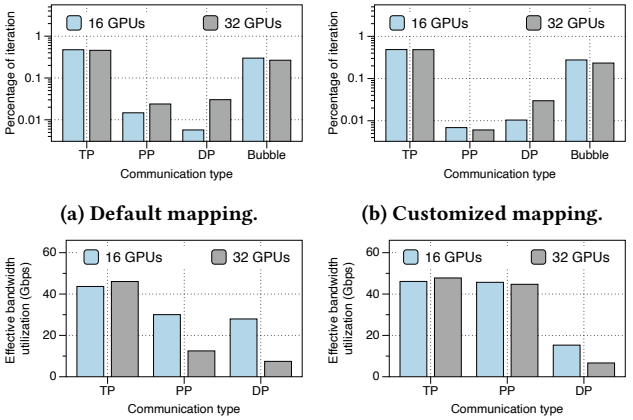
Dynamic patterns. To illustrate this, we evaluate a GPT-based MoE model with 760M parameters, distributed over 8 RTX3090 GPUs, with 8 EP for expert layers and 8 DP for other layers. The traffic heatmaps during MoE training are depicted in Fig. 4. The red diagonal squares in the heatmap represent a combination of Ring AllReduce traffic (from DP) and all-to-all traffic (from EP), while the blue squares indicate exclusively all-to-all traffic.

The results indicate that the all-to-all traffic is non-uniform within each iteration and dynamic across different iterations. We also observe that the total volume of all-to-all traffic varies in different iterations (Fig. 4b is "darker" than Fig. 4a). This variation is attributed to the adoption of a top-2 gating algorithm during training, where each token can be routed to one or two experts, leading to fluctuating traffic volumes. A trend we observed during our experiments is that as the training progresses, tokens increasingly tend to select two experts, consequently increasing the total traffic volume.

Semi-predictability. The gate network in MoE is trained with a target for load balance (*i.e.*, uniformly distributing tokens across experts). To assess its effectiveness, we analyze the uniformity of token distribution over the first 100 iterations and develop a *nonuniform factor* which is defined as the disparity between the highest and lowest traffic volumes of all-to-all traffic per iteration. We evaluate three GPT-based MoE models with sizes of 125M, 350M, and 760M. The results in Fig. 5 show a consistent decrease in the nonuniform factor, indicating the efficiency of the gate network in achieving more uniform token distribution as training progresses.

4 OVERHEAD ANALYSIS

In this section, we analyze the influence of various factors, including model architecture, training scale, parallelism strategy, hardware platform, and network protocol, on the communication overhead of typical models. In the body of this paper, we primarily focus on analyzing the overhead of GPT models. We also examine five DP models as listed in Table 2, uncovering the determinant factors of their communication overhead. Due to space limitation, we leave detailed analyses of these DP models in Appendix B.



(a) Default mapping. (b) Customized mapping.

(c) Default mapping. (d) Customized mapping.

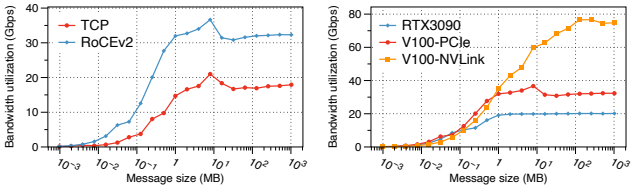
Figure 6: Communication ratio and effective bandwidth utilization during GPT training.

4.1 Overhead in GPT Training

The GPT models listed in Table 2 are evaluated with two training configurations: a 1.5B parameter model trained across 16 RTX3090s with $(p, t, d) = (4, 2, 2)$, and a 3B parameter model trained across 32 RTX3090s with $(p, t, d) = (4, 2, 4)$. The configurations for the 1.5B and 3B models are set as $(l, h, s, gb, b, m) = (48, 1600, 1024, 512, 32, 8)$ and $(54, 2000, 1024, 1024, 32, 8)$ (notations explained in Table 4), respectively. For the pipeline scheduling, we adopt the "1F1B" scheme [17], an illustration of which is depicted in Fig. 9.

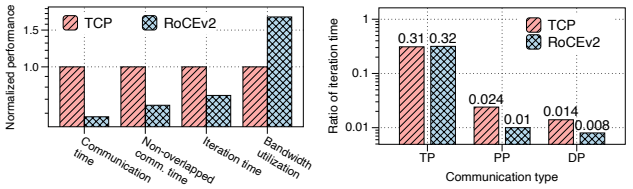
Diverging from pure DP training, GPT models employ the PTD-P training, introducing several distinct communication phases: TP, PP, DP, and bubble time. We utilize the DeepSpeed framework's integrated `communication log` tool to accurately record the execution time of communication operations. Notably, for PP, after a GPU completes its current computation, it initiates a `recv` operation to await the intermediate results of subsequent micro-batch. The `recv` time includes not only active traffic transmission but idle periods as well. Thus, we regard only the active transmission duration of `recv` as PP time, leaving the idle period as bubble time.

Parallelism mapping influences overhead. We assess the influence of parallelism mapping on communication overhead, across two training scales, and with two parallelism mappings: DeepSpeed's default mapping and a customized mapping strategy (see §3.1). As illustrated in Fig. 6, our findings reveal that both the training scale and parallelism mapping affect communication overhead. Specifically, we observe that: (1) TP communication occupies the largest share of communication time (about 50%), attributed to its large volume of traffic; (2) At the 32-GPU scale, DP time experiences a marked increase compared to the 16-GPU scale, due to the expanded DP group size and reduced bandwidth efficiency; (3) Comparing the default and customized mapping strategies, the latter modifies the distribution of traffic on heterogeneous network links, allocating PP communication within machines and DP communication between machines, which results in reduced PP overhead and increased DP overhead.



(a) TCP vs. RoCEv2. (b) Three hardware platforms.

Figure 7: AllReduce benchmarks, assessing the impact of network transport and hardware platform.



(a) VGG16 Training. (b) GPT-3B Training.

Figure 8: Performance of TCP and RoCEv2 in realistic VGG16 and GPT-3B training over V100-PCIe platform.

4.2 Impact of Protocol and Platform

In the preceding experiments, we employ the default configuration: RTX3090 platform and RoCEv2 protocol. Here, we assess how variations in network protocols and hardware platforms influence the communication overhead.

Network protocol. We first compare TCP and RoCEv2 on the V100-PCIe platform using AllReduce benchmarks. We utilize two machines, each with 4 GPUs, forming an 8-worker AllReduce group. The effective bandwidth utilization of the AllReduce operation is shown in Fig. 7a. The results reveal the superior performance of RoCEv2, e.g., offering a 1.8× higher throughput for messages over 10MB compared to TCP. We then assess TCP and RoCEv2 on realistic VGG16 and GPT-3B training. For VGG16, we measure the normalized communication time, iteration time, and bandwidth utilization of RoCEv2 over TCP (Fig. 8a). The results demonstrate that RoCEv2 cuts communication and iteration times by 2× and 1.5×, respectively. For GPT-3B, we measure the ratio of TP, PP, and DP times over iteration (Fig. 8b). As the results show, RoCEv2 achieves a 2.5× and 1.6× reduction on PP and DP communication, respectively. Note that TP is not affected by transport protocols, as its traffic is within intra-machine domains.

Hardware platform. With RoCEv2 set as the protocol, we evaluate the communication time across three hardware platforms. We form an 8-worker AllReduce group and measure the performance of AllReduce benchmarks. As illustrated in Fig. 7b, the V100-NVLink platform obtains the best performance, showing 2.2× and 3.7× higher throughput than V100-PCIe and RTX3090, attributed to its superior interconnection capacity. The difference between RTX3090 and V100-PCIe is due to the `NCCL_P2P` feature being enabled in the V100 but not in the RTX3090. We observe consistent results in the performance of realistic training, here omitted as space limitation.

5 COMMUNICATION ESTIMATION

5.1 Analytical Formulation

The analytical formulation aims to estimate the communication overhead of GPT models. Note that the formulation of pure DP models roughly equals the DP process of GPT models. The used notations are illustrated in Table 4. Our analysis and experiments leverage mixed-precision training, utilizing 16-bit precision for model parameters, activations, and gradients. We begin the analysis with a dissection of the iteration time.

Iteration time. As depicted in Fig. 9, the TP `allreduce` operation is executed multiple times during the processing of each micro-batch. PP `send` and `recv` operations occur at the boundaries of pipeline stages, while the DP `allreduce` operation takes place at

Notation	Explanation
p, t, d	(p, t, d) for the pipeline parallel size, tensor parallel size, and data parallel size, respectively.
N	Total number of model parameters.
l	Number of transformer block layers
h	Hidden size
s	Sequence length
gb, b	Global and micro-batch size, respectively
m	Number of micro-batches per iteration
$C_{TP,PP,DP}$	Effective bandwidth utilization of TP, PP, DP
F	GPU computation capacity (*i.e.*, peak FP16 FLOP/s)

Table 4: Notations used in GPT models' setting.

the end of each iteration. The iteration time is determined by the total working time at $stage_1$, which completes the backward computation of the last micro-batch at the latest. Assuming $rank_0$ is at $stage_1$, the iteration time is the sum of computation, communication (including TP, PP, and DP), and bubble time at $rank_0$:

$$T_{iter} = T_{comp} + T_{TP} + T_{PP} + T_{DP} + T_{bubble} \tag{1}$$

Note that the DP `allreduce` could potentially overlap with backward computation. However, DeepSpeed does not implement this feature in its default PTD-P training. Hence, our analysis assumes no overlap between them.

TP, PP, and DP time. For TP, each `allreduce` operation generates $2bsh$ bytes of traffic within a TP group. If Ring-AllReduce is adopted, each `AllReduce` actually generates $2bsh \times \frac{2(t-1)}{t}$ bytes of traffic. There are four such operations per micro-batch and transformer block. Given that the recomputation (by default enabled) adds two additional `allreduce`, the total comes to six `allreduce` operations per micro-batch and transformer block. With the no. of transformer blocks allocated to one stage being l/p, the TP time per iteration is calculated as follows, where T_{TP}^{mb} represents the TP time for one micro-batch:

$$T_{TP} = m \times T_{TP}^{mb} = m \times \frac{l}{p} \times \frac{6 \times 2bsh \times 2(t-1)}{t \times C_{TP}} \tag{2}$$

For PP, each `send`/`recv` operation transfers $2bsh$ bytes of data between two GPUs in adjacent pipeline stages. At $rank_0$, one `send` and one `recv` operation occur per micro-batch, leading to the following formulation for PP time:

$$T_{PP} = m \times T_{PP}^{mb} = m \times \frac{2 \times 2bsh}{C_{PP}} \tag{3}$$

After completing all backward computations, an `allreduce` is used to aggregate model parameters (16-bit parameter). Assuming a uniform distribution of model parameters N across pipeline stages

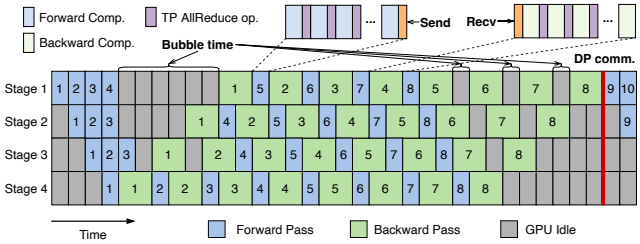


Figure 9: Illustration of pipeline scheduling with 4 stages and 8 micro batches used in this work.

(a simplification, as the actual distribution may slightly vary), the DP time at $rank_0$ is calculated as:

$$T_{DP} = \frac{2N}{p \times t} \times \frac{2(d-1)}{d \times C_{DP}} \tag{4}$$

Bubble time. Considering the computation time for one micro-batch, including both forward and backward passes, as T^{mb}_{comp}, under the "1F1B" scheduling scheme, we can derive the formulation of bubble time at $rank_0$:

$$T_{bubble} = (p-1) \times (T^{mb}_{comp} + T^{mb}_{PP} + T^{mb}_{TP}) \tag{5}$$

Furthermore, the total iteration time can be given by:

$$T_{iter} = m \times (T^{mb}_{comp} + T^{mb}_{PP} + T^{mb}_{TP}) + T_{bubble} + T_{DP}, \tag{6}$$

If T_{DP} is excluded, the ratio of bubble time, R_{bubble}, can be approximated as a constant value:

$$R_{bubble} = T_{bubble}/T_{iter} \approx (p-1)/(p-1+m) \tag{7}$$

Computation time. It is estimated that each model parameter and input token requires roughly eight floating-point operations (FLOPs) for computation, two for the forward pass, two for the recomputation, and four for the backward pass [11, 18]. The detailed rationale behind this approximation is discussed in Appendix C. Adopting this value, the computation requirement for each micro-batch at $rank_0$ is calculated as:

$$FLOP^{mb}_{required} = 8 \times \frac{N}{p \times t} \times b \times s \tag{8}$$

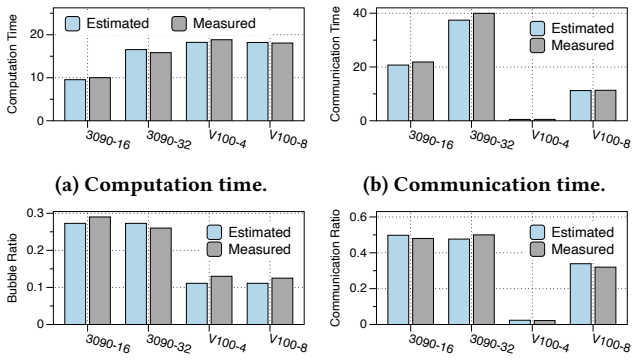
The computation time is then the ratio of required FLOPs to the GPU's capacity (FLOP/s). Given the unattainability of the GPU's peak computational capacity F in practice, a factor μ is introduced to represent the GPU utilization rate. Thus, the computation time per iteration at $rank_0$ is estimated by:

$$T_{comp} = \frac{m \times FLOP^{mb}_{required}}{\mu F} = \frac{8m \times N \times b \times s}{p \times t \times \mu F} \tag{9}$$

5.2 Evaluation of Accuracy

To assess the accuracy of the analytical formulation, we compare its estimations against realistic data derived from our experiments. This evaluation assesses the estimation of computation time (T_{comp}), bubble ratio (R_{bubble}), communication time (T_{comm}), and communication ratio ($R_{comm} = T_{comm}/T_{iter}$).

The evaluation covers four experimental configurations: 16 RTX3090s with a 1.5B GPT model, 32 RTX3090s with a 3B GPT model, 4 V100s with a 1.5B GPT model, and 8 V100s with a 3B GPT model. During our experiments, we observed that the GPU utilization rate,



(a) Computation time. **(b) Communication time.**

(c) Bubble ratio. **(d) Communication ratio.**

Figure 10: Comparision between the analytical model and realistic results over four experiment settings.

μ, remains relatively constant across different training scales but varies depending on the hardware platform. Based on the actual measurement, we apply a μ of 0.3 for RTX3090 and 0.4 for V100. We speculate the distinction is due to V100's larger GPU memory, which enables more model parameters per GPU, thus enhancing the utilization efficiency of GPU's parallel SMs. Existing works [11, 18] mention that A100 could achieve a greater μ. For C_{TP}, C_{PP}, and C_{DP}, given the training scale and hardware platform, these values remain constant and can thus be accurately determined using NCCL micro-benchmarks, as adopted by Fig. 7. Note that for the estimation of R_{bubble}, we adopt the simplified form in Eq. 7.

As illustrated by the results in Fig. 10, the analytical formulation achieves approximately 90% accuracy across the majority of our experiments, demonstrating its capability in estimating the performance characteristics of GPT model training.

Summary. This analytical formulation precisely formulates the traffic volume and required computation FLOPs, offering a mathematical understanding of the predictability characteristic of GPT models. For execution time metrics, it utilizes empirical values, including GPU utilization rate and effective bandwidth utilization.

6 CONCLUSION & FUTURE WORK

In this paper, we experimentally evaluate the influence of various factors on the communication pattern and overhead in distributed training. We present a comprehensive analysis of the characteristics of communication patterns and propose an analytical formulation to estimate communication overhead. For future work, we plan to: (1) broaden our experimental setting to incorporate more advanced GPUs and larger training scales to verify our current findings; (2) explore more domain-specific features; (3) conduct an in-depth dissection for communication overhead, e.g., identifying the factors influencing the effective bandwidth utilization.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their constructive suggestions. The HKUST affiliated authors are supported by the Key-Area Research and Development Program of Guangdong Province (2021B0101400001), the Hong Kong RGC TRS T41-603/20R, the GRF 16213621, the ITF ACCESS, the NSFC 62062005, and the TACC [31]. Kai Chen is the corresponding author.

REFERENCES

[1] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. 2015. {Information-Agnostic} Flow Scheduling for Commodity Data Centers. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 455–468.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[3] NVIDIA Mellanox ConnectX-5. 2020. https://www.nvidia.com/en-us/networking/ethernet/connectx-5/.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[5] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research* 23, 120 (2022), 1–39.

[6] Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A {GPU} cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 485–500.

[7] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS operating systems review* 42, 5 (2008), 64–74.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[9] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of {Large-Scale}{Multi-Tenant}{GPU} clusters for {DNN} training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 947–960.

[10] Zhihao Jia, Matei Zaharia, and Alex Aiken. 2019. Beyond Data and Model Parallelism for Deep Neural Networks. *Proceedings of Machine Learning and Systems* 1 (2019), 1–13.

[11] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems* 5 (2023).

[12] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. 2014. Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems* 27 (2014).

[13] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. 2020. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020).

[14] Wenxue Li, Chaoliang Zeng, Jinbin Hu, and Kai Chen. 2023. Towards Fine-Grained and Practical Flow Control for Datacenter Networks. In *2023 IEEE 31st International Conference on Network Protocols (ICNP)*. IEEE, 1–11.

[15] Wenxue Li, Junyi Zhang, Yufei Liu, Gaoxiong Zeng, Zilong Wang, Chaoliang Zeng, Pengpeng Zhou, Qiaoling Wang, and Kai Chen. 2024. Cepheus: Accelerating Datacenter Applications with High-Performance RoCE-Capable Multicast. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 908–921.

[16] Megatron-DeepSpeed. 2021. https://github.com/microsoft/Megatron-DeepSpeed.

[17] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 1–15.

[18] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. 2021. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

[19] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. 2019. A generic communication scheduler for distributed DNN training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 16–29.

[20] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[21] Sudarsanan Rajasekaran, Manya Ghobadi, Gautam Kumar, and Aditya Akella. 2022. Congestion control in machine learning clusters. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*. 235–242.

[22] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

[23] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[24] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[25] Hao Wang, Yuxuan Qin, ChonLam Lao, Yanfang Le, Wenfei Wu, and Kai Chen. 2023. Preemptive Switch Memory Usage to Accelerate Training Jobs with Shared In-Network Aggregation. In *2023 IEEE 31st International Conference on Network Protocols (ICNP)*. IEEE, 1–12.

[26] Hao Wang, Han Tian, Jingrong Chen, Xinchen Wan, Jiacheng Xia, Gaoxiong Zeng, Wei Bai, Junchen Jiang, Yong Wang, and Kai Chen. 2024. Towards {Domain-Specific} Network Transport for Distributed {DNN} Training. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 1421–1443.

[27] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. 2023. {TopoOpt}: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 739–767.

[28] Yiding Wang, Decang Sun, Kai Chen, Fan Lai, and Mosharaf Chowdhury. 2023. Egeria: Efficient dnn training with knowledge-guided layer freezing. In *Proceedings of the Eighteenth European Conference on Computer Systems*. 851–866.

[29] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, et al. 2023. {SRNIC}: A scalable architecture for {RDMA}{NICs}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1–14.

[30] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. {MLaaS} in the wild: Workload analysis and scheduling in {Large-Scale} heterogeneous {GPU} clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 945–960.

[31] Kaiqiang Xu, Xinchen Wan, Hao Wang, Zhenghang Ren, Xudong Liao, Decang Sun, Chaoliang Zeng, and Kai Chen. 2021. Tacc: A full-stack cloud computing infrastructure for machine learning tasks. *arXiv preprint arXiv:2110.01556* (2021).

[32] Chaoliang Zeng, Xudong Liao, Xiaodian Cheng, Han Tian, Xinchen Wan, Hao Wang, and Kai Chen. 2024. Accelerating Neural Recommendation Training with Embedding Scheduling. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 1141–1156.

[33] Zhen Zhang, Chaokun Chang, Haibin Lin, Yida Wang, Raman Arora, and Xin Jin. 2020. Is network the bottleneck of distributed training?. In *Proceedings of the Workshop on Network Meets AI & ML*. 8–13.

[34] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, et al. 2022. Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 559–578.

[35] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 523–536.

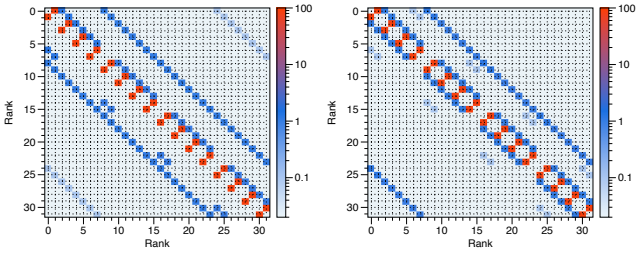
A ADDITIONAL PATTERN ANALYSIS

A.1 Mapping Influences Traffic Pattern

As mentioned in §3, we construct a customized parallelism mapping scheme, whose principle is to allocate TP groups and PP stages within machines and distribute DP groups across machines. The illustration of it from the logical strategy $(p, t, d) = (4, 2, 4)$ to 32 GPUs (4 machines) is shown in Fig. 12. The corresponding traffic heatmap is shown in Fig. 11b. As the results show, the customized parallelism mapping results in a distinct traffic heatmap, compared to the heatmap in Fig. 11a, demonstrating that parallelism mapping influences traffic patterns too.

B MORE ABOUT OVERHEAD

As mentioned in §4, we also evaluate five DP models as listed in Table 2 using the RTX3090 platform and measure their communication time, non-overlapped communication time, and the ratios over end-to-end training. We vary the training scale from 8 GPUs to 32



(a) Default mapping. (b) Customized mapping.

Figure 11: Traffic heatmap of a GPT-3B model with the logical strategy $(p, t, d) = (4, 2, 4)$.

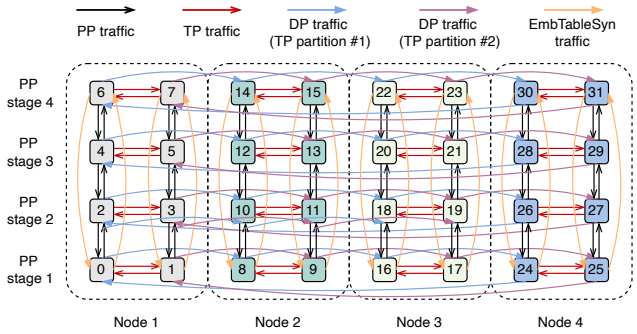
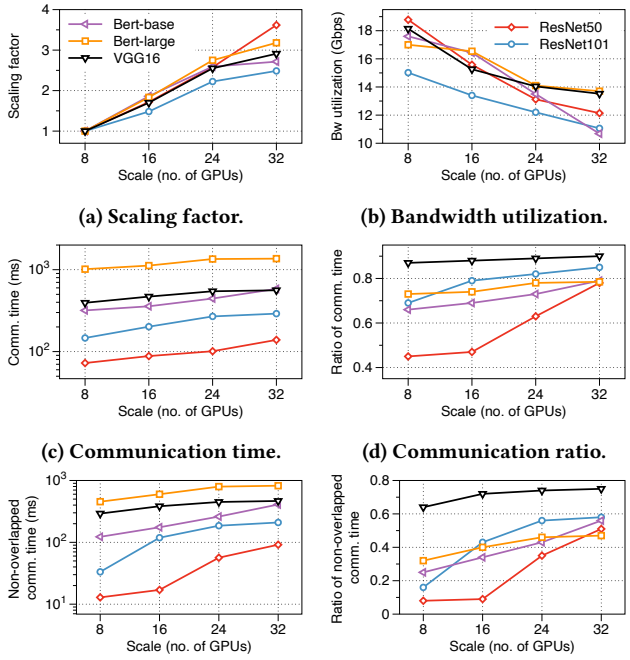


Figure 12: Customized parallelism mapping from the logical strategy $(p, t, d) = (4, 2, 4)$ to 32 GPUs (4 nodes).

GPUs and assess the scaling factor, with results illustrated in Fig. 13. The per-GPU batch size is fixed to 32 for CV models (*i.e.*, ResNet50, ResNet101, and VGG16) and 12 for NLP models (*i.e.*, Bert-base and Bert-large) in all training scales, forming a weak-scaling setting.

Effect of model architecture. The communication overhead is determined by model architecture (see Fig. 13c, 13d, 13e, and 13f). Specifically, VGG16 and ResNet50 are observed with the highest and lowest communication ratios, respectively. This is because VGG16 has FC layers, with a significant amount of parameters to aggregate but relatively low computation requirements, causing a high communication ratio. By contrast, ResNets are constructed using convolutional neural network layers, which are parameter-efficient.

Impact of training scale. Fig. 13a illustrates a constantly decreasing scaling factor with training scales across all models. We observe that, with weak scaling, the computation time across different scales remains almost the same (also mentioned by prior works [33]), while the communication time increases on a larger scale, thus causing a decreasing scaling factor. The increasing communication time is due to the decreasing *effective bandwidth utilization*, as shown in Fig. 13b.



(a) Scaling factor. (b) Bandwidth utilization.

(c) Communication time. (d) Communication ratio.

(e) Non-overlapped communication time. (f) Non-overlapped communication ratio.

Figure 13: Communication overhead of pure DP models with varying training scale.

C MORE ABOUT FORMULATION

C.1 Estimation of FLOP Requirement

As summarized in [18], the number of parameters in a GPT model, N, can be computed as (V is the vocabulary size):

$$N = 12lh^2(1 + \frac{13}{12h} + \frac{V+s}{12lh}) \tag{10}$$

Given batch size b and sequence length s, the number of required FLOPs per iteration, P, can be approximately computed as (recomputation enabled by default):

$$P = 96bslh^2(1 + \frac{s}{6h} + \frac{V}{16lh}) \tag{11}$$

When $h >> s$, the ratio of P over N is about $8bs$, meaning that each model parameter and input token requires roughly eight FLOPs for computation.