LLM-CCL: An LLM-Driven Generator for Efficient Collective Communication Schedules

***Abstract*—Collective communication is a key determinant of GPU utilization and the efficiency of large-scale machine-learning workloads. Recent research has sought to synthesize custom collective schedules tailored to specific topologies and communication workloads, addressing the inefficiencies of vendor-provided algorithms. State-of-the-art solutions typically formulate schedule generation as a mixed-integer linear programming (MILP) problem. However, in thousand-GPU clusters and beyond, these methods still face a major challenge: managing the high computational overhead of MILP solving.**

In this paper, we present LLM-CCL, an LLM-driven generator for efficient collective communication schedules. First, LLM-CCL uses a large language model to search for a high-performance *Propagation Plan*. It then leverages symmetry within collective schedules to transform the plan directly into a complete collective schedule, avoiding the lengthy optimization process. Experimental results show that LLM-CCL improves collective communication performance by up to 90% on our 64-GPU V100 testbed compared with state-of-the-art approaches. It scales schedule generation to clusters with up to 16,384 GPUs, while maintaining high schedule quality as the cluster grows. To the best of our knowledge, LLM-CCL is the first system to demonstrate collective communication schedule synthesis for cluster configurations exceeding 10,000 GPUs.

I. Introduction

As model sizes, context windows, and GPU clusters continue to grow [13], collective communication has become a critical performance factor in large-scale machine-learning systems [16], [40], [43]. Training large language models with trillions of parameters [15], [45] generates immense collective communication traffic , and such training increasingly relies on clusters that scale to hundreds of thousands of GPUs [3], [35], [44]. Production systems often rely on expert-designed collective communication templates, such as ring, tree, CollNet [1], [17], [21], [32], and halving-doubling (HD) [37]. These algorithms are simple and robust, but they are generally tuned to only a limited range of topologies and message sizes [11], [24], [34], [46], [47]. On complex topologies such as torus [22], Clos [12], and multi-rail [30], with heterogeneous interconnects such as NVLink, InfiniBand (IB), and PCIe, fixed templates can lead to imbalanced link utilization [38], [47].

Recent work [6], [8], [29], [41] has proposed approaches that synthesize collective communication schedules to adapt to varying topologies and collective demands. These approaches generally formulate schedule synthesis as a mixed-integer linear program (MILP). The cost of solving these MILPs is a key scalability bottleneck for solver-backed synthesizers. For example, SyCCL reports that TECCL did not finish synthesizing a 512-GPU AllGather schedule within a 10-hour time limit [8]. SyCCL accelerates synthesis by exploiting symmetry in collective demands and network topologies, achieving up to a 17,286× synthesis-time speedup over TECCL on a multi-rail topology [8]. However, SyCCL's sketch decomposes the original problem into subproblems that are still solved by TECCL's MILP backend. Consequently, MILP solving remains a dominant cost even after decomposition.

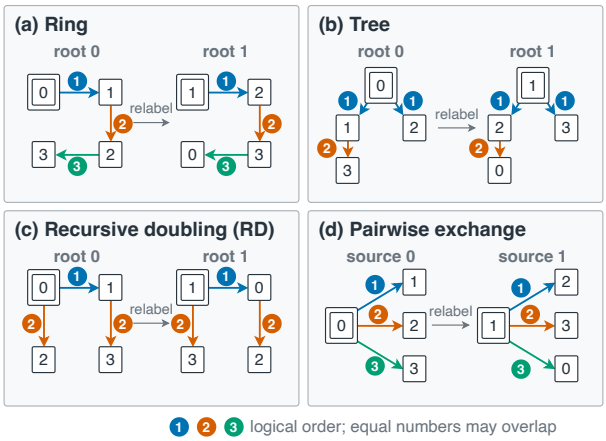


Fig. 1. Symmetry in expert-designed collective algorithms. Each communication pattern on the right is obtained from its counterpart on the left by relabeling rank IDs while preserving communication edges and logical steps. Ring, tree, and pairwise exchange use cyclic rotations of rank IDs, whereas recursive doubling swaps ranks 0 and 1 and ranks 2 and 3.

This scalability bottleneck raises a fundamental question: *Is MILP-based optimization necessary for collective schedule synthesis?* Existing solutions already confine MILP solving to subproblems. Within each such subproblem, we observe that all GPUs receive the same amount of data and are connected by identical links, suggesting that the schedule can instead be constructed through symmetry. Such symmetry is also common in expert-designed collective algorithms [32], [37], [46]. As Fig. 1 illustrates, one GPU's schedule can be obtained from another's by relabeling rank IDs. Accordingly, we introduce *symmetry-driven schedule construction*, which directly generates complete schedules without MILP-based synthesis.

Eliminating MILP-based completion still leaves the problem of finding a good decomposition. The decomposition can be represented as a list of compact transmissions ; we call this list a Propagation Plan (Section IV-B). The enumeration-

and-pruning methods used by existing solutions can take a long time on large clusters and may discard high-performance Propagation Plans. The space of possible Propagation Plans is combinatorially large, making it difficult to design an effective search strategy. We reason that LLMs are well suited to this high-level search problem. Recent work has shown that LLMs can help solve combinatorial search and program-generation tasks by proposing structured candidates and iteratively refining them based on feedback [25], [39]. Similarly, collective schedule design often resembles expert reasoning: identifying the topology's hierarchy, inferring bottlenecks, choosing a propagation strategy, and revising the design based on performance observations.

However, using an LLM to find a good decomposition presents three challenges. First, representing the topology in a form that an LLM can understand is difficult. Adjacency matrices and lists are lengthy and can obscure the topology's structure, whereas images are intuitive but difficult to reference precisely and require multimodal capabilities [48]. Second, existing evolutionary approaches are inefficient because they provide only a score as feedback [9], [23], [25], [28], [31], [39]: the LLM learns whether a solution is good or bad, but not why. Third, long iterative searches accumulate extensive exploration histories, and placing the entire history in the context degrades LLM performance [26].

To address these challenges and exploit symmetry in collectives and topologies, we propose LLM-CCL, an efficient LLM-driven framework for synthesizing high-performance collective schedules. First, we propose TopoDSL, a compact, parameterized topology language (Section IV-A) designed to be read and written by both humans and LLMs. TopoDSL describes the entire topology using a small number of parameterized connection rules, enabling the LLM to reason about topology at a high level. Second, we design an Assessment Pipeline to give more detailed profile feedback to the LLM. Third, we design a Context Agent and a Reflection Agent to select history records and summarize the plan proposed by the LLM. We also introduce symmetry-driven schedule construction: deterministic symmetry expansion transforms each legal Propagation Plan into a complete schedule. The coordination mechanism enforces a deterministic workflow comprising the Proposal Agent, Assessment Pipeline, Reflection Agent, and Context Agent, enabling the LLM to search efficiently and iteratively for high-performance collective schedules.

We make the following contributions:

- We introduce symmetry-driven schedule construction, which uses topological and collective symmetries as a deterministic construction rule rather than solely as a solver-reduction mechanism.
- We design and implement LLM-CCL, an iterative LLM-driven synthesis framework combining TopoDSL and Propagation Plans with a deterministic Assessment Pipeline and three LLM agents: the Proposal, Reflection, and Context Agents.
- For AllGather/AllToAll on V100 Clos/H800 multi-rail, LLM-CCL achieves up to 1.9× the bandwidth of SyCCL/ForestColl , and finds NCCL-beating schedules for 16,384 GPUs within 10.4 minutes. Code is available at https://anonymous.4open.science/r/llm-ccl-021D.

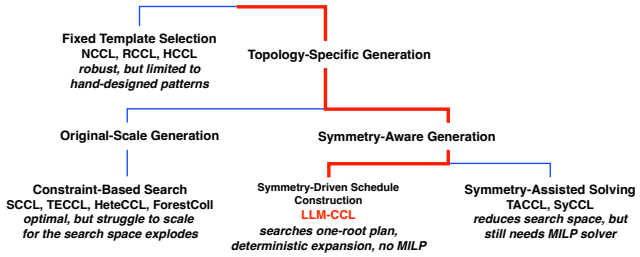


Fig. 2. Design space for collective communication schedule generation. The red path marks LLM-CCL's choices.

II. Motivation

Fixed collective libraries [1], [17], [21], [32] are robust and easy to deploy, but their limited set of manually designed patterns cannot cover the rapidly growing combinations of topology, collective primitive, and message size [8], [19], [29], [41], [47]; automatic synthesis should instead tailor each schedule to its target setting. The remainder of this section follows the red path in Fig. 2 and motivates LLM-CCL's three key design choices: exploiting symmetry, using it for direct schedule construction rather than only solver reduction, and using an LLM to search the remaining plan space.

A. Why Exploit Symmetry?

Recent work [6], [8], [19], [29], [41] has modeled collective schedule synthesis as an optimization problem, for example using satisfiability modulo theories (SMT) or mixed-integer linear programs (MILPs). These formulations encode the collective demand, topology, and link characteristics as constraints, with the goal of minimizing collective completion time. However, their complexity grows rapidly with the size of the network and the number of participating devices [8], [29], [41]. SCCL cannot synthesize a 16-GPU AllGather schedule within 24 hours [8]. TECCL uses a heuristic to divide the original problem into multiple stages and solve each stage independently, which may incur up to a 20% performance loss [8], [29].

However, modern GPU clusters and collective primitives both expose strong symmetry [8], and this symmetry is not accidental. Large-scale cluster networks are usually built from repeated units such as rails, leaves, spines, and pods [3], [35]. Operators prefer such regular designs because they reduce the operational complexity of a data center. If each server or rack had different outgoing bandwidth or a different connection pattern, scheduling jobs and maintaining the cluster would become substantially harder. Collective communication also exhibits semantic symmetry: for example, all GPUs in a collective may send or receive an equal amount of data.

These two forms of symmetry can reduce the number of distinct choices that a synthesizer must consider, although schedule expansion and evaluation can still scale with the number of GPUs. A synthesizer that generates the entire schedule

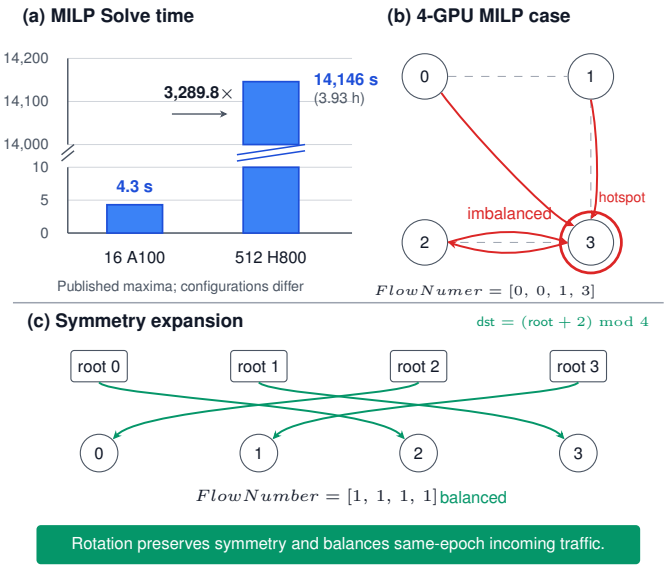


Fig. 3. Motivation for symmetry-driven schedule construction. (a) Published maximum MILP solve times for two different GPU configurations. (b) A four-GPU subset of a 256-GPU MILP-synthesized schedule shows imbalanced same-step incoming traffic. (c) Deterministic symmetry expansion balances the traffic through rotation.

independently may repeatedly search over isomorphic choices. Exploiting topology and collective symmetry lets it search and reuse representative communication structures.

B. Why Symmetry-Driven Schedule Construction Instead of Symmetry-Assisted MILP?

Recent efforts (e.g., TACCL [41] and SyCCL [8]) have shown that symmetry can substantially reduce the search space of schedule synthesis. However, they still rely on MILP to complete the schedule, which remains a scalability bottleneck. TACCL allows users to apply rotational symmetry constraints to further reduce the search space. SyCCL observes that encoding additional symmetry constraints in the MILP formulation can make the problem harder to solve. SyCCL uses a two-phase approach to synthesize collective schedules. First, it decomposes the collective demand into sub-demands across topology groups and communication stages and enumerates candidate sketches with symmetry-aware pruning. It then uses a TECCL-style MILP solver to synthesize sub-schedules for the sub-demands in each generated sketch combination and combines them into a complete schedule. Thus, SyCCL retains this MILP bottleneck. Fig. 3(a) contrasts two reported maximum SyCCL synthesis times: 4.3 s for 16 A100 GPUs and 14,146 s for 512 H800 GPUs. Because these cases use different hardware and configurations, they do not constitute a controlled scaling comparison. They nevertheless illustrate that MILP completion can dominate synthesis time. Moreover, SyCCL's 512-GPU profile attributes nearly all remaining synthesis cost to TECCL within symmetric groups; sketch exploration is sub-second.

A solution that is optimal under an epoch-discretized MILP need not be optimal in actual execution. As Fig. 3(b) illustrates, a schedule synthesized by an MILP can produce uneven traffic: in the same step, three GPUs send traffic to GPU 3. Such a gap between the model and measured behavior can result from epoch discretization. A TECCL-style MILP model divides time into discrete epochs, and the solver seeks a schedule that minimizes the discretized objective while satisfying all constraints. When the epoch duration is too large, two schedules with different performance may be considered to have the same duration because they occupy the same number of epochs in the MILP model. Conversely, choosing a smaller epoch duration increases the number of decision variables and can make solving prohibitively expensive. Under the same sub-demand, deterministic symmetry expansion can produce more balanced schedules by ensuring that equivalent GPUs follow the same propagation pattern, as shown in Fig. 3(c). This approach avoids the epoch-discretization trade-off of MILP-based schedule construction and balances the illustrated logical endpoint load. More fundamentally, a TECCL-style MILP can optimize only the network effects explicitly represented by its variables and constraints. Encoding fine-grained routing, contention, bandwidth sharing, and queueing would further enlarge an already expensive formulation. SyCCL does use a simulator to rank the complete schedules produced from different sketch combinations; however, this evaluation occurs only after its MILP has synthesized the sub-schedules and therefore cannot recover alternatives that the discretized MILP model did not generate.

LLM-CCL instead closes the search loop around simulation: the LLM proposes a compact Propagation Plan, deterministic symmetry expansion constructs its complete schedule, and a flow-level simulator evaluates it. This separation allows richer network behavior to be modeled without translating every effect into MILP variables and constraints, providing a more faithful performance signal for subsequent proposals. A symmetric collective demand alone does not require independently instantiated root schedules to use the same root-relative mapping. Rotation couples these choices by construction: under $\mathsf{dst} = (\mathsf{root} + 2) \bmod 4$, the destination map is a permutation, so every GPU receives one flow and the resulting per-GPU flow counts are $\texttt{FlowNumber} = [1, 1, 1, 1]$. This guarantee concerns balanced logical endpoint load across the four rotated roots illustrated in Fig. 3(c).

Therefore, LLM-CCL takes symmetry-driven construction a step further: it uses symmetry to construct schedules rather than only to reduce a solver's search space. The system searches for a Propagation Plan and deterministically expands it into a complete collective schedule. This eliminates target-scale optimization over chunk-link-epoch variables while delegating dependency recovery and schedule instantiation to the expander.

C. Why Use an LLM?

After adopting symmetry-driven schedule construction, the remaining challenge is how to divide the entire problem into smaller subproblems. SyCCL [8] uses enumeration and symmetry-aware pruning to search over candidate sketches. Although symmetry removes many equivalent candidates, its

pre-pruning sketch space grows combinatorially with the number of topology groups and the number of GPUs per group.

LLM-CCL uses a Propagation Plan for the same divide-and-conquer strategy. Although its search space is much smaller than that of a full schedule, it remains large on large-scale topologies, requiring efficient exploration.

Recent work has shown that LLMs can solve combinatorial-search and program-generation tasks by proposing structured candidates and refining them based on evaluator feedback [23], [25], [31], [39]. Tool-interactive critique, verbal reflection, and self-improving system agents further show how structured external feedback can guide subsequent proposals [18], [42], [50]. Similarly, collective schedule design resembles expert reasoning: identifying topology hierarchy, inferring bottlenecks, choosing a propagation strategy, and revising the design based on performance observations. This motivates us to investigate LLMs for this high-level search.

III. System Overview

We introduce LLM-CCL, an LLM-driven generator for efficient collective communication schedules. The system decomposes schedule synthesis into an iterative pipeline: (1) take the topology as input ; (2) propose Propagation Plans; (3) verify, expand, and evaluate the proposals; (4) reflect on the Propagation Plans; (5) adjust the context; and (6) return the final schedule when the search budget is reached. To support this workflow, we design two abstractions—TopoDSL and the Propagation Plan—and four system modules: the Proposal Agent, Assessment Pipeline, Reflection Agent, and Context Agent.

- **TopoDSL:** a compact, parameterized language that describes topology symmetry using a few connection rules.
- **Propagation Plan:** a topology-aware root template for chunk propagation across symmetric groups.
- **Proposal Agent:** an LLM agent that generates candidate Propagation Plans from a fixed task specification and a round-dependent search context.
- **Assessment Pipeline:** a deterministic pipeline that verifies each proposed plan, applies deterministic symmetry expansion to transform legal plans into complete schedules, and evaluates them with a profile-guided flow simulator.
- **Reflection Agent:** an LLM agent that interprets the round winner and converts the assessment result into a reusable explanation of the candidate's communication strategy.
- **Context Agent:** an LLM agent that maintains the historical exploration pool and curates useful plans, errors, and reflections for the next prompt.

A. Workflow of LLM-CCL

The workflow of LLM-CCL is illustrated in Fig. 4, where LLM denotes that the step is driven by an LLM.

a) TopoDSL and collective demand: TopoDSL and the collective demand are the workflow inputs. TopoDSL describes the target topology, and the user provides a collective demand (1). The TopoDSL specification (Section IV-A) describes the topology's symmetry structure so that an LLM can understand it .

b) Proposal Agent: Given the TopoDSL description and collective demand, the Proposal Agent generates a batch of candidate Propagation Plans (2) using a fixed task specification and a round-dependent search context. The fixed specification defines the topology, demand, plan contract, and bottleneck-profile schema; the context starts with a legal ring-based seed and later contains curated programs, reflections, profiles, and recurring failure patterns. Each candidate is a program that expresses a root-0 plan as `(step, layer, group, srcs, dsts)` transmissions, from which the deterministic backend derives chunk-level events, physical routes, and dependency edges. The Propagation Plan specification is given in Section IV-B .

c) Assessment Pipeline: After the Proposal Agent generates candidate Propagation Plans, the Assessment Pipeline assesses them (3). It verifies each plan, expands every legal plan into a complete collective schedule with a deterministic backend, and evaluates its performance. The verifier returns errors for invalid plans, whereas the assessment returns algorithm bandwidth and a structured bottleneck profile for valid plans .

After assessing the batch, the system selects the highest-scoring valid candidate as the round winner. It updates the global best only if this winner improves the incumbent score.

d) Reflection Agent: The Reflection Agent analyzes the round winner (4). It generates a structured reflection that summarizes the winner's propagation strategy and relates its strengths and weaknesses to the score and profile. This reflection is added to the search context for the next round. It serves as a compressed representation of the winner's program and assessment result, whose full representations are too token-expensive to retain in the context.

e) Context Agent: After reflection, the Context Agent builds the bounded search context for the next round (5) from the generated plans, assessment results, winner's reflection, and current pool. It selects records by performance and structural diversity, stores valid candidates with their scores and profiles, records verifier errors for invalid candidates, and maintains the global best separately. A deterministic error-analysis component normalizes and categorizes these errors, tracks their frequencies, and summarizes recurring failures.

The next-round prompt retains the global best's complete program and reflection, compact summaries of selected plans, the failure summary, and relevant bottleneck information. Reusing the fixed task specification while omitting the complete search history prevents prompt size from growing linearly with the number of rounds and feeds curated evidence back to the Proposal Agent. When the search budget is reached, the system returns the best collective schedule (6).

IV. Design of LLM-CCL

This section presents LLM-CCL's two compact search abstractions—TopoDSL and the Propagation Plan—and its

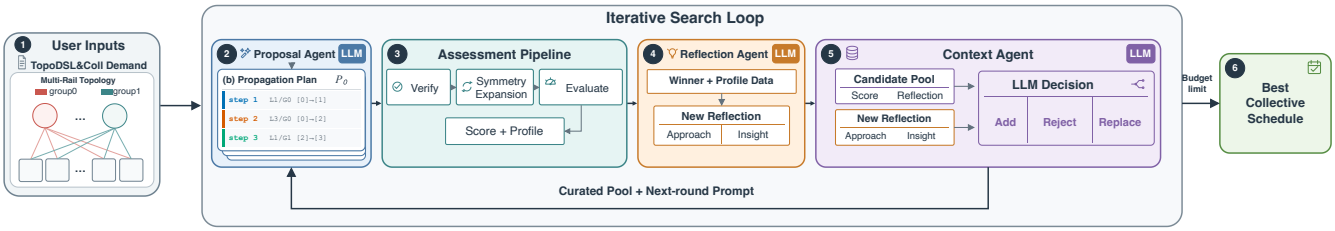


Fig. 4. Overview of the LLM-CCL workflow. Steps 2–5 form an iterative search loop that proposes, assesses, reflects on, and curates Propagation Plans. User inputs enter from the left, and the best collective schedule is returned when the search budget is reached.

```
topology = MultiRailTopology(
  gpu_num=512,
  coll_bytes="1MB",
  collective=CollectiveType.ALLGATHER,

  layer1=LayerSpec(
    1, LinkSpec("150GB/s", "10.5us"),
    group_num=64, node_num=8,
    node_type=NodeType.GPU),

  layer2=LayerSpec(
    2, LinkSpec("45GB/s", "0us"),
    group_num=512, node_num=1,
    node_type=NodeType.NIC),

  layer3=LayerSpec(
    3, LinkSpec("45GB/s", "21.5us"),
    group_num=8, node_num=64,
    node_type=NodeType.SWITCH))
```

Fig. 5. An abridged TopoDSL example for a 512-GPU multi-rail topology.

Assessment Pipeline for plan verification, schedule expansion, and performance evaluation.

A. Topology Abstraction with TopoDSL

TopoDSL is LLM-CCL's topology-facing interface. It describes the target cluster in a form that the Proposal Agent can read and the Assessment Pipeline can instantiate deterministically.

Rather than enumerating physical edges, TopoDSL represents a structured topology with a few parameterized connection rules. This embedded DSL uses `BaseTopology`, `NodeType`, `LinkSpec`, and `LayerSpec`; Fig. 5 shows an abridged 512-GPU multi-rail example.

a) Layer and group abstractions: A `LayerSpec` describes a class of repeated connections: a layer identifier, bandwidth and latency through `LinkSpec`, the number of groups, and each group's scale and node type. Topology connection functions define the group's connection rules.

In Fig. 5, Layer 1 represents host-local communication. Its 64 groups correspond to 64 hosts, each containing eight GPUs connected by the same host-local rule. Layer 2 represents the repeated GPU–NIC mapping. Its 512 groups associate each GPU with one NIC. Layer 3 represents inter-host communication through eight rails; each rail group contains the corresponding endpoints from all hosts. TopoDSL also provides an interface for users to specify the connection rules for each layer.

Formally, executing the connection rule of layer d produces a layer-specific graph $\mathcal{H}_d = (\mathcal{N}_d, \mathcal{E}_d)$. The connected components of $\mathcal{H}_d$ determine the communication groups $\mathcal{G}_d$. For a group $g \in \mathcal{G}_d$, its GPU projection is

$$\mathcal{V}_{d,g} = \{v \mid v \text{ is a GPU in group } g\}. \tag{1}$$

The pair (d, g) therefore identifies a concrete communication domain. For example, $(1, h)$ identifies the GPUs within host h, whereas $(3, r)$ identifies the GPUs that can communicate through rail r. These domains later become the topology-aware actions available to a Propagation Plan.

The example describes 512 GPUs with three connection rules instead of a 512×512 connectivity matrix or a long edge list. Scaling a fixed topology family mainly changes parameters and loop ranges, leaving its layers and rules unchanged. TopoDSL thus preserves algorithmically useful repetition without expanding it in the LLM prompt.

TopoDSL also serves as a system-wide topology contract. Given a user-provided definition, the Proposal Agent observes its source code (including layers, group counts, connection rules, and link costs) while the frontend executes the same definition to derive the connection mapping and compute $\mathcal{V}_{d,g}$. The verifier uses these sets to reject cross-domain transmissions, while the evaluator uses the instantiated links and their bandwidth and latency to model network behavior. Generation, validation, and evaluation therefore share one topology definition.

TopoDSL is intentionally limited to topology representation, compacting a physical topology into topology-aware communication domains.

B. Propagation Plan: A Compact Schedule Representation

To separate algorithm design from mechanical schedule construction, the LLM searches for a Propagation Plan , a root-level propagation template. Collective-specific symmetry instantiates it for the remaining roots or source–destination pairs (Section IV-C1).

1) Representation and Semantics: A Propagation Plan is an ordered sequence of high-level transmissions $\tau_i = (k_i, d_i, g_i, S_i, D_i)$, corresponding to `(step, layer, group, srcs, dsts)`. Here, k_i is a logical step, d_i and g_i identify a TopoDSL layer and a group, and S_i and D_i are source and destination GPU sets within that communication domain:

$$S_i, D_i \subseteq \mathcal{V}_{d_i,g_i}, \qquad S_i \cap D_i = \varnothing. \tag{2}$$

A root-r plan $P_r = \langle \tau_1, \tau_2, \ldots, \tau_m \rangle$ describes how GPU r's chunk reaches all other GPUs. The `step` field expresses

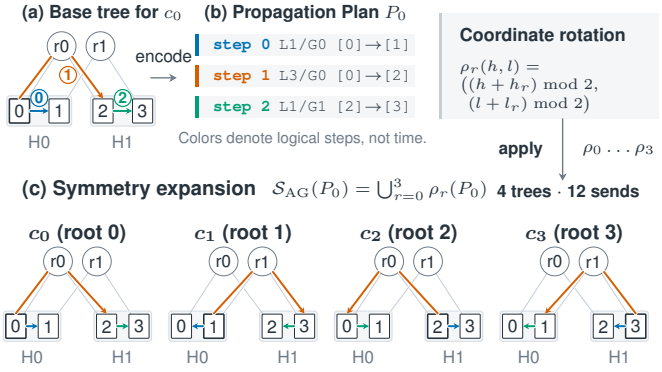


Fig. 6. From a Propagation Plan to a complete AllGather schedule. (a) A representative tree propagates chunk c_0 over two 2-GPU hosts connected by two rails. (b) P_0 records the same three logical transmissions using topology-aware communication domains. (c) Coordinate-wise rotation instantiates P_0 for all four roots, yielding the complete four-GPU AllGather schedule with 12 point-to-point flows. Colors show logical order, not timing.

logical causality: same-step transmissions may execute concurrently, although routing and link contention determine actual concurrency and completion time. A GPU may relay the chunk only after receiving it in an earlier step.

For a multi-source transmission, LLM-CCL deterministically partitions the destination list equally among sources, compactly expressing symmetric fan-out without source ambiguity.

For example, consider two hosts containing GPUs $\{0, 1\}$ and $\{2, 3\}$. Layer 1 represents host-local communication, while Layer 3, Group 0 connects GPUs 0 and 2 through an inter-host rail. A root-0 Propagation Plan can be written as:

```
(step=0, layer=1, group=0, srcs=[0], dsts=[1])
(step=1, layer=3, group=0, srcs=[0], dsts=[2])
(step=2, layer=1, group=1, srcs=[2], dsts=[3])
```

Fig. 6 visualizes this compact plan, its representative propagation tree, and the complete AllGather schedule after deterministic symmetry expansion.

The plan propagates the root chunk within its source host, sends it across hosts to relay GPU 2, and finishes within the destination host. It omits the inter-host physical route, other roots' chunk-level flows, and simulator dependencies, all recovered deterministically.

A Propagation Plan lies between fixed algorithm templates and complete schedules. Unlike a fixed ring or tree, it can vary stage ordering, layer/group selection, relay placement, fan-out, and cross-domain concurrency, expressing chains, trees, hierarchical propagation, and multi-rail parallelism. Deterministic symmetry expansion supplies the full flow set.

Compared with link-level synthesis, a Propagation Plan removes two dimensions from the LLM-visible representation: the LLM searches one root-level template rather than a full schedule for every collective demand and uses logical steps rather than physical-link epochs. Deterministic expansion reconstructs the remaining collective instances, point-to-point flows, and dependencies. This compression nevertheless leaves a reception-order space that can grow exponentially with the number of GPUs even under bounded depth ; the space is enlarged by additional choices in relays, layers/groups, fan-out, and transmission decomposition. Topology constraints prune illegal candidates but do not make exhaustive enumeration practical.

2) Compression and Remaining Search Complexity: The Propagation Plan removes much low-level state, but its search space remains combinatorial.

The selection among the remaining plans is a discrete, variable-length, structurally constrained optimization problem that depends on stage count, relay placement, topology hierarchy, cross-group concurrency, and post-expansion contention. Its objective emerges only after checking, expansion, and evaluation, providing no convenient analytical gradient. Yet plans expose compositional edits, such as advancing intra-host fan-out, replacing a long cross-host chain with a shallow tree, distributing relays across rails, or reducing concurrency in a congested domain. Conditioned on TopoDSL, the current plan, validation errors, prior strategies, and bottleneck profiles, the LLM proposes and prioritizes such edits under a limited search budget.

Accordingly, LLM-CCL uses the LLM as a structure-aware proposal generator, not a correctness checker or optimal solver. It neither reduces the search space's cardinality nor certifies legality; the deterministic verifier, expander, and evaluator retain these responsibilities (Sections IV-C1 and IV-C5).

C. Assessment Pipeline

The Assessment Pipeline deterministically converts a candidate Propagation Plan into performance feedback by verifying its legality, expanding it into a complete logical schedule, and evaluating that schedule with a profile-guided flow simulator.

1) Plan Verification and Deterministic Symmetry Expansion: A Propagation Plan covers only one root chunk and is therefore neither a complete collective schedule nor a directly executable one . It omits the remaining root-chunk flows for AllGather , along with the flow sequences and rounds for every source–destination chunk in AllToAll. Collective-specific deterministic symmetry expansion lowers its high-level transmissions into point-to-point flows with recovered dependencies, then instantiates the single-root structure for the complete demand. The LLM thus searches for one compact plan while deterministic components handle correctness-critical schedule construction.

2) Expansion Contract and Dependency Recovery: The verifier accepts a root-0 plan only if (1) every transmission's sources and destinations lie in its communication domain, (2) each source has obtained the root chunk in a strictly earlier step, and (3) GPU 0 is the unique initial owner and every other GPU appears exactly once as a destination. These invariants make every valid plan a directed propagation tree rooted at GPU 0.

Before expansion, the system normalizes each high-level transmission into single-source transmissions. Given ordered sources $S_i = (s_1, \dots, s_p)$ and destination list D_i, the verifier requires $|D_i|$ to be divisible by p and partitions D_i into p

equal, contiguous subsets, one per source. This makes every source–destination assignment unambiguous.

The resulting plan graph records the unique transmission that delivers the root chunk to each non-root GPU v. If a later transmission uses v as a relay, v's incoming flow becomes a dependency of every outgoing flow generated by that later transmission. Chunk ownership, not the LLM, thereby determines dependencies.

Each point-to-point flow records

$$(\mathit{chunk}, \mathit{src}, \mathit{dst}, \mathit{step}, \mathit{layer}, \mathit{origin})$$

where `origin` identifies the generating root-0 transmission. This provenance lets the evaluator map bottleneck flows back to editable `step`, `layer`, `group`, `srcs`, and `dsts` fields.

3) Deterministic Symmetry Expansion Across Collective Demands: A Propagation Plan specifies how one root chunk propagates through topology-aware communication domains. The expander turns this single-root template into the complete collective schedule by relabeling devices, selecting the required source–destination paths, and recovering their data dependencies.

Let each GPU v have a D-dimensional topology coordinate $\mathbf{c}(v) = (c_1(v), \ldots, c_D(v))$, where dimension d has size n_d. For a target source s, define the coordinate offset $\boldsymbol{\delta}_s = \mathbf{c}(s) - \mathbf{c}(0)$ and the rotation

$$\rho_s(v) = \mathbf{c}^{-1}\left(\left(\mathbf{c}(v) + \boldsymbol{\delta}_s\right) \bmod \mathbf{n}\right), \qquad \mathbf{n} = (n_1, \ldots, n_D), \tag{3}$$

where the modulo operation is applied component-wise. Because $\mathbf{c}(0) + \boldsymbol{\delta}_s = \mathbf{c}(s)$, the rotation maps the canonical root to the target source: $\rho_s(0) = s$. The mapping is valid when $\mathbf{c}$ is bijective over the topology coordinate set and each domain used by the plan is mapped to an isomorphic domain with the same communication relation and link costs.

For the two-host example in Fig. 6, ρ_1 maps GPU IDs $(0, 1, 2, 3)$ to $(1, 0, 3, 2)$. Therefore, the representative path $0 \to 2 \to 3$ becomes $1 \to 3 \to 2$, while preserving its dependency order.

The expander applies this mapping to the representative path for each required delivery. It then connects every flow to the flow that produced its input chunk, so a GPU forwards a chunk only after receiving it. Multiple ready flows from the same GPU are assigned distinct logical steps, whereas flows from different GPUs may share a step. These steps express causality; physical completion time is determined later by routing, link contention, and dependencies in the simulator. For ReduceScatter, the corresponding AllGather trees are traversed in reverse and incoming contributions are reduced before they are forwarded, so each rank receives the reduction for its output shard .

4) Correctness Scope: The verifier and expander jointly guarantee that the expanded schedule satisfies the collective demand, every transfer conforms to its specified layer and group domains, and every required chunk originates at its source, reaches its destinations, and is forwarded only after arrival. For ReduceScatter, the reversed trees also preserve exactly-once contribution coverage and reduce-before-forward ordering.

This contract can be satisfied by common topology families such as multi-rail, Clos, and Alibaba HPN, but does not hold for arbitrary irregular topologies or asymmetric or uneven demands such as AllToAll with a non-uniform destination set.

5) Performance Evaluation and Profile-Guided Attribution: A scalar score ranks Propagation Plans but does not identify which compact root-0 plan fields to revise, because the simulator evaluates flows from the complete logical schedule. For each valid plan P, it returns collective completion time $T(P)$ and score $1/T(P)$, yet equal times can reflect different propagation depths, transfers, and resource contention. We therefore construct a structured bottleneck profile that maps per-flow timing back to editable fields.

For each flow f along the completion path, the simulator records

$$\Delta t_f = t_f^{\text{prop}} + t_f^{\text{busy}} + t_f^{\text{wait}}, \tag{4}$$

where t_f^{prop} includes fixed scheduling and route-propagation delay, t_f^{busy} is accumulated link-busy transmission time, and t_f^{wait} is queueing or bandwidth-sharing delay.

Starting from the flow whose reception completes last, the profiler walks backward through dependencies, selecting the predecessor whose send or receive completion most recently released the current flow:

$$f_{i-1} = \arg\max_{p \in \text{pred}(f_i)} t_{p \to f_i}^{\text{release}}. \tag{5}$$

Reversing this sequence yields an ordered late-arriving chain from an initial flow to the final flow. Each entry records its chunk, actual endpoints, release and completion times, and the decomposition in Eq. (4).

Using provenance retained during symmetry expansion (Section IV-C1), the profiler maps every flow to its canonical root-plan transmission. This includes flows for rotated chunks and different source–destination pairs.

This mapping provides heuristic guidance for reflection: a large propagation component suggests examining depth or serial stages; a large link-busy component suggests examining the selected topology layer; and substantial waiting suggests examining fan-out, relay placement, or concurrent use of shared domains. The profile captures only one late-arriving chain, not the full critical DAG, and cannot identify a unique physical cause of resource waiting.

6) Fast Performance Estimation via Symmetry Quotients: At scale, materializing every expanded flow is prohibitively expensive: an N-GPU AllGather has N rotated root-plan instances and $N(N-1)$ logical flows, making in-loop simulation too slow and memory-intensive.

Our profile-guided simulator instead uses a symmetry quotient. It groups flows that are equivalent across GPU roots and across symmetry-equivalent topology groups and links, simulates one representative per class, and records class size as a multiplicity that weights traffic on each modeled resource. The quotient thus models the dependency constraints and resource effects needed to rank candidates (including link

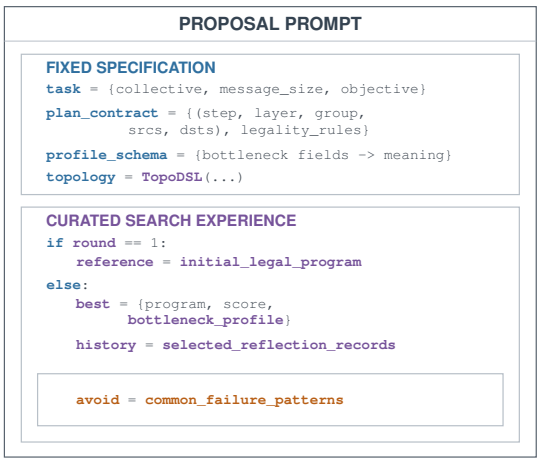


Fig. 7. Prompt Structure.

bandwidth and latency, serialization, and contention) without instantiating all logical flows. For a 16,384-GPU AllGather, it tracks 524,256 flow states rather than 268,419,072 logical flows, reducing the simulated state count by a factor of 512. Selected schedules are subsequently re-evaluated with the independently developed SyCCL simulator and, where available, measured on real hardware.

D. Reflection Agent

After deterministic verification and evaluation, the Reflection Agent receives the round winner's program, score, and bottleneck profile. It returns two short records: `Approach` describes the propagation strategy, and `Insight` connects the measured bottleneck to editable plan fields. For example, a profile dominated by link-busy time prompts an insight about serialization and relay placement; these records accompany the winning program in the next round. Correctness is enforced by the Assessment Pipeline before this reflection step.

E. Context Agent

The Context Agent maintains legal candidates with scores, profiles, and reflections. Its LLM-guided admission decisions are `ADD`, `REPLACE`, or `REJECT`, which retain every new best, preserve structurally diverse approaches, and prefer higher-scoring or simpler representatives among near-duplicates. The configurable pool holds up to 150 entries and typically fewer than 25 in our runs.

Prompt assembly uses a fixed task specification (topology, collective, message size, objective, plan contract, and profile schema) and deterministically compresses the retained experience. Figure 7 illustrates the prompt structure. The initial prompt contains a legal seed; subsequent prompts contain the best program and profile in full; compact `{index, score, node_id, reflection}` records for other entries; and the ten most frequent normalized verifier-error patterns. Pool capacity and compact records bound retained history across rounds. Each round evaluates up to four proposals, selects the best legal one, and retains the incumbent if none improves it.

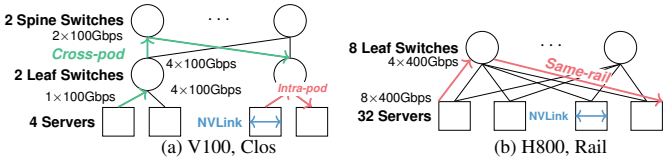


Fig. 8. GPU cluster topologies used in evaluation.

V. Evaluation

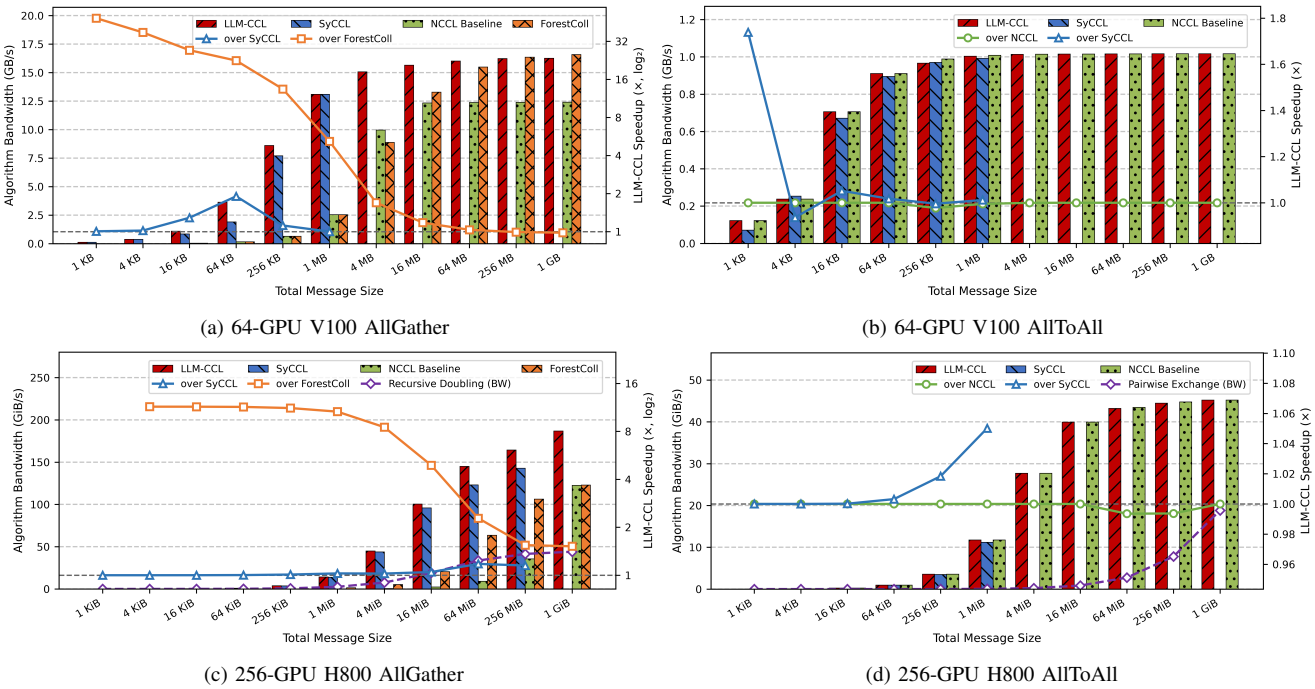
We evaluated LLM-CCL in terms of schedule quality (i.e., the performance of synthesized schedules) and search efficiency (i.e., search time) across cluster topologies, collective types, and data sizes. We compared LLM-CCL against the state-of-the-art schedule synthesizers SyCCL [8] and ForestColl [52], as well as the open-source NCCL library [32]. To assess LLM-CCL's design, we evaluated the effectiveness of its individual components. We also evaluated end-to-end training performance for GPT-6.7B [5] and Qwen3-8B [49] on our DGX-2 testbed under different parallelism settings.

A. Setup

a) Topology: We used two topologies commonly found in GPU clusters. Fig. 8(a) depicts a DGX-2 cluster consisting of four servers. Each server is equipped with 16 NVIDIA V100 GPUs and one 100-Gbps RDMA NIC. The servers are connected through a two-layer Clos network, with each ToR switch connecting two servers. Fig. 8(b) depicts an H800 cluster consisting of 32 servers. Each server is equipped with 8 NVIDIA H800 GPUs and 8 RDMA NICs, each operating at 400 Gbps. The servers are interconnected through a multi-rail network; GPUs with the same local ID are connected to the same switch. In both clusters, the GPUs within each server are interconnected through an NVSwitch fabric over NVLink. V100-related tests were conducted on a real testbed, except for the scalability tests. H800-related tests used parameters measured from a real-world cluster and were executed in simulation. We used the SyCCL simulator, which is based on ASTRA-sim [4]. Using the independently developed SyCCL simulator helps ensure that the observed gains stem from the algorithmic mechanisms discovered by the LLM, rather than from overfitting to a simulator developed as part of our system.

b) Synthesis: We ran schedule synthesis for SyCCL and ForestColl on a server equipped with two 48-core Intel Xeon Platinum 8468V CPUs (192 threads in total) and 1 TB of memory. For SyCCL on the multi-rail topology, we used the same parameter settings reported in its paper and provided in the examples of its open-source codebase. We set a 10-hour solve-time limit for both SyCCL and ForestColl. For LLM-CCL, we used DeepSeek-V4 Flash [15] as the backbone model and a two-hour search budget. Inference accounts for 90.9% of search wall-clock time. Across 46 synthesis runs, we consumed a total of 12.66M input and 24.97M output tokens, costing $0.15–0.20 per run [14].

c) Metric: We report algorithm bandwidth (AlgBW) as the metric for evaluating schedule performance. All message sizes refer to total collective payloads.



(a) 64-GPU V100 AllGather

(b) 64-GPU V100 AllToAll

(c) 256-GPU H800 AllGather

(d) 256-GPU H800 AllToAll

Fig. 9. Schedule performance on the V100 and H800 topologies.

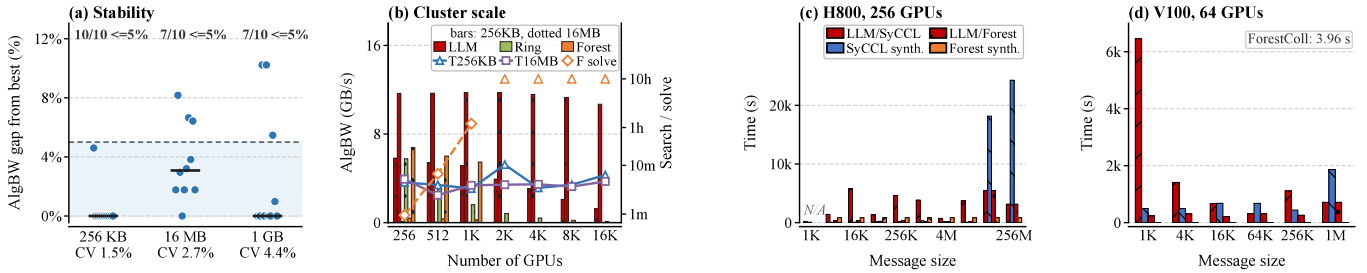


Fig. 10. Search stability, efficiency, and scalability of LLM-CCL. (a) AlgBW gap from the best of ten two-hour runs; bars show medians and shading marks the region within 5% of the best. (b) AlgBW, LLM-CCL time to first beat NCCL, and ForestColl solve time versus cluster size; upward triangles mark 10-hour ForestColl timeouts. (c)–(d) Active LLM-CCL time to reach each baseline's AlgBW versus that baseline's synthesis time on H800 and V100; the legend in (c) applies to (d).

B. Results

1) Schedule Performance: We follow existing work [8], [29] and compare AllGather and AllToAll schedules for total message sizes from 1 K i B to 1 G i B on a 64-GPU V100 testbed and a 256-GPU H800 configuration.

a) AllGather: Figs. 9(a) and 9(c) compare LLM-CCL with ForestColl, SyCCL, and NCCL. On V100, from 1 KB to 1 MB, LLM-CCL delivers $5.2\times$ — $49.2\times$ the algorithm bandwidth of NCCL, $1.0\times$ — $1.9\times$ that of SyCCL, and $5.2\times$ — $48.8\times$ that of ForestColl. At small sizes, NCCL's default ring incurs excessive per-hop overhead, while ForestColl is optimized for large-message throughput; both therefore trail LLM-CCL and SyCCL. At 1 MB, LLM-CCL and SyCCL achieve the same bandwidth despite using different schedules.

SyCCL reports no result above 1 MB on V100 because finding better schedules for larger messages requires more epochs, causing the MILP solver to reach the 10-hour limit. From 4 MB to 1 GB, LLM-CCL achieves, on average, $1.18\times$ the algorithm bandwidth of throughput-optimized ForestColl and up to $1.51\times$ that of NCCL.

On the 256-GPU H800 topology, SyCCL fails to return a result for the largest message size before our 10-hour timeout, even after heuristic parameter tuning. In contrast, LLM-CCL produces schedules for all message sizes and adapts its strategy to the dominant bottleneck. At small sizes, where per-hop overhead dominates, it discovers a two-dimensional same-rail broadcast combined with intra-host fan-out, minimizing hop count. At larger sizes, where bandwidth contention dominates, it discovers a three-stage schedule: source-host fan-out, per-rail inter-host multicast, and destination-host fan-out. This mechanism distributes traffic across multiple rails to reduce link contention. We also compare LLM-CCL with Recursive Doubling, a widely used AllGather algorithm, shown by the purple curve. LLM-CCL outperforms Recursive Doubling

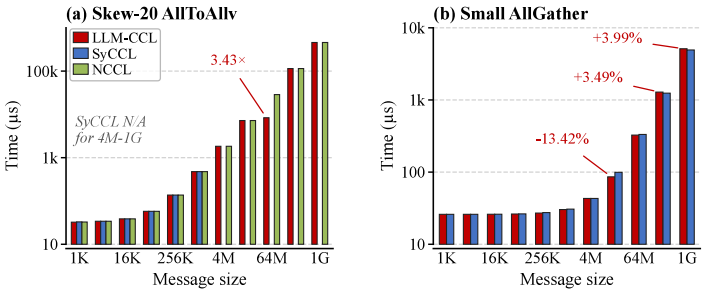


Fig. 11. Completion times (μs; logarithmic axes) for (a) skew-20 AllToAllv with fixed total traffic on 256 H800 GPUs and (b) AllGather on 16 H800 GPUs . The legend in (a) applies to (b); K/M/G denote KiB/MiB/GiB. N/A denotes unavailable SyCCL results. The callout in (a) shows speedup over NCCL; percentages in (b) show time differences relative to SyCCL, with negative values favoring LLM-CCL.

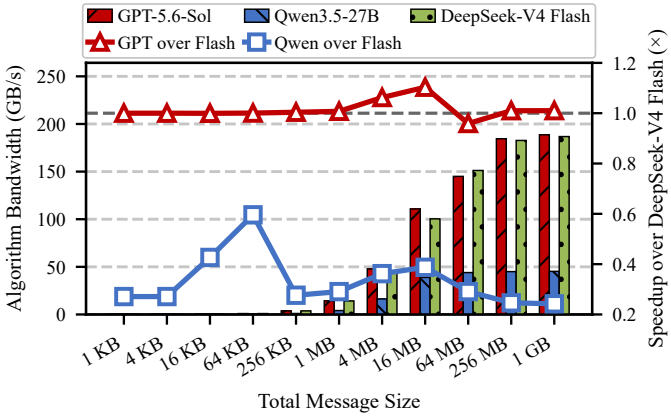


Fig. 12. Backbone sensitivity on 256-GPU H800 AllGather.

across all message sizes. Overall, LLM-CCL delivers up to 17% more algorithm bandwidth than SyCCL and 1.52× — 11.44× the bandwidth of ForestColl; it also outperforms NCCL on H800.

Finding. LLM-CCL maintains high AllGather quality across small- and large-message regimes, with up to 1.9× the algorithm bandwidth of SyCCL.

b) AllToAll: Unlike AllGather, AllToAll decomposes into independent point-to-point transfers, leaving limited room for schedule reordering. NCCL's PXN AllToAll is already near-optimal on both topologies. ForestColl does not support AllToAll and is excluded. As Figs. 9(b) and 9(d) show, LLM-CCL matches NCCL on both V100 and H800. On H800, it also matches SyCCL where SyCCL reports results. The only exception is V100 at 256 KB, where LLM-CCL trails NCCL by 2.3%. This single-point gap falls within the run-to-run variation reported in the stability experiment below. We also compare LLM-CCL with fixed Pairwise Exchange on H800, a widely used AllToAll algorithm. LLM-CCL outperforms fixed Pairwise Exchange across all eleven sizes.

c) AllToAllv: We stress-test schedules synthesized for uniform AllToAll on a skew-20 hot-receiver workload on 256 H800 GPUs with fixed total traffic (Fig. 11(a)). At 1 MiB and 1 GiB, AlgBW drops by 89.2% and 95.2%, respectively, for both LLM-CCL and NCCL relative to uniform traffic. Across all tested sizes, LLM-CCL matches or outperforms NCCL. The speedup at 64 MiB is incidental to the selected multi-hop route via a rotating relay GPU on the destination host.

d) AllToAllv: Small-instance comparison. We also compare AllGather on two H800 hosts (16 GPUs), using two-hour LLM-CCL and ten-hour SyCCL budgets and the SyCCL simulator for both. Across eleven sizes from 1 KiB to 1 GiB, LLM-CCL matches or beats SyCCL in nine of eleven message sizes, with the two exceptions at 3.49% and 3.99% (Fig. 11(b)). This is because SyCCL combines complementary chunk paths to balance intra-host and cross-host bandwidth. Disabling this combination in SyCCL increases its large-message times by 14%–17%, identifying the mechanism behind the two differences.

2) Search Stability: Because LLM proposal generation is stochastic, we evaluate run-to-run stability. We repeat the search ten times for three representative AllGather message sizes (256 KiB , 16 MiB , 1 GiB — small, medium, large) on the 256-GPU H800 topology, each with the same two-hour budget.

As Fig. 10(a) shows, the coefficients of variation in algorithm bandwidth are 1.5%, 2.7%, and 4.4% for the three message sizes, respectively. All ten 256-K i B runs and seven of ten runs for each larger size produce schedules within 5% of the highest observed algorithm bandwidth. These results confirm that LLM-CCL's output is stable across independent runs; the stochasticity of LLM generation does not translate into meaningful performance variance.

3) Backbone Sensitivity: We test whether LLM-CCL requires a specific backbone LLM (Figure 12) . We replace DeepSeek-V4 Flash with GPT-5.6-Sol [33] and Qwen3.5-27B [36], keeping all other settings unchanged (11 AllGather sizes, 256-GPU H800). All three backbones produce legal schedules that pass verification in all 33 cases, confirming that correctness is guaranteed by the deterministic backend regardless of model choice. GPT-5.6-Sol achieves equal or higher algorithm bandwidth in 9 of 11 cases (strictly higher in 7), suggesting that a stronger reasoning model can further improve search quality. Qwen3.5-27B produces legal schedules in all cases but achieves 67% lower bandwidth on average than DeepSeek-V4 Flash , indicating that search quality degrades with weaker models. We retain DeepSeek-V4 Flash as the default for cost and reproducibility reasons. The system remains correct and operational across the evaluated backbones, while schedule quality depends on model capability.

4) Time to Target Schedule Quality: We evaluate search efficiency using *time to target*: the active search time required for LLM-CCL to first generate a schedule whose AlgBW is at least as high as that of SyCCL or ForestColl. We consider matched AllGather cases with available baseline schedules and search traces, covering ten message sizes on the 256-GPU H800 multi-rail topology and six message sizes on the 64-GPU V100 Clos topology. Active search time is measured

TABLE I

END-TO-END TENSOR-PARALLEL TRAINING. ITERATION TIMES ARE IN MILLISECONDS; "N/L" LISTS NCCL/LLM-CCL ITERATION TIMES. AG/RS GAIN IS $T_{\text{NCCL}}/T_{\text{LLM-CCL}} - 1$; FOR EXAMPLE, 0.47× DENOTES A 1.47× SPEEDUP.

Model	TP	Iter. N/L	Reduction	AG gain	RS gain
GPT-6.7B	32	336.5/**260.3**	22.6%	0.47×	0.15×
GPT-6.7B	64	350.3/**179.8**	48.7%	1.56×	0.85×
Qwen3-8B	32	572.6/**324.6**	43.3%	0.89×	0.48×
Qwen3-8B	64	513.5/**429.9**	16.3%	0.80×	0.53×

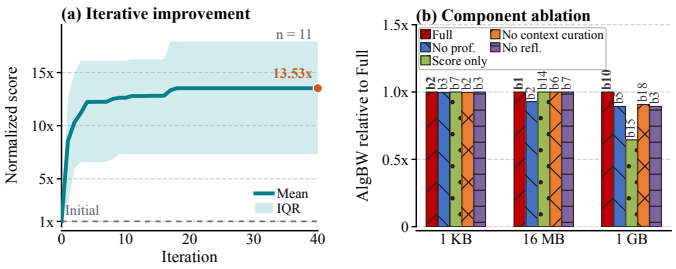


Fig. 13. Iterative improvement and component ablation of LLM-CCL. (a) Best-so-far evaluator score normalized to the initial solution; the line and band show the mean and interquartile range across 11 AllGather tasks. (b) Final AlgBW relative to the full system for 256-GPU H800 AllGather; labels show the first batch reaching each configuration's final quality (batch 0 is the initial program).

from the timestamp of the first search event to the timestamp of finding the first candidate that meets the target.

Figs. 10(c)–(d) show that LLM-CCL progressively reaches competitive schedule quality without waiting for the search budget to expire. Across the corresponding cases, LLM-CCL reaches the SyCCL quality target in a median of 1,412 seconds (23.5 minutes), with times ranging from 145 seconds to 1.8 hours. The median time to the ForestColl target is 305 seconds (5.1 minutes), with a range of 172 seconds to 5,466 seconds. The advantage is most pronounced when MILP synthesis exhibits a long tail. For the 64 MB and 256 MB H800 cases, LLM-CCL reaches the SyCCL quality target 3.32× and 7.70× faster than SyCCL completes synthesis, respectively. These results demonstrate LLM-CCL's anytime behavior: iterative search yields competitive schedules while avoiding long solver delays.

5) Ablation Study:

a) Iterative search vs. the initial solution: We first isolate whether iterative search improves upon the LLM's initial solution. For each of the 11 AllGather tasks, we track the best evaluator score observed through each iteration and normalize it by the score of the initial LLM-generated solution. As Fig. 13(a) shows, the arithmetic mean rises from 1.00× at initialization to 13.53× after 40 iterations. Most of the gain appears in the first several iterations, followed by another improvement before iteration 20. Because the evaluator score is the inverse of schedule completion time, this increase reflects faster schedules rather than merely greater proposal diversity. Thus, the iterative evaluation and refinement loop materially improves upon one-shot LLM generation.

b) Search-component ablations: We compare the full system with profile-free and score-only feedback, and separately ablate context curation and reflection, on 1 KB, 16 MB, and 1 GB AllGather workloads on the 256-GPU H800 topology. Each generation batch contains up to four candidates. Fig. 13(b) reports the final AlgBW relative to the full configuration and annotates the first batch reaching that quality.

At 1 KB, every variant is within 0.4% of the full system. At 16 MB, all variants except the profile-free configuration match its final AlgBW; the full system reaches that quality in batch 1, compared with batches 14, 6, and 7 for score-only feedback, no context curation, and no reflection, respectively. At 1 GB, the effects extend to final quality: profile-free feedback and no reflection reach 89.3% of the full-system AlgBW, no context curation reaches 90.8%, and score-only feedback reaches 64.4%. These results show that the components can improve both search efficiency and the attainable quality within a fixed search budget, especially for bandwidth-dominated workloads.

6) Scalability: We evaluate the scalability of LLM-CCL on simulated Clos topologies ranging from 256 to 16,384 GPUs, a 64× increase in cluster size, using 256 KB and 16 MB AllGather workloads. As shown in Fig. 10(b), LLM-CCL outperforms NCCL in all 14 evaluated cases, achieving arithmetic mean speedups of 356.65× for 256 KB and 30.03× for 16 MB. SyCCL times out on these cases. ForestColl returns schedules at 256, 512, and 1,024 GPUs, but its solve time grows from 57.0 seconds to 4,322.0 seconds and exceeds the 10-hour limit at every larger scale.

More importantly, LLM-CCL retains much more algorithm bandwidth as the cluster grows. From 256 to 16,384 GPUs, its algorithm bandwidth drops by factors of only 4.69× for 256 KB and 1.09× for 16 MB, whereas NCCL's drops by factors of 64.00× and 54.56×, respectively. For 16 MB, LLM-CCL sustains 10.69–11.75 GB/s across seven cluster sizes, while ring bandwidth falls from 5.75 to 0.11 GB/s.

Finding. LLM-CCL maintains both schedule quality and search efficiency from 256 to 16,384 GPUs.

7) End-to-End Training Performance: To validate that the generated schedules translate to real training speedups, we conduct end-to-end tensor-parallel training experiments on the DGX-2 testbed (V100, NVSwitch), the largest multi-node platform available to us for training runs. We evaluate GPT-6.7B and Qwen3-8B under two configurations: TP=32 on two DGX-2 nodes with 32 GPUs and TP=64 on four DGX-2 nodes with 64 GPUs. We profile actual executions of both models and record the invoked collective communication primitives and their corresponding message sizes. AllGather and ReduceScatter are the primary primitives in the collected traces since AllReduce is treated as ReduceScatter followed by AllGather . For each observed primitive–message-size pair, we synthesize an offline LLM-CCL schedule and execute the corresponding schedule in the training workload. We compare LLM-CCL against NCCL with default autotuning under otherwise identical model and training configurations.

Table I shows iteration-time reductions of 16.3% –48.7% across all four configurations. For GPT-6.7B at TP= 32 and

TP= 64, the AllGather times (NCCL/LLM-CCL, in ms) are 56.9/38.6 and 100.9/39.4, respectively, while the ReduceScatter times are 69.6/60.4 and 109.4/59.2. For Qwen3-8B, the corresponding AllGather times are 77.7/41.1 and 139.6/77.4, and the ReduceScatter times are 94.9/64.3 and 163.9/106.8. RS gains trail AG gains because the current MSCCL-based runtime lacks NCCL's reduction-specific optimizations. TP-scaling trends differ because the models operate in different message-size regimes. Qwen3-8B is already in the large-message regime at TP= 32, with a 1.16 GiB vocabulary-output AG versus ∼196.5 MiB for GPT-6.7B. Its AG gain does not increase at TP= 64 , whereas GPT's smaller messages expose a widening inter-node advantage. Together with overlap and residual computation, this explains why GPT's iteration-time reduction increases with TP while Qwen's decreases .

VI. Discussion and Limitations

A. Dependence on the simulator

LLM-CCL relies on simulator feedback to estimate the performance of a candidate schedule on a physical GPU cluster. Independent re-evaluation with the SyCCL simulator reduces the risk of overfitting to simulator-specific timing details: schedules discovered using our simulator remain effective under this separately developed simulator (Section V). Compared with the discrete timing models used by MILP-based synthesizers, a flow-level simulator can naturally incorporate routing, contention, bandwidth sharing, and queueing. Unmodeled jitter, kernel-launch and memory/cache costs matter only for tiny payloads (under 1 KiB) .

B. Asymmetric workloads

Some parallelization strategies, such as Mixture-of-Experts (MoE) expert parallelism, generate skewed or data-dependent traffic in which different GPUs may send or receive unequal volumes of data. Such workloads do not directly satisfy the symmetric collective-demand assumption underlying deterministic symmetry expansion. Supporting them would require extending the demand representation with the traffic volume for each source–destination pair and introducing load-aware expansion or a fast online scheduling heuristic. The symmetric schedule produced by LLM-CCL may serve as a balanced initial schedule that an online scheduler subsequently adapts to the observed traffic skew; evaluating this approach is left to future work. As a fallback, we treat asymmetric traffic as uniform. Given the dynamic fluctuations in MoE traffic, we plan to integrate specialized systems such as DeepEP, MoonEP, or EPIC [7], [10], [51] in future work.

VII. Related Work

Prior work most relevant to LLM-CCL spans three areas: collective communication schedule optimization, LLM agents for systems and software engineering, and evaluation-guided LLM search. We focus on how these approaches differ from LLM-CCL in their search objects, correctness mechanisms, and schedule-construction procedures.

A. Collective Communication Schedule Optimization

Prior approaches to collective schedule optimization fall into three broad categories. First, open-source collective communication libraries, including NCCL [32], RCCL [1], Gloo [17], and HCCL [21], provide implementations of established algorithms such as ring and tree schedules, together with mechanisms for runtime selection or tuning. Second, topology-aware optimizers such as Blink [47] and Themis [38] optimize collective execution for a given topology or scheduling objective. Their scope depends on each system's algorithm families and topology or collective assumptions.

Third, automatic synthesizers, including SCCL [6], TACCL [41], TECCL [29], ForestColl [52], SyCCL [8], and HeteCCL [19], formulate schedule construction as a search or optimization problem, often using constraint solving. They generate schedules for specified topologies, collectives, and message sizes, but the underlying search or solving cost can grow rapidly with cluster scale and topological complexity. Building on SyCCL's symmetry-based decomposition, LLM-CCL assigns symmetry a constructive role: a deterministic symmetry expander transforms a valid single-root Propagation Plan into a complete collective schedule, removing MILP solving from the plan-to-schedule path.

B. LLM Agents for Systems and Software Engineering

LLM agents increasingly generate or revise systems and software artifacts using compilation, testing, or execution feedback [20], [27], [50]. Unlike these general workflows, collective schedule synthesis must satisfy hard topology, causality, and receiver-coverage constraints before evaluation. LLM-CCL therefore confines the LLM to a compact root-0 plan; deterministic components verify and expand it, then evaluate the resulting schedule.

C. Evaluation-Guided LLM Search and Optimization

Evaluation-guided systems use external scores or search history in generate–evaluate–improve loops to refine programs and heuristics [23], [25], [31], [39]. LLM-CCL applies this pattern under hard correctness constraints: static verification rejects illegal candidates before simulation, and a structured bottleneck profile maps flow-level timing to editable Propagation Plan fields. This provides domain-specific feedback while leaving schedule correctness to deterministic components.

VIII. Conclusion

This paper presented LLM-CCL, an LLM-driven framework that represents structured topologies with TopoDSL and searches compact Propagation Plans. A deterministic verifier and symmetry expander transform each legal root-0 plan into a complete collective schedule without MILP-based completion, while simulator feedback guides iterative refinement. Across V100 Clos and H800 multi-rail workloads, LLM-CCL achieves up to $1.9\times$ the algorithm bandwidth of state-of-the-art synthesizers, generates schedules for up to 16,384 GPUs, and reduces end-to-end training iteration time by 16.3%–48.7%.

REFERENCES

[1] Advanced Micro Devices, Inc., "ROCm Collective Communication Library (RCCL)," GitHub repository, 2024, accessed: 2026-08-01. [Online]. Available: https://github.com/ROCm/rccl

[2] "Amazon EC2 R6a Instances," https://aws.amazon.com/ec2/instance-types/r6a/.

[3] J. Araujo, A. Chow, M. Handley, R. Lewis, C. Paasch, J. Padhye, M. Papamichael, G. Steinbrecher, A. Tootoonchian, L. Yuan, S. Anantharamu, A. Dosi, M. Garg, M. Ghazi, T. Hoefler, D. Jayasinghe, J. Jose, A. Kabbani, G. Lu, Y. Wang, K. Doddapaneni, M. Garimella, V. Jain, Y. Le, H. Nagulapalli, S. Narayanan, R. Pan, R. Sabesan, R. Sivaramu, R. Sohan, E. Davis, D. Dumitrescu, M. Kalkunte, B. Mitra, G. Morandin, A. Popa, C. Raiciu, E. Spada, J. Spillane, N. Vaidya, A. Barnea, I. Burstein, E. Cohen, Y. Friedman, N. Katz, M. Moshref, Y. Shpigelman, S. Shuler, S. Shyman, and S. Sur, "Resilient ai supercomputer networking using mrc and srv6," 2026. [Online]. Available: https://arxiv.org/abs/2605.04333

[4] ASTRA-SIM Contributors, "ASTRA-Sim," GitHub repository, ASTRA-SIM, Jul. 2026, accessed: 2026-07-27. [Online]. Available: https://github.com/astra-sim/astra-sim

[5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H.-T. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[6] Z. Cai, Z. Liu, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, and O. Saarikivi, "Synthesizing optimal collective algorithms," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 62–75. [Online]. Available: https://doi.org/10.1145/3437801.3441620

[7] J. Cao, Q. Li, Y. Liu, J. Gao, Y. Zhang, S. Shi, Z. Chen, Y. Wang, J. Zhang, K. He, E. Zhai, J. Dong, B. Fu, and D. Cai, "Balancing and beyond: Communication-centric optimizations in expert parallelism," in *Proceedings of the ACM SIGCOMM 2026 Conference*, ser. SIGCOMM '26. New York, NY, USA: Association for Computing Machinery, 2026, p. 550–563. [Online]. Available: https://doi.org/10.1145/3789240.3829201

[8] J. Cao, S. Shi, J. Gao, W. Liu, Y. Yang, Y. Xu, Z. Zheng, Y. Guan, K. Qian, Y. Liu, M. Xu, T. Wang, N. Wang, J. Dong, B. Fu, D. Cai, and E. Zhai, "Syccl: Exploiting symmetry for efficient collective communication scheduling," in *Proceedings of the ACM SIGCOMM 2025 Conference*, ser. SIGCOMM '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 645–662. [Online]. Available: https://doi.org/10.1145/3718958.3750499

[9] M. Cemri, S. Agrawal, A. Gupta, S. Liu, A. Cheng, Q. Mang, A. Naren, L. E. Erdogan, K. Sen, M. Zaharia, A. Dimakis, and I. Stoica, "AdaEvolve: Adaptive LLM driven zeroth-order optimization," 2026. [Online]. Available: https://arxiv.org/abs/2602.20133

[10] Y. Chen, C. Li, Y. Wang, and M. Wei, "Moonep: A perfectly balanced expert parallelism library via dynamic redundant experts," https://github.com/MoonshotAI/MoonEP, 2026.

[11] S. Cho, H. Son, and J. Kim, "Logical/Physical Topology-Aware Collective Communication in Deep Learning Training," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Los Alamitos, CA, USA: IEEE Computer Society, Mar. 2023, pp. 56–68. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HPCA56546.2023.10071117

[12] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, Mar. 1953. [Online]. Available: https://doi.org/10.1002/j.1538-7305.1953.tb01433.x

[13] D. De Sensi, L. Pichetti, F. Vella, T. De Matteis, Z. Ren, L. Fusco, M. Turisini, D. Cesarini, K. Lust, A. Trivedi, D. Roweth, F. Spiga, S. Di Girolamo, and T. Hoefler, "Exploring gpu-to-gpu communication: Insights into supercomputer interconnects," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '24. IEEE Press, 2024. [Online]. Available: https://doi.org/10.1109/SC41406.2024.00039

[14] "Models & Pricing | DeepSeek API Docs," https://api-docs.deepseek.com/quick_start/pricing/.

[15] DeepSeek-AI, A. Xu, B. Lin, B. Xue, B. Wang, B. Xu, B. Wu, B. Zhang, C. Lin, C. Dong, C. Ling, C. Lu, C. Zhao, C. Deng, C. Hou, C. Xu, C. Shao, C. Ruan, C. Sun, D. Dai, D. Guo, D. Yang, D. Chen, D. Li, D. Ji, E. Li, F. Wei, F. Lin, F. Yuan, F. Xia, F. Dai, G. Hao, G. Chen, G. Cao, G. Meng, G. Li, H. Yu, H. Zhang, H. Xu, H. Li, H. Liang, H. Zhang, H. Luo, H. Wei, H. Yuan, H. Zhang, H. Luo, H. Chen, H. Ji, H. Zhang, H. Ding, H. Tang, H. Cao, H. Gao, H. Qu, H. Zeng, J. Yang, J. Zhu, J. Luo, J. Song, J. Yu, J. Huang, J. Cai, J. Liang, J. Zhou, J. Ye, J. Li, J. Xu, J. Hu, J. Yang, J. Chen, J. Yan, J. Chen, J. Zhou, J. Xiang, J. Yuan, J. Cheng, J. Zhou, J. Zhu, J. Yu, J. Sun, J. Ran, J. Jiang, J. Qiu, J. Li, J. Zheng, J. Song, K. Dong, K. Gao, K. Guan, K. Zhou, K. Huang, K. Yu, L. Wang, L. Zhang, L. Wang, L. Xia, L. Zhang, L. Zhao, L. Guo, L. Luo, L. Ma, L. Zhu, L. Wang, L. Cai, L. Zhang, L. Chen, M. Di, M. Xu, M. Mei, M. Wang, M. Zhang, M. Zhang, M. Tang, M. Li, M. Zhou, M. Han, N. Wang, P. Huang, P. Wang, P. Cong, P. Wang, P. Zhang, Q. Wang, Q. Zhu, Q. Li, Q. Chen, Q. Du, Q. Jiang, R. Tian, R. Xu, R. Lu, R. Xu, R. Ge, R. Zhang, R. Pan, R. Wang, R. Chen, R. Yin, R. Xu, R. Shen, R. Zhang, R. Chen, S. Liu, S. Lu, S. Sun, S. Zhou, S. Chen, S. Cai, S. Nie, S. Wu, S. Chen, S. Hu, S. Liu, S. Hu, S. Ma, S. Wang, S. Yu, S. Zhou, S. Pan, S. Yu, S. Zhou, T. Ni, T. Yun, T. Jin, T. Pei, T. Ye, T. Lin, T. Ji, T. Cui, T. Yue, T. Yu, T. Wang, W. Zhang, W. Xiao, W. Zeng, W. An, W. Zhao, W. Liu, W. Liang, W. Pang, W. Luo, W. Yao, W. Gao, W. Yang, W. Huang, W. Hou, W. Zhang, W. Ma, X. Gao, X. He, X. Wang, X. Wang, X. Bi, X. Liu, X. Wang, X. Chen, X. Zhang, X. Nie, X. Sun, X. Wang, X. Cheng, X. Liu, X. Xie, X. Liu, X. Liu, X. Yu, X. Li, X. Yang, X. Zhang, X. Chen, X. Wang, X. Su, X. Chen, X. Lin, X. Fu, Y. Yan, Y. Wang, Y. Ma, Y. Luo, Y. Zhang, Y. Xu, Y. Ma, Y. Huang, Y. Li, Y. Li, Y. Xu, Y. Zhao, Y. Sun, Y. Wang, Y. Qian, Y. Shao, Y. Yu, Y. Zhang, Y. Ding, Y. Shi, Y. Wu, Y. Xiong, Y. Ma, Y. He, Y. Tang, Y. Zhou, Y. Luo, Y. Zhong, Y. Piao, Y. Wang, Y. Zhang, Y. Chen, Y. Tan, Y. Wei, Y. Ma, Y. Liu, Y. Yang, Y. Guo, Y. Wu, Y. Wu, Y. Li, Y. Cheng, Y. Ou, Y. Xu, Y. Li, Y. Wang, Y. Yang, Y. Xu, Y. Wu, Y. Meng, Y. Zou, Y. Zha, Y. Xiong, Y. Chen, Y. Lin, Y. Cao, Y. Wang, Y. Zhang, Y. Yan, Y. Lin, Y. Gu, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. Zhou, Y. Huang, Z. Wu, Z. Wang, Z. Zhao, Z. Ren, Z. Zhang, Z. Sha, Z. Fu, Z. Ju, Z. Xu, Z. Xie, Z. Zhang, Z. Gao, Z. Hao, Z. Gou, Z. Ma, Z. Yan, Z. Shao, Z. Huang, Z. Chen, Z. Wu, Z. Ren, Z. Wu, Z. Li, Z. Zhang, Z. Xu, Z. Wang, Z. Qu, Z. Gu, Z. Zhu, Z. Li, Z. Zhang, Z. Xie, Z. Gao, Z. Wan, Z. Pan, and Z. Yao, "Deepseek-v4: Towards highly efficient million-token context intelligence," 2026. [Online]. Available: https://arxiv.org/abs/2606.19348

[16] A. Gibiansky, "Bringing HPC techniques to deep learning," Baidu Research Technical Blog, Feb. 2017, accessed: 2026-07-21. [Online]. Available: https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

[17] Gloo Contributors, "Gloo," GitHub repository, 2024, accessed: 2026-08-01. [Online]. Available: https://github.com/pytorch/gloo

[18] Z. Gou, Z. Shao, Y. Gong, Y. Shen, Y. Yang, N. Duan, and W. Chen, "CRITIC: Large language models can self-correct with tool-interactive critiquing," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=Sx038qxjek

[19] C. Hei, F. Li, J. Li, J. Cao, C. Gao, X. Sha, T. Liu, D. Zhang, E. Zhai, and X. Wang, "HeteCCL: Synthesizing Near-Optimal collective communication schedules for heterogeneous GPU clusters," in *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. Renton, WA: USENIX Association, May 2026, pp. 2533–2551. [Online]. Available: https://www.usenix.org/conference/nsdi26/presentation/hei

[20] D. Huang, J. M. Zhang, M. Luck, Q. Bu, Y. Qing, and H. Cui, "Agentcoder: Multi-agent-based code generation with iterative testing and optimisation," 2024. [Online]. Available: https://arxiv.org/abs/2312.13010

[21] Huawei Technologies Co., Ltd., "CANN HCCL," Gitcode repository, 2024, accessed: 2026-08-01. [Online]. Available: https://gitcode.com/cann/hccl

[22] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, C. Young, X. Zhou, Z. Zhou, and

D. A. Patterson, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589350

[23] R. T. Lange, Y. Imajuku, and E. Cetin, "Shinkaevolve: Towards open-ended and sample-efficient program evolution," in *The Fourteenth International Conference on Learning Representations*, 2026. [Online]. Available: https://openreview.net/forum?id=lKEdGCoDNC

[24] D. Lim and J. Kim, "Tidalmesh: Topology-driven allreduce collective communication for mesh topology," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025, pp. 1526–1540.

[25] F. Liu, T. Xialiang, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, and Q. Zhang, "Evolution of heuristics: Towards efficient automatic algorithm design using large language model," in *Proceedings of the 41st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, Eds., vol. 235. PMLR, 21–27 Jul 2024, pp. 32 201–32 223. [Online]. Available: https://proceedings.mlr.press/v235/liu24bs.html

[26] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024. [Online]. Available: https://aclanthology.org/2024.tacl-1.9/

[27] Q. Liu, M. Zou, H. Zhang, D. Du, Y. Xia, and H. Chen, "Sharpen the spec, cut the code: A case for generative file system with SYSSPEC," 2025. [Online]. Available: https://arxiv.org/abs/2512.13047

[28] S. Liu, S. Agarwal, M. Maheswaran, M. Cemri, Z. Li, Q. Mang, A. Naren, E. Boneh, A. Cheng, M. Z. Pan, A. Du, K. Keutzer, A. Cheung, A. G. Dimakis, K. Sen, M. Zaharia, and I. Stoica, "EvoX: Meta-evolution for automated discovery," 2026. [Online]. Available: https://arxiv.org/abs/2602.23413

[29] X. Liu, B. Arzani, S. K. R. Kakarla, L. Zhao, V. Liu, M. Castro, S. Kandula, and L. Marshall, "Rethinking machine learning collective communication as a multi-commodity flow problem," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 16–37. [Online]. Available: https://doi.org/10.1145/3651890.3672249

[30] K. Mandakolathur and S. Jeaugey, "Doubling All2All performance with NVIDIA Collective Communication Library 2.12," NVIDIA Technical Blog, Feb. 2022, accessed: 2026-07-21. [Online]. Available: https://developer.nvidia.com/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/

[31] A. Novikov, N. Vũ, M. Eisenberger, E. Dupont, P.-S. Huang, A. Z. Wagner, S. Shirobokov, B. Kozlovskii, F. J. R. Ruiz, A. Mehrabian, M. P. Kumar, A. See, S. Chaudhuri, G. Holland, A. Davies, S. Nowozin, P. Kohli, and M. Balog, "AlphaEvolve: A coding agent for scientific and algorithmic discovery," 2025. [Online]. Available: https://arxiv.org/abs/2506.13131

[32] NVIDIA Corporation, "NVIDIA Collective Communication Library (NCCL)," GitHub repository, 2024, accessed: 2026-08-01. [Online]. Available: https://github.com/NVIDIA/nccl

[33] OpenAI, "GPT-5.6: Frontier intelligence that scales with your ambition," https://openai.com/index/gpt-5-6/, Jul. 2026.

[34] J. Pjesivac-Grbovic, "Towards automatic and adaptive optimizations of mpi collective operations," Dec. 2007. [Online]. Available: https://trace.tennessee.edu/handle/20.500.14382/23716

[35] K. Qian, Y. Xi, J. Cao, J. Gao, Y. Xu, Y. Guan, B. Fu, X. Shi, F. Zhu, R. Miao, C. Wang, P. Wang, P. Zhang, X. Zeng, E. Ruan, Z. Yao, E. Zhai, and D. Cai, "Alibaba HPN: A data center network for large language model training," in *Proceedings of the ACM SIGCOMM 2024 Conference*. Association for Computing Machinery, 2024, pp. 691–706. [Online]. Available: https://doi.org/10.1145/3651890.3672265

[36] Qwen Team, "Qwen3.5: Towards native multimodal agents," February 2026. [Online]. Available: https://qwen.ai/blog?id=qwen3.5

[37] R. Rabenseifner, "Optimization of collective reduction operations," in *Computational Science – ICCS 2004*. Springer Berlin Heidelberg, 2004, pp. 1–9. [Online]. Available: https://doi.org/10.1007/978-3-540-24685-5_1

[38] S. Rashidi, W. Won, S. Srinivasan, S. Sridharan, and T. Krishna, "Themis: A network bandwidth-aware collective scheduling policy for distributed training of dl models," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. Association for Computing Machinery, 2022, pp. 581–596. [Online]. Available: https://doi.org/10.1145/3470496.3527382

[39] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. R. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, P. Kohli, and A. Fawzi, "Mathematical discoveries from program search with large language models," *Nature*, vol. 625, no. 7995, pp. 468–475, 2024. [Online]. Available: https://doi.org/10.1038/s41586-023-06924-6

[40] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," 2018. [Online]. Available: https://arxiv.org/abs/1802.05799

[41] A. Shah, V. Chidambaram, M. Cowan, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, O. Saarikivi, and R. Singh, "TACCL: Guiding collective algorithm synthesis using communication sketches," in *20th USENIX Symposium on Networked Systems Design and Implementation*, 2023, pp. 593–612. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/shah

[42] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 36. Curran Associates, Inc., 2023, pp. 8634–8652. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf

[43] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019. [Online]. Available: https://arxiv.org/abs/1909.08053

[44] M. Si, P. Balaji, Y. Chen, C.-H. Chu, A. Gangidi, S. Hasan, S. Iyengar, D. Johnson, B. Liu, R. Ren, D. Shah, A. J. Shetty, G. Steinbrecher, Y. Wang, B. Wu, X. Xie, J. Yang, M. Yang, K. Yu, M. Yu, C. Zhao, W. Bland, D. Boyda, S. Gumudavelli, P. Kannan, C. Lumezanu, R. Miao, Z. Qu, V. Ramesh, M. Samoylov, J. Seidel, S. Sundaresan, F. Tian, Q. Tan, S. Zhang, Y. Zhao, S. Zheng, A. Zhu, and H. Zeng, "Collective communication for 100k+ gpus," 2025. [Online]. Available: https://arxiv.org/abs/2510.20171

[45] K. Team, T. Bai, Y. Bai, Y. Bao, M. C., J. Cai, X. Cai, P. Cao, Y. Cao, Z. Chai, Y. Charles, H. S. Che, G. Chen, G. Chen, G. Chen, H. Chen, J. Chen, J. Chen, J. Chen, K. Chen, P. Chen, R. Chen, W. Chen, X. Chen, Y. Chen, Y. Chen, Y. Chen, Y. Chen, Y. Chen, Y. Chen, Y. Chen, Z. Chen, D. Cheng, Y. Cheng, J. Cui, J. Cui, A. Dai, J. Deng, H. Ding, R. Ding, S. Ding, M. Dong, M. Dong, Y. Dong, Y. Dong, A. Du, C. Du, D. Du, J. Du, Y. Du, Y. Fan, J. Feng, Q. Feng, Y. Feng, K. Fu, Q. Fu, F. Gao, H. Gao, J. Gao, T. Gao, W. Gao, S. Geng, J. Gong, L. Gong, S. Gong, X. Gong, Q. Gu, Y. Gu, S. Guan, H. Guo, S. Guo, X. Guo, Z. Guo, B. Hao, W. Hao, X. Hao, D. He, H. He, L. He, Q. He, W. He, X. He, X. He, Y. He, Y. He, C. Hong, T. Hong, H. Hu, J. Hu, R. Hu, W. Hu, Y. Hu, Z. Hu, L. Hua, J. Huang, K. Huang, R. Huang, S. Huang, W. Huang, Y. Huang, Z. Huang, Z. Huang, Y. Hui, C. Jia, Y. Jiang, Z. Jiang, Z. Jiang, W. Jin, X. Jin, Y. Jing, H. Kong, G. Lai, A. Li, C. Li, C. Li, C. Li, F. Li, G. Li, H. Li, J. Li, J. Li, L. Li, L. Li, L. Li, L. Li, W. Li, W. Li, X. Li, Y. Li, Y. Li, Y. Li, Y. Li, Z. Li, Z. Li, Z. Li, Z. Li, Z. Li, J. Lin, X. Lin, Y. Lin, Z. Lin, Z. Lin, B. Liu, B. Liu, C. Liu, L. Liu, S. Liu, S. Liu, S. Liu, T. Liu, W. Liu, Y. Liu, Y. Liu, Y. Liu, Y. Liu, Z. Liu, Z. Liu, E. Lu, H. Lu, L. Lu, T. Lu, Z. Lu, A. Luo, G. Luo, J. Luo, Y. Luo, B. Lyu, W. Lyu, S. Mao, Y. Mei, X. Men, M. Ni, Y. Niu, S. Pan, S. Peng, Z. Qi, R. Qin, Z. Qin, Z. Qin, H. Qiu, J. Qiu, J. Qiu, B. Qu, Y. Qu, Z. Shang, Y. Shao, H. Shen, J. Shi, J. Shi, L. Shi, S. Shi, W. Siu, P. Song, X. Song, J. Su, Y. Su, Z. Su, L. Sui, J. Sun, J. Sun, S. Sun, S. Sun, T. Sun, Y. Sun, Y. Tai, C. Tang, H. Tang, S. Tang, Z. Tang, C. Tian, R. Tian, Y. Tian, W. Tu, C. Wang, C. Wang, C. Wang, D. Wang, F. Wang, H. Wang, H. Wang, H. Wang, H. Wang, H. Wang, J. Wang, J. Wang, J. Wang, J. Wang, L. Wang, S. Wang, S. Wang, S. Wang, S. Wang, S. Wang, T. Wang, W. Wang, X. Wang, X. Wang, X. Wang, X. Wang, Y. Wang, Y. Wang, Y. Wang, Y. Wang, Y. Wang, Y. Wang, Y. Wang, Y. Wang, Z. Wang, Z. Wang, Z. Wang, Z. Wang, Z. Wang, C. Wei, M. Wei, S. Wei, Z. Wen, F. Wu, H. Wu, R. Wu, W. Wu, X. Wu, Y. Wu, Y. Wu, Y. Wu, Z. Wu, X. Xian, C. Xiang, Y. Xiang, B. Xiao, C. Xiao, X. Xiao, J. Xie, X. Xie, Y. Xie, Z. Xie, B. Xing, Y. Xiong, B. Xu, B. Xu, J. Xu, J. Xu, J. Xu, J. Xu, L. H. Xu, Q. Xu, S. Xu, S. Xu, T. Xu, T. Xu, W. Xu, X. Xu, Y. Xu, Y. Xu, Y. Xu, Z. Xu, H. Xue, J. Yan, Y. Yan, F. Yang, G. Yang, H. Yang, J. Yang, R. Yang, W. Yang, X. Yang, X. Yang,

Y. Yang, Y. Yang, Y. Yang, Y. Yang, Z. Yang, Z. Yang, Z. Yang, Z. Yang, H. Yao, D. Ye, H. Ye, W. Ye, Z. Ye, B. Yin, H. Yin, X. Yin, C. Yu, H. Yu, L. Yu, S. Yu, S. Yu, T. Yu, E. Yuan, M. Yuan, T. Yue, W. Yue, Y. Yue, D. Zha, H. Zhan, B. H. Zhang, D. Zhang, F. Zhang, H. Zhang, H. Zhang, H. Zhang, J. Zhang, J. Zhang, J. Zhang, K. Zhang, M. Zhang, P. Zhang, Q. Zhang, R. Zhang, R. Zhang, S. Zhang, S. Zhang, X. Zhang, X. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, Z. Zhang, Z. Zhang, B. Zhao, C. Zhao, F. Zhao, J. Zhao, J. Zhao, S. Zhao, W. Zhao, X. Zhao, X. Zhao, Y. Zhao, Z. Zhao, H. Zheng, H. Zheng, R. Zheng, S. Zheng, T. Zheng, H. Zhong, L. Zhong, L. Zhong, M. Zhou, Q. Zhou, R. Zhou, R. Zhou, X. Zhou, Y. Zhou, Z. Zhou, J. Zhu, L. Zhu, X. Zhu, Y. Zhu, Y. Zhu, Z. Zhu, C. Zhuang, W. Zhuang, and X. Zu, "Kimi k3: Open frontier intelligence," 2026. [Online]. Available: https://arxiv.org/abs/2607.24653

[46] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in MPICH," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, Feb. 2005. [Online]. Available: https://doi.org/10.1177/1094342005051521

[47] G. Wang, S. Venkataraman, A. Phanishayee, N. Devanur, J. Thelin, and I. Stoica, "Blink: Fast and generic collectives for distributed ml," in *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds., vol. 2, 2020, pp. 172–186. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2020/file/cd3a9a55f7f3723133fa4a13628cdf03-Paper.pdf

[48] H. Wei, Y. Sun, and Y. Li, "Deepseek-ocr 2: Visual causal flow," 2026. [Online]. Available: https://arxiv.org/abs/2601.20552

[49] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu, "Qwen3 technical report," 2025. [Online]. Available: https://arxiv.org/abs/2505.09388

[50] G. Zhang, S. Zhu, A. Wei, Z. Song, A. Nie, Z. Jia, N. Vijaykumar, Y. Wang, and K. Olukotun, "AccelOpt: A self-improving LLM agentic system for AI accelerator kernel optimization," 2025. [Online]. Available: https://arxiv.org/abs/2511.15915

[51] C. Zhao, S. Zhou, L. Zhang, C. Deng, Z. Xu, Y. Liu, K. Yu, J. Li, and L. Zhao, "Deepep: an efficient expert-parallel communication library," https://github.com/deepseek-ai/DeepEP, 2025.

[52] L. Zhao, S. Maleki, Y. Wang, Z. Wang, Z. Yang, H. Pourreza, and A. Krishnamurthy, "ForestColl: Throughput-optimal collective communications on heterogeneous network fabrics," in *23rd USENIX Symposium on Networked Systems Design and Implementation*, 2026, pp. 2067–2093. [Online]. Available: https://www.usenix.org/conference/nsdi26/presentation/zhao-liangyu

AI USE

The authors used OpenAI Codex for two limited purposes. First, the AI system was used throughout the manuscript to identify and revise grammatical errors, awkward phrasing, and unclear sentences. Second, it was used to assist with writing portions of the implementation code, specifically portions of the experimental implementation and supporting scripts. The AI system was not used to generate the scientific ideas, claims, experimental results, analyses, figures, or conclusions presented in this paper. All AI-assisted text and code were carefully reviewed, edited, tested, and validated by the authors. The authors take full responsibility for the accuracy, originality, and integrity of the manuscript and the associated code.