Coflow Scheduling for LLM Training

Xinchen Wan
Hong Kong University of Science and Technology
Hong Kong, China
xinchen.wan@connect.ust.hk

Xinyu Yang
Hong Kong University of Science and Technology
Hong Kong, China
xinyu.yang@connect.ust.hk

Kaiqiang Xu
Hong Kong University of Science and Technology
Hong Kong, China
kxuar@cse.ust.hk

Xudong Liao
Hong Kong University of Science and Technology
Hong Kong, China
xliaoaf@connect.ust.hk

Yilun Jin
Hong Kong University of Science and Technology
Hong Kong, China
yilun.jin@connect.ust.hk

Yijun Sun
Hong Kong University of Science and Technology
Hong Kong, China
ysuneb@connect.ust.hk

Zhenghang Ren
Hong Kong University of Science and Technology
Hong Kong, China
zrenak@cse.ust.hk

Han Tian
University of Science and Technology of China
Hefei, China
henrytian@ustc.edu.cn

Kai Chen
Hong Kong University of Science and Technology
Hong Kong, China
kaichen@cse.ust.hk

Abstract

Training large language models (LLMs) generates diverse coflows within a cluster, requiring optimized scheduling to enhance communication-computation overlap and minimize training time. Existing schedulers inadequately handle contention both across and within coflows, resulting in suboptimal performance.

We present Hermod, a comprehensive coflow scheduler that orchestrates all coflow types for LLM training. The key insight behind Hermod is that coflows can be uniquely characterized by three model factors—microbatch ID, coflow type, and layer ID—enabling optimal scheduling decisions. Leveraging this insight, Hermod applies model-factor–driven inter-coflow priority scheduling aligned with the LLM training DAG. Preliminary simulation results show potential for performance improvements.

CCS Concepts

• **Networks** → **Cloud computing**; **Data center networks**.

Keywords

Coflow scheduling, Large language model

ACM Reference Format:
Xinchen Wan, Xinyu Yang, Kaiqiang Xu, Xudong Liao, Yilun Jin, Yijun Sun, Zhenghang Ren, Han Tian, and Kai Chen. 2025. Coflow Scheduling for LLM Training. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3718958.3750467

SIGCOMM '25, September 8–11, 2025, Coimbra, Portugal

ACM ISBN 979-8-4007-1524-2/2025/09…$15.00
https://doi.org/10.1145/3718958.3750467

1 Introduction

Large language models (LLMs) have achieved impressive results across tasks such as machine translation, code generation, and dialogue systems [3, 5, 13]. However, training these models remains resource-intensive due to their massive scale and complexity. To meet computational and memory demands, LLM training is distributed across thousands of accelerators, requiring coordinated parallelization strategies for efficient scalability [6, 9, 15].

Training LLMs involves multiple forms of parallelism, including data, tensor, sequence, expert, and pipeline parallelism. Each introduces distinct communication patterns—e.g., `AllReduce` and `AllGather` for data and tensor parallelism (DP/TP) [8, 14, 16, 17, 19, 20], `AlltoAll` for expert parallelism (EP) [10, 11], and `send/recv` for pipeline parallelism (PP) [7, 12]—modeled as coflows within the network [4]. Optimizing these coflows is essential for minimizing job completion time (JCT) and ensuring efficient communication-computation overlap.

Coflow schedulers aim to improve JCT by reordering communications based on training data dependencies. Approaches such as ByteScheduler [14] and Lina [10] have shown promise by optimizing specific coflow types: ByteScheduler improves data parallelism via layer-wise DP coflows overlap, while Lina prioritizes EP coflows over DP coflows in backward passes. These techniques, though effective within their scope, address only subsets of the communication landscape in LLM training.

Despite their benefits, existing schedulers [10, 14] are limited by their local, endhost-focused scope and fail to holistically coordinate coflows across the entire cluster. They also overlook PP, whose coflows introduce strict sequential dependencies that, if unaccounted for, hinder overall throughput. Furthermore, current schedulers neglect flow-level imbalances within coflows, leading to internal bottlenecks.

This work addresses this challenge by introducing Hermod, a comprehensive coflow scheduler for LLM training. Through an in-depth analysis of the training DAG, we identify three model-specific

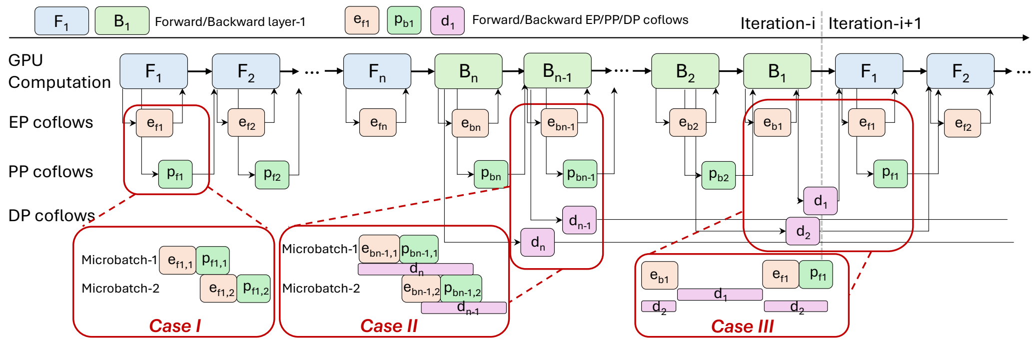


Figure 1: Computation and coflows in LLM training.

	Case I	*Case II*	*Case III*
Priority factor	MicrobatchID (Lower is better)	Coflow type (EP & PP > DP)	LayerID (Lower is better)

Table 1: Priority factors used to assign priority for three cases.

factors—microbatch ID, coflow type, and layer ID—that directly inform scheduling priorities. Hermod leverages these factors to enforce inter-coflow prioritization aligned with the DAG, enabling comprehensive cluster-wide coordination across all communication patterns. The remainder of this paper presents the design of Hermod (§2) and evaluates its effectiveness (§3).

2 Hermod Design

2.1 LLM Training DAG with Coflows

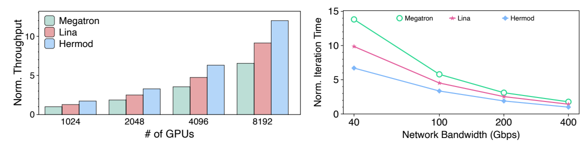
We revisit the LLM training DAG from a coflow scheduling perspective, focusing on optimizing coflow overlaps across inter-host parallelisms—EP, PP, and DP—to minimize job completion time (JCT). Our goal is to derive a cluster-wide scheduling policy that aligns network resource allocation with training dependencies.

In typical LLM training, the DAG is fixed by the topology, parallelization strategy, and worker placement. This DAG dictates coflow dependencies across iterations. Figure 1 illustrates a conventional DAG for training an MoE model with ZeRO-2 and 1F1B optimizations. Each iteration consists of forward propagation (FP) and backward propagation (BP). Within each iteration, EP coflows occur twice per MoE layer group for token dispatch and gathering, followed by PP coflows to transmit intermediate results between layers [18]. These operations repeat across layers and microbatches. During BP of the current iteration and FP of the next, DP coflows perform `ReduceScatter` and `AllGather` for gradient synchronization and parameter updates [9, 14, 16, 24].

A systematic analysis of the DAG identifies three key cases of coflow overlap, marked by red rectangles in Figure 1: (I) EP–PP overlaps across microbatches, (II) EP, PP, and DP overlaps during BP, and (III) cross-iteration overlaps between FP EP/PP coflows and BP DP coflows. These overlaps underscore the need for fine-grained coflow prioritization to mitigate resource contention and inefficient communication ordering.

2.2 Model Factor-Driven Priority Assignment

From the above cases, we identify three critical model factors—microbatch ID (MID), coflow type (CType), and layer ID (LID)—that uniquely characterize and prioritize coflows. These factors are application-defined, static throughout training, and easily encoded for transport-



(a) Scalability **(b) Different bandwidths**

Figure 2: Simulation results when training Qwen-MoE.

layer scheduling. We propose a model factor–driven scheduling strategy that assigns priorities to minimize critical-path blockage and maximize communication–computation overlap.

As summarized in Table 1, coflows with smaller MIDs receive the highest priority, followed by CType (EP/PP over DP) and then LID (lower layers prioritized). This ordering favors earlier microbatches and time-critical coflows, ensuring efficient training progression. The strict priority hierarchy is enforced cluster-wide across all iterations. We also analyze all potential conflicts to maintain consistency , resulting in a deterministic, conflict-free priority scheme across the cluster.

3 Evaluation And Future Work

Simulation Setup. We build our simulator on FlexFlow [1] and htsim [2], following the methodology of [21]. We extend FlexFlow to generate DAGs representing both computation and communication for all coflow types. Simulations use Qwen1.5-MoE [23] as the default model, comparing Hermod with Megatron and Lina [10]. Unless otherwise noted, we simulate a cluster of 8-GPU servers connected via a full-bisection fat-tree with 100Gbps links and 1μs propagation delay.

Scalability. We evaluate scalability by increasing cluster size from 1,024 to 8,192 GPUs. As shown in Figure 2a, Megatron suffers from poor throughput due to uncoordinated coflow handling. Lina improves on this with endhost prioritization but overlooks PP coflows. Hermod achieves up to 1.3× higher throughput by performing cluster-wide coflow scheduling.

Network Bandwidth. We assess performance under varying bandwidths (40–400Gbps), shown in Figure 2. Hermod consistently outperforms baselines. At 40Gbps, Hermod yields up to 1.78× speedup over Megatron via coordinated prioritization. At 400Gbps, network bottlenecks diminish, but Hermod still provides up to 2× higher throughput by mitigating EP/PP blocking and enabling DP overlap.

Future Work. We plan to extend Hermod to better resolve intra-coflow contention (*e.g.*, imbalanced EP coflows) and validate its effectiveness in real-world LLM training under larger clusters [22].

References

[1] Flexflow source code. https://github.com/flexflow/FlexFlow, 2024.

[2] htsim packet-level simulator. https://github.com/nets-cs-pub-ro/NDP/wiki/NDP-Simulator, 2024.

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

[4] Jiamin Cao, Yu Guan, Kun Qian, Jiaqi Gao, Wencong Xiao, Jianbo Dong, Binzhang Fu, Dennis Cai, and Ennan Zhai. Crux: Gpu-efficient communication scheduling for deep learning training. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 1–15, New York, NY, USA, 2024. Association for Computing Machinery.

[5] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[6] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, et al. Rdma over ethernet for distributed training at meta scale. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 57–70, 2024.

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

[8] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed {DNN} training in heterogeneous {GPU/CPU} clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 463–479, 2020.

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

[10] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed {MoE} training and inference with lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 945–959, 2023.

[11] Juncai Liu, Jessie Hui Wang, and Yimin Jiang. Janus: A unified distributed training framework for sparse mixture-of-experts models. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 486–498, 2023.

[12] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM symposium on operating systems principles*, pages 1–15, 2019.

[13] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

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

[15] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, et al. Alibaba hpn: A data center network for large language model training. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 691–706, 2024.

[16] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[17] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[18] Xinchen Wan, Kaiqiang Xu, Xudong Liao, Yilun Jin, Kai Chen, and Xin Jin. Scalable and efficient full-graph gnn training for large graphs. *Proceedings of the ACM on Management of Data*, 1(2):1–23, 2023.

[19] Xinchen Wan, Hong Zhang, Hao Wang, Shuihai Hu, Junxue Zhang, and Kai Chen. Rat-resilient allreduce tree for distributed machine learning. In *Proceedings of the 4th Asia-Pacific Workshop on Networking*, pages 52–57, 2020.

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

[21] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. {TopoOpt}: Co-optimizing network topology and parallelization strategy for distributed training jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 739–767, 2023.

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

[23] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[24] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P Xing. Poseidon: An efficient communication architecture for distributed deep learning on {GPU} clusters. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 181–193, 2017.