Location Matters: LLM-Guided Joint Optimization of In-Network Aggregation Placement and Routing for DML Workloads

Long Luo, *Member, IEEE*, Yanan Huang, *Student Member, IEEE*, Xixi Chen, *Student Member, IEEE*, Yongsheng Zhao, *Student Member, IEEE*, Hongfang Yu, *Senior Member, IEEE*, and Schahram Dustdar, *Fellow, IEEE*

***Abstract*—In-network aggregation (INA) accelerates gradient aggregation in distributed machine learning (DML) by alleviating communication bottlenecks, but its effectiveness crucially depends on two location decisions: where to deploy INA functions and where to aggregate gradient flows. Most existing methods optimize INA placement and gradient flow routing independently, missing the advantages of joint optimization. This paper presents LLMINA, which leverages Large Language Models (LLMs) to automate the heuristic design for joint INA placement and gradient aggregation, aiming to minimize makespan (i.e., the total time required for all DML jobs to complete gradient aggregation). Directly using LLMs to generate end-to-end solutions is infeasible due to problem complexity and LLM limitations. Instead, LLMINA uses LLMs to generate heuristics for INA placement through an evolutionary process, and then applies an optimization-based heuristic for gradient routing that takes into account DML workload characteristics. Experiments across diverse network topologies and workloads show that LLMINA can significantly reduce makespan compared to state-of-the-art baselines. These results underscore that location matters for both INA deployment and aggregation, and highlight the potential of LLM-guided heuristic design for complex network resource optimization.**

***Index Terms*—In-network aggregation, distributed machine learning, large language model (LLM), joint optimization, heuristic.**

I. Introduction

MACHINE learning, particularly with deep neural networks (DNNs), has become the cornerstone of advances in natural language processing, computer vision, and a broad spectrum of AI-driven applications and fields [1], [2]. As both model complexity and dataset sizes continue to grow exponentially, distributed machine learning (DML) has become indispensable for meeting computational demands by leveraging multiple GPUs and nodes [3], [4], [5], [6], [7], [8]. However, this distributed paradigm imposes a persistent communication bottleneck: workers must frequently synchronize massive gradient tensors, saturating network bandwidth and restricting system efficiency [9], [10], [11], [12], [13], [14], [15], [16].

In-network aggregation (INA) has emerged as a promising solution to alleviate this communication overhead. By allowing programmable switches or smart NICs to aggregate gradient data directly within the network [9], [12], [17], [18], [19], [20], INA ensures that only the aggregated result is forwarded, rather than transmitting multiple individual gradient updates from each worker. This approach significantly reduces network traffic and latency, thereby accelerating the completion of DML jobs. Despite its promise, fully implementing the benefits of INA is fundamentally constrained by the scarcity and complexity of the orchestration of network resources [21], [22]. Typically, only a handful of switches or smart NICs are available for INA within a cluster, each with limited on-chip memory and computation bandwidth (e.g., tens of MB per switch [23]), which is significantly less than the hundreds of MB often demanded by state-of-the-art DNNs. This resource scarcity leads to a crucial infrastructure-level challenge: *where to place INA functions* to maximize coverage and accessibility for various DML jobs. Beyond physical placement, these scarce INA resources must also be efficiently shared by multiple coexisting DML jobs with heterogeneous communication demands. At the operational level, it is critical to decide *where each worker's gradient flow should be aggregated*, that is, to determine which subset of workers should have their gradients aggregated at each INA node. Placement and routing decisions are fundamentally intertwined: suboptimal placement may cause network hotspots or overload certain aggregators, while poor routing can lead to imbalanced utilization and reduced throughput.

Recent research has significantly advanced the field of optimizing INA resource utilization wihtin DML systems [9], [24], [25], [26], [27]. A substantial body of work has concentrated on granular aspects of INA optimization. For instance, specific efforts (e.g., GRID [24]) have focused on optimizing gradient flow routing strategies to proactively mitigate aggregator hotspots, thereby ensuring balanced load distribution. Concurrently, other research avenues have explored the strategic placement of INA functionalities within the network fabric. The primary objective

Received 23 September 2025; revised 19 December 2025; accepted 8 January 2026. Date of publication 14 January 2026; date of current version 29 January 2026. This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFB2904600. Recommended for acceptance by Dr. Chau Yuen. *(Corresponding author: Long Luo.)*

Long Luo, Yanan Huang, Xixi Chen, Yongsheng Zhao, and Hongfang Yu are with the University of Electronic Science and Technology of China, Chengdu 610054, China (e-mail: llong@uestc.edu.cn; cxxii@std.uestc.edu.cn; yuhf@uestc.edu.cn).

Schahram Dustdar is with Distributed Systems Group, TU Wien, 1040 Vienna, Austria (e-mail: dustdar@dsg.tuwien.ac.at).

Digital Object Identifier 10.1109/TNSE.2026.3654163

here is to minimize overall network traffic, as demonstrated by solutions such as TAPINA [27]), which aim to reduce communication overhead by processing gradients closer to their sources. Despite these advancements, a critical limitation pervades the existing literature: most proposed solutions address either the placement of INA functionalities or the routing of gradient flows in isolation. This segregated approach often operates under simplified assumptions, such as single-job scenarios where network contention is minimal. Crucially, these methods tend to overlook the profound and intricate coupling that exists between the INA location, the gradient flow routing policies, and the resulting resource contention. This interdependency is particularly pronounced and prevalent in real-world multi-job DML clusters, where multiple concurrent jobs compete for limited network and switch in-network computing resources. Consequently, a significant research gap persists: the absence of a holistic and integrated framework capable of jointly optimizing both INA placement and routing decisions. Such a framework could be indispensable for effectively managing complex resource constraints and achieving truly efficient and scalable DML deployments in multi-job environments.

To address this challenge, we propose LLMINA, a joint INA function placement and gradient flow routing method designed for multi-job DML clusters with limited in-network computing resources. Our central optimization objective is to minimize the makespan, the total time required to complete all gradient aggregations for coexisting DML jobs, thereby accelerating job completion and improving overall cluster efficiency. We first develop a mixed-integer linear programming (MILP) model that captures the coupling between placement and routing decisions under realistic INA and bandwidth constraints. This joint optimization problem is combinatorially complex and makes exact solutions intractable for practical-scale clusters. To tackle this, we introduce a principled divide-and-conquer framework that decomposes the challenging joint problem into two tractable subproblems, each of which is quickly solved with a well-designed heuristic. First, we leverage large language models (LLMs) to automatically generate and iteratively improve heuristic algorithms for INA placement via evolutionary search. This overcomes the limitations of manual heuristic design and adapts effectively to varying cluster topologies and workloads. Then, given INA placements, we design an efficient optimization-based heuristic for gradient flow routing, which balances aggregator utilization and minimizes the communication makespan. This modeling-driven, LLM-guided approach bridges the gap between theoretical optimality and practical deployment, fully realizing the potential of INA acceleration in multi-job DML clusters. In summary, the main contributions of this work are as follows:

- We present the first mathematical formulation for jointly optimizing INA function placement and gradient flow routing under realistic resource constraints for multiple coexisting DML jobs.
- We develop an LLM-guided heuristic algorithm for INA placement and a specialized optimization-based heuristic for gradient flow routing, together effectively balancing aggregator loads and minimizing the overall gradient aggregation makespan for DML workloads.
- We demonstrate through extensive simulations that LLMINA reduces gradient aggregation makespan significantly compared to state-of-the-art baselines across a wide range workload scenarios.

The remainder of this paper is structured as follows. Section III motivates our approach by demonstrating, via an illustrative example, the critical need for jointly optimizing INA placement and gradient flow routing. Subsequently, Section IV provides a detailed mathematical formulation of the problem. Section V introduces our proposed algorithm framework and elaborates on its detailed design for addressing the formulated optimization problem. In Section VI, we present and analyze comprehensive experimental results, and Section VII summarizes our contributions.

II. Related Work

A. In-Network Aggregation for DML

In Distributed Machine Learning (DML) system, efficient gradient communication among participating workers frequently poses a critical bottleneck. To address this, in-network aggregation (INA) techniques, utilizing programmable network elements like switches, have emerged as a promising paradigm [11], [12], [49], [50], [51], [52]. Early INA systems, exemplified by SwitchML and ATP, demonstrated the feasibility of performing gradient aggregation directly within the network fabric. However, these pioneering efforts often relied on fixed INA function placements and rigid routing policies, which inherently limited their adaptability and resource utilization efficiency in complex multi-job workload environments. Subsequent research aimed to overcome these limitations by enhancing the flexibility and scope of INA. Several works explored augmenting aggregation capabilities through memory sharing mechanisms or by expanding deployment points to additional network nodes [32], [33], thereby broadening the physical locations for aggregation. Currently, significant attention has been paid to optimizing the gradient data flow. Flexible gradient routing strategies [19], [24], [25], [26], [45], [46] and INA placement optimization methodologies [27], [43], [44] have been proposed to improve aggregation efficiency. Nevertheless, a common characteristic of these studies is their tendency to address INA placement and gradient routing as largely decoupled problems, often optimizing one given the other, and frequently focusing on single-job workloads.

Some recent works also have studied integrative platforms and protocols to tackle the practical complexities of deploying INA in production-scale environments. These efforts address critical aspects such as dynamic resource sharing across concurrent DML jobs, adaptation to hardware constraints (e.g., limited switch memory or processing power), and improving overall system-level throughput. Notable contributions include INAaaS [53], a generic framework for on-demand cloud INA with a unified interface; NetPack [42], a job placement system for INA-enabled clusters using novel resource estimation and optimization algorithms; GISA [18], a generic INA service for line-rate acceleration and traffic reduction; and an aggregator-aware protocol [54] for multi-tenant DML featuring decoupled congestion control and straggler-aware allocation. Additionally,

TABLE I
COMPARISON OF OPTIMIZATION SCOPE AND CAPABILITIES OF RELATED WORK

Category	Ref.	Workload	Optimization of "Locations"			Lossless
			Worker Placement	Aggregator Placement	Aggregation Routing	
System Impl.	[11], [12], [28]–[31]	Single	✗	✗	✗	✓
Memory Opt.	[32]–[35]	**Multi**	✗	✗	✗	✓
Transport Proto.	[36], [37]	Single	✗	✗	✗	✓
Approximation	[38]–[40]	Single	✗	✗	✗	✗
Job Scheduling	[41]	Single	✓	✗	✗	✓
Stat. Job Opt.	[42]	Multi	✓	✗	✓	✓
INA Placement	[27], [43], [44]	Single	✗	✓	✗	✓
Routing Co-design	[19], [24], [45]–[48], [25], [26]	Single	✗	✗	✓	✓
LLMINA (Ours)	-	**Multi**	✗	✓	✓	✓

Worker Placement: Deciding locations of DML workers (servers). **Aggregator Placement**: Deciding which switches perform aggregation (INA functions). **Aggregation Routing**: Constructing aggregation trees/paths in the network.

related works like [19] offer aggregator-aware routing optimization for Clos datacenter networks, addressing routing constraints and switch limitations to achieve substantial throughput gains. Other research work further explores synergy with novel network technologies. For instance, NetReduce [31] design a transport-transparent INA primitive for for modern multi-rack data centers. AQINA [38] proposes the joint utilization of INA and quantized gradient communication. Concurrently, P4INC-AOI [55] harnesses in-network computing and all-optical interconnects (AOI) to jointly optimize resource allocation and minimize AOI reconfigurations, specifically targeting DML acceleration.

Gap in prior work: Despite these advancements, a critical gap persists in the holistic, joint optimization of INA function placement and gradient flow routing within complex, multi-job, and resource-constrained cluster environments. While prior research has individually addressed placement or routing, or focused on simplified single-job scenarios as summarized in Table I, the interplay between these decisions in heterogeneous, concurrent-job settings, particularly under practical hardware limitations and evolving network conditions, remains significantly under-explored. This work aims to bridge this gap by presenting a unified framework that jointly optimizes both INA placement and gradient routing for enhanced system-wide performance.

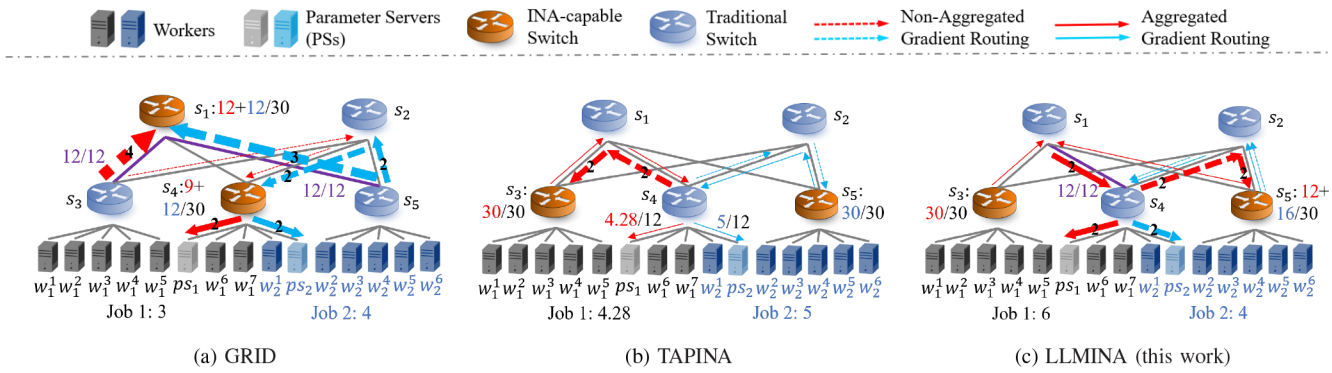
B. Automated Algorithm Design Leveraging LLMs

Recent advancements in generative AI have introduced powerful tools for network optimization [56], [57], [58]. While some studies [59], [60] employ graph diffusion models to generate numerical solutions for MEC or general network challenges, LLMINA follows a distinct paradigm. By leveraging the symbolic reasoning of Large Language Models (LLMs), our approach evolves interpretable heuristics that ensure deterministic execution and seamless alignment with complex protocol logic. This methodology fundamentally departs from traditional learning-based optimization, which primrily utilizes neural-based models—such as Reinforcement Learning (RL) or Graph Neural Networks (GNNs), to optimize parameters or decision policies in an opaque, black-box manner. Beyond serving as direct optimizers, LLMs have sparked a paradigm shift in networking by acting as automated optimizers [61], [62], [63], [64], predictors [65], [66], [67], [68], extractors [69], [70], [71], [72], and designers [73], [74], [75], [76]. Early efforts primarily focused on modular algorithmic construction, such as autonomously generating mutation operators for differential evolution or synthesizing hybrid metaheuristics by integrating diverse search paradigms [77], [78].

Subsequent advancements have pushed the boundaries from component-level generation to more holistic program synthesis. Notable examples include the FunSearch framework [75], where LLMs are employed to discover novel algorithmic building blocks and assemble them into functional programs, and FunBO [79], which utilizes LLMs for the automated generation of acquisition functions in Bayesian optimization. More recently, significant progress has been made in the integration of LLMs with evolutionary computation methodologies. Systems such as EoH [73], [80] and ReEvo [81] showcase the power of LLM-driven evolutionary algorithms. These approaches empower LLMs not only to generate initial heuristic designs but also to iteratively evolve them in response to performance feedback, effectively automating the discovery and refinement of high-performing heuristics for both single-objective and multi-objective optimization problems, and even for guiding hyper-heuristic design in a language-based manner.

Beyond the synthesis of individual components or evolutionary processes, LLMs are increasingly being utilized to generate complete metaheuristic frameworks. Several studies have demonstrated the ability of LLMs to act as the core optimizer or the primary code generator for entire optimization systems. Exemplary work includes the Zoological Search Optimization framework [82], which uses LLM to guide the search process, and dedicated LLM-based systems such as LLaMEA [83] and LLM-EA [84], where LLM plays a central role in the design of the search strategy or in the generation of the underlying optimization code.

Gap in prior work: While LLMs have demonstrated significant progress in automating algorithm design for classical combinatorial optimization problems (i.e., traveling salesman problem (TSP) [63], [84], [85], bin packing variants (both static



(a) GRID (b) TAPINA (c) LLMINA (this work)

Fig. 1. Comparison of gradient aggregation strategies in a spine-leaf network with two concurrent training jobs (distinguished by color) and two INA-capable switches. (a) Gradient routing only (i.e., GRID [24]) achieves an average aggregation rate of 3.5; (b) INA function placement focused (i.e., TAPINA [27]) achieves 4.64; (c) Joint optimizing placement and routing (this work) achieves 5.0. Arrow colors distinguish flows from different jobs; solid/dashed arrows depict aggregated/unaggregated gradient flows; thickness and labels show flow volume, rate, and link utilization.

and online) [73], [75], [86], [87], [88], and flow-shop scheduling [89], [90], [91], [92], [93]), their application to more complex and specialized domains, particularly intricate network resource allocation problems, remains largely unexplored. Our work aims to bridge this gap by investigating the efficacy of LLMs in generating effective algorithmic solutions for these challenging and underexplored scenarios.

III. A Motivating Example

To illustrate the necessity of jointly optimizing INA placement and gradient flow routing, we consider a representative spine-leaf network (Fig. 1) with two concurrent training jobs and a budget for INA deployment on only two switches. Each INA switch has a processing capacity limit and all network links are bandwidth constrained.

Consider the spine-leaf network depicted in Fig. 1, featuring two spine switches (s_1, s_2) and three Top-of-Rack (ToR) switches (s_3, s_4, s_5). We assume a budget allowing INA functionality deployment on at most two switches. Each potential INA switch has a processing capacity constraint of 30 units (e.g., aggregated gradient messages per time unit), and all links have a uniform bandwidth capacity of 12 units. Two DT jobs compete for resources in this multi-tenant environment using the Parameter Server (PS) architecture: Job 1 with 7 workers (w_1–w_7) and Job 2 with 6 workers (w_1–w_6). For simplicity, we assume both jobs train models of equivalent size, implying similar gradient volumes per worker. The optimization objective is to maximize the average gradient sending rate across both jobs. We compare the following three strategies.

- Routing-only optimization (GRID [24]): As shown in Fig. 1(a), only distributing the gradient flows from the workers of a single job to different INA switches for aggregation, could result in link congestion. For instance, the path from ToR s_4 to the spine s_2, which caps the gradient sending rate at an average of 3.5 units. Even with optimal routing, poor placement prevents full utilization of available network and compute resources.
- Placement-centric optimization (TAPINA [27]): Fig. 1(b) illustrates the impact of optimizing placement, where all gradient flows for a given job are routed to a single INA switch to maximize traffic reduction. This shifts the bottleneck from network bandwidth to switch processing capacity, boosting the average gradient sending rate to 4.64 units, a significant gain over GRID (a 32.6% gain over GRID). Nevertheless, forcing all gradients to a single aggregator is a rigid policy that may waste spare INA capacity and hinder load balancing under varying traffic conditions.
- Joint optimization (LLMINA, this work): Our method (Fig. 1(c)) jointly determines both INA placement and flexible gradient flow routing for the workers of each DML job. For example, most workers from Job 1 can aggregate at s_3, while overflow is routed to s_5; Job 2's workers can be split between two aggregation points or sent directly to the PS as needed. This flexibility ensures both link and compute capacities are efficiently utilized, leading to the highest rate of 5.0 units (42.9% and 7.8% higher than GRID and TAPINA, respectively).

This example highlights the shortcomings of optimizing placement or routing in isolation: each leads to distinct bottlenecks and leaves potential inferior performance. Only through unified, joint optimization can we fully exploit the capacity of INA-enabled networks, especially for multiple DML job scenarios. This motivates the approach presented in this work.

IV. Mathematical Formulation: Joint Optimization of INA Function Placement and Gradient Routing

This section provides a formal definition of the joint optimization problem of INA function placement and gradient routing for multiple coexisting DML jobs. We begin by presenting the system model, detailing the training architecture, gradient aggregation workflow, and resource constraints. Next, we introduce the mathematical formulation, explicitly specifying the objective function and constraints that guide both INA placement and gradient routing decisions. This rigorous formulation lays the foundation for the algorithmic solutions proposed in subsequent sections.

A. System Model

Cluster network and training architecture. We model the distributed training cluster as a graph comprising compute nodes N, switches S, and network links $\mathcal{E}$. Each DML job j employs a standard PS architecture, with a set of workers $W_j = \{w_j^1, \ldots, w_j^{|W_j|}\}$ and a dedicated PS node d_j. In each iteration, workers compute gradients locally and transmit them for aggregation prior to the global model update at the PS.

Gradient aggregation model. We leverage the in-network aggregation (INA) capabilities of programmable switches, incorporating the following three key design strategies to enhance efficiency and flexibility. (1) Hierarchical aggregation: INA-capable switches serve as intermediate aggregation points, enabling partial gradient aggregation prior to final aggregation at the parameter server (PS). This hierarchical structure alleviates load on the PS and reduces overall network traffic. (2) Flexible intra-job routing: Workers within each job may distribute their gradient flows across multiple INA switches or send them directly to the PS, rather than being restricted to a single aggregator. This flexibility allows adaptive load balancing and improved utilization of link resources. Furthermore, each INA switch can aggregate gradients for multiple jobs concurrently, subject to its processing capacity. (3) Single in-network aggregation hop: Following prior work [24], [26], each gradient flow may be aggregated by at most one INA switch before reaching the PS; flows not aggregated in-network are sent directly to the PS. This restriction simplifies flow management, avoids the complexity of multi-hop aggregation, and maintains tractability while retaining the key benefits of INA. Together, these design choices enable efficient resource utilization, reduced communication overhead, and manageable system complexity in the optimization of INA for multi-job DML workloads.

B. Mathematical Formulation

We now formulate the joint optimization of INA function placement and gradient flow routing. Key notations are summarized in Table II.

Objective: Our objective is to minimize the makespan t, i.e., the total time required for all jobs to complete gradient aggregation. Let m_j denote the total gradient data volume of job j, and let γ_j denote the sending rate at which each worker of job j transmits gradients to aggregation points. The completion constraint for each job and makespan is:

$$\frac{m_j}{\gamma_j} \leq t, \quad \forall j \in J \tag{1}$$

To linearize the problem, we introduce a variable $\alpha = \frac{1}{t}$ and restate the objective as maximizing α, yielding the equivalent formulation:

$$\max \alpha \tag{2}$$

$$\frac{\gamma_j}{m_j} \geq \alpha, \quad \forall j \in J \tag{3}$$

Constraints: The optimization is subject to the following constraints.

1) INA placement budget: The number of network switches within the network equipped with INA functionality is limited by budget K:

$$\sum_{s \in S} x_s \leq K \tag{4}$$

where x_s is a binary variable indicating INA deployment at switch s.

2) Gradient flow routing: For each worker w in job j, its gradient data must be routed entirely to a single aggregation node v, either an INA-enabled switch or the PS, i.e., no partial splitting:

$$y_{jws} \leq x_s, \qquad \forall j \in J, w \in W_j, s \in S \tag{5}$$

$$\sum_{v \in S \cup \{d_j\}} y_{jwv} = 1, \qquad \forall w \in W_j, j \in J \tag{6}$$

where y_{jwv} denotes whether worker w in job j choose node v as its aggregation point (1 if chosen, 0 otherwise).

3) INA switch processing capacity: The total incoming gradient flow at an INA-enabled switch cannot exceed its processing capacity C_s:

$$\sum_{j \in J} \sum_{w \in w_j} \gamma_{jws} \leq C_s \times x_s, \quad \forall s \in S \tag{7}$$

where γ_{jws} is the flow rate from worker w in job j to INA switch s.

4) Network link capacity: The total gradient traffic traversing any link e must not exceed its bandwidth C_e^{bw}:

$$\sum_j \left(\ell_{je}^{\text{worker}\to\text{switch}} + \ell_{je}^{\text{worker}\to\text{PS}} + \ell_{je}^{\text{switch}\to\text{PS}}\right) \leq C_e^{\text{bw}}, \forall e \tag{8}$$

TABLE II
KEY NOTATIONS USED IN PROBLEM FORMULATION

Notations	Description
J, j	Set of DML jobs; index of a job
W_j, w	Set of workers of job j; index of a worker in W_j
d_j	Parameter Server (PS) node of job j
m_j	Total gradient data volume of job j
S, s	Set of switches; index of a switch in S
E, e	Set of links; index of a link in E
C_e^{bw}	Bandwidth capacity of link e
C_s	INA processing capacity of switch s
$I_{u\to v}(e)$	Indicator: 1 if link e is on the path from node u to node v; 0 otherwise
K	INA budget: maximum number of switches deployable with INA functionality
Variables	**Description**
x_s	Binary: 1 if INA function is deployed on switch s, 0 otherwise
y_{jwv}	Binary: 1 if worker w of job j route its gradient flow to node $v \in S \cup \{d_j\}$ for aggregation; 0 otherwise
γ_j	Gradient sending rate of job j
γ_{jw}	Gradient sending rate of worker w in job j
γ_{jws}	Rate at which worker w of job j sends gradient flow to switch s for aggregation
γ_{jwd_j}	Rate at which worker w of job j sends gradient flow directly to PS d_j
γ_{js}	Rate at which INA switch s sends its aggregated gradient flow to PS for job j
t	Makespan: time to complete gradient aggregation for all jobs in J
α	Auxiliary variable: inverse of makespan $(1/t)$

where $\ell_{je}^{X\to Y}$ denotes the gradient traffic on link e from X to Y for job j. The total traffic on link e for each job j consists of: (1) $\ell_{je}^{\text{worker}\to\text{switch}}$, gradients from workers to INA switches, computed as $\sum_{w\in W_j}\sum_{s\in S} I_{w\to s}(e)\cdot\gamma_{jws}$; (2) $\ell_{je}^{\text{worker}\to\text{PS}}$, gradients sent directly from workers to the PS, calculated as $\sum_{w\in W_j} I_{w\to d_j}(e)\cdot\gamma_{jwd_j}$; (3) $\ell_{je}^{\text{switch}\to\text{PS}}$, aggregated gradients sent from INA switches to the PS, given by $\sum_{s\in S} I_{s\to d_j}(e)\cdot\gamma_{js}$. Here, $I_{u\to v}(e)$ equals 1 if link e lies along the path from node u to v, and 0 otherwise.

5) Gradient sending rate constraint: Finally, (9)–(13) govern the logic of gradient transmission rates. Specifically, (9) and (10) utilize a Big-M formulation to ensure logical consistency, gradient traffic can only flow through assigned aggregation points. Most importantly, (13) captures the physical characteristic of Synchronous SGD: the overall training speed γ_j is bounded by the slowest worker (i.e., $\gamma_j \le \gamma_{jw}, \forall w$). This constraint explicitly models the straggler effect, forcing the optimizer to improve the performance of the bottleneck worker to maximize global throughput.

$$\gamma_{jws} \le M\cdot y_{jws}, \quad \forall j,w,s \tag{9}$$

$$\gamma_{jwd_j} \le M\cdot y_{jwd_j}, \quad \forall j,w,s \tag{10}$$

$$\gamma_{jws} \le \gamma_{js}, \quad \forall j,w,s \tag{11}$$

$$\gamma_{jw} = \sum_{v\in S}\gamma_{jws} + \gamma_{jwd_j}, \quad \forall w\in W_j, j\in J \tag{12}$$

$$\gamma_j \le \gamma_{jw}, \quad \forall j\in J, w\in W_j \tag{13}$$

where M is a sufficiently large constant, and other variables are as previously defined (see Table II).

Complete formulation: The overall joint optimization problem can be expressed as:

$$\text{P1} \qquad \max \alpha$$

$$\text{s.t.} \qquad (3)\text{–}(13)$$

$$var.\begin{cases} x_s\in\{0,1\}, & \forall s\in S \\ y_{jwv}\in\{0,1\}, & \forall j\in J, w\in W_j, v\in S\cup\{d_j\} \\ \gamma_{jwv}\ge 0, & \forall j\in J, w\in W_j, s\in S\cup\{d_j\} \\ \gamma_{jw}\ge 0, & \forall j\in J, w\in W_j \\ \gamma_{js}\ge 0, & \forall j\in J, s\in S \\ \gamma_j\ge 0, & \forall j\in J \\ \alpha>0 \end{cases}$$

After obtaining the optimal α from the above formulation, the corresponding makespan can be computed as $t=\frac{1}{\alpha}$.

Complexity analysis: The problem P1 is formulated as a mixed-integer linear program (MILP) as it involves both contentious variables ($\boldsymbol{\gamma}$ and α) and binary variables for both INA function placement ($\boldsymbol{x}$) and gradient routing ($\boldsymbol{y}$). This inherent discreteness and the coupled nature of these decisions render P1 an NP-hard problem. Specifically, the decision space for the $|S|$ INA placement variables offers $\mathcal{O}(2^{|S|})$ possibilities, while the $\mathcal{O}(|J|\cdot W_{\max}\cdot|S|)$ gradient routing variables introduce significant combinatorial complexity. The exponentially growing search space makes it computationally infeasible to find an exact solution using brute-force or standard MILP solvers for large network instances.

V. LLMINA: Algorithmic Framework and Design Details

A. Algorithmic Framework Overview

To address the computational complexity of problem P1, we adopt a divide-and-conquer strategy by decomposing it into two sequential subproblems:

- INA function placement (IFP): This subproblem focuses on determining the optimal deployment locations for INA functionalities within the network, with respect to INA budget constraints. Specifically, it solves for the integer variables $\boldsymbol{x}$ in P1, which indicate the placement decisions.
- Gradient routing (GR): Given the INA placement $\boldsymbol{x}$ obtained from the IFP subproblem, this stage determines, for each DML job, the aggregation node (either an INA-enabled switch or a PS) to which each worker should route its generated gradient flow, subject to link bandwidth and INA switch processing capacity constraints. This subproblem specifically solves for the integer variables $\boldsymbol{y}$ in P1.

This hierarchical decomposition enables efficient handling of the assignment of integer variables $\boldsymbol{x}$ and $\boldsymbol{y}$ in P1, thereby simplifying the overall solution process.

LLMINA addresses the joint optimization challenge through a two-stage logic designed for rapid execution. First, the IFP problem is solved using a pre-designed heuristic, which is evolved by an LLM entirely offline. At runtime, this optimized algorithm is directly invoked without any online LLM interaction. Second, the GR problem is handled by an efficient manual heuristic. By decoupling these tasks and eliminating online LLM interaction, LLMINA ensures sub-second decision latency while effectively balancing INA and link resources to minimize communication makespan. Detailed algorithm designs for each stage follow in the subsequent subsections.

B. LLM-Guided Evolutionary Heuristic for IFP

Fig. 2 illustrates our LLM-driven heuristic design for the IFP problem, which adapts the Evolution of Heuristics (EoH) framework [73]. We leverage the advanced reasoning and code generation capabilities of Large Language Models (LLMs) to automatically generate and iteratively refine heuristic functions tailored for the specific INA placement constraints of DML workloads. As shown, the framework integrates prompt engineering with a population-based evolutionary search: the LLM is utilized to generate the core heuristic logic (e.g., greedy rules), and the subsequent evolutionary mechanism systematically evaluates and optimizes these generated functions over a predetermined number of generations (N). Upon meeting the termination condition, the single heuristic algorithm with the highest fitness score is selected and outputted as the final optimized IFP algorithm for deployment. The following subsections detail the specific steps involved in each iteration of this evolutionary loop.

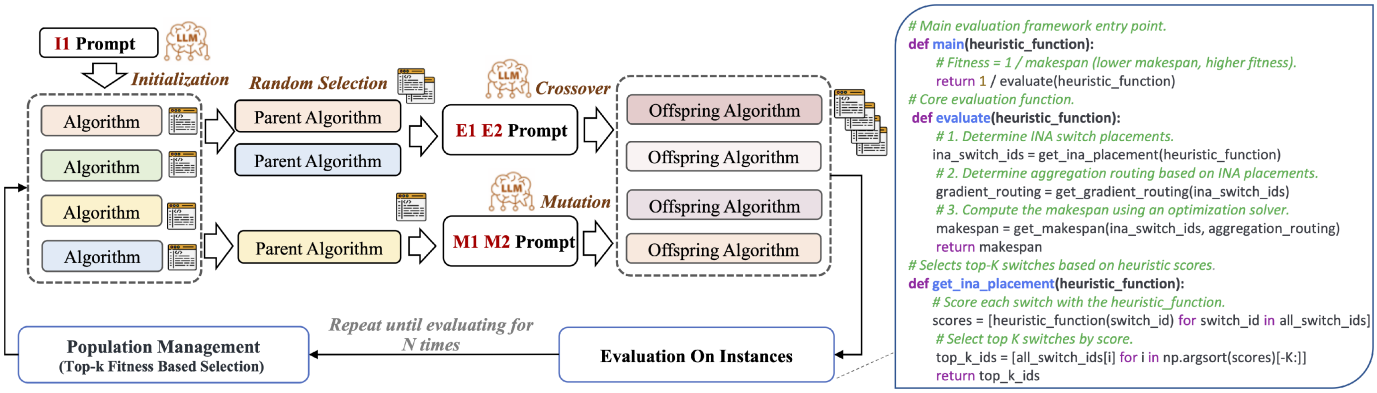


Fig. 2. The framework for offline evolutionary synthesis of INA placement heuristics. The process leverages LLMs to iteratively generate and refine algorithm code (left) to maximize fitness, defined as the inverse of the makespan (right), prior to deployment.

Prompt-based Initialization: The evolutionary process begins with prompt-driven algorithm generation. The LLM (e.g., GPT-4o-min) is provided with a carefully crafted prompt (**I1**), which details the IFP objectives, constraints, input/output formats, and practical considerations (as depicted in Fig. 8). This prompt also instructs the LLM to generate complete a Python algorithm that assign a priority score to a network switch. When scoring a switch, we encourage LLM to considering factors such as its distance to workers, proximity to PS, and traffic reduction achieved through on-switch gradient aggregation. To promote diversity and creativity within the initial heuristic population, the prompt further requests the LLM to "describe your new algorithm and main steps in one sentence" prior to code generation.

Evolutionary operators: The heuristic population is successively improved across generations using two core evolutionary operators, crossover and mutation, each facilitated by specialized prompt templates:

- Crossover (**E1**, **E2**): Given two parent algorithms, the LLM is prompted to either to synthesize a structurally distinct new heuristic (E1), or to identify the common backbone for further elaboration (E2), thereby facilitating the recombination of salient features (see Fig. 8).
- Mutation (**M1**, **M2**): The LLM is prompted to augment a given heuristic by either modifying the algorithmic procedure (M1) or or adjusting hyperparameters and coefficients within the scoring function (M2), enhancing both topological and parametric diversity.

Algorithm Evaluation: Each LLM-generated heuristic is evaluated using the procedure given in see the right panel of Fig. 2. Specifically, we first compute INA candidate scores using the heuristic and select the top K switches for INA function deployment, subject to resource budget. For the resulting placement, solve the gradient routing subproblem (see Sec. V-C) to obtain the gradient communication makespan. The fitness of a heuristic is defined as the inverse of the makespan, thus favoring algorithms that achieve faster aggregation.

Population Management: Following the evaluation step, the population management module implements a Top-K fitness based selection strategy. This process ranks the evaluated heuristics (including the offspring from Crossover and Mutation) according to their fitness scores. To maintain a constant population size, a fixed number (q) of heuristics with the highest fitness scores are selected to serve as the parent pool for the next evolutionary generation.

Evolutionary process and example heuristic: The framework iteratively applies prompt-based initialization (I1), crossover (E1, E2), and mutation (M1, M2) over multiple generations, as shown in Fig. 2. After a pre-defined number of generations, the top-performing heuristics are retained as the final output. An illustrative instance is depicted in Fig. 9: here, the LLM synthesizes a heuristic (*calculate_switch_priority*) that scores each switch based on communication cost, proximity to workers and PSs (using shortest-path computation), and type-specific weighting. For example, switches located near a larger number of workers or PSs, particularly those at advantageous edge locations, receive higher priorities to host INA functionalities.

By integrating sophisticated prompt engineering with the generative reasoning capabilities of LLMs, our evolutionary framework enables the automated discovery of heuristics closely tailored to the INA placement problem, harnessing both the creativity and analytical strength of LLMs.

C. Optimization-Based Bottleneck-Aware Heuristic for GR

We present a heuristic for routing gradient flows from multiple workers to the available aggregation locations (i.e., INA switches and the PS) identified by the IFP problem. This is formalized as subproblem $P1_x(\boldsymbol{y})$, an MILP involving only binary variable $\boldsymbol{y}$, for which standard "relax and round" techniques are often employed. However, these conventional approaches do not account for the unique characteristics and constraints of DML jobs, often resulting in imbalanced assignments or resource contention, as shown in our experiments. Rather than directly rounding the fractional values of $\boldsymbol{y}$, our approach leverages the gradient rate variables γ obtained from the relaxed solution to guide more effective routing decisions.

Our heuristic is implemented as a two-phase process. First, we solve the LP relaxation of $P1_x(\boldsymbol{y})$, yielding optimal rates $\boldsymbol{\gamma_{jws}}$ and $\boldsymbol{\gamma_{jwd_j}}$ for each worker to send gradient data towards INA switches and the PS, respectively. These results are then

used to inform the subsequent two-phase assignment workflow, described as follows:

Phase I: Initial Assignment. This phase constructs an initial mapping from workers to aggregators based on γ_{jws} and γ_{jwd_j}. Due to synchronization constraints in distributed training, workers within the same DML job typically operate at a uniform sending rate determined by the slowest member. To reflect physical locality, we partition the workers of each job according to their Top-of-Rack (ToR) switch connection: those connected to the same ToR form a group, denoted as W_k^j for the k-th group in job j. For each group W_k^j:

- We compute group-level sending rates $\gamma_s^k = \min_{w \in W_k^j} \gamma_{jws}$ for each INA switch $s \in S^{\text{INA}}$, and $\gamma_{d_j}^k = \min_{w \in W_k^j} \gamma_{jwd_j}$ for the PS, capturing the bottleneck rate within the group.
- Workers are then assigned to aggregators in proportion to these rates. Specifically, the number of workers assigned to each aggregator $\ell \in S^{\text{INA}} \cup PS$ is approximately $n_\ell^k = \lfloor |W_k^j| \times \rho_\ell^k \rfloor$, where $\rho_\ell^k = \frac{\gamma_\ell^k}{\sum_{\ell' \in S^{\text{INA}} \cup d_j} \gamma_{\ell'}^k}$. We ensure $\sum_{\ell \in S^{\text{INA}} \cup d_j} n_\ell^k = |W_k^j|$. This proportional allocation routes more gradient flows to aggregators with higher capacity, closely emulating the optimal LP solution.

Phase II: Bandwidth-Aware Adjustment. As the PS often represents a major communication bottleneck, this phase refines the assignment to avoid PS overloading:

- We calculate $N_{d_j}^{\text{cap}} = \frac{\text{uplink\_cap}(d_j)}{\gamma_j}$, the maximum number of flows that can traverse through the PS link at the rate γ_j derived from the relaxed LP, where uplink_cap(d_j) is the uplink bandwidth capacity of the PS.
- The actual load generated by the initial assignment, $N_{d_j}^{\text{load}}$, is the sum of direct worker flows to the PS, plus aggregator-to-PS edge flows: $N_{d_j}^{\text{load}} = \sum_k (\sum_{w \in W_k^j} \mathbb{I}(w \to d_j) + \sum_{s \in S^{\text{INA}}} \mathbb{I}(s \to d_j))$, where $\mathbb{I}(\cdot)$ is an indicator for the routes to PS d_j.
- If $N_{d_j}^{\text{load}} > N_{d_j}^{\text{cap}}$, we iteratively reassign worker flows from the PS to INA switches with available capacity and less load. If necessary, we reroute aggregated flows from over-assigned INA switches to others, ensuring all adjustments maintain feasible loads at the PS, $N_{d_j}^{\text{load}} \leq N_{d_j}^{\text{cap}}$.

By identifying group-wise rate bottlenecks and adaptively balancing aggregator utilization, our method more closely aligns routing decisions with the relaxed rate solutions. This approach stands in contrast to conventional rounding methods, which may fail to fully exploit the optimality of the relaxed solutions.

VI. Performance Evaluation

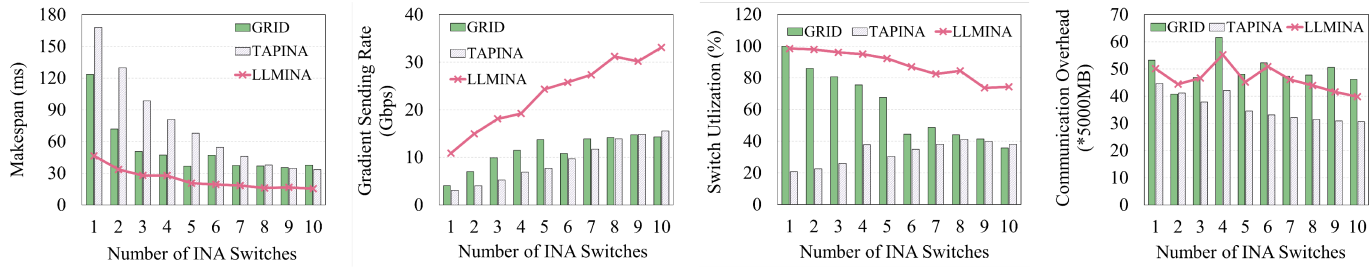
A. Experimental Methodology

Network topologies: We evaluate LLMINA on two representative data center network topologies: Spine-leaf and Fat-tree. The Spine-leaf topology features 5 spine switches, 10 leaf (ToR) switches, and 200 servers, with each leaf switch connected to 20 servers via 100 Gbps links; each leaf–spine link operates at 400 Gbps. The Fat-tree topology includes 4 core switches, 8 aggregation switches, 8 edge (ToR) switches, and 160 servers, with each server linked to its ToR switch at 100 Gbps, edge–aggregation links at 500 Gbps, and aggregation–core links at 250 Gbps, capturing typical oversubscription in hierarchical designs. To model realistic in-network computation constraints, we limit INA-enabled switches to a budget of K, each supporting up to 750 Gbps of processing throughput. This setup enables us to assess INA placement under practical bandwidth and compute capacity limitations.

Baselines: We compare LLMINA with two state-of-the-art approaches: TAPINA [27] and GRID [24]. TAPINA jointly optimizes INA function placement and job prioritization, requiring all workers of a job to aggregate gradients at the same INA switch (or PS), while allowing INA switches to be shared across jobs to conserve switch budget. GRID is designed to maximize gradient sending rates through optimal routing, but it targets single-job scenarios and does not optimize INA placement. For a fair comparison, we deploy INA functions at the same locations as LLMINA when evaluating GRID, and sequentially process each job to adapt it for multiple jobs.

Performance metrics: Our primary performance evaluation metric is the *gradient aggregation makespan*. This metric quantifies the maximum time interval required for gradient aggregation completion across all concurrent DML jobs. For an individual job, the time to finalize gradient aggregation is intrinsically governed by two key factors: the volume of gradient data (directly proportional to model size) and the achieved gradient sending rate. Consequently, the overall system makespan is dictated by the job with the longest aggregation time, thereby effectively pinpointing performance bottlenecks and comprehensively assessing the aggregate efficiency of the system. To provide a more holistic understanding of system performance beyond the makespan, we also collect and analyze the following metrics: *the gradient sending rate* per DML job, *the average INA resource utilization* of the network switches, and the *total communication overhead*. The gradient sending rate per job offers insight into the individual job's network throughput capabilities. The INA resource utilization of switches is crucial because it reflects the degree to which the network infrastructure is being utilized by the aggregation process. The total communication overhead, qualified by the aggregate volume of traffic traversing all network links, quantifies the overall network burden imposed by the DML jobs.

Foundation models: Our evaluation of LLMINA uses several popular LLMs, including GPT-4o-mini, GPT-3.5-Turbo, Llama-3.1-8B, and Qwen2.5-7B-Instruct. Unless otherwise specified, LLMINA employs GPT-4o-mini as the default base model due to its consistently strong performance (see Table IV for details). Closed-source GPT models (e.g., GPT-4o-mini, GPT-3.5-Turbo) are accessed via API calls, while open-source models (e.g., Llama-3.1-8B, Qwen2.5-7B-Instruct) are deployed locally on NVIDIA RTX 3090 GPUs for inference. The LLM-driven INA placement algorithm is developed on the foundation of LLM4AD [94], an open-source Python-based platform that leverages LLMs for automatic algorithm design.



(a) Gradient aggregation makespan (b) Gradient sending rate (c) INA resource utilization (d) Communication overload

Fig. 3. Impact of INA resource budgets. LLMINA consistently reduces makespan (up to 74.1%) and boosts sending rate (up to 3.7×) by maintaining superior and sustained INA resource utilization.

TABLE III
SUMMARY OF CONSTRAINT GROUPS IN P1 FORMULATION

Constraint Group	Equation	Physical Intuition / Feasibility
Placement Budget	Eq. (4)	Limits hardware costs by bounding the number of deployed INA switches within budget K.
Flow Routing	Eq. (5)–(6)	Enforces each worker connects to exactly one aggregator to avoid flow splitting.
Switch Capacity	Eq. (7)	Workload feasibility: total packet rate must not exceed the switch ASIC's processing capability (C_s).
Link Bandwidth	Eq. (8)	Congestion control: aggregate traffic on any link must stay within physical bandwidth limits (C_e^{bw}).
Rate Logic	Eq. (9)–(13)	Couples routing decisions with flow rates and captures the straggler effect: synchronous training speed is bottlenecked by the slowest worker (Eq. 13).

The LLM-driven heuristic search was conducted over 10 evolutionary generations using a Top-K Fitness-Based Selection strategy with a population size $q = 4$. In each generation, 4 offspring were produced via crossover (E1, E2) and mutation (M1, M2). Heuristic fitness was rigorously defined as the inverse of the normalized makespan, averaged across a comprehensive set of instances varying in switch processing capacity, the number of concurrent DML jobs, and the INA switch budget, ensuring optimal robustness across diverse conditions.

B. General Performance Results

First, we evaluate the overall effectiveness of LLMINA under varying resource constraints and workload characteristics. Specifically, we conduct two sets of experiments to compare the overall performance: (i) under different INA resource budgets, and (ii) across diverse workloads.

(1) *Performance comparison across varying INA resource budgets:* To thoroughly investigate the impact of INA resource constraints, we vary the number of INA-capable switches in the network. Each experimental trial involves eight concurrent DML jobs. For each job, the model size is randomly sampled from [10, 1000] MB. The number of workers per job (inclusive of a single parameter server) is selected from the interval [10, $S_{\max}$]. The value of $S_{\max}$ is set to ensure full utilization of the available server pool guaranteeing a minimum worker count per job. Within each job, one worker is randomly designated as PS; all other nodes functioned as workers responsible for gradient computation and transmission. Workers are consolidated and deployed under a consecutive set of ToR switches.

As depicted in Fig. 3(a) and (b), the gradient aggregation makespan and gradient sending rate exhibit the expected inverse correlation. All methods consistently demonstrate a reduction in makespan and a corresponding improvement in sending rate as the number of INA-capable switches, increases from 1 to 10. This observed trend validates the efficacy of enhanced in-network aggregation capacity, as a higher availability of INA switches enables a larger proportion of gradient traffic to be aggregated locally, thereby mitigating link contention and augmenting overall communication efficiency. In direct comparisonour proposed approach, LLMINA, consistently outperforms both GRID and TAPINA across all INA budget settings. For instance, under the constraint of budget $K = 1$, GRID incurs a makespan of 120 ms and TAPINA approximately 170 ms. In contrast, LLMINA achieves a substantially lower makespan of 45 ms. This translates to a reduction of 53.3% compared to GRID and 74.1% compared to TAPINA. respectively, LLMINA demonstrates a significantly higher rate of 10 Gbps. This constitutes an impressive performance gain of 150% over GRID and 233% over TAPINA. Even under a relaxed budget of $K = 8$, LLMINA maintains substantial performance advantages, achieving a 56.3% reduction in makespan over GRID and 57.3% over TAPINA, while boosting the sending rate by 120.1% and 124.5%, respectively.

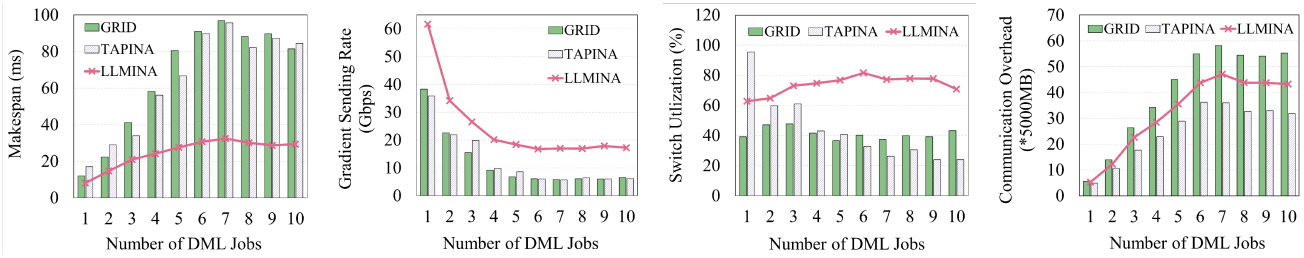
Complementing these efficiency gains, Fig. 3(c) compares the average INA resource utilization. GRID exhibits high utilization, starting at 100% for $K = 1$ and decreasing sharply to 38–44% for larger K, suggesting heavy initial INA resource leveraging that saturates at lower K. TAPINA shows consistently lower utilization, beginning at 21% for $K = 1$ and and moderately increasing to 35–38% for larger K, indicating a more conservative resource usage strategy. LLMINA demonstrates an initial high utilization of 98% at $K = 1$ and stabilizes between 75–77% for larger K. This pattern suggests LLMINA intelligently scales down its INA resource demand as more switches become available, while maintaining significant usage, indicating an efficient allocation strategy that balances resource usage with communication performance.

Finally, Fig. 3(d) reports the communication overhead. TAPINA achieves the lowest overhead due to its proximity-based routing strategy, which selects the nearest INA switch for each job's workers. However, this localized optimization, while reducing immediate overhead, limits TAPINA's ability to fully

TABLE IV
MAKESPAN (MICROSECONDS) COMPARISON USING DIFFERENT LLMS.

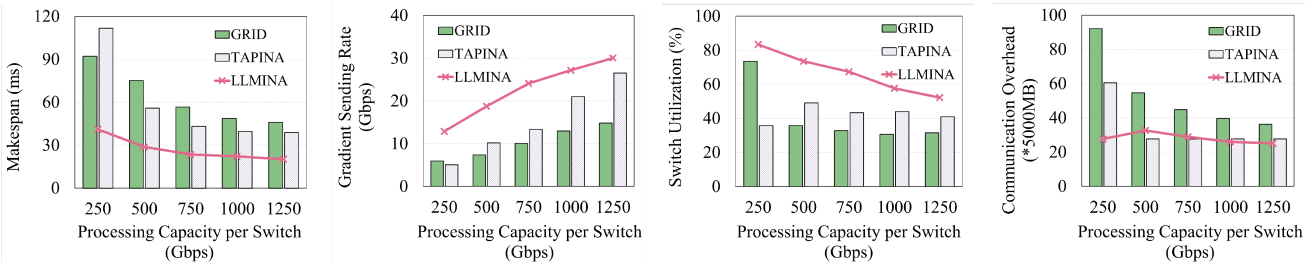
No. of DML jobs	1	2	3	4	5	6	7	8	9	10
GPT-4o-mini	**7.84**	**10.78**	**13.30**	**14.87**	**17.21**	**21.05**	**20.59**	**17.71**	**18.10**	**17.97**
M2 (GPT-3.5)	7.84	13.39	16.91	19.42	21.14	26.09	28.72	26.78	25.51	24.87
M3 (Llama)	16.37	16.37	18.15	21.13	25.03	27.88	27.88	26.49	26.40	27.43
M4 (Qwen)	13.68	16.02	18.15	21.13	25.03	27.88	27.85	26.49	26.40	27.43

Note: M1=GPT-4o-mini, M2=GPT-3.5-Turbo, M3=Llama-3.1-8B(Local), M4=Qwen2.5-7B-Instruct(Local)



(a) Gradient aggregation makespan (b) Gradient sending rate (c) INA resource utilization (d) Communication overload

Fig. 4. Impact of workload density. LLMINA demonstrates superior robustness, achieving a lower makespan (up to 68% reduction) and over 2.9× higher sending rates, especially where baselines degrade sharply.



(a) Gradient aggregation makespan (b) Gradient sending rate (c) INA resource utilization (d) Communication overload

Fig. 5. Impact of switch processing capacity. LLMINA efficiently exploits INA resources, reducing makespan by up to 63.2%. Its performance is often limited by network link bandwidth, rather than switch processing capacity.

leverage available INA resources, particularly under larger K, ultimately leading to load imbalance and reduced resource utilization LLMINA and GRID exhibit similar levels of communication overhead, with differences within 15%. Critically, despite this comparable overhead, LLMINA achieves a significantly lower makespan.

(2) *Performance comparison across varying workloads:* We further evaluate the impact of varying workloads, specifically the number of concurrent DML jobs. For these experiments, the number of INA switches is set to three in the network.

As shown in Figs. 4(a) and 5(d), increasing job concurrency from 1 to 6 intensifies network contention, causing makespan to rise and sending rate to fall across all methods. Beyond 6 jobs, a slight improvement occurs as localized communication reduces inter-rack traffic. LLMINA consistently outperforms GRID and TAPINA, particularly under high contention. For instance, with three jobs, LLMINA offers 49.1% lower makespan and 70.6% higher sending rate than GRID. With eight jobs, these advantages extend to 65.8% and 179.5%, respectively. GRID degrades sharply due to its greedy routing, while TAPINA's local optimization limits scalability. Fig. 4(c) shows that LLMINA and GRID exhibit decreasing INA switch utilization as job count rises, reflecting balanced workload distribution and approaching aggregation saturation. Conversely, TAPINA's utilization increases, peaking at 95.4% with one job. This high single-job utilization stems from its memory-aware heuristic that dedicates one switch per job, failing to leverage remaining resources and leading to poor scalability and load imbalance under higher concurrency.

Fig. 4(d) reports communication overhead. As job count increases, overhead grows nearly linearly for all methods due to increased traffic. TAPINA achieves the lowest overhead via proximity routing, but this limits resource utilization. LLMINA and GRID show comparable overhead (within 20%), yet LLMINA achieves a significantly lower makespan.

(3) *Performance comparison across varying INA switch capacity:* We further investigate the impact of INA switch processing capacity by varying it from 250 Gbps to 1250 Gbps.

As illustrated in Fig. 5(a) and (b), increasing processing capacity naturally reduces the aggregation makespan and boosts sending rates across all schemes. However, LLMINA consistently outperforms the baselines in both resource-constrained and resource-abundant scenarios. Specifically, under a constrained capacity of 250 Gbps, LLMINA reduces the makespan by approximately 55.4% compared to GRID and 63.2% relative to TAPINA. Even

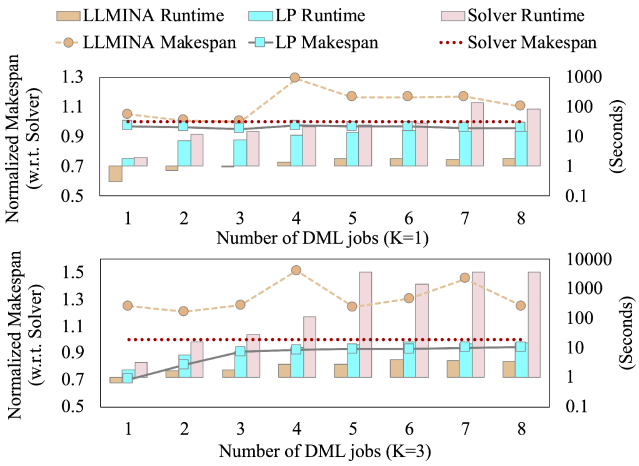


Fig. 6. Optimality gap and runtime analysis. LLMINA closely tracks the LP relaxation lower bound, offering a tight approximation to the optimal solution with negligible runtime overhead (< 1 s) compared to the solver's exponential complexity.

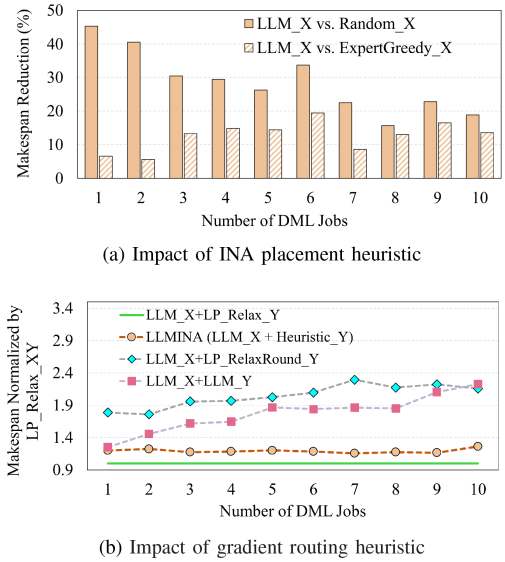


Fig. 7. Component-wise performance comparison. (a) Placement analysis: LLMINA reduces makespan by 45% over ExpertGreedy. (b) Routing analysis: LLMINA outperforms both and LP-rounding baseline and pure LLM-guided routing.

with an ample capacity of 1250 Gbps, LLMINA sustains its advantage and achieves makespan reductions of 56.0% over GRID and 48.0% over TAPINA. This demonstrates its ability to fully exploit the additional computing power.

Fig. 5(c) offers further insights into resource efficiency, showing that LLMINA exhibits a decreasing trend in switch utilization as processing capacity increases, indicating it fully utilizes path resources until constrained by link bandwidth limits, rather than switch processing constraints. In contrast, GRID and TAPINA show less pronounced utilization trends, suggesting their performance is limited by suboptimal use of switch capacity. Notably, LLMINA maintains higher resource utilization than both baselines across all tested conditions.

Finally, the communication overhead analysis in Fig. 5(d) confirms that LLMINA achieves a strategic balance. It reduces traffic volume by 54.2% compared to GRID at low capacity while avoiding the throughput limitations inherent in the rigid routing strategy employed by TAPINA.

C. Component-Wise Performance Results

(1) *Optimality and runtime analysis:* We evaluate LLMINA's optimality and efficiency by comparing its makespan and execution time against the LP-relaxation lower bound and an exact solver. As shown in Fig. 6, LLMINA consistently tracks the LP lower bound, maintaining a tight approximation to the optimal solution across varying job scales ($K = 1, 3$). Crucially, while the exact solver exhibits exponential complexity—with runtimes escalating to over 10^3 seconds, LLMINA consistently completes within sub-second latency ($< 1s$). These results demonstrate that LLMINA effectively sidesteps the state-space explosion of traditional optimization, providing near-optimal results with negligible overhead suitable for DML environments.

The results above show that the superior performance of LLMINA derives from its joint optimization of INA placement and gradient flow routing. To better understand the individual contribution of each component, we further conduct two sets of ablation experiments, each isolating one aspect.

(2) *Effectiveness of INA placement policy:* We evaluate placement efficiency by comparing our LLM-generated placement policy (LLM_X), against two baselines under a unified routing policy: Random_X and ExpertGreedy_X. The latter ranks candidate switches using a multi-objective metric considering network proximity and gradient absorption capacity. As shown in Fig. 7(a), LLM_X consistently achieves the lowest makespan across all scenarios. It outperforms Random_X by 28% on average (up to 45% in light workloads). More importantly, LLM_X yields a 13% average (19% peak) improvement over the ExpertGreedy_X heuristic. These results demonstrate that while human-defined heuristics rely on static scoring, LLM-guided placement heuristic effectively captures the intricate and nonlinear dependencies between topology and workload dynamics. By leveraging high-level reasoning, LLMINA identifies superior placement configurations that transcend the limitations of manual designs, significantly accelerating gradient aggregation.

(3) *Effectiveness of gradient flow routing policy:* We also evaluate routing efficiency by comparing LLMINA against three benchmarks: the theoretical LP-relaxed lower bound (coined with LLM_X+LP_Relax_Y), a standard max-rounding baseline (LLM_X+LP_RelaxRound_Y), and a pure LLM-generated heuristic (coined with LLM_X+LLM_Y). As shown in Fig. 7(b), the max-rounding baseline yields significant performance degradation ($1.78\times$–$2.29\times$ of the LP-relaxed lower bound). This stems from its rigid binary assignment, which disrupts the delicate load balancing of the relaxed solution and triggers congestion hotspots. Similarly, the end-to-end LLM heuristic only marginally outperforms the baseline. This suggests that the numerically constrained nature of gradient routing favors specialized algorithms over LLMs, as the latter struggle to navigate strictly constrained search spaces despite their excellence in abstract decision-making. In contrast, LLMINA consistently achieves near-optimal performance (within $1.16\times$–$1.26\times$ of the bound). By employing a proportional routing strategy and explicitly accounting for PS-side bandwidth bottlenecks, LLMINA

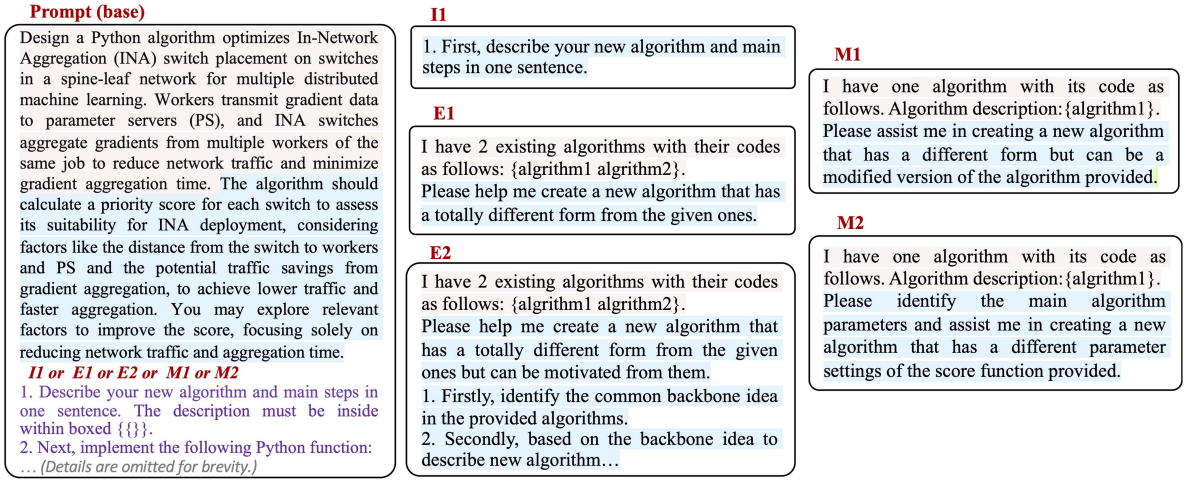


Fig. 8. Prompt engineering examples: Initialization (I1), Evolutionary (E1, E2), and Mutation (M1 M2).

TABLE V
AVG. TOKEN USAGE AND COSTS.

Model ID	Input (k)	Output (k)	Cost ($)
M1 (GPT-4o)	48.872	18.849	0.01864
M2 (GPT-3.5)	**59.556**	**27.880**	**0.02978**
M3 (Llama)	51.078	28.498	0.000
M4 (Qwen)	59.276	23.186	0.000

```
def calculate_switch_priority(
    graph, num_jobs, job_sizes, switch_id,
    workers_per_job, ps_list_for_jobs, switch_type
):
    """ Calculate the priority score of a switch for INA placement."""
    comm_score = sum(len(workers) for workers in
workers_per_job)
    prox_score = 0.0
    for i in range(num_jobs):
        for worker in workers_per_job[i]:
            dist = shortest_path_length(graph, switch_id, worker)
            if dist > 0: prox_score += job_sizes[i] / (dist ** 2)
        dist = shortest_path_length(graph, switch_id,
ps_list_for_jobs[i])
        if dist > 0: prox_score += job_sizes[i] / (dist ** 2)
    if switch_type == "edge_switch": weight = 1.5
    elif switch_type == "core_switch": weight = 1.2
    else: weight = 1.0
    return (prox_score + comm_score) * weight
```

Fig. 9. Heuristic generated by LLM for evaluating the suitability of a switch for INA function placement.

effectively avoids uplink overload. These results indicate that LLMs are currently better suited as co-design tools for high-level abstraction rather than direct solvers for complex, low-level network optimization.

(4) *Sensitivity to LLM capabilities and operational costs:* We further evaluate the impact of different LLMs on heuristic quality and operational overhead. As shown in Table IV, GPT-4o-mini consistently yields the most efficient routing and placement strategies, outperforming GPT-3.5-Turbo, Llama-3.1-8B, and Qwen2.5-7B by average margins of 22.1%, 32.4%, and 31.3%, respectively. While GPT-3.5 and the two local models (Llama and Qwen) exhibit comparable performance, the superior reasoning of GPT-4o-mini translates into significantly lower makespans, particularly as the number of DML jobs increases. Beyond performance, Table V highlights the economic and computational efficiency of our approach. GPT-4o-mini not only achieves the best results but also maintains a lower footprint, requiring fewer input/output tokens than GPT-3.5-Turbo and incurring a negligible cost (avg. $0.0186 per run). Notably, although local models (Llama-3.1-8B, Qwen2.5-7B) eliminate API costs, their higher makespans justify the minimal expense of using a more capable model like GPT-4o-mini. These results suggest that for complex network optimization, leveraging advanced yet lightweight models provides an optimal balance between solution quality and resource expenditure.

Why location matters? In summary, the above experimental results validate that *location*, the placement of INA functions and the dynamic routing of flows to those points, is the primary determinant of DML performance. Baselines like GRID and TAPINA fail to map aggregation capacity to heterogeneous demands, causing resource underutilization and severe congestion. By contrast, LLMINA's holistic approach resolves these bottlenecks, achieving near-optimal makespans where others degrade. This confirms that the location decisions requires a unified optimization framework to navigate the intricate interplay between topology and traffic.

VII. CONCLUSION

In this paper, we introduce LLMINA, an effective approach to accelerating gradient aggregation in DML workloads through in-network aggregation (INA). LLMINA addresses the substantial aggregation demands of concurrent training jobs by jointly optimizing INA functions placement and the of gradient flow routing. We formulate this challenge as a mixed integer linear program and introduce a practical two-stage method that combines an LLM-guided heuristic with an optimization-based one. Our extensive experimental evaluations validate the effectiveness of LLMINA, demonstrating that joint optimization of INA placement and routing is crucial to fully realize the performance benefits of INA resources in DML systems. Furthermore, our LLM-driven methodology highlights the significant promise of integrating advanced generative AI to tackle complex network optimization tasks.

Limitations and future work: While LLMINA proves effective for in-network gradient aggreagtion for DML workloads, its generalizability to broader network optimization challenges remains unexplored. Furthermore, our methodology relies on manual sub-problem decomposition. Future work could explore automated partitioning frameworks, leveraging LLMs as agentic orchestrators to autonomously decompose complex tasks. We also plan to integrate real-time feedback to handle network dynamics, ensuring robustness against volatility and shifting traffic patterns.

APPENDIX

See Figs. 8 and 9.

REFERENCES

[1] J. Wang et al., "Generative AI based secure wireless sensing for ISAC networks," *IEEE Trans. Inf. Forensics Secur.*, vol. 20, pp. 5195–5210, 2025.

[2] J. Wang et al., "Generative artificial intelligence assisted wireless sensing: Human flow detection in practical communication environments," *IEEE J. Sel. Areas Commun.*, vol. 42, no. 10, pp. 2737–2753, Oct. 2024.

[3] D. Narayanan et al., "Efficient large-scale language model training on GPU clusters using megatron-LM," in *Proc. Int. Conf. High Perform. Comput. Netw., Storage Anal.*, 2021, pp. 1–15.

[4] H. Zhou, Z. Li, H. Yu, L. Luo, and G. Sun, "NBSync: Parallelism of local computing and global synchronization for fast distributed machine learning in WANs," *IEEE Trans. Serv. Comput.*, vol. 16, no. 6, pp. 4115–4127, Nov.–Dec. 2023.

[5] A. Gangidi et al., "RDMA over ethernet for distributed training at meta scale," in *Proc. ACM SIGCOMM Conf.*, 2024, pp. 57–70.

[6] R. Zhang, X. Chi, W. Zhang, G. Liu, D. Wang, and F. Wang, "Unimodal training-multimodal prediction: Cross-modal federated learning with hierarchical aggregation," *IEEE Trans. Mobile Comput.*, vol. 24, no. 10, pp. 10009–10023, Oct. 2025.

[7] Z. Yuchen, L. Long, L. Zonghang, S. Gang, and Y. Hongfang, "Which cluster meets my deadline: A budget-aware scheduler for distributed training jobs in heterogeneous environments," in *Proc. IEEE Int. Conf. Commun.*, 2025, pp. 1–6.

[8] Y. Shao et al., "Distributed graph neural network training: A survey," *ACM Comput. Surv.*, vol. 56, no. 8, pp. 1–39, 2024.

[9] S. Dong et al., "Mina: Fine-grained in-network aggregation resource scheduling for machine learning service," in *Proc. IEEE Conf. Comput. Commun.*, 2025, pp. 1–10.

[10] Z. Wang, Z. Xu, J. Xi, Y. Wang, A. Shrivastava, and T. E. Ng, "{ZEN}: Empowering distributed training with sparsity-driven data synchronization," in *Proc. 19th USENIX Symp. Operating Syst. Des. Implementation (OSDI 25)*, 2025, pp. 537–556.

[11] A. Sapio et al., "Scaling distributed machine learning with in-network aggregation," in *Proc. Symp. Netw. Syst. Des. Implementation*, 2021, pp. 785–808.

[12] C. Lao et al., "ATP: In-network aggregation for multi-tenant learning," in *Proc. Symp. Netw. Syst. Des. Implementation*, 2021, pp. 741–761.

[13] F. Liang, Z. Zhang, H. Lu, V. Leung, Y. Guo, and X. Hu, "Communication-efficient large-scale distributed deep learning: A comprehensive survey," 2024, *arXiv:2404.06114*.

[14] S. Luo, P. Fan, K. Li, H. Xing, L. Luo, and H. Yu, "Efficient parameter synchronization for peer-to-peer distributed learning with selective multicast," *IEEE Trans. Serv. Comput.*, vol. 18, no. 1, pp. 156–168, Jan.–Feb. 2025.

[15] L. Luo, C. Zhang, H. Yu, Z. Li, G. Sun, and S. Luo, "Energy-efficient hierarchical collaborative learning over LEO satellite constellations," *IEEE J. Sel. Areas Commun.*, vol. 42, no. 12, pp. 3366–3379, Dec. 2024.

[16] Z. Li et al., "Accelerating geo-distributed machine learning with network-aware adaptive tree and auxiliary route," *IEEE/ACM Trans. Netw.*, vol. 32, no. 5, pp. 4238–4253, Oct. 2024.

[17] A. Feng, D. Dong, F. Lei, J. Ma, E. Yu, and R. Wang, "In-network aggregation for data center networks: A survey," *Comput. Commun.*, vol. 198, pp. 63–76, 2023.

[18] J. Xia, W. Wu, L. Luo, D. Guo, and G. Cheng, "In-network aggregation as a generic service for distributed applications," *IEEE Trans. Netw.*, vol. 33, no. 6, pp. 2977–2992, Dec. 2025.

[19] S. Luo, X. Yu, K. Li, and H. Xing, "Releasing the power of in-network aggregation with aggregator-aware routing optimization," *IEEE/ACM Trans. Netw.*, vol. 32, no. 5, pp. 4488–4502, Oct. 2024.

[20] H. Zhu, W. Jiang, Q. Hong, and Z. Guo, "When in-network computing meets distributed machine learning," *IEEE Netw.*, vol. 38, no. 5, pp. 238–246, Sep. 2024.

[21] M. Nickel and D. Göhringer, "A survey on architectures, hardware acceleration and challenges for in-network computing," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 18, no. 1, pp. 1–34, 2024.

[22] Z. Li et al., "Accelerating distributed graph learning by using collaborative {In-Network} multicast and aggregation," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC 25)*, 2025, pp. 233–247.

[23] "Barefoot tofino," accessed 2025. [Online]. Available: https://www.barefootnetworks.com/technology/#tofino.

[24] J. Fang, G. Zhao, H. Xu, C. Wu, and Z. Yu, "GRID: Gradient routing with in-network aggregation for distributed training," *IEEE/ACM Trans. Netw.*, vol. 31, no. 5, pp. 2267–2280, Oct. 2023.

[25] J. Fang, H. Xu, G. Zhao, Z. Yu, B. Shen, and L. Xie, "Accelerating distributed training with collaborative in-network aggregation," *IEEE/ACM Trans. Netw.*, vol. 32, no. 4, pp. 3437–3452, Aug. 2024.

[26] J. Liu et al., "Inart: In-network aggregation with route selection for accelerating distributed training," in *Proc. ACM Web Conf.*, 2024, pp. 2879–2889.

[27] H. Kim, H. Lee, and S. Pack, "Traffic-aware in-network aggregation placement for multi-tenant distributed machine learning," in *Proc. 32nd Int. Conf. Comput. Commun. Netw.*, 2023, pp. 1–9.

[28] Z. Li et al., "Pa-atp: Progress-aware transmission protocol for in-network aggregation," in *Proc. IEEE 31st Int. Conf. Netw. Protoc.*, 2023, pp. 1–11.

[29] J. Ye, Y. Peng, Y. Li, Z. Li, and J. Huang, "Asynchronous control based aggregation transport protocol for distributed deep learning," *IEEE Trans. Comput.*, vol. 74, no. 4, pp. 1362–1376, Apr. 2025.

[30] G. Zhang, D. Dong, T. Li, and S. Yang, "Libra: High-precision congestion control for datacenter networks with in-network allreduce," *Comput. Netw.*, vol. 269, 2025, Art. no. 111407.

[31] S. Liu et al., "In-network aggregation with transport transparency for distributed training," in *Proc. 28th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Volume 3, 2023, pp. 376–391.

[32] B. Zhao, C. Liu, J. Dong, Z. Cao, W. Nie, and W. Wu, "Enabling switch memory management for distributed training with in-network aggregation," in *Proc. IEEE Conf. Comput. Commun.,* 2023, pp. 1–10.

[33] H. Wang, Y. Qin, C. Lao, Y. Le, W. Wu, and K. Chen, "Preemptive switch memory usage to accelerate training jobs with shared in-network aggregation," in *Proc. IEEE Int. Conf. Netw. Protoc.*, 2023, pp. 1–12.

[34] H. Wang, Y. Qin, C. Lao, Y. Le, W. Wu, and K. Chen, "Preemptive switch memory usage to accelerate training jobs with shared in-network aggregation," in *Proc. IEEE 31st Int. Conf. Netw. Protoc.*, 2023, pp. 1–12.

[35] Y. Li et al., "Flexible job scheduling with spatial-temporal compatibility for in-network aggregation," *IEEE Trans. Comput.*, vol. 74, no. 4, pp. 1322–1333, Apr. 2025.

[36] Y. Li et al., "Straggler-aware gradient aggregation for large-scale distributed deep learning system," *IEEE/ACM Trans. Netw.*, vol. 32, no. 6, pp. 4917–4930, Dec. 2024.

[37] H. Lee, J. Lee, H. Kim, and S. Pack, "Straggler-aware in-network aggregation for accelerating distributed deep learning," *IEEE Trans. Serv. Comput.*, vol. 16, no. 6, pp. 4198–4204, Nov.–Dec. 2023.

[38] Z. Su, S. Luo, K. Li, H. Xing, and B. Peng, "Efficient in-network aggregation with adaptive quantization," in *Proc. 9th Asia-Pacific Workshop Netw.*, 2025, pp. 247–248.

[39] H. Lee, H. Kim, C. Bae, Y. Kim, and S. Pack, "Quantization-aware in-network aggregation for minimizing communication overhead," in *Proc. Companion 19th Int. Conf. Emerg. Netw. EXperiments Technol.*, 2023, pp. 57–58.

[40] S. Luo, Z. Liu, Q. Rao, K. Li, and H. Xing, "Progress and bandwidth aware partial model synchronization with selective reduce," *IEEE Trans. Netw. Sci. Eng.*, vol. 13, pp. 1358–1373, 2026.

[41] Y. Qiu, G. Zhao, H. Xu, H. Huang, and C. Qiao, "PARING: Joint task placement and routing for distributed training with in-network aggregation," *IEEE/ACM Trans. Netw.*, vol. 32, no. 5, pp. 4317–4332, Oct. 2024.

[42] B. Zhao, W. Xu, S. Liu, Y. Tian, Q. Wang, and W. Wu, "Training job placement in clusters with statistical in-network aggregation," in *Proc. 29th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Volume 1, 2024, pp. 420–434.

[43] R. Segal, C. Avin, and G. Scalosub, "Constrained in-network computing with low congestion in datacenter networks," in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 1639–1648.

[44] L. Luo, S. Yang, H. Wu, H. Yu, B. Lei, and S. Gao, "Maximizing aggregation throughput for distributed training with constrained in-network computing," in *Proc. IEEE Int. Conf. Commun.*, 2023, pp. 3652–3657.

[45] J. Fang, G. Zhao, H. Xu, Z. Yu, B. Shen, and L. Xie, "GOAT: Gradient scheduling with collaborative in-network aggregation for distributed training," in *Proc. IEEE/ACM 31st Int. Symp. Qual. Service*., 2023, pp. 1–10.

[46] H. Zhu, Z. Guo, and M. Ye, "Dina: Toward determined in-network aggregation for distributed machine learning," *IEEE Trans. Netw.*, vol. 33, no. 4, pp. 1454–1468, Aug. 2025.

[47] L. Luo, X. Chen, S. Yang, W. Fan, and H. Yu, "Accelerating communication for distributed training with collaborative in-network aggregation," in *Proc. IEEE 24th Int. Conf. Commun. Technol.*, 2024, pp. 681–687.

[48] J. Bao, G. Zhao, H. Xu, H. Wang, and P. Yang, "Ingo: In-network aggregation routing with batch size adjustment for distributed training," in *Proc. IEEE/ACM 32nd Int. Symp. Qual. Service*., 2024, pp. 1–10.

[49] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computation is a dumb idea whose time has come," in *Proc. 16th ACM Workshop Hot Topics Netw.*, 2017, pp. 150–156.

[50] R. L. Graham et al., "Scalable hierarchical aggregation protocol (SHArP): A hardware architecture for efficient data reduction," in *Proc. 1st Int. Workshop Commun. Optimizations HPC (COMHPC)*., 2016, pp. 1–10.

[51] R. L. Graham et al., "Scalable hierarchical aggregation and reduction protocol (sharp) tm streaming-aggregation hardware design and evaluation," in *Proc. Int. Conf. High Perform. Comput.*, 2020, pp. 41–59.

[52] N. Gebara, M. Ghobadi, and P. Costa, "In-network aggregation for shared machine learning clusters," *MLSys*, vol. 3, pp. 829–844, 2021.

[53] H. Wang, D. Sun, J. Hu, and K. Chen, "Enabling in-network acceleration over the cloud," in *Proc. IEEE Conf. Comput. Commun.,* 2025, pp. 1–10.

[54] Z. Li et al., "A2tp: Aggregator-aware in-network aggregation for multi-tenant learning," in *Proc. 18th Eur. Conf. Comput. Syst.*, 2023, pp. 639–653.

[55] X. Xie, B. Tang, X. Chen, and Z. Zhu, "P4INC-AOI: All-optical interconnect empowered by in-network computing for DML workloads," *IEEE Trans. Netw.*, vol. 33, no. 3, pp. 1236–1251, Jun. 2025.

[56] G. Sun et al., "Large language model (LLM)-enabled graphs in dynamic networking," *IEEE Netw.*, vol. 39, no. 1, pp. 235–242, Jan. 2025.

[57] H. Du et al., "Reinforcement learning with LLMs interaction for distributed diffusion model services," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 47, no. 10, pp. 8838–8855, Oct. 2025.

[58] H. Luo et al., "Toward edge general intelligence with multiple-large language model (multi-LLM): Architecture, trust, and orchestration," *IEEE Trans. Cogn. Commun. Netw.*, vol. 11, no. 6, pp. 3563–3585, Dec. 2025.

[59] R. Liang et al., "GDSG: Graph diffusion-based solution generator for optimization problems in MEC networks," *IEEE Trans. Mobile Comput.*, vol. 24, no. 10, pp. 10264–10277, Oct. 2025.

[60] R. Liang et al., "Diffusion models as network optimizers: Explorations and analysis," *IEEE Internet Things J.*, vol. 63, no. 6, pp. 16–24, Jun. 2025.

[61] F. Liu et al., "Large language model for multiobjective evolutionary optimization," in *Proc. Int. Conf. Evol. Multi-Criterion Optim.*, 2025, pp. 178–191.

[62] S. Liu, C. Gao, and Y. Li, "Large language model agent for hyper-parameter optimization," 2024, *arXiv:2402.01881*.

[63] C. Yang et al., "Large language models as optimizers," in *Proc. 12th Int. Conf. Learn. Representations*, 2023, pp. 1–41.

[64] S. Brahmachary et al., "Large language model-based evolutionary optimizer: Reasoning with elitism," *Neurocomputing*, vol. 622, 2025, Art. no. 129272.

[65] H. Hao, X. Zhang, and A. Zhou, "Large language models as surrogate models in evolutionary algorithms: A preliminary study," *Swarm Evol. Computation*, vol. 91, 2024, Art. no. 101741.

[66] N. Egami, M. Hinck, B. Stewart, and H. Wei, "Using imperfect surrogates for downstream inference: Design-based supervised learning for social science applications of large language models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, vol. 36, pp. 68589–68601 .

[67] G. Jawahar, M. Abdul-Mageed, L. V. Lakshmanan, and D. Ding, "LLM performance predictors are good initializers for architecture search," 2023, *arXiv:2310.16712*.

[68] E. Soares, V. Sharma, E. V. Brazil, R. Cerqueira, and Y.-H. Na, "Capturing formulation design of battery electrolytes with chemical large language model," in *Proc. AI Accelerated Mater. Des.-NeurIPS 2023 Workshop*, 2023.

[69] A. Kristiadi, F. Strieth-Kalthoff, M. Skreta, P. Poupart, A. Aspuru-Guzik, and G. Pleiss, "A sober look at llms for material discovery: Are they actually good for Bayesian optimization over molecules?," 2024, *arXiv:2402.05015*.

[70] X. Wu, Y. Zhong, J. Wu, B. Jiang, and K. C. Tan, "Large language model-enhanced algorithm selection: Towards comprehensive algorithm representation," 2023, *arXiv:2311.13184*.

[71] H. Du et al., "Mixture of experts for network optimization: A large language model-enabled approach," 2024, *arXiv:2402.09756*.

[72] A. Memduhoğlu, N. Fulman, and A. Zipf, "Enriching building function classification using large language model embeddings of openstreetmap tags," *Earth Sci. Informat.*, vol. 17, no. 6, pp. 5403–5418, 2024.

[73] F. Liu et al., "Evolution of heuristics: Towards efficient automatic algorithm design using large language model," in *Proc. 41st Int. Conf. Mach. Learn.*, 2024, pp. 32201–32223.

[74] E. Hemberg, S. Moskal, and U.-M. O'Reilly, "Evolving code with a large language model," *Genet. Program. Evolvable Machines*, vol. 25, no. 2, 2024, Art. no. 21.

[75] B. Romera-Paredes et al., "Mathematical discoveries from program search with large language models," *Nature*, vol. 625, no. 7995, pp. 468–475, 2024.

[76] Y. J. Ma et al., "Eureka: Human-level reward design via coding large language models," 2023, *arXiv:2310.12931*.

[77] M. Pluhacek, J. Kovac, A. Viktorin, P. Janku, T. Kadavy, and R. Senkerik, "Using LLM for automatic evolvement of metaheuristics from swarm algorithm soma," in *Proc. Genet. Evol. Computation Conf. Companion*, 2024, pp. 2018–2022.

[78] M. Pluhacek, A. Kazikova, A. Viktorin, T. Kadavy, and R. Senkerik, "Investigating the potential of ai-driven innovations for enhancing differential evolution in optimization tasks," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 2023, pp. 1070–1075.

[79] V. Aglietti et al., "Funbo: Discovering acquisition functions for bayesian optimization with funsearch," 2024, *arXiv:2406.04824*.

[80] S. Yao, F. Liu, X. Lin, Z. Lu, Z. Wang, and Q. Zhang, "Multi-objective evolution of heuristic using large language model," in *Proc. AAAI Conf. Artif. Intell.*, 2025, vol. 39, no. 25, pp. 27144–27152.

[81] H. Ye et al., "Reevo: Large language models as hyper-heuristics with reflective evolution," *in Proc. Adv. Neural Inf. Process. Syst.*, 2024, vol. 37, pp. 43571–43608.

[82] R. Zhong, Y. Xu, C. Zhang, and J. Yu, "Leveraging large language model to generate a novel metaheuristic algorithm with crispe framework," *Cluster Comput.*, vol. 27, no. 10, pp. 13835–13869, 2024.

[83] N. v. Stein and T. Bäck, "LLaMEA: A large language model evolutionary algorithm for automatically generating metaheuristics," *IEEE Trans. Evol. Comput.*, vol. 29, no. 2, pp. 331–345, Apr. 2025.

[84] S. Liu, C. Chen, X. Qu, K. Tang, and Y.-S. Ong, "Large language models as evolutionary optimizers," in *Proc. IEEE Congr. Evol. Comput.*, 2024, pp. 1–8.

[85] F. Liu, X. Tong, M. Yuan, and Q. Zhang, "Algorithm evolution using large language model," 2023, *arXiv:2311.15249*.

[86] P. V. T. Dat, L. Doan, and H. T. T. Binh, "Hsevo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using LLMs," in *Proc. AAAI Conf. Artif. Intell.*, vol. 39, no. 25, 2025, pp. 26931–26938.

[87] X. Wu et al., "Efficient heuristics generation for solving combinatorial optimization problems using large language models," in *Proc. 31st ACM SIGKDD Conf. Knowl. Discov. Data Mining V. 2*, 2025, pp. 3228–3239.

[88] Y. Shi, J. Zhou, W. Song, J. Bi, Y. Wu, and J. Zhang, "Generalizable heuristic generation through large language models with meta-optimization," 2025, *arXiv:2505.20881*.

[89] H. Abgaryan, A. Harutyunyan, and T. Cazenave, "LLMs can schedule," 2024, *arXiv:2408.06993*.

[90] J. Huang, X. Li, L. Gao, Q. Liu, and Y. Teng, "Automatic programming via large language models with population self-evolution for dynamic job shop scheduling problem," 2024, *arXiv:2410.22657*.

[91] X. Yang, L. Zhang, H. Qian, L. Song, and J. Bian, "Heuragenix: Leveraging llms for solving complex combinatorial optimization challenges," 2025, *arXiv:2506.15196*.

[92] P. T. Amarasinghe, S. Nguyen, Y. Sun, and D. Alahakoon, "Ai-copilot for business optimisation: A framework and a case study in production scheduling," 2023, *arXiv:2309.13218*.

[93] Z. Iklassov, Y. Du, F. Akimov, and M. Takac, "Self-guiding exploration for combinatorial problems," in *Proc. Adv. Neural Inf. Process. Syst.*, 2024, vol. 37, pp. 130569–130601.

[94] F. Liu et al., "Llm4ad: A platform for algorithm design with large language model," 2024, *arXiv:2412.17287*.