JEEVES: The Valet Who Masters the Art of Cross-DC Training Scheduling

Haotian Deng[1], Xuebin Song[2], Menghao Zhang[2], Yuan Yang[1], Mingwei Xu[1]
[1]Tsinghua University [2]Beihang University

Abstract

As model sizes continue to grow and the capacity of a single data center becomes insufficient, training models across multiple data centers efficiently is becoming increasingly important. In this paper, we first show that existing parallelism strategies perform poorly under limited bandwidth and high latency of cross-DC links. To address this, we propose JEEVES, a framework that extends the pipeline parallelism across DCs to minimize iteration time under memory constraints. We identify that the key lies in a good schedule of computation and communication, and propose communication-aware schedule, memory-aware stage division and inter-replica coordinated schedule. Simulations show that JEEVES improves iteration time by up to 43% when training a 175B-parameter model.

CCS Concepts

• **Networks → Network algorithms**.

Keywords

Large scale training, Pipeline parallelism, High performance computing, Cross-DC training

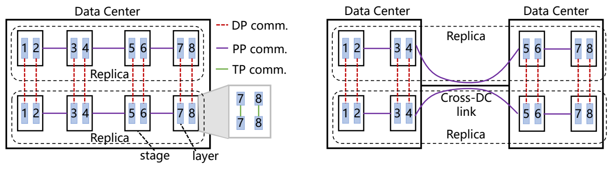
ACM Reference Format:
Haotian Deng, Xuebin Song, Menghao Zhang, Yuan Yang, Mingwei Xu. 2025. JEEVES: The Valet Who Masters the Art of Cross-DC Training Scheduling. In *The 24th ACM Workshop on Hot Topics in Networks (HotNets '25), November 17–18, 2025, College Park, MD, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3772356.3772387

1 Introduction

Large language models (LLMs) [2, 23] have achieved remarkable success recently. Adhering to the scaling law [12], the number of parameters and the size of training datasets are continuously expanding. Consequently, tens of thousands of GPUs are required to train an LLM within a reasonable time [11]. The ever-increasing scale of LLMs makes it infeasible for a single data center (DC) to sustain the training process, due to limitations in power, space, and other factors [4]. Thus, it has become an inevitable trend to combine multiple DCs for LLM training. However, unlike the intra-DC network with abundant bandwidth and extremely low latency, the cross-DC network suffers from relatively limited bandwidth and significantly higher latency. This heterogeneity introduces a new challenge: during the training, data transmission, rather than computation, can become the new bottleneck. The total training time can be substantially prolonged, and computation resources may be wasted while waiting for data transmission. Therefore, an effective parallelism solution for cross-DC LLM training is critically needed.

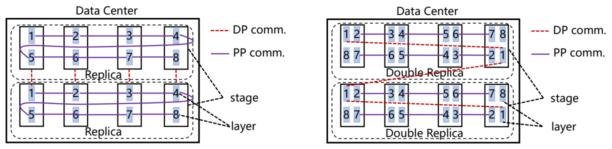
In a DC, GPUs are coordinated under various parallelism strategies, including data parallelism (DP), tensor parallelism (TP), pipeline parallelism (PP), sequence parallelism (SP), expert parallelism (EP), etc., and these strategies are combined to improve training efficiency and reduce training time. Each parallelism strategy introduces different traffic patterns, so when it comes to cross-DC scenarios, careful selection and design of parallelism strategies is crucial to minimize cross-DC traffic and avoid significant communication overhead that could affect training time. We conduct comprehensive simulations to evaluate typical parallelism strategies in a simple two-DC scenario (§2.2). We find that PP, which partitions the model layers into stages and batches into microbatches for pipelining, performs better among all parallelism strategies. However, there still remains a significant performance loss. Deep analysis reveals that the intrinsic reason for this result is the ill-suited scheduling for PP.

Current PP scheduling methods focus solely on computation to determine the execution order of microbatches on each stage [5, 8, 17] (referred to as computation operations hereafter). These methods are designed for intra-DC scenarios, so the communication overhead is largely ignored. However, in cross-DC scenarios, long transmission times cannot be ignored and lead to delayed computation operations. Consequently, computation-only scheduling approaches tend to introduce pipeline bubbles and underutilized compute resources. Furthermore, some studies propose increasing the number of inter-stage communications [14–16, 18] to reduce idle time for intra-DC training, but when directly applied to cross-DC scenarios, these approaches introduce additional



(a) Intra-DC. (b) Inter-DC.

Figure 1: LLM parallelism strategy.



(a) VP placement. (b) DualPipe placement.

Figure 2: VP & DualPipe.

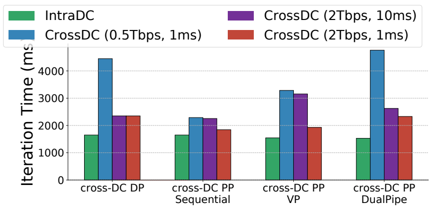


Figure 3: Iteration time when training GPT-3 (175B).

cross-DC communications, which may negate the benefits of finer-grained scheduling and even degrade performance.

To overcome these problems, we propose JEEVES, a framework designed to optimize parallelism and scheduling for cross-DC training. We extend the existing PP scheduling model by defining communication operations across DCs and introducing dependencies between computing operations and communication operations, where the transmission time of each microbatch is estimated based on data size, cross-DC bandwidth, and propagation delay. With this model, we propose an improved PP scheduling algorithm that can adapt to various cross-DC network settings. When executing this algorithm, two factors stand in the way of reducing training time. The first factor is that GPU memory limitations impose a critical constraint on scheduling, further exacerbating the expansion of training time. To address this, we propose a memory-aware stage division to partition model layers, mitigating the training time expansion caused by imbalanced memory consumption under a unified stage division. Another factor is contention for the cross-DC bandwidth among model replicas that expands the transfer time. We design a virtual dependency between communication operations of different replicas and introduce non-blocking sending to reduce the transmission time of successive microbatches.

We implement an operation-level simulator to evaluate JEEVES under varying inter-DC bandwidth, latency, and memory constraints. We simulate a training iteration of a GPT-3 model with 175B parameters using the same parallelism strategy as MegaScale [11]. Evaluation results show that JEEVES achieves iteration time with only a 2% increase compared to intra-DC training, even under low-bandwidth and high-latency inter-DC connections, without increasing memory usage. Compared to the widely adopted 1F1B [5, 18] schedule, JEEVES reduces iteration time by up to 43%.

2 Background and Motivation

2.1 Background

LLM parallelization. Data parallelism (DP) distributes input data across multiple model replicas, with gradient synchronization performed after each iteration. Model parallelism includes inter-layer strategies such as pipeline parallelism (PP), which partitions model layers into stages and splits the input batch into microbatches for pipelined execution, and intra-layer strategies that divide computations within a layer across devices. These include tensor parallelism (TP) for splitting matrix multiplications, sequence parallelism (SP) for dividing operations like LayerNorm and Dropout, and expert parallelism (EP) for distributing MLP computations across multiple experts. Each parallelism strategy leads to distinct communication patterns. In practice, these strategies are combined to scale training [11, 18]. Figure 1(a) shows an example of such combined parallelism and the associated communication.

Cross-DC training. Due to geographical constraints, DCs are often separated by long distances, resulting in inter-DC links with limited bandwidth, typically several terabits per second (Tbps) [1], and relatively high latency, ranging from 0.1 ms to tens of milliseconds [21]. Given the high communication overhead on such links, it is widely accepted that the large volume of traffic induced by intra-layer parallelism, e.g., TP, SP and EP should be confined within the DC [4, 19].

2.2 Extending Parallelism Across DCs

A key challenge in LLM training across DCs is the optimal placement of the multiple model replicas in different DCs. NVIDIA and Google propose to extend DP, distributing different model replicas across DCs, leveraging strategies such as hierarchical all-reduce [19] and federated-style training [3, 4, 24] to reduce the cross-DC traffic introduced by gradient synchronization. Federated-style training lowers synchronization overhead by decreasing the frequency of model updates, but this may adversely affect convergence. Hierarchical all-reduce mitigates cross-DC traffic by aggregating gradients within each DC before global synchronization. However, substantial communication overhead persists, as each iteration still requires transferring a full set of model gradients.

Another approach is to extend PP, co-locating the same layer across all replicas within the same DC, while assigning different layers of a replica to different DCs, as shown in Figure 1(b). In this way, gradient synchronization traffic is entirely confined within each DC, and only the inter-layer activations between adjacent layers need to be transferred across DCs. This approach can reduce the total volume of cross-DC traffic. For example, for a GPT-3 model with 175 billion parameters and a batch size of 3096, gradient synchronization requires transferring at least 175 GB per iteration (assuming 8 bits per gradient), while cross-DC activation traffic amounts to only 75 GB per iteration [11].

We run simulations to compare these two approaches (referred to as cross-DC DP and cross-DC PP), under various inter-DC bandwidth and latency, with the same experimental setup in §4. For cross-DC DP, we adopt hierarchical all-reduce. While no specific strategy for cross-DC PP placements currently exists, various intra-DC placement strategies have been proposed to enhance performance apart from sequentially placing model layers onto multiple devices as in Figure 1(a). These strategies can be broadly categorized into two types: 1) VP placement [6, 16, 18], which forms a tighter pipeline by assigning multiple non-contiguous layer subsets to each stage (Figure 2(a)); 2) DualPipe placement [14, 15], which introduces a bidirectional pipeline by storing a duplicate copy of model parameters in the opposite direction (Figure 2(b)). We extend these strategies to cross-DC training. Figure 3 shows the results for a single iteration. VP and DualPipe placements outperform sequential placement by 6.4% and 7.3% respectively in intra-DC training scenarios, but their performance significantly degrades in cross-DC scenarios. This degradation stems from the additional cross-DC communication they introduce: VP increases cross-DC traffic by introducing more communication across stages, while DualPipe places replicas of the same layer across DCs and incurs extra synchronization overhead. On the other hand, cross-DC PP with sequential placement outperforms cross-DC DP in all bandwidth settings due to the lower cross-DC traffic it generates. Prior work compares cross-DC PP with sequential placement and cross-DC DP with ring all-reduce, and reports similar observations [22]. Beyond iteration time, prior work notes that higher cross-DC traffic can also increase costs, as cloud providers often charge more for cross-DC traffic [22]. Therefore, the cross-DC PP with sequential placement holds more potential for cross-DC training.

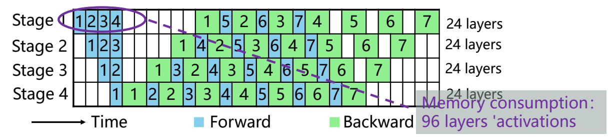
Furthermore, evaluation results show that the iteration time of cross-DC PP with sequential placement increases by up to 38.7% compared to intra-DC training. The key reason behind this lies in scheduling operations within the model pipeline. Figure 4(a) shows an example timeline of the model pipeline during intra-DC training. We visualize the computation operations on each stage, where communication between stages is negligible due to high-bandwidth, low-latency links within DCs, and thus is not visualized. Each microbatch undergoes a forward pass through all stages, followed by a backward pass in the reverse direction. The operations on each stage are well scheduled, minimizing idle time and iteration time. However, when extending to cross-DC scenarios, the timeline can be significantly impacted as cross-DC communication time can be comparable to or even exceed computation time. For example, in GPT-3 training with 64 replicas across a 2 Tbps, 1 ms cross-DC link, each replica receives ~30 Gbps, leading to a microbatch cross-DC transfer (~25MB) of 10 ms, roughly equal to the computation time for forward and backward passes. However, traditional scheduling strategies are designed for intra-DC environments, where communication overhead is minimal. They cannot adaptively change the operation order when faced with such high communication overhead. Figure 4(b) illustrates the resulting timeline when applying the same schedule to cross-DC scenarios without modification. Besides computation operations on each stage, we visualize cross-DC communication in both directions. The idle time on each stage increases, wasting computational resources and extending the iteration time. To mitigate this, a scheduling strategy that adapts to high cross-DC communication overhead is in urgent need.

3 Problem Analysis & JEEVES Design

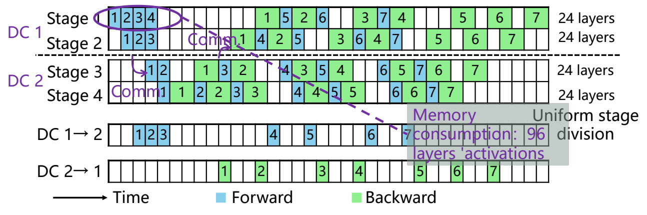
We present JEEVES, an framework for optimizing the schedule of PP for cross-DC training. We identify three limitations of extending the intra-DC scheduling strategies across DCs and propose corresponding solutions: (1) Unaware of communication overhead: we propose a scheduling algorithm that optimizes iteration time under varying cross-DC link conditions (§3.1). (2) Uneven memory consumption: we introduce an uneven stage division method to balance memory overhead across stages, achieving a good trade-off between iteration time and memory consumption (§3.2). (3) Replicas' competition on inter-DC bandwidth: we propose an inter-replica coordinated scheduling strategy (§3.3).

3.1 Comm-aware Schedule

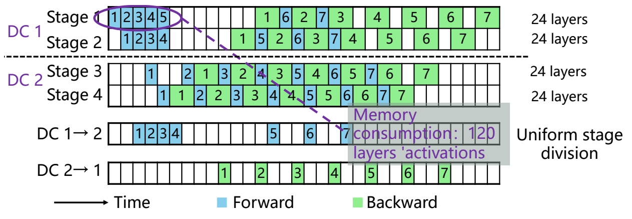
Following the usual workflow, before executing the pipeline, each stage is initialized with an ordered list of computation operations and an ordered list of communication operations, where one operation is executed at a time in sequence. Each operation depends on one or more preceding operations and becomes executable once all its dependencies are resolved. Dependencies are typically introduced to enforce the data dependencies for a microbatch's operations. Specifically, computation operations depend on previous activations, and output activations are sent after completion. During execution, after the forward computation of each microbatch on



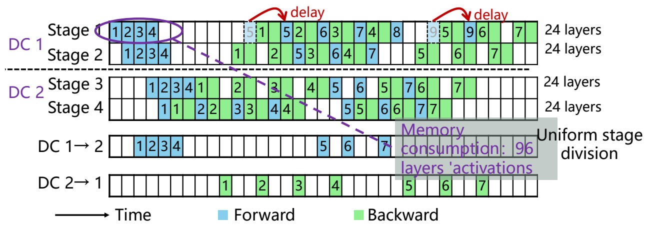
(a) Baseline (Intra-DC).



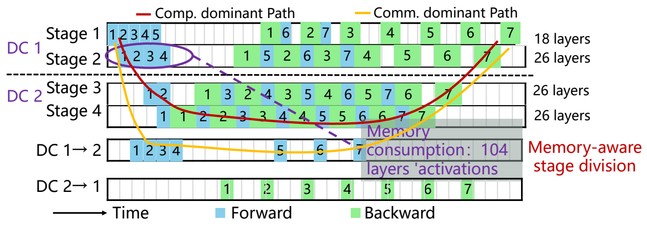
(b) Cross-DC Naive Extension.



(c) Comm-Aware Schedule.



(d) Comm-Aware Schedule with Tight Memory Constraints.

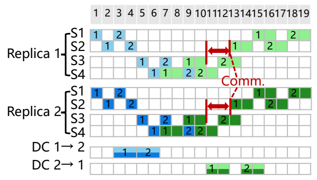


(e) Comm-Aware Schedule with Memory-Aware Division.

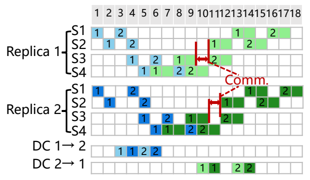
Figure 4: Stepwise Optimization for Cross-DC Training.

each stage, the resulting activation must be retained for the backward pass, consuming substantial memory [14].

Challenge. The goal of scheduling is to assign the order of operations within each list to minimize iteration time. There are two main challenges for the scheduling algorithm: (1) A schedule may perform differently under various cross-DC link bandwidth and latency, so the algorithm must be adaptable. (2) If memory consumption exceeds the limits during execution, an out-of-memory error will occur. Therefore, the algorithm must ensure that memory consumption stays within the constraints for each stage.



(a) Replicas compete for bandwidth.



(b) Inter-replica coordinated communication.

Figure 5: Handling different replicas' communication.

Strawman solution. For intra-DC training, rule-based approaches are commonly used but are ineffective in addressing the challenges. There are two types of methods: one employs the 1F1B strategy [5, 6, 8, 18], each stage begins with a fixed number of forward computations followed by alternating forward and backward operations. However, the appropriate number of forward computations varies for each network condition, making it hard to generate an effective schedule in cross-DC scenarios. The other uses the Forward-first strategy [10, 13], which executes all forward computations first for each stage. While this method minimizes idle time, it results in higher memory usage.

Our approach. For the first challenge, we identify that idle time mainly arises from blocking caused by the predefined operation order. Therefore, our solution is to initially avoid specifying the operation list. Instead, we simulate execution along a timeline without pre-assigned lists. An operation can be executed as soon as its dependencies are satisfied and the corresponding stage is idle. And we gather the order of operations during the simulation as output schedule.

The greedy method can only reduce idle time but cannot guarantee memory consumption constraints. To optimize memory usage, we introduce two additional rules. First, to prevent exceeding memory limits during execution, we maintain a memory counter for each stage, which increases after executing a forward computation and decreases after executing a backward computation. If the counter exceeds the memory limit, the forward operation is blocked and will not be scheduled until the counter decreases. Second, since the forward and backward passes go in opposite directions, there are cases where forward and backward operations from different microbatches are ready simultaneously. The choice of which to execute first affects memory usage. We define a simple but effective rule: prioritize executing backward operations. This ensures earlier release of memory on the current stage and allows subsequent backward operations to begin sooner, freeing up memory for future stages.

Figure 4(c) and Figure 4(d) show examples of generated schedules with different memory constraints, where each stage can accommodate a maximum of five and four microbatches' activations, respectively. The schedule adapts to different constraints and reduces iteration time compared to

Figure 4(b). Note that on stage 3 while the forward computations of microbatch 3 and 4 can be executed, the backward computation of microbatch 1 is executed first, allowing its backward computation at stage 1 to start earlier and release memory, enabling the computation of microbatch 6 or microbatch 5 to begin as soon as possible.

3.2 Memory-aware Stage Division

Comparing Figure 4(a) and 4(c), we observe that cross-DC training results in higher peak memory consumption than intra-DC training. This is because long cross-DC communication delays backward computations, which in turn delay memory release (e.g., microbatch 1's backward on stage 1). When memory consumption exceeds the constraint, the schedule must be adjusted to delay some forward computations. As shown in Figure 4(d), microbatch 5's forward computation is delayed until microbatch 1's backward computation completes, increasing iteration time. To mitigate this, we reduce peak memory consumption by balancing memory across stages. Since memory consumption is proportional to the number of layers and earlier stages typically consume more, we can move layers from earlier to later stages. Figure 4(e) shows an example where 18 layers are placed in the first stage, and the remaining layers are evenly distributed across the other stages. With exact the same schedule, peak memory consumption is reduced from storing activations of 120 layers (5 microbatches) in stage 1 to 104 layers (4 microbatches) in stage 2.

Challenge. Since microbatch computation time is also proportional to the number of layers per stage, uneven stage division improves memory balance at the cost of longer computation time on the largest stage. Since pipeline iteration time is largely determined by the slowest stage, this also increases overall iteration time. Thus the challenge is how to balance the trade-off between the extra iteration time introduced by uneven stage division and memory constraints.

Our approach. We model the problem of finding the stage division that minimizes iteration time as an optimization problem, which takes each stage's layer number as a variable. The key lies in formulating the objective, i.e., the iteration time of an ideal schedule under memory constraints, given the stage division. We divide the objective into two parts, i.e., the ideal iteration time without memory constraints and the extra time introduced by memory constraints, and sum these two parts accordingly. The variables represent the number of layers assigned to each stage, whose sum equals the total number of layers in the model (on the order of 100). The number of variables equals the PP dimension (on the order of 10), making the problem small-scale and solvable for the optimal solution within a short time. Currently, we solve this problem with a commercial solver, i.e., gurobi [7], as the stage number is typically small and thus can be solved timely. We leave exploring more efficient methods for future work.

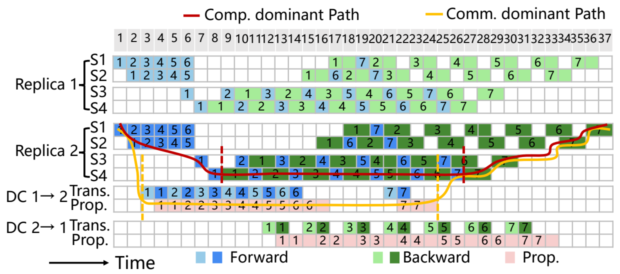


Figure 6: JEEVES's schedule (uniform stage division).

The ideal iteration time without memory constraint is determined by the critical path, i.e., a sequence of operations that defines the lower bound of iteration time. In cross-DC training, we observe two such paths. The first is the computation-dominant path (red line in Figure 4(e)), driven by the total computation time on the slowest stage. This path includes the forward and communication operations of the first microbatch on preceding stages, the forward and backward computations of all microbatches on the slowest stage, and the backward computation and communication of the last microbatch on succeeding stages. The second is the communication-dominant path (yellow line), determined by the total time to transfer all microbatch's activations across DCs. This path includes the computation of the first microbatch in the first DC, the transmission of all microbatches' activations, and the computation and communication of the last microbatch. The ideal iteration time is the longer of the two, assuming the ideal schedule without memory constraints avoids extra idle time.

As shown in Figure 4(d), the extra time introduced by memory constraints consists of two parts. First, the delay from waiting for the earliest memory release (e.g., microbatch 5's delay), which can be calculated using the latest start time of each microbatch on stage 1 without introducing idle time on the last stage and the completion time of microbatch 1's computation. Second, the accumulated delay caused by delayed backward computations, such as microbatch 9's forward is delayed due to waiting for microbatch 5's backward to release memory. This delay accumulates periodically, with the period depending on how many microbatches' activations can fit in stage 1 (in Figure 4(d), the period is 4 microbatches).

3.3 Inter-replica Coordinated Schedule

During intra-DC training, model replicas are typically placed on different devices, and the DC ensures sufficient bandwidth between any two points. As a result, there is no competition for compute or communication resources between replicas. In contrast, during cross-DC training, replicas must share limited cross-DC link bandwidth resources.

Strawman solution. Each replica typically schedules its own computation and communication operations independently for intra-DC training. When this is extended to cross-DC, the communication between replicas competes for bandwidth, leading to increased transfer time for each microbatch's activation, as shown in Figure 5(a).

Our approach. To avoid contention, we propose a coordinated scheduling of communication operations across replicas. Instead of transferring different replicas' microbatches simultaneously, we use the full bandwidth to transfer one microbatch at a time, thereby reducing transfer time and enabling earlier start of subsequent computations for that microbatch. We then alternate the transmission between replicas, sending one microbatch from each replica at a time. This approach ensures that, apart from the first microbatch of each replica waiting for the previous replica's transfer to complete, the remaining communications start at different time, thus avoiding waiting or competition. To implement this strategy, we introduce a global communication list containing all cross-DC communications to ensure sequential execution of communication operations. As illustrated in Figure 5(b), as replica 1's microbatch 1 is sent from DC1 to DC2 at full speed, the subsequent communication from DC2 to DC1 for this microbatch starts earlier than microbatch 1 of replica 2, thus both messages transmit at full speed.

Another issue is about transmission of adjacent messages. Typically, adjacent messages are sent in a blocking manner during training, with the next message only sent after receiving the previous message's acknowledgment (ACK). This has minimal overhead inside DCs with microsecond latency but causes significant delays across DCs with millisecond latency. To address this, we propose non-blocking transmission. Once the last packet of the previous message is sent, the system immediately signals the next replica without waiting for ACK[1]. The signal only needs to be transmitted within the DC, resulting in minimal delay. The next replica starts transmission upon receiving the signal.

3.4 Putting It All Together

Given the training settings, including the number of replicas, number of model stages, computation time for each layer, and cross-DC link bandwidth and latency, JEEVES first determines the stage division based on the memory-aware stage division. Then, the inter-replica coordinated schedule generates the global communication operation list and the comm-aware schedule generates the operation list for each stage. Figure 6 shows an example. The computation-dominant path is minimized as no extra idle time is introduced along this path, while the communication-dominant path is minimized as almost no propagation delay is introduced.

[1]In this work, we assume a lossless network. However, due to potential packet losses, retransmissions from previous messages may overlap with the transmission of new ones. We leave the analysis of whether to prioritize retransmissions or transmit both concurrently as future work.

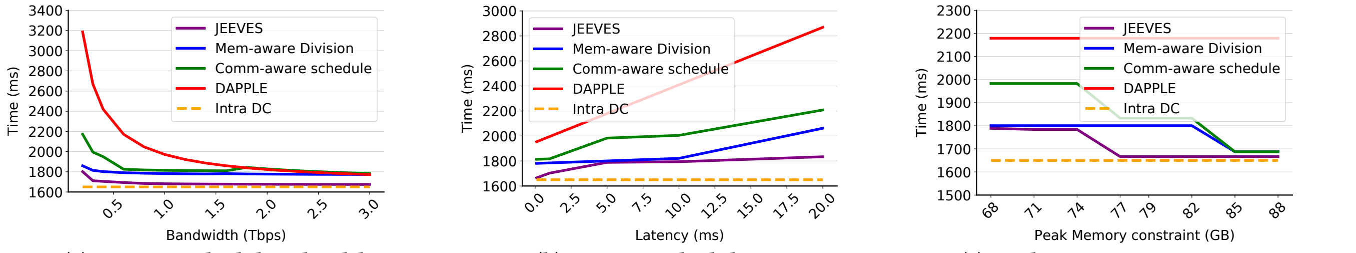
4 Evaluation

We evaluate JEEVES through simulations, modeling the duration of computation operations based on the profiled results, and calculating the duration of communication operations using the message, bandwidth and latency via the $\alpha-\beta$ model [9]. We choose the GPT-3 model with 175B parameters, hidden size 12288, sequence length 2048, and 96 layers. We extend the practical parallelism strategies used in Megascale for intra-DC training to cross-DC training [11]. We employ 64 model replicas, each consisting of 8 stages, with each stage divided across 8 GPUs using TP. For pipeline parallelism, we adopt the sequential placement to replace Megascale's VP placement. We set a batch size of 2072 and a microbatch size of 1. For cross-DC links, we set the default bandwidth as 1Tbps and latency as 1ms.

We compare JEEVES with the following baselines: 1) DAPPLE [5]: SOTA intra-DC schedule for sequential pipeline placement, extended to cross-DC. 2) Intra-DC: Intra-DC training with the same GPU count and parallelism strategies. 3) Comm-aware schedule: JEEVES's variant with comm-aware scheduling, excluding memory-aware stage division and intra-replica coordination. 4) Mem-aware schedule: JEEVES's variant with comm-aware scheduling and memory-aware stage division, excluding intra-replica coordination.

Figure 7 (a) shows results with various cross-DC link bandwidth. As bandwidth increases, the iteration time for cross-DC training decreases, with JEEVES consistently outperforming others at all levels. At 1 Tbps, JEEVES improves performance by 17.6% compared to DAPPLE, and only takes 2% more time than intra-DC training. As the bandwidth increases, the iteration time is mainly influenced by the computation-dominant path, and is less affected by bandwidth. When the bandwidth is lower, the iteration time is mainly influenced by the communication-dominant path. In this case, each of JEEVES's components contributes significantly to the overall performance improvements.

Figure 7 (b) shows that latency significantly affects iteration time in cross-DC training, as each microbatch requires two rounds of cross-DC communication, resulting in a high number of messages. JEEVES benefits from the non-blocking transmission, which prevents the cumulative latency from different messages. Although latency's impact cannot be fully eliminated, JEEVES prevents iteration time from increasing linearly with latency. With a latency of 20 ms, JEEVES reduces iteration time by 36.7% compared to DAPPLE, with only an 11% increase compared to intra-DC training.



(a) Cross-DC link bandwidth **(b) Cross-DC link latency** **(c) Peak memory consumption**

Figure 7: Simulations on an iteration of training a GPT-3 (175B) model with 4096 GPUs.

Figure 7 (c) shows the performance under different memory limits. The DAPPLE method does not adaptively adjust the schedule as memory capacity increases, resulting in no performance improvement. In contrast, JEEVES can fully utilize the available memory within a certain range through better scheduling. When memory exceeds a certain threshold, computation speed becomes the primary bottleneck, and performance no longer improves.

5 Conclusion and Future Work

In this paper, we identify the limitations of current cross-DC training and introduce JEEVES, a framework for scheduling pipeline parallelism across DCs. We present preliminary evaluations under varying bandwidth, latency, and memory constraints, demonstrating its promising performance. In our ongoing work, we plan to enhance the design, build a prototype based on the mainstream framework [18, 20], extend JEEVES to multiple DCs under network fluctuations, and conduct extensive experiments. We hope JEEVES offers valuable insights for cross-DC training.

Acknowledge

We thank our shepherd and the anonymous HotNets reviewers for their valuable comments. This work is supported in part by the National Natural Science Foundation of China (No. 62402025), the Fundamental Research Funds for the Central Universities, and the National Key Research and Development Program of China under Grant 2022YFB2901300. Menghao Zhang, Yuan Yang and Mingwei Xu are the corresponding authors.

References

[1] Dacoso GmbH. n.d.. *Managed DCI — High Performance Connectivity Between Data Centers.* Accessed: 2025-06-22.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

[3] Arthur Douillard, Yanislav Donchev, Keith Rush, Satyen Kale, Zachary Charles, Zachary Garrett, Gabriel Teston, Dave Lacey, Ross McIlroy, Jiajun Shen, et al. 2025. Streaming diloco with overlapping communication: Towards a distributed free lunch. *arXiv preprint arXiv:2501.18512* (2025).

[4] Arthur Douillard, Qixuan Feng, Andrei A Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. 2023. Diloco: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105* (2023).

[5] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, et al. 2021. DAPPLE: A pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 431–445.

[6] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[7] Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[8] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. 2018. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377* (2018).

[9] Roger W Hockney. 1994. The communication challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel computing* 20, 3 (1994), 389–398.

[10] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems* 32 (2019).

[11] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. 2024. {MegaScale}: Scaling large language model training to more than 10,000 {GPUs}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 745–760.

[12] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[13] Joel Lamy-Poirier. 2023. Breadth-first pipeline parallelism. *Proceedings of Machine Learning and Systems* 5 (2023), 48–67.

[14] Shigang Li and Torsten Hoefler. 2021. Chimera: efficiently training large-scale neural networks with bidirectional pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.

[15] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, and .... 2024. DeepSeek-V3 Technical Report. *arXiv preprint arXiv:2412.19437* (Dec. 2024). Technical Report.

[16] Ziming Liu, Shenggan Cheng, Haotian Zhou, and Yang You. 2023. Hanayo: Harnessing wave-like pipeline parallelism for enhanced large model training efficiency. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

1–13.

[17] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. 2021. Memory-efficient pipeline-parallel dnn training. In *International Conference on Machine Learning*. PMLR, 7937–7947.

[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] NVIDIA. 2024. *Turbocharge LLM Training across Long-Haul Data Center Networks with the NVIDIA NeMo Framework*. https://developer.nvidia.com/blog/turbocharge-llm-training-across-long-haul-data-center-networks-with-nvidia-nemo-framework/

[20] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 3505–3506.

[21] Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, et al. 2020. Annulus: A dual congestion control loop for datacenter and wan traffic aggregates. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 735–749.

[22] Foteini Strati, Paul Elvinger, Tolga Kerimoglu, and Ana Klimovic. 2024. ML training with Cloud GPU shortages: Is cross-region the answer?. In *Proceedings of the 4th Workshop on Machine Learning and Systems*. 107–116.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[24] Ying Zhu, Yang Xu, Hongli Xu, Yunming Liao, Zhiwei Yao, and Liusheng Huang. 2025. Cross-region Model Training with Communication-Computation Overlapping and Delay Compensation. *arXiv preprint arXiv:2504.17672* (2025).