

A Distributed Training Adaptive Parallel Strategy Generation Algorithm for Multi-DCs in the MAN

Yunuo Yang* **, Rui Han* **, Botao Ma*

*National Network New Media Engineering Research Center, Institute of Acoustics,
Chinese Academy of Sciences, Beijing, China

**School of Electronic, Electrical and Communication Engineering,
University of Chinese Academy of Sciences, Beijing, China
yangyn@dsp.ac.cn, hanr@dsp.ac.cn, mabt@dsp.ac.cn

Abstract—The challenges of distributed training across data centers (DCs) in metropolitan area network (MAN) is under-explored: prevailing pipeline-parallel (PP) methods assume single-data-center(single-DC), homogeneous and high-bandwidth low-latency clusters with full resource visibility, thus misfit cross-data-center (cross-DC) constraints of privacy and high-latency links. We present Chain-based Dynamic Programming (CDynP), a privacy-preserving planner that passes compressed state summary (SS) along a data-center chain to co-optimize model partitioning and device mapping for minimal iteration time without exposing any internal information. In cross-DC evaluations, CDynP limited iteration-time increase to 8.3% when scaling from one to four DCs—substantially less than competing baselines — highlighting its adaptability. Across larger-scale training tasks and degraded network conditions, it consistently achieved top performance, and its search was both rapid and high-quality, underscoring robustness and scalability. These results establish CDynP as a practical, privacy-compliant approach for leveraging dispersed compute islands in MAN for large-scale AI training.

Keywords— Adaptive Parallel Strategy, Multi-DC Collaboration, MAN, Pipeline Parallelism, Privacy-Preserving Optimization

I. INTRODUCTION

In recent years, advances in generative artificial intelligence (AI) have driven a sharp escalation in global computing demand. Computational requirements for training state-of-the-art (SOTA) models are increasing at a steep rate [1], resulting in acute supply shortages across cloud platforms [2]. Scaling within individual DCs is hindered by constraints in energy, cooling, and physical space.

Meanwhile, global compute capacity remains fragmented; numerous small and medium-sized DCs hold idle resources but are geographically dispersed, forming isolated “compute islands.” MANs, delivering millisecond latency and up to 11.3 GB/s throughput within regions [3], present a potential unifying backbone for multi-DC collaborative training.

Among the various parallel paradigms in distributed training, PP is recognized as the most suitable infrastructure for cross-data-center (Inter-DC) collaborative training due to

its low communication volume and inherent support for the overlap of computation and communication [2]-[5].

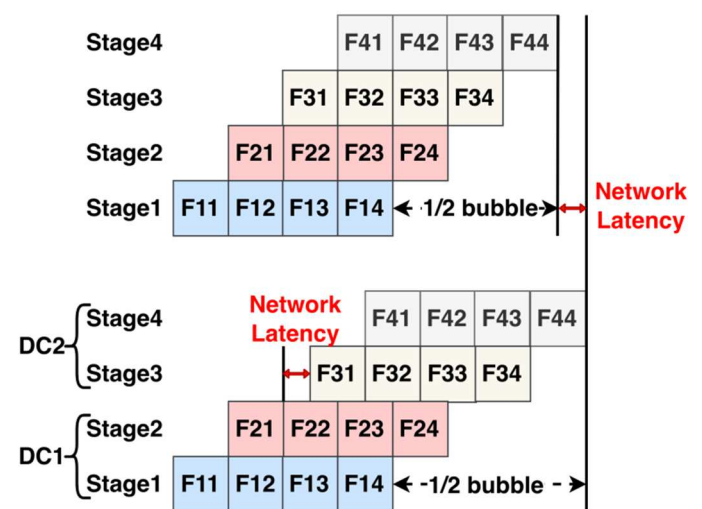


Figure 1. The Inter-DC latency magnifies the bubble.

PP divides the model into a series of consecutive stages and interweaves micro-batch communication with computation (Figure 1). However, designing efficient PP strategies within MANs faces two major challenges. First, a privacy and resource visibility conflict exists: DCs conceal detailed topology and workloads, sharing only coarse-grained metrics such as available GPUs. Consequently, it becomes impossible to derive the parallel strategy that achieves the highest training efficiency, as decisions are made based on incomplete information. Second, MAN latency-although low-are significantly higher than intra-data-center (intra-DC) interconnects. This latency magnifies idle time nonlinearly, degrading throughput, as shown in Figure1.

However, most existing algorithms assume homogeneous high-speed clusters with transparent resource visibility, making them unsuitable for privacy-constrained Inter-DC environments, often leading to suboptimal per-iteration training time. To address this gap, this work proposes an automated method for near-optimal parallel strategy

generation in MAN-scale multi-DC training, explicitly targeting the minimization of per-iteration training time.

We first formulate a mathematical optimization model integrating PP and data parallelism (DP), distinguishing intra- and inter-DC communication costs. Building on this, we introduce CDynP, a decentralized algorithm that conducts local optimizations via SS inheritance from prior hops, ensuring no internal data exposure. To validate the effectiveness of the proposed method, we develop a simulation platform modeling varied MAN topologies, cluster sizes, and neural models. Evaluations under multi-DC, large-model, and degraded-network scenarios show that CDynP markedly optimize the per-iteration training time. It can still maintain leading performance as task and network complexity increase, and achieve rapid strategy search while retaining solution quality. Ultimately, this work provides a privacy-preserving solution to interconnect fragmented compute islands in MANs, fostering efficient collaborative AI training.

II. RELATED WORK

This section reviews task scheduling mechanisms in computing power networks (CPNs) and adaptive parallelism strategies in distributed training.

A. Task Scheduling in CPN

CPNs facilitate unified orchestration across cloud, edge, and end devices. Recent work improves scheduling flexibility through adaptive task granularity via decomposition and generation.

Task decomposition divides complex applications into subtasks. Abdullah et al. [6] present an automated black-box method that refactors monolithic web applications into microservices using access logs and unsupervised learning, enhancing scalability and performance via autoscaling. Likewise, Wang et al. [7] propose WMG-Dcom, a weighted-graph framework for multi-granularity mapping in cloud-edge-end systems.

Task generation leverages granular computing to tailor workloads to heterogeneous resources. Liu et al. [8] use dependency-based generation for better priority scheduling, whereas Wei et al. [9] integrate component aggregation, mixed-integer programming, and reinforcement learning to minimize execution time and energy in mobility-aware deployments.

B. Adaptive Parallelism in Distributed Training

As models and hardware grow more complex, adaptive parallelism optimizes distributed training by selecting DP, tensor parallel (TP), and PP combinations based on topology and architecture, often minimizing per-iteration time or balancing stage loads. PP is preferred for cross-DC coordination due to reduced communication.

Early PP hybrids use dynamic programming (DynP) for Min-Max stage-time optimization. PipeDream [10] combined PP with DP via the 1F1B schedule to overlap computation and communication. AMP [11] automated partitioning and micro-batch sizing with advanced cost models. SPP [12] added

network awareness through recursive ordering and partitioning to cut latency.

Heuristic approaches, like Galvatron's [13] decision-tree search for multidimensional exploration and Yuan et al.'s [14] evolutionary scheduling for low-bandwidth edges, enable finer control in expansive search spaces.

C. Summary

Although CPN task scheduling improves general flexibility, it rarely considers pipeline bottlenecks caused by sequential dependencies. Conversely, adaptive parallelism techniques are tailored to distributed training but overwhelmingly assume homogeneous high-bandwidth LAN environments and full resource visibility, making them unsuitable for MAN scenarios with inter-/intra-DC latency disparities and strict privacy constraints in multi-DC collaboration. We address these combined limitations by proposing a privacy-preserving automated framework that generates efficient parallel strategies specifically for MAN-based multi-DC distributed training.

III. PROBLEM DEFINITION AND SYSTEM MODELING

This section formalizes the metropolitan-scale, multi-DC collaborative training framework. We first define the problem scenario and system model, then develop a comprehensive cost model integrating computation, communication, and synchronization delays. From this foundation, the generation of parallel strategies is formulated as a constrained global optimization problem.

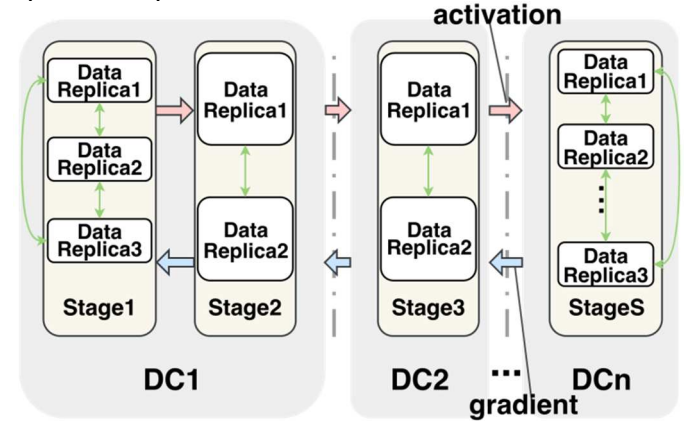


Figure 2. The parallel paradigm across DCs is PP. Within each DC, it is a hybrid architecture combining PP and DP.

A. Scenario Description

The system comprises K heterogeneous DCs connected via MAN. The workload is a Transformer-based large model TLM with L layers. As in Figure 2, synchronous PP governs inter-DC training under network constraints, while PP with DP is employed internally. Though described for PP-DP, the approach generalizes to TP and expert parallelism (EP). The optimization seeks to minimize per-iteration latency by determining: (1) a partition of TLM into S contiguous stages, and (2) a mapping of these stages to K DCs with specified data replica counts.

B. Mathematical Formulation

In alignment with the distributed training scenario addressed in this paper (as outlined in Section 1) and drawing upon the bubble time analysis from GPipe [15], we adopt the minimization of per-iteration training time as our primary optimization objective. The resulting optimization problem is formulated as follows. The constant parameters, input variables, and decision variables used in this formulation are summarized in Tables 1 and 2.

$$\text{Minimize: } T_{\text{iter}} = (M+S-1) \times \max_{s \in [1, S]} T_{\text{stage}}^s$$

This formula reflects the initial filling time required to start the assembly line as well as the throughput characteristics in the stable state. Here, T_{stage}^s represents the duration of stage S , since in PP computation and communication can overlap, the larger value is taken to represent the actual stage time:

$$T_{\text{stage}}^s = \max(T_{\text{comm}}^{s-1, s}, T_{\text{comp}}^s), \quad s \in [1, S]$$

The first term $T_{\text{comm}}^{s-1, s}$ denotes the total communication time on the communication channel between layers e_s and s_s :

$$T_{\text{comm}}^{s-1, s} = \frac{d_{e_s, s_s}^f + d_{e_s, s_s}^b}{r_s \square r_{s+1} \square \min_{i \in \mathcal{A}(s-1), j \in \mathcal{A}(s)} B_{i,j}}, \quad s \in [1, S-1]$$

The second term $T_{\text{comp}}^s = \frac{\sum_{l=s_s}^{e_s} (p_l^f + p_l^b)}{r_s} + A_s$ is the training time on the s -th stage, including processing time over layers s_s to e_s , replicated on r_s GPUs and time taken by the corresponding AllReduce operation. Here A_s denotes the time for AllReduce operation of layers s_s to e_s replicated over GPUs:

$$A_s = \frac{2(r_s-1)}{r_s} \square \frac{\sum_{l=s_s}^{e_s} a_l}{\min_{i,j \in \mathcal{A}(s)} B_{i,j}}$$

Constraints:

1. Layer continuity: Layers $[s_s, e_s]$ must form a continuous and non-overlapping sequence covering $[1, L]$.
2. Resource limits: GPU allocation per DC must not exceed its capacity: $\sum_{s: \text{map}(s)=k} r_s \leq G_k, \quad \square k$.
3. Privacy preservation: To prevent information leakage, intra-DC bandwidth B_k^{intra} , GPU capacities G_k , and mappings $\text{map}(s)$ are abstracted in the solution space.

IV. PARALLEL STRATEGY GENERATION MECHANISM BASED ON CHAIN-LIKE INTERACTION

The optimization problem formulated in the Section 3 is NP-hard[16], with a search space that grows exponentially with model depth and GPU count, making exact solutions impractical for large heterogeneous clusters. Coupled with the need for rapid deployment, exhaustive search or learning-based planners are unsuitable due to prohibitive computation and latency.

Inspired by the DynP-based strategy in Section 2.B, we leverage optimal substructure to extend DynP to multi-DC settings and propose CDynP, a privacy-preserving algorithm that builds a near-optimal global plan through hop-by-hop exchanges of compressed SS. Rather than centralized control, each DC makes local decisions based on the SS received from its predecessor and forwards an updated summary to the next, thereby assembling a global policy along the chain.

This chapter first presents the single-DC parallel strategy generation algorithm (Section A) as a key component of CDynP, then analyzes how these local decisions can be integrated into a global strategy via a chain-based interaction mechanism (Section B).

A. Single-CD Parallel Strategy Formulation Based on DynP

Each DC on the chain execute DynP locally, aiming to minimize the maximum stage execution time within the constraints $[L_{\text{start}}, L_{\text{end}}]$ and GPU capacity G_k .

Let $dp[s][l][g]$ denote the minimum maximum stage execution time, when we partition the first l consecutive layers in TLM into s stages and place the s stages over g GPUs. $dp[s][l][g]$ can be recursively computed as follows:

$$dp[s][l][g] = \min_{0 \leq l' < l, 0 \leq g' < g} \{ \max(dp[s-1][l'][g'], T_{\text{comm}}(l', l, g, g'), T_{\text{comp}}(l', l, g-g')) \}$$

This captures: (i) the minimum maximum stage execution time for the s stages covering l' layers with g' GPUs, (ii) T_{comm} between the $(s-1)$ th stage and the s -th stage, and (iii) T_{comp} for layers $l'+1$ to l using $g-g'$ GPUs.

The algorithm enumerates all (s, l, g) states, for each trying all layer splits and GPU assignments. Each enumeration computes T_{comm} , T_{comp} , and prior stage latency, resulting in complexity $O(SL^2G^2)$.

TABLE 1. SYSTEM MODEL PARAMETERS

Symbol	Description
L	Total number of layers in the model
K	Total number of collaborating DCs
M	Number of micro-batches per training iteration
G_k	Number of GPUs available in DC_k
a_l	The parameter size for layer l
p_l^f, p_l^b	Forward/backward computation time for processing a single micro-batch in layer l
$d_{l,l+1}^f, d_{l,l+1}^b$	Amount of activation data (forward) or gradient data (backward) transferred between layers l and $l+1$
$B_{i,j}$	Bandwidth between GPU_i and GPU_j
B_k^{intra}	Intra-DC high-speed interconnect bandwidth (e.g., NVLink/InfiniBand)
$B_{i,j}^{\text{inter}}$	Inter-DC bandwidth between DC_i and DC_j
$\text{Delay}_{i,j}$	End-to-end network latency between DC_i and DC_j

TABLE 2. DECISION VARIABLES

Symbol	Description
$\mathcal{S}$	Set of all stages that TLM is partitioned into ($S= \mathcal{S} $: total number of pipeline stages)
s_s, e_s	Start and end layer indices for stage s ($1 \leq s_s \leq e_s \leq L$)
$\mathcal{F}: \mathcal{S} \rightarrow \mathcal{P}(V)$	Device mapping function from stages to sets of GPUs. ($ \mathcal{F}(s) = r_s$, the number of replicas of stage S)

B. CDynP Framework

As outlined in Section A, DynP is restricted to solving model partition and resource mapping within fixed model layer intervals $[L_{\text{start}}, L_{\text{end}}]$ on single-DC. However, multi-DC environments introduce the challenge of assigning each center DC_k an interval $[s_k, e_k]$, effectively determining the cut-points

between DCs. To fill the gap, we propose an extended CDynP framework for global optimization.

1) Mechanism Overview: The mechanism involves two roles: a coordinator node, the DC which receives the user's request and orchestrates planning, and a set of executor nodes, DCs that run the distributed training task. These executor nodes are organized into a logical chain that underpins the CDynP procedure (Figure 3).

Phase 1 – Heuristic Pre-partitioning: To reduce the vast search space, the coordinator node divides the model into coarse-grained segments, then sends the pre-partitioned scope to the first executor node in the chain.

Phase 2 – Forward CDynP: The head node in the execution chain performs an independent DynP and send compressed output -SS- to next node. Each intermediate node executes the local DynP based on the received SS and forward an updated SS downstream. Once tail node completes its DP step, its output contains near-optimal global strategy covering the entire model.

Phase 3 – Global Strategy & Chain Backtracking: The tail node scans its output to identify the configuration yielding the minimal global iteration cost. Through the chain backtracking process, optimal boundary decisions are progressively determined and propagated upstream, thereby completing the global allocation.

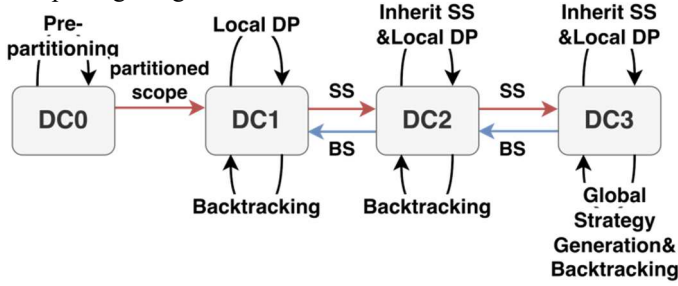


Figure 3. The schematic diagram of the CDynP workflow, in which the chain includes three executor nodes. Boundary state (BS) is the necessary boundary information required for backtracking.

2) Phase Details: This section provides a detailed description of the various stages.

Phase 1 – Heuristic Pre-partitioning

Computational capacity of DC determine initial boundary allocations:

$$r_k = \frac{G_k}{\sum_{i=1}^K G_i}, \quad L_k^{target} = \lfloor r_k L \rfloor$$

Boundaries are computed cumulatively across centers, forming initial ranges $[start_k, end_k]$. A soft-boundary mechanism further extends boundaries using bias $(L/8)$:

$$start_k^{exp} = \max(1, start_k - Bias)$$

$$end_k^{exp} = \min(L, end_k + Bias)$$

In Phase2, the local DynP operates on this expanded range. After expansion, overlapping regions $([start_k^{exp}, end_{k-1}^{exp}])$ occur between ranges assigned to neighboring DCs, and the subsequent algorithm will select optimal cut-points within these overlapping areas.

Phase 2 – Forward CDynP

Before detailing the CDynP procedure, we redefine $dp[s][l][g]$ to cover multi-DC scenarios: it represents the division of layers from $l+start_k^{exp}-1$ into s stages across g GPUs in DC_k , covering all k upstream DCs, with the minimal maximum stage time. The stage dimension S_{max} is bounded by: $S_{max} = \min(end_k^{exp}, \sum_{i=1}^K G_i)$. This allocation guarantees sufficient capacity to inherit upstream cumulative stages, maintaining seamless optimization across the CDynP chain.

Next, we will introduce the execution process of this stage. Each DC executes iterative “inheritance-planning-compression” cycles across DCs to progressively construct the global strategy.

First, the previous node (DC_{k-1}) compresses $dp[s][l][g]$ array into SS to conceal allocation patterns while retaining minimal maximum stage time:

$$SS_{k-1}[s][l] = \min_{1 \leq g \leq G_{k-1}} dp_{k-1}[s][l][g]$$

Upon receiving $SS_{k-1}[s][l]$, DC_k sets its local DynP array initialization as:

$$dp[s][l][0] = SS_{k-1}[s][l + \Delta_{offset}]$$

Here $g=0$ is a special virtual state representing workload fully handled by upstream nodes. The Δ_{offset} maps upstream indices to the local index, ensuring seamless integration into the local search space.

Then, DC_k executes DynP locally on expanded range (Section 4.A), but it is important to note that when performing state transitions, the cost of cross-domain communication needs to be taken into account:

$$T_{comm} = \frac{d_{l,l+1}^f + d_{l+1,l}^b}{B_{k-1,k}^{inter}} + Delay_{k-1,k}$$

Finally, DC_k compresses DynP outputs and sends to DC_{k+1} .

Phase 3 – Global Strategy & Chain Backtracking

After computation, the tail node scans $dp[s][l][g]$ for (s^*, g^*) that minimizes the objective function. The tail node backtracks according to its local plan array to determine the partition and mapping strategy for itself.

When the backtracking process reaches a boundary state with $g^*=0$, this indicates that the current stage's starting point is an cut-point across DCs. At this point, the tail node transmits the corresponding state upstream. Upon receiving the state, the preceding node repeats the same backtracking procedure until the head of the chain is reached.

3) Theoretical Analysis and Proof: This section demonstrates the validity of the CDynP framework, focusing on proving that it can obtain the optimal solution under privacy constraints. In terms of performance, it is equivalent to the global enumeration-combination method (EC). The EC exhaustively enumerates all possible cut-points within the overlapping area, and based on the divided layer ranges each DC runs the local DynP respectively (refer to Section 4.A), and selects the optimal result from the obtained combinations.

Let S_{EC} denote the set of all strategies generated by EC, and S_{CDP} denote the set of solutions generated by the CDynP

framework. The optimal equivalence criterion requires that CDynP achieves the same global minimum as EC:

$$\min_{\pi \in S_{CDP}} T(\pi) = \min_{\pi' \in S_{FPE}} T(\pi')$$

Proof (Mathematical Induction)

- Base Case($k=1$): For $\square C_1$, both algorithms execute the DynP within the range of $[0, end_1^{exp}]$, and the solution spaces are equivalent.
- Inductive Hypothesis: Assume that for the preceding $k-1$ DCs, $\min_{\pi \in S_{CDP}} T(\pi) = \min_{\pi' \in S_{FPE}} T(\pi')$
- Inductive Step: For the DC_k , the $dp_k[s][l][g]$ combined with local computation ($g>0$) or direct continuation ($g=0$), extends the optimal solution seamlessly from DC_{k-1} to DC_k . According to the optimal substructure property of DynP, the global optimal policy $\pi_k^\square$ includes the cut-point $l^\square$ between DC_{k-1} and DC_k , the optimal strategy $\pi_{k-1}^\square$ of DC_{k-1} and a certain strategy π' of DC_k . The cost of $\pi_{k-1}^\square$ is necessarily captured by $collapse_dp_{k-1}[s][l]$. When performing DynP on DC_k , the previous states ($l, 0$) will be enumerated. When $l=l^\square$ is enumerated, since it corresponds to the optimal solution, it will be captured by the transfer equation.

Note: The optimality of CDynP is relative to the pre-defined overlapping areas. This is a necessary trade-off to balance computational efficiency and search breadth.

V. EXPERIMENTAL EVALUATION AND RESULTS ANALYSIS

A. Settings

We emulate a MAN environment using NetworkX (version 3.1), running on a local MacBook Pro with a 2.4 GHz Quad-Core Intel Core i5 CPU and 8 GB of RAM. The specifications of the simulated target system are detailed in Table 3. It is important to note that, due to hardware constraints, our evaluation is conducted on a simulation platform rather than through large-scale physical deployment. However, the relative performance trends observed in our results (as depicted in Figure. 4-9) robustly demonstrate the superiority of our method.

1) Evaluation Metrics : We report two core metrics to comprehensively assess system performance.

- Per-iteration training time (T_{iter}): the decisive indicator of training efficiency, captures computation time, communication overhead, and idle waiting due to pipeline bubbles.
- Strategy search time: the time required for the algorithm to produce a parallelization strategy, which directly determines the scheduler's responsiveness and its ability to support online scheduling.

2) Baselines: We compare against three representative methods: (i) standard DP as a performance reference; (ii) PipeDream, a classic pipeline partitioning approach; and (iii) Synchronous Pipeline Planning (SPP) [12], a network awareness planer for hybrid parallel strategies. Given the

significant divergence in latency sensitivity between inter-DC and intra-DC environments, we adapt these baselines to the cross-DC setting using a two-stage process. First, we partition the model into coarse-grained segments proportional to the aggregate computational power of each data center. Secondly, we perform strategy search independently within each local environment. While optimal allocation under PP may not strictly follow computational proportionality due to the overlap of computation and communication, this heuristic provides a fair resource-to-task alignment. Furthermore, optimizing the adaptation strategy itself would require complex modeling of PP-specific communication and computation timing, which falls outside the scope of this study.

TABLE 3. HARDWARE AND SIMULATION CONFIGURATION

Configuration Item	Specification
GPUs per Rack	4×NVIDIA V100 (32 GB, 14 TFLOPS)
Inter-GPU Interconnect	NVLink 2.0 (~300 GB/s per link)
Network Topology	3 DCs(Fits multi-DC collaboration; otherwise, excessive inter-DC latency), Full-Mesh
Intra-DC Interconnect	Bandwidth: 32-40 Gbps, Latency: < 0.1 ms
Inter-DC Link Models	Bandwidth: 1-2 Gbps, Latency: 1-3 ms GPT-1.9B, BERT-72
Micro-batch Size	16
Sequence Length	1920

B. Results and Analysis

1) Training Efficiency : CDynP employs chain-based DynP to realize end-to-end global optimization across DCs, embedding inter-DC bandwidth-aware cost control into DynP transitions. This mechanism biases layer cut-points toward those with lower communication overhead and, crucially, incorporates the influence of inter-DC network conditions into intra-DC decisions, enabling adaptive strategy adjustments that mitigate cross-DC latency impacts. As a result, CDynP dynamically balances loads and optimizes resource utilization across DCs. Extensive benchmarks across diverse scenarios demonstrate substantial performance gains.

Different numbers of DCs: Under a fixed total GPU budget, increasing the number of DCs from one to four extended training times for PP approaches due to higher inter-domain communication costs (Figure 4). CDynP's iteration time increased by only 8.3%, markedly lower than the 16.0% observed for SPP and 18.2% for PipeDream, an improvement attributable to dynamic load balancing guided by the states of preceding stages. In comparison, methods with fixed partitions incurred roughly 10% more imbalance in their slowest stage. Interestingly, DP's performance improved in geographically dispersed deployments, as distribution helped alleviate intra-DC gradient synchronization bottlenecks.

Different scale of models : From BERT-Large to GPT-1.9B, CDynP consistently outperformed competitors (Figure 5). While PipeDream minimizes maximum stage time, it neglects the bubble latency introduced by additional stages.

For GPT-1.9B, CDynP reduces training time by 10.1% compared to SPP, achieved by shortening the longest stage duration and selecting $S=5$ to limit bubble overhead. In contrast, SPP fixes $S=6$, which increases bubble latency. CDynP's global optimization objective explicitly balances computational workload against bubble overhead, effectively mitigating the nonlinear complexity associated with scaling to larger models.

Different numbers of micro-batches: With micro-batch sizes M from 4 to 16, CDynP reliably exceeded baselines, underscoring its prowess in curbing pipeline bubbles (Figure 6). As described by $T_{iter} \propto (S+M-1)$, the bubble impact scales with M . CDynP and SPP incorporate this dependency into their models, selecting fewer stages for smaller M values. Conversely, DP and PipeDream do not employ global coupling, which leads to higher performance fluctuations under varying configurations.

2) Network Adaptability : CDynP incorporates the network bandwidth/delay of Inter/Intra-DC into the DynP transfer equation. The algorithm can sense changes in network conditions and make corresponding adjustments to the pipeline division points. By leveraging the characteristics of PP computation-communication overlap, it minimizes the impact of communication overhead on the overall execution time. A large number of benchmark tests have shown that CDynP can maintain leading performance even when the Intra-Dc/Rack network environment deteriorates.

Different inter-rack bandwidth levels: Across intra-DC bandwidths ranging from 5 to 100 Gbps, CDynP consistently achieved superior performance, with T_{iter} fluctuating by only 0.44 s (Figure 7). Under bandwidth-constrained conditions (5–10 Gbps), CDynP recorded 4.85 s per iteration, significantly outperforming SPP's 8.05 s. These findings demonstrate CDynP's capability to detect network bottlenecks and dynamically adjust partitioning, in contrast to SPP's localized optimization strategy, which exacerbates the influence of constrained links on overall pipeline performance.

Different inter-DC bandwidth/delay levels : Under degraded MAN conditions, CDynP demonstrated the highest resilience, with iteration time increasing at the slowest rate among all methods (Figure 8). By incorporating inter-domain communication costs into its state-transition model, CDynP can sense low-quality links and automatically adjust layer partitioning across centers, enabling computation and communication to overlap and effectively masking latency. In contrast, competing approaches focus solely on intra-DC optimization, rendering them insensitive to inter-domain variability and resulting in sharp performance degradation as network quality deteriorates.

3) Algorithm Efficiency : In the 32-GPU configuration, CDynP achieved a search time of 39.27 s, which is $13.4 \times$ faster than SPP's 528.00 s (Figure 9). This gain originates from the precise selection of the dimensions of its DynP process and leveraging a chain-structured formulation, thereby

avoiding the combinatorial explosion and scalability bottlenecks that constrain SPP. While PipeDream attained the shortest search time (1.43 s), it neglects the impact of the stage count S on pipeline efficiency, compromising global optimality. Within a reasonable computational budget, CDynP reduced T_{iter} by 16% versus PipeDream and by 5% versus SPP, demonstrating strong practical utility. It is worth noting that in Phase2, each executor node transmits a compressed SS of size $O(SL/K)$ to the next node. However, compared to the overall search time, the Inter-DC delay added by this operation can be considered negligible.

VI. CONCLUSIONS AND FUTURE WORK

This paper present a CDynP framework for privacy-preserving, multi-DC collaborative training in MAN. By integrating PP and DP paradigms into a unified model that captures inter/intra-DC network disparities, CDynP enables decentralized strategy generation through local decisions and compressed state transfer along the execution chain. Experiments show that CDynP achieves consistently lower iteration latency and faster strategy generation than optimized baselines, with strong scalability under larger models and degraded network conditions. These results highlight CDynP as a practical approach for efficiently leveraging dispersed compute resources in privacy-constrained, MAN AI training scenarios.

Our immediate priority for future research is to validate the proposed framework on a physical Metropolitan Area Network (MAN) cluster. Beyond physical validation, we plan to focus on dynamic adaptive scheduling against network volatility, extending support to fine-grained paradigms such as tensor parallelism and MoE, and integrating deeply heterogeneous intra-DC hardware resources. Furthermore, we envision incorporating emerging network architectures—such as ICN with on-path routing—to pick the best DC hop-by-hop, dynamically producing global strategies.

ACKNOWLEDGMENT

This work was supported by the National Key R&D Program of China Application Demonstration of Polymorphic Network Environment for computing from the eastern areas to the western. (Project No. 2023YFB2906404).

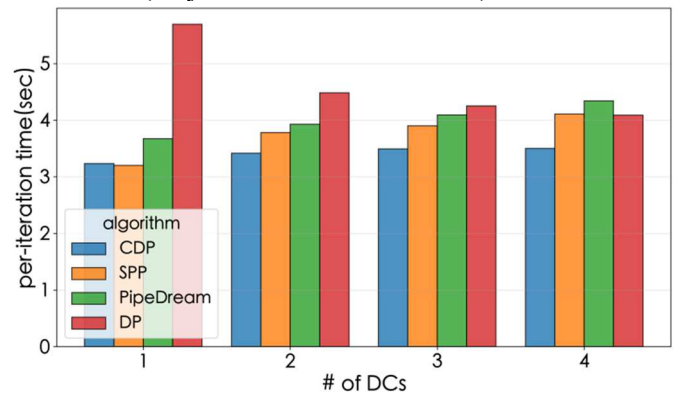


Figure 4. Training time: different # of DCs

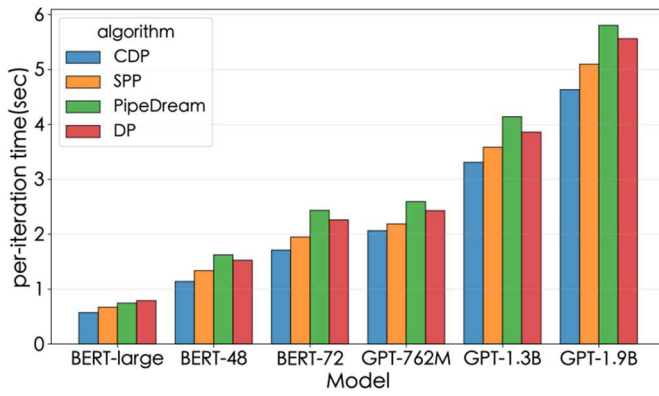


Figure 5. Training time: different scale of models

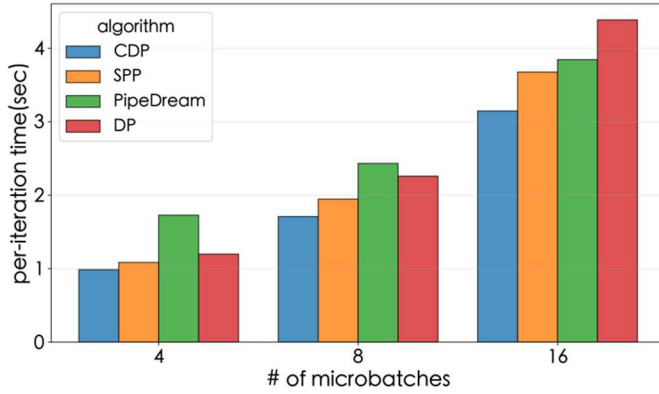


Figure 6. Training time: different # of micro-batches

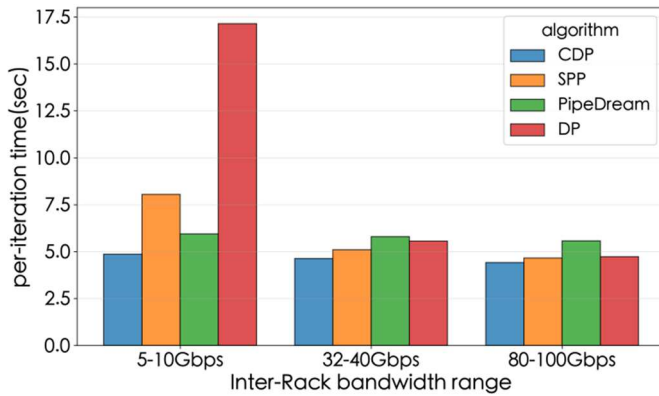


Figure 7. Training time: different inter-rack bandwidth levels

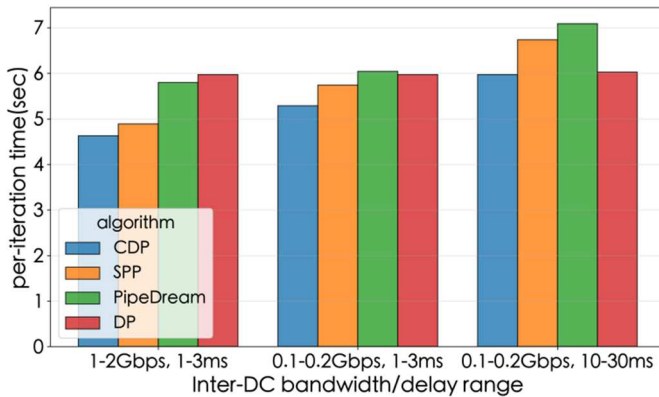


Figure 8. Training time: different inter-DC bandwidth/delay levels

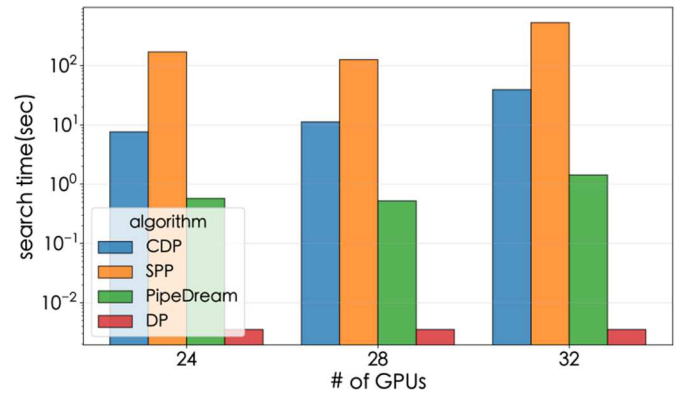


Figure 9. Search time: different # of GPUs

REFERENCES

- [1] Epoch AI. (2025) Training compute of frontier AI models grows by 4 - 5 $\times$ per year. Epoch AI. [Online]. Available: <https://epoch.ai/blog/training-compute-of-frontier-ai-models-grows-by-4-5x-per-year>
- [2] F. Strati, P. Elvinger, T. Kerimoglu, and A. Klimovic, "ML training with cloud GPU shortages: Is cross-region the answer?," in Proc. 4th Workshop Mach. Learn. Syst. (EuroMLSys' 24), Athens, Greece, Apr. 2024, pp. 107 - 116, doi: 10.1145/3642970.3655843.
- [3] T. Chen, A. Kubicek, L. Huang and T. Hoefler, "CrossPipe: Towards optimal pipeline schedules for cross-datacenter training," in Proc. 2025 USENIX Annu. Tech. Conf. (USENIX ATC' 25), Santa Clara, CA, USA, Jul. 2025, pp. 1089 - 1108.
- [4] S. Smith et al., "Using DeepSpeed and Megatron to train Megatron-Turing NLG 530B, a large-scale generative language model," arXiv:2201.11990 [cs.CL], Jan. 2022. [Online]. Available: <https://arxiv.org/abs/2201.11990>
- [5] H. Liao et al., "UB-Mesh: A hierarchically localized nD-FullMesh data center network architecture," IEEE Micro, vol. 45, no. 5, pp. 20 - 29, Sep. - Oct. 2025, doi: 10.1109/MM.2025.3592688.
- [6] A. M. Abdullah, W. Iqbal, and A. Erradi, "Unsupervised learning approach for web application auto-decomposition into microservices," J. Syst. Softw., vol. 151, pp. 243 - 257, May 2019, doi: 10.1016/j.jss.2019.02.031.
- [7] Z. Wang, "Multi-granularity decomposition of componentized network applications based on weighted graph clustering," J. Web Eng., vol. 21, no. 03, pp. 815 - 844, Mar. 2022.
- [8] X. Liu, "Task Decomposition and Scheduling Analysis in Large-scale Network Management", Xinxu Xitong Gongcheng, no. 2, p. 2, Feb. 2013, doi: 10.3969/j.issn.1001-2362.2013.02.036.
- [9] L. Wei et al., "Task decomposition of network services for deploying in heterogeneous network computing nodes," in Proc. 2021 IEEE 26th Int. Workshop Comput. Aided Model. Des. Commun. Links Netw. (CAMAD), Porto, Portugal, Oct. 2021, pp. 1 - 6, doi: 10.1109/CAMAD52502.2021.9617805.
- [10] D. Narayanan et al., "PipeDream: Generalized pipeline parallelism for DNN training," in Proc. 27th ACM Symp. Oper. Syst. Princ. (SOSP' 19), Huntsville, ON, Canada, Oct. 2019, pp. 1 - 15, doi: 10.1145/3341301.3359646.
- [11] D. Li, H. Wang, E. Xing, and H. Zhang, "AMP: Automatically finding model parallel strategies with heterogeneity awareness," in Proc. 36th Int. Conf. Neural Inf. Process. Syst. (NeurIPS' 22), New Orleans, LA, USA, Nov. - Dec. 2022, pp. 6630 - 6639.
- [12] Z. Luo et al., "Efficient pipeline planning for expedited distributed DNN training," in Proc. IEEE INFOCOM 2022 - IEEE Conf. Comput. Commun., London, U.K., May 2022, pp. 340 - 349, doi: 10.1109/INFOCOM48880.2022.9796787.

- [13] X. Miao et al., “Galvatron: Efficient transformer training over multiple GPUs using automatic parallelism,” *Proc. VLDB Endowment*, vol. 16, no. 3, pp. 470 – 479, Nov. 2022, doi: 10.14778/3570690.3570697.
- [14] B. Yuan et al., “Decentralized training of foundation models in heterogeneous environments,” in *Proc. 36th Int. Conf. Neural Inf. Process. Syst. (NeurIPS’ 22)*, New Orleans, LA, USA, Nov. – Dec. 2022, pp. 25464 – 25477.
- [15] Y. Huang et al., “GPipe: Efficient training of giant neural networks using pipeline parallelism,” in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS’ 19)*, Vancouver, BC, Canada, Dec. 2019.
- [16] A. Benoit and Y. Robert, “Mapping pipeline skeletons onto heterogeneous platforms,” *J. Parallel Distrib. Comput.*, vol. 68, no. 6, pp. 790 – 808, Jun. 2008.