JOINT COMMUNICATION-COMPUTATION OPTIMIZATION FOR MULTI-GPU HETEROGENEOUS GNN TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Full-graph training of heterogeneous graph neural networks (GNNs) remains difficult to scale because irregular relation-specific workloads and cross-GPU aggregation jointly limit computation and communication efficiency. This bottleneck is particularly acute in learning-assisted large-scale supply chain production planning, where candidate-variable screening depends on preserving long-range heterogeneous dependencies, making sampling an unsuitable substitute for full-graph execution. We propose a phase-aware multi-GPU framework that jointly optimizes workload organization and cross-GPU communication while preserving message-passing semantics. At preparation time, a lightweight surrogate selects the graph workload granularity by trading scheduling flexibility against one-time materialization overhead. During recurrent training, asynchronous aggregation overlaps cross-GPU reduction with local computation, while a net-benefit controller selects the communication granularity and reverts to blocking execution when overlap is not worthwhile. The resulting transformations preserve relation-specific full-graph aggregation up to floating-point reduction order. Experiments on five consecutive industrial supply-chain graphs show that the selected preparation configuration is within 0.709% of the empirical optimum on a held-out snapshot. The complete framework reduces mean steady-state epoch time by 62.50% and 50-epoch wall-clock time by 58.50%, with the additional preparation cost amortized after approximately four epochs.

1 INTRODUCTION

Learning-to-optimize (L2O) provides a promising way to accelerate repeatedly solved optimization problems by exploiting information from previously solved instances (Bengio et al., 2021). Large-scale supply-chain production planning (SCPP) is a representative setting, where historical solutions can be used to predict the activity of candidate decision variables and screen unlikely candidates before solving the reduced mathematical program (Gasse et al., 2019). Because candidate relevance depends on heterogeneous material, resource, temporal, and transportation dependencies, graph neural networks (GNNs) provide a natural representation for such learning-assisted optimization. Importantly, these dependencies can span long relational paths through context entities such as inventory and capacity nodes; removing such paths can change which candidates appear relevant even when the prediction target is local. This makes sampling or subgraph training unsuitable when the original relation-specific dependencies must be preserved, motivating the full-graph heterogeneous GNN setting studied in this work.

However, full-graph training on industrial heterogeneous graphs is difficult to scale. Relation types differ substantially in cardinality, destination degree, and transformation cost, leading to irregular computation and workload imbalance. Multi-GPU parallel execution expands memory and computation capacity, but introduces cross-GPU aggregation and synchronization overhead. Existing systems have improved scalability through graph partitioning, communication planning, prefetching, and communication-computation overlap (Zheng et al., 2020; Gandhi & Iyer, 2021; Wan et al., 2022; Du et al., 2024; Sarkar et al., 2024; Luo et al., 2024). These techniques typically optimize a particular execution component: workload organization determines the decomposition presented to the runtime, while communication mechanisms operate on the communication pattern

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

induced by that decomposition. In heterogeneous full-graph training, however, the two decisions are inherently coupled because the organization of relation-specific destination workloads directly determines which partial aggregates must cross device boundaries.

This coupling creates a non-trivial granularity trade-off. Finer workload organization can improve load balance and expose more opportunities for communication-computation overlap, but also increases packing, indexing, collective-launch, and synchronization overhead. The trade-off is further complicated by different execution frequencies: graph organization and device-local materialization are paid once for a graph snapshot, whereas aggregation and communication recur at every layer and epoch. A configuration that is attractive in preparation may therefore be inefficient during recurrent training, while maximizing communication overlap may introduce more overhead than runtime benefit. This raises the central question of this work: *how should a multi-GPU system jointly organize heterogeneous full-graph workloads and schedule cross-GPU aggregation when the two decisions are coupled but their costs are incurred over different execution horizons?*

To address this question, we propose a phase-aware joint communication-computation optimization framework for exact heterogeneous GNN training on a single multi-GPU server. During graph preparation, relation-specific destination workloads are organized into reusable physical scheduling units, and a lightweight runtime surrogate selects their granularity by balancing workload-distribution flexibility against one-time preparation overhead. During recurrent training, cross-GPU aggregation is decomposed along the destination dimension so that completed reductions overlap with subsequent local aggregation. Rather than maximizing overlap itself, a net-benefit model selects the communication granularity according to its reduction in critical-path communication after accounting for decomposition overhead, and retains blocking execution when asynchronous overlap is not beneficial.

We make the following three contributions:

- **A phase-aware joint communication-computation formulation.** We formulate multi-GPU parallel full-graph heterogeneous GNN training by explicitly coupling workload organization with the cross-GPU aggregation pattern it induces. The objective accounts for both one-time graph preparation and recurrent synchronized execution, making the amortization horizon explicit rather than optimizing preparation cost or communication overlap in isolation.
- **Structure-aware workload organization with provable semantic preservation.** We define relation–destination TARGETGROUPs as semantic aggregation units and pack their device-local fragments into reusable SUPERBLOCKs that expose configurable physical scheduling granularity. A graph-dependent runtime surrogate selects this granularity by balancing workload-distribution flexibility against preparation overhead. We prove that reducing device-local partial aggregates recovers the original relation-specific full-graph aggregation up to floating-point reduction order.
- **Exact communication-computation overlap driven by net runtime benefit.** We develop a destination-wise asynchronous aggregation scheme that overlaps cross-GPU reduction with local computation without altering full-graph message-passing semantics. The communication granularity is selected according to net critical-path benefit rather than overlap ratio alone, with blocking reduction as an explicit fallback. Our experiments further show that the maximum-overlap configuration is not necessarily the runtime-optimal one.

On five consecutive industrial heterogeneous graphs, the selected preparation configuration is within 0.709% of the empirical optimum on a held-out snapshot. The complete framework reduces mean steady-state epoch time by 62.50% and 50-epoch wall-clock time by 58.50%, with the additional preparation cost amortized after approximately four epochs.

2 RELATED WORK

Graph learning for structured decision support. Learning-to-optimize methods use solved instances to improve later optimization without necessarily replacing the mathematical model (Bengio et al., 2021). Graph representations are particularly useful when variable relevance is governed by relational constraints: learned policies have been used to select branching decisions (Khalil et al., 2017), and graph convolutional models can imitate strong branching while retaining an exact solver as the final decision mechanism (Gasse et al., 2019). Supply chains are also naturally relational. Existing

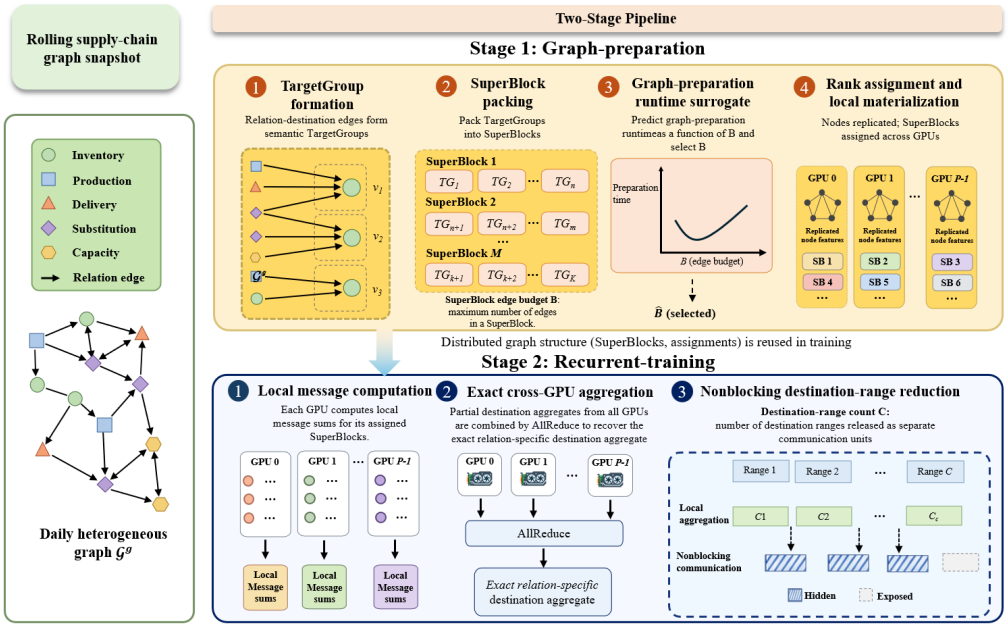


Figure 1: Overview of the proposed joint communication-computation optimization framework. Graph preparation forms semantic relation–destination TARGETGROUPS, packs their device-local fragments into reusable SUPERBLOCKS, selects a graph-specific workload granularity, and materializes the multi-GPU parallel execution state. Recurrent training computes local partial aggregates and reduces disjoint destination ranges nonblockingly. The graph topology and heterogeneous aggregation semantics are unchanged.

studies use graph-based representations for inventory control, probabilistic supply and inventory prediction, and supply-relationship forecasting (Kotecha & del Rio Chanona, 2025; Ahn et al., 2024; Xiang et al., 2025). These works establish the value of structured representations, but their main objective is predictive or decision quality. Our focus is different: we study how the full heterogeneous graph learner itself can be trained efficiently when its typed aggregation structure must be preserved.

Heterogeneous graph models retain multiple entity and relation types because different edges need not carry interchangeable semantics. Relation-aware projection and attention (Chairatanakul et al., 2021), heterogeneous contrastive learning (Zhang et al., 2023), and dynamic heterogeneous embedding (Nguyen et al., 2023; Li et al., 2024) illustrate how structural, feature, and temporal heterogeneity can be represented. Rolling planning graphs exhibit all three forms: operational entities have different types, constraints induce typed dependencies, and consecutive snapshots preserve much of the topology while their attributes change. These modeling considerations motivate relation-specific full-graph aggregation, but they do not determine how that aggregation should be organized across GPUs.

Scalable and parallel GNN training. Sampling and clustering reduce the active graph footprint and improve locality (Chiang et al., 2019). Such methods are effective when stochastic or subgraph training is acceptable; they are not interchangeable with the full-graph setting studied here because excluding a dependency changes the information available to a candidate-variable classifier. Parallel systems instead partition topology, features, or computation. DistDGL supports large partitioned graphs (Zheng et al., 2020), while recent surveys identify remote access, irregular computation, workload imbalance, communication, and synchronization as recurring bottlenecks (Lin et al., 2023; Shao et al., 2024). Feature-only partitioning with communication planning (Du et al., 2024) and continuous prefetching (Sarkar et al., 2024) further show that data placement and movement must be considered together with computation. Our node-replicated, edge-partitioned setting removes repeated remote reads of node states and shifts the principal partition-induced traffic to the reduction of partial destination aggregates.

Graph partitioning and workload organization. Classical edge-cut assigns vertices and minimizes cross-partition edges, whereas vertex-cut distributes edges and replicates selected vertex state. PowerGraph demonstrated that vertex cuts can balance highly skewed natural graphs (Gonzalez et al., 2012). Recent multi-GPU GNN studies likewise explore vertex-cut organization and compare the runtime and memory effects of alternative partitions (Cao et al., 2023; Merkel et al., 2025). However, an edge count or cut objective does not fully characterize a heterogeneous GNN workload. Incoming edges with different relation labels invoke distinct transformations and must contribute to different relation-specific outputs. Our TARGETGROUP makes this semantic aggregation boundary explicit; SUPERBLOCK packing then introduces a separate physical granularity for scheduling without collapsing relation types.

Communication-computation overlap. P3 pipelines graph computation and communication at scale (Gandhi & Iyer, 2021); PipeGCN overlaps inter-partition feature and gradient communication with intra-partition computation (Wan et al., 2022); and heterogeneous scheduling methods jointly place and order parallel GNN tasks (Luo et al., 2024). These systems demonstrate that communication volume and communication exposed on the critical path are distinct. Yet finer overlap is not free: each additional range can incur launch latency, indexing, buffer operations, and synchronization. Moreover, in our setting the communication units are induced by the preceding organization of typed destination workloads. The proposed framework therefore differs in two respects. It coordinates computation organization with the resulting cross-GPU aggregation schedule, and it selects overlap granularity by *net wall-clock benefit* rather than hidden communication alone.

3 PROBLEM SETTING AND SYSTEM OVERVIEW

3.1 ROLLING PLANNING AS HETEROGENEOUS GRAPH LEARNING

We study a learning-assisted rolling-horizon supply-chain production planning application. Each planning cycle coordinates production, inventory, material substitution, resource capacity, and inter-plant transportation over multiple periods. Material-balance equations couple production with inventories and component requirements; transportation redistributes material subject to source, destination, and lead time; substitution introduces alternative component paths; and capacity constraints connect many activities through shared resources. As orders, forecasts, inventories, capacities, and material availability change, a new optimization instance is solved. We refer to the information available in one cycle as a *planning snapshot*.

Historical optimal solutions provide activity labels for candidate transportation and substitution variables. A heterogeneous GNN learns an activity score for each candidate in a new snapshot; sufficiently low-score candidates can be removed before the reduced mathematical program is solved. Thus the learner reduces the search space, while the original optimization model remains responsible for feasibility and the final plan. The graph also retains unlabeled context entities, such as inventory or capacity nodes, because they connect candidate decisions through material and resource constraints. This distinction matters operationally: dropping a context node or sampling away a long-range dependency may change which candidates appear relevant even when the prediction target is local. Appendix A details the application pipeline and graph schema.

3.2 WHY THE PARALLEL PROBLEM IS JOINT

The planning snapshot is represented as a directed heterogeneous graph. Nodes correspond to typed planning entities and edges instantiate typed operational dependencies within and across periods. All incoming messages with relation r and destination v contribute to one relation-specific output. This relation–destination structure is simultaneously a semantic property of the GNN and a source of execution irregularity: destination degrees, relation cardinalities, memory-access patterns, and transformation costs vary substantially.

In a node-replicated, edge-partitioned implementation, every GPU can construct messages from local edges without fetching remote source states. When edges for one relation–destination output reside on several GPUs, however, their partial aggregates must be reduced before the layer can consume the output. Workload organization therefore determines both the slowest-GPU computation time and the set and shape of cross-GPU reductions. Communication scheduling then determines which portion of

those reductions is hidden behind useful computation. Treating either component in isolation misses this causal link.

The costs also occur at different frequencies. Sorting, packing, assignment, and tensor materialization are paid once for a graph snapshot, whereas message computation and cross-GPU aggregation repeat across layers and epochs. We consequently organize the joint problem into a preparation phase and a recurrent-training phase. This is a decomposition by execution horizon, not an assumption that computation and communication are independent: the prepared graph structure is precisely the interface through which the first phase constrains the second.

Heterogeneous full-graph message passing. Let $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ be a directed heterogeneous graph, with relation-specific edge set $\mathcal{E}_r$ for $r \in \mathcal{R}$. At layer ℓ, the message on $(u, v) \in \mathcal{E}_r$ is

$$m^{(\ell)}_{u\to v,r} = \Phi^{(\ell)}_r(h^{(\ell)}_u, h^{(\ell)}_v, a_{uv}), \qquad s^{(\ell)}_{r,v} = \operatorname*{AGG}_{(u,v)\in\mathcal{E}_r} m^{(\ell)}_{u\to v,r}. \tag{1}$$

Different relations retain distinct transformations and aggregates. Our implementation uses a sum followed by the original relation-specific normalization; the arguments below also apply to decomposable associative reductions. Full-graph training means that every edge in every $\mathcal{E}_r$ participates in the layer computation. We do not use sampling, stale node representations, or asynchronous parameter updates.

Execution model. We consider P devices on one server. Node features, intermediate node states, and model parameters are replicated, while each relation's edges form a disjoint device partition, $\mathcal{E}_r = \biguplus_{p=1}^{P} \mathcal{E}_{r,p}$. Replication removes repeated remote feature fetches but requires each destination's device-local partial aggregates to be combined. The design is suitable when node states fit on each device and edge structure dominates partitioned storage; Appendix E discusses this scope.

Objective. Let B control preparation-time workload granularity and C control recurrent communication granularity. For an N-epoch run on snapshot g, we minimize

$$T^g_{\text{total}}(B, C; N) = T^g_{\text{prep}}(B) + \sum_{e=1}^{N} \max_{p\in[P]} T^g_{e,p}(B, C). \tag{2}$$

The maximum reflects synchronized execution. Equation 2 is deliberately finite-horizon: a preparation decision is useful only if recurrent savings amortize its one-time cost. We keep hardware, the edge-partition policy, the GNN operator, and collective implementation fixed and optimize the physical organization and schedule induced on top of them.

4 JOINT COMMUNICATION-COMPUTATION OPTIMIZATION

The framework exposes three different units rather than forcing one graph partition to serve every purpose. A TARGETGROUP is the semantic unit defined by a relation and destination; it specifies which messages must be aggregated together. A SUPERBLOCK is the physical preparation unit used to assign and materialize complete GPU-local TARGETGROUP fragments. A destination range is the recurrent communication unit whose partial outputs are reduced together. Separating these roles is central: the semantic unit is fixed by the GNN, whereas the two physical units can be configured to reduce runtime.

4.1 EXACT EDGE-PARTITIONED AGGREGATION

For a relation–destination pair (r, v), define the semantic aggregation unit $Q_{r,v} = \{(u, v) \in \mathcal{E}_r\}$ and its device-local fragment $Q^p_{r,v} = Q_{r,v} \cap \mathcal{E}_{r,p}$. Device p computes $s^p_{r,v} = \sum_{(u,v)\in Q^p_{r,v}} m_{u\to v,r}$, after which a cross-device reduction gives $s_{r,v} = \sum_p s^p_{r,v}$.

Proposition 1 (Semantic preservation). *If $\{\mathcal{E}_{r,p}\}_{p=1}^{P}$ is a disjoint cover of $\mathcal{E}_r$ and every edge uses the same relation-specific message function as in Equation 1, reducing the device-local partial aggregates recovers the unpartitioned relation–destination aggregate, up to floating-point reduction order.*

The result follows by rearranging a finite sum over the disjoint fragments; the full proof and the mean-normalization case are in Appendix B. The proposition establishes the semantic boundary for all subsequent optimizations: scheduling may change where or when a message is computed, but not which messages contribute to a typed destination aggregate.

4.2 PHASE I: STRUCTURE-AWARE GRAPH PREPARATION

Exposing every $Q^p_{r,v}$ directly to the scheduler creates millions of tiny objects in our graphs. The preparation method therefore packs complete device-local fragments into physical SUPERBLOCKS. An ordinary SUPERBLOCK S satisfies

$$W(S) = \sum_{Q \in S} |Q| \leq B, \tag{3}$$

where fragments from different relation types may share a SUPERBLOCK while their semantic boundaries and relation identifiers remain intact. A fragment larger than B is retained as an oversized singleton unless an implementation memory bound requires splitting. The resulting units are assigned with the fixed load-aware rule used by the runtime and materialized into device-local edge tensors, destination maps, reduction buffers, and metadata. Because these structures depend on topology rather than trainable parameters, they are reused across epochs.

The edge budget B does not uniquely determine the physical object count. Let $K_g(B)$ be the number of SUPERBLOCKS realized on graph g. It depends on the full distribution of relation–destination fragment sizes, so two graphs with the same number of edges can respond differently to the same budget. Exhaustively materializing every B for each new graph would negate the configuration benefit. We fit the following low-dimensional preparation-time surrogate on preceding snapshots:

$$\widehat{T}_{\text{prep}}(K) = \beta_0 + \beta_1 K + \frac{\beta_2}{K}, \qquad K > 0. \tag{4}$$

The linear term captures per-object packing, indexing, and materialization cost; the inverse term represents the loss of distribution flexibility when a few coarse indivisible units determine the slowest-device runtime. If $\beta_1, \beta_2 > 0$, Equation 4 is strictly convex and has continuous minimizer $K^* = \sqrt{\beta_2/\beta_1}$. For a new graph g', we compute $K_{g'}(B)$ cheaply from fragment sizes and choose $\widehat{B} = \arg\min_{B \in \mathcal{B}} \widehat{T}_{\text{prep}}(K_{g'}(B))$. The model is used to rank feasible configurations, not to reconstruct every short-range runtime fluctuation.

4.3 PHASE II: DESTINATION-RANGE OVERLAP

After preparation, local aggregation and cross-device reduction recur in every layer. For relation r, the recurrent scheduler partitions its ordered destination set into C mutually disjoint and collectively exhaustive ranges, $\mathcal{D}_r = \biguplus_{c=1}^{C} \mathcal{D}_{r,c}$. All device-local partial aggregates for a destination remain in the same range. Once local aggregation for range c completes, its nonblocking all-reduce begins while range $c+1$ is computed; the runtime waits only before the reduced values are consumed.

Let $A_c(C)$ and $R_c(C)$ be local aggregation and reduction time for range c. The useful communication hidden by the next local-computation window is

$$H(C) = \sum_{c=1}^{C-1} \min\{R_c(C), A_{c+1}(C)\}, \quad E(C) = \sum_{c=1}^{C} R_c(C) - H(C), \tag{5}$$

where $E(C)$ is exposed communication and the last range produces an unavoidable terminal tail. Finer ranges can increase $H(C)$, but they also increase collective-launch, range-indexing, buffer-management, and synchronization overhead. We therefore optimize net runtime benefit rather than overlap ratio.

With E_{blk} denoting exposed communication under blocking reduction, define $\Delta E(C) = E_{\text{blk}} - E(C)$ and decomposition overhead $O(C)$. In the calibrated operating region we fit

$$\widehat{\Delta E}(C) = \gamma_0 - \frac{\gamma_1}{C}, \quad \widehat{O}(C) = \eta_0 + \eta_1 C, \quad \widehat{U}(C) = \widehat{\Delta E}(C) - \widehat{O}(C). \tag{6}$$

For $\gamma_1, \eta_1 > 0$, $\widehat{U}$ is strictly concave and its continuous maximizer is $C^* = \sqrt{\gamma_1/\eta_1}$. We select $\widehat{C} = \arg\max_{C \in \mathcal{C}} \widehat{U}(C)$ and enable nonblocking reduction only if $\widehat{U}(\widehat{C}) > 0$; otherwise, the scheduler falls back to blocking reduction. This fallback is part of the policy, not merely an error-handling path.

Correctness and cost. Destination ranges partition output coordinates, not graph edges. Proposition 1 applies independently to every destination, so concatenating all reduced ranges recovers the blocking relation output. The transformation adds $O(C)$ range descriptors and collective invocations but leaves the message and edge work unchanged. Graph preparation reduces the scheduler-visible object count from the number of nonempty fragments $|\{Q^p_{r,v}\}|$ to $K_g(B)$, while retaining $O(|\mathcal{E}|)$ edge storage. Detailed proofs and a complexity discussion appear in Appendices B–C.

4.4 Phase coordination and deployment procedure

The two configuration decisions are coordinated through the partitioned graph structure rather than optimized by a costly Cartesian sweep over (B, C). For each historical snapshot, the preparation profiler records the realized mapping $B \mapsto K_g(B)$ and the synchronized preparation time. These observations fit Equation 4. When a new snapshot arrives, the system forms its relation–destination fragments, computes candidate $K_{g'}(B)$ values without materializing every candidate, and selects $\widehat{B}$. Only the selected organization is packed, assigned, and materialized. This establishes the device-local ordering, destination maps, and partial-aggregation layout used throughout training.

The recurrent controller is conditional on that organization. Short calibration runs measure local aggregation, total reduction, exposed reduction, and decomposition overhead for feasible C values on preceding snapshots. Equation 6 then ranks destination-range counts for the calibrated environment. At deployment, the controller chooses $\widehat{C}$ only when its estimated net benefit is positive; otherwise it uses the exact blocking schedule. The operational sequence is therefore:

1. form relation–destination TargetGroups and their device-local fragments;
2. predict preparation cost through the realized SuperBlock count and select $\widehat{B}$;
3. pack and assign SuperBlocks, then materialize the reusable parallel graph state;
4. select a positive-benefit destination-range count $\widehat{C}$, or retain blocking reduction;
5. for every layer and epoch, compute local partial aggregates and launch range reductions in dependency-safe order.

This staged procedure is computationally different from optimizing the phases independently. The recurrent model is evaluated only after a concrete preparation structure is fixed, and total performance is assessed with Equation 2, which includes the one-time cost. It is also different from jointly enumerating every (B, C) pair: the latter requires repeated materialization and training measurements for each new graph, potentially spending more time on tuning than it saves. The proposed decomposition exploits which state is reusable and which operations recur while preserving the causal dependence from workload organization to communication.

Design implications. Three consequences follow. First, the best edge budget cannot be inferred from $|\mathcal{E}|$ alone because $K_g(B)$ depends on the full fragment-size distribution. Second, the largest C is not generally optimal because overlap opportunities saturate while per-range overhead continues to grow. Third, the number of training epochs matters: even a configuration with lower recurrent runtime is undesirable when its additional preparation cost is not recovered within the planned horizon. These implications motivate the phase-specific experiments in Section 5.

5 Evaluation

We evaluate three questions: **RQ1**: Does phase-aware execution reduce end-to-end training time after one-time costs are included? **RQ2**: Can the preparation surrogate transfer a low-regret granularity to a later graph? **RQ3**: Why is maximizing communication overlap alone insufficient for selecting C?

Table 1: Industrial graph snapshots. "Groups" are nonempty relation–destination TARGETGROUPS.

Snapshot	D1	D2	D3	D4	D5
Edges (10^6)	5.422	5.354	5.062	5.050	5.433
Groups (10^6)	3.610	3.584	3.389	3.541	3.591

Table 2: End-to-end runtime on the held-out Day-5 graph.

Metric	Direct edges	Proposed
Time through epoch 1 (s)	75.97	182.66
Steady epoch, mean (s)	77.04	**28.89**
Steady epoch, std. (s)	0.62	**0.50**
50-epoch runtime (s)	3851.03	**1598.14**
Steady-epoch reduction	–	62.50%
Total-runtime reduction	–	58.50%

5.1 SETUP

Graphs and chronological protocol. The data comprise five consecutive daily snapshots from a rolling-horizon industrial supply-chain planning system. Each snapshot is converted into a directed heterogeneous graph containing typed production, inventory, transportation, substitution, capacity, and temporal dependencies. The partitioned edge sets contain 5.05–5.43 million edges and 3.39–3.61 million nonempty relation–destination groups (Table 1). Days 1–4 fit both runtime models; Day 5 is excluded from fitting and used for configuration selection and end-to-end evaluation. This chronological split tests transfer from previous executions to the next arriving graph instead of tuning on the test graph.

System and model. All experiments are conducted on a single server equipped with four 64-GB PCIe AI accelerators, one 32-core 7185 processor, 512 GB of host memory, and a 100-Gb/s compute network. The software stack consists of DTK 26.04, [framework and version], and [collective communication library and version]. Across all methods, node states are replicated, while the edge partitions, model parameters, random seed, and timing boundaries are kept identical to ensure a controlled comparison. The direct-edge implementation serves as the end-to-end baseline. The blocking variant uses SUPERBLOCK organization but performs full-range blocking reduction, whereas the nonblocking variants use the same prepared graph with $C \in \{2, \ldots, 8\}$. Appendix D provides the complete profiling protocol and metric definitions.

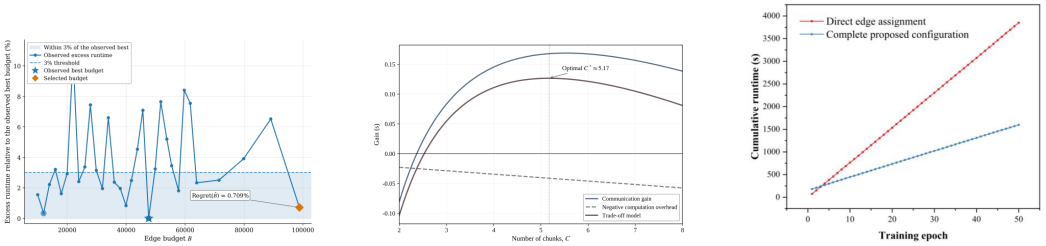
For Phase I, we profile 32 budgets from 10,000 to 64,000 edges in increments of 2,000 plus {70,000, 80,000, 90,000, 100,000}; each graph–budget pair is repeated twice. The 128 aggregated configurations from Days 1–4 fit Equation 4; all 32 Day-5 configurations remain unseen. For Phase II, blocking reduction and $C \in \{2, \ldots, 8\}$ are measured on Days 1–4 after fixing graph organization. Preparation and first-epoch warm-up are excluded when fitting recurrent models.

5.2 RQ1: END-TO-END RUNTIME

Table 2 compares the original direct-edge implementation with the complete framework on held-out Day 5. Mean runtime over epochs 2–50 falls from 77.04 to 28.89 seconds, a 62.50% reduction. The proposed run has a larger time through epoch 1 because it materializes reusable execution structures, but its 50-epoch wall-clock time falls by 58.50%. The additional 106.69 seconds are recovered after approximately four epochs (Figure 2c). Epoch-time standard deviation also decreases from 0.62 to 0.50 seconds, so the mean improvement is not obtained at the cost of greater epoch-to-epoch variability within this execution.

5.3 RQ2: TRANSFERRING PREPARATION GRANULARITY

On Day 5, the preparation surrogate obtains MAE 2.329 s, RMSE 3.352 s, NRMSE 2.823%, and $R^2 = -0.257$. The negative R^2 means that the compact model does *not* faithfully predict candidate-



(a) Preparation regret. (b) Net-benefit decomposition. (c) Cumulative runtime.

Figure 2: Configuration and end-to-end results on the held-out operating regime. The orange marker in (a) is the surrogate-selected workload granularity; (b) shows why maximizing overlap differs from maximizing net benefit; (c) shows that preparation is amortized after approximately four epochs.

level runtime variation and should not be presented as a general runtime predictor. Its configuration decision is nevertheless useful: the selected budget incurs 0.709% measured regret relative to the fastest of 32 candidates and lies in the broad low-runtime region in Figure 2a. The evidence supports a narrower claim: within five consecutive, structurally similar snapshots, $K_g(B)$ is sufficient to transfer a near-optimal operating region. It does not establish transfer across unrelated graph families or hardware.

5.4 RQ3: Overlap is not the objective

The measured overlap ratio grows from 14.57% at $C = 2$ to 33.01% at $C = 8$, but exposed communication is nonmonotonic: it is 0.5013 s for blocking reduction, 0.3350 s at $C = 3$, 0.5075 s at $C = 4$, and 0.3019 s at $C = 8$. Equation 6 selects $C = 5$, whereas overlap ratio alone would select $C = 8$. Figure 2b explains the difference: the marginal reduction in exposed communication diminishes while decomposition overhead grows. The result validates the controller's central design choice—select the schedule by net critical-path benefit and retain blocking execution when no candidate is beneficial.

5.5 Discussion and limitations

The evaluation isolates a systems effect under a fixed model, partition, device count, and graph family: structure-aware reusable organization plus adaptive destination-range reduction substantially lowers runtime without changing the mathematical message-passing operator. It also exposes two failure modes that a configuration method must handle. First, a low-dimensional surrogate may identify a useful operating region despite poor pointwise R^2; selection regret and prediction error therefore answer different questions. Second, the schedule with maximum measured overlap need not minimize recurrent runtime after launches and synchronization are included. Both observations support configuration objectives aligned with final wall-clock time.

The end-to-end comparison also demonstrates why the two phases should not be reported separately. If evaluation started only after materialization, the proposed method would appear to save 62.50% per epoch immediately. Including time through the first epoch reveals an initial disadvantage of 106.69 seconds and a break-even point near epoch four. Equation 2 makes this trade-off visible and allows deployment to reject a preparation configuration for very short runs even when it is favorable in steady state. Conversely, reporting preparation time alone would miss the much larger recurrent benefit obtained from reuse.

The result does not imply that the preparation surrogate is accurate in an ordinary regression sense. Its negative held-out R^2 shows that short-range candidate ordering contains variation not explained by $K_g(B)$, potentially from allocation, device scheduling, relation ordering, and transient system state. The broad low-runtime region in Figure 2a makes the weaker configuration task easier: several neighboring budgets incur similar costs. The appropriate conclusion is therefore that the surrogate transfers a low-regret region across the evaluated consecutive snapshots, not that its functional form universally predicts graph-preparation time.

The evidence is confined to one server, a fixed device count, and five similarly structured proprietary snapshots. It supports temporal transfer within this regime, not transfer to unrelated graphs or inter-node clusters. Node replication also requires node states to fit on every device. Public benchmarks, device scaling, matched-semantics baselines, component ablations, profiler breakdowns, and numerical-equivalence tests remain necessary for broader claims.

6 CONCLUSION

This paper formulates mutli-GPU parallel full-graph heterogeneous GNN training as a joint communication-computation optimization problem across two execution horizons. Relation–destination groups preserve semantic aggregation, SUPERBLOCKs organize reusable physical computation, and destination ranges schedule recurrent communication. Their configurations balance workload flexibility against preparation cost and hidden communication against decomposition overhead. Disjoint-cover reductions preserve the original full-graph relation outputs.

On five consecutive industrial planning graphs, the preparation surrogate selects a held-out configuration with 0.709% regret, while the communication controller selects $C = 5$ rather than the maximum-overlap setting $C = 8$. Together, the two stages reduce mean recurrent-epoch time by 62.50% and 50-epoch wall-clock time by 58.50%, amortizing preparation after approximately four epochs.

REFERENCES

H.-I. Ahn, Y. C. Song, S. Olivar, H. Mehta, and N. Tewari. GNN-based probabilistic supply and inventory predictions in supply chain networks. *arXiv preprint arXiv:2404.07523*, 2024.

Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. LogGP: Incorporating long messages into the LogP model. *Journal of Parallel and Distributed Computing*, 44(1):71–79, 1997.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Kaidi Cao, Rui Deng, Shiwen Wu, Edward W. Huang, Karthik Subbian, and Jure Leskovec. Communication-free distributed GNN training with vertex cut. *arXiv preprint arXiv:2308.03209*, 2023.

N. Chairatanakul, X. Liu, and T. Murata. PGRA: Projected graph relation-feature attention network for heterogeneous information network embedding. *Information Sciences*, 570:769–794, 2021.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 1–12, 1993.

Bin Du, Jing Liu, Zihan Luo, Chuan Wu, Qizhen Zhang, and Hai Jin. Expediting distributed GNN training with feature-only partition and optimized communication planning. In *IEEE INFOCOM 2024*, pp. 1321–1330, 2024.

Swapnil Gandhi and Anand Padmanabha Iyer. P3: Distributed deep graph learning at scale. In *15th USENIX Symposium on Operating Systems Design and Implementation*, pp. 551–568, 2021.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. PowerGraph: Distributed graph-parallel computation on natural graphs. In *10th USENIX Symposium on Operating Systems Design and Implementation*, pp. 17–30, 2012.

Roger W. Hockney. The communication challenge for MPP: Intel paragon and meiko CS-2. *Parallel Computing*, 20(3):389–398, 1994.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

N. Kotecha and A. del Rio Chanona. Leveraging graph neural networks and multi-agent reinforcement learning for inventory control in supply chains. *Computers & Chemical Engineering*, 199:109111, 2025.

H. Li, W. Zheng, F. Tang, Y. Song, B. Yao, and Y. Zhu. Dynamic heterogeneous attributed network embedding. *Information Sciences*, 662:120264, 2024.

Haiyang Lin, Ming Yan, Xiaocheng Ye, Dongrui Fan, Shirui Pan, Wenzhi Chen, and Yuan Xie. A comprehensive survey on distributed training of graph neural networks. *Proceedings of the IEEE*, 111(12):1572–1606, 2023.

Zihan Luo, Yixin Bao, and Chuan Wu. Optimizing task placement and online scheduling for distributed GNN training acceleration in heterogeneous systems. *IEEE/ACM Transactions on Networking*, 32(5):3715–3729, 2024.

Nikolai Merkel, Daniel Stoll, Ruben Mayer, and Hans-Arno Jacobsen. An experimental comparison of partitioning strategies for distributed graph neural network training. In *Proceedings of the 28th International Conference on Extending Database Technology*, pp. 171–184, 2025.

H. Nguyen, R. Hamidi Rad, F. Zarrinkalam, and E. Bagheri. DyHNet: Learning dynamic heterogeneous network representations. *Information Sciences*, 646:119371, 2023.

Avishek Sarkar, Sayan Ghosh, Nathan R. Tallent, and Ali Jannesari. MassiveGNN: Efficient training via prefetching for massively connected distributed graphs. In *2024 IEEE International Conference on Cluster Computing*, pp. 62–73, 2024.

Yingxia Shao, Hongzheng Li, Xizhi Gu, Hongbo Yin, Yawen Li, Xupeng Miao, Wentao Zhang, Bin Cui, and Lei Chen. Distributed graph neural network training: A survey. *ACM Computing Surveys*, 56(8):1–39, 2024.

Cheng Wan, Youjie Li, Cameron R. Wolfe, Anastasios Kyrillidis, Nam Sung Kim, and Yingyan Lin. PipeGCN: Efficient full-graph training of graph convolutional networks with pipelined feature communication. In *International Conference on Learning Representations*, 2022.

L. Xiang, Q. Hu, X. Zhang, W. Lan, and B. Liu. Graph neural Poisson models for supply chain relationship forecasting. *arXiv preprint arXiv:2508.12044*, 2025.

Q. Zhang, Z. Zhao, H. Zhou, X. Li, and C. Li. Self-supervised contrastive learning on heterogeneous graphs with mutual constraints of structure and feature. *Information Sciences*, 640:119026, 2023.

Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. DistDGL: Distributed graph neural network training for billion-scale graphs. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms*, pp. 36–44, 2020.

A Industrial learning-assisted optimization application

A.1 Rolling-horizon planning snapshots

Each planning cycle receives updated orders, forecasts, inventory states, material availability, production resources, capacity, substitution options, and transportation conditions. A mathematical program

jointly determines production, inventory, substitution, capacity use, and inter-plant transportation over a finite horizon. Material balance, inventory transitions, resource use, substitutions, and lead times tightly couple candidate decisions across products, plants, and periods. When new information arrives, the horizon advances and a new planning snapshot is solved. Consecutive snapshots differ in parameters while retaining substantial entity and constraint structure (Figure 3).

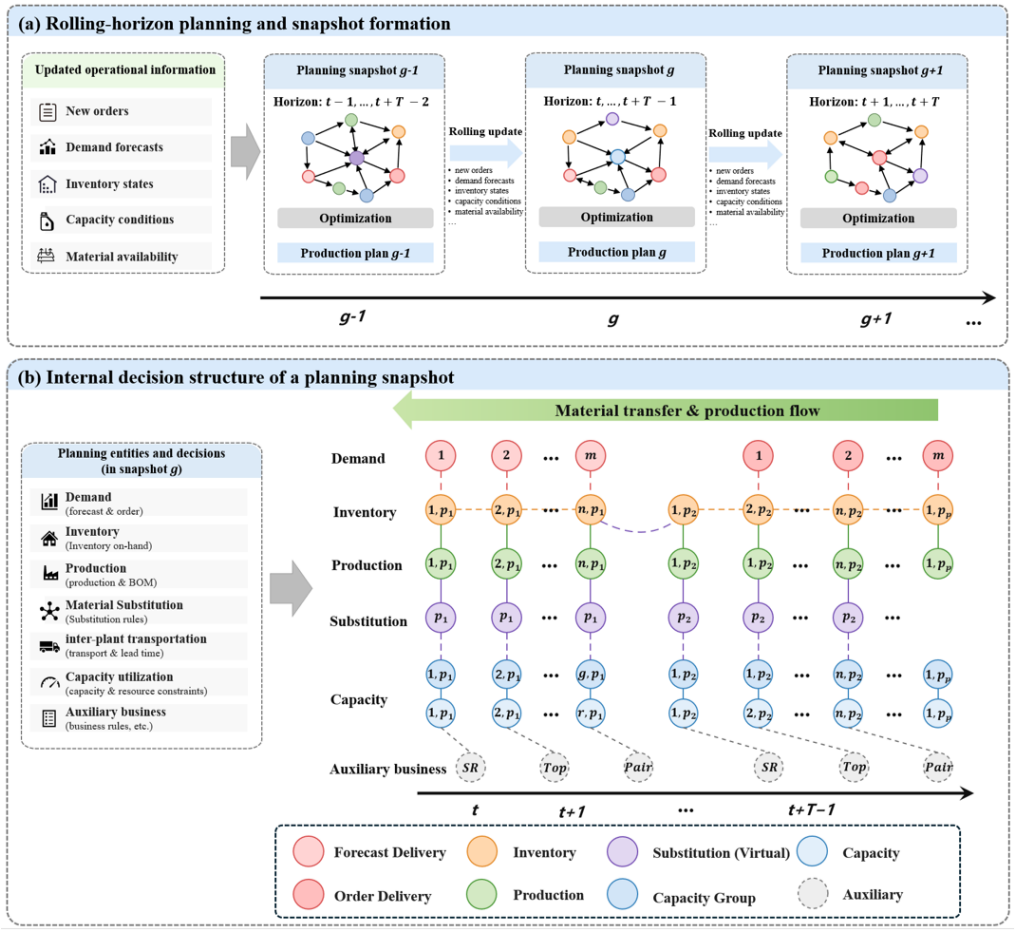


Figure 3: Rolling-horizon planning and the formation of successive graph snapshots.

A.2 Role of the heterogeneous learner

For a historical snapshot, a candidate variable is labeled active if the optimized solution selects it or, for a continuous decision, assigns it a nonzero value. A heterogeneous GNN predicts activity scores for candidate variables in the next snapshot. Low-score candidates can be screened before the reduced mathematical program is solved; predictions can also initialize solver heuristics. The learner therefore assists search, while the mathematical model retains responsibility for feasibility and final decisions (Figure 4).

The decision graph includes demand, inventory, production, substitution, capacity, and source–destination context nodes. Typed edges encode material consumption, production output, inventory carryover, substitution, capacity association, and transportation lead-time dependencies, including reverse relations when the reverse message direction has distinct semantics. Context nodes without activity labels remain in the graph because they connect supervised candidates through operational constraints. Node and edge attributes include identifiers and values such as item, plant, period, resource, demand, inventory, bill-of-material coefficients, production rates, substitution ratios, capacity, and lead times.

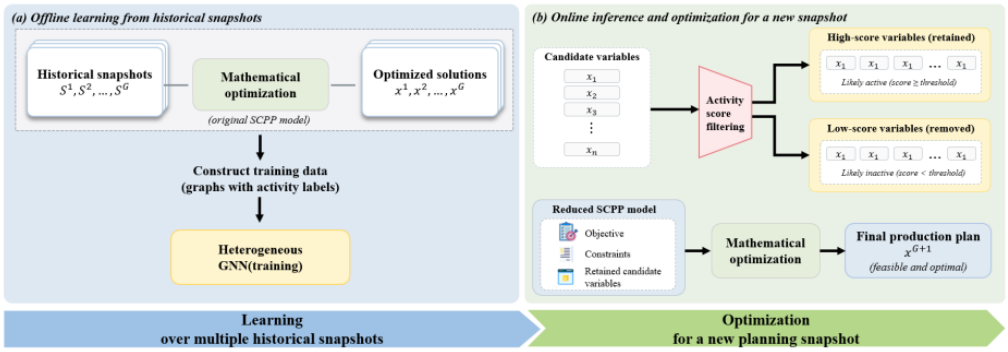


Figure 4: The learner operates before the final mathematical optimization solve.

B Correctness proofs

B.1 Relation-specific parallel aggregation

Proof of Proposition 1. Fix relation r and destination v. Because $\{\mathcal{E}_{r,p}\}_{p=1}^{P}$ is a disjoint cover of $\mathcal{E}_r$, the fragments $Q_{r,v}^{p} = Q_{r,v} \cap \mathcal{E}_{r,p}$ are a disjoint cover of $Q_{r,v}$. Hence

$$\sum_{p=1}^{P} s_{r,v}^{p} = \sum_{p=1}^{P} \sum_{(u,v)\in Q_{r,v}^{p}} m_{u\to v,r} = \sum_{(u,v)\in Q_{r,v}} m_{u\to v,r} = s_{r,v}. \tag{7}$$

For mean aggregation, dividing the recovered sum by the global relation-specific destination degree $|Q_{r,v}|$ gives the same normalized output as unpartitioned execution. Floating-point collectives may change summation order and therefore the final low-order bits, but not the aggregation semantics or contributing edge set. □

B.2 Destination-range assembly

Proposition 2 (Exact destination-range assembly)**.** *If $\{\mathcal{D}_{r,c}\}_{c=1}^{C}$ is a disjoint cover of $\mathcal{D}_r$ and each range performs the reduction in Proposition 1, concatenating the reduced range outputs gives the same relation output as blocking full-range reduction, up to floating-point reduction order.*

Proof. Every destination $v \in \mathcal{D}_r$ appears in exactly one range. Proposition 1 recovers its full relation-specific aggregate within that range. Because the ranges are disjoint and exhaustive, their concatenation contains exactly one recovered aggregate for every destination and no additional output coordinate. □

B.3 Surrogate extrema

For Equation 4, $\widehat{T}'_{\text{prep}}(K) = \beta_1 - \beta_2/K^2$ and $\widehat{T}''_{\text{prep}}(K) = 2\beta_2/K^3 > 0$ when $\beta_2 > 0$. Thus the surrogate is strictly convex and its stationary point is the unique minimizer $K^* = \sqrt{\beta_2/\beta_1}$ for $\beta_1 > 0$.

For Equation 6, $\widehat{U}'(C) = \gamma_1/C^2 - \eta_1$ and $\widehat{U}''(C) = -2\gamma_1/C^3 < 0$ when $\gamma_1 > 0$. The unique continuous maximizer is $C^* = \sqrt{\gamma_1/\eta_1}$ for $\eta_1 > 0$. Deployment evaluates only feasible integer candidates and applies the positive-benefit fallback.

C Complexity and applicability

Let Q_g be the number of nonempty device-local TargetGroup fragments and $K_g(B)$ the realized number of SuperBlocks. Preparation associates every local edge with a fragment and packs fragments into $K_g(B)$ objects. It does not reduce the $O(|\mathcal{E}|)$ message-passing work or edge storage; it

reduces the number of scheduler-visible objects from Q_g to $K_g(B)$ in the common regime $K_g(B) \ll Q_g$. Materialized state includes local edge indices, source/destination maps, global destination degrees, relation labels, reduction buffers, and SUPERBLOCK/range metadata.

During recurrent training, destination-range decomposition leaves the edge set, message functions, and output size unchanged. Its incremental cost grows with C through descriptors, collective launches, event synchronization, and output assembly. Even if the total payload is nearly constant, a larger number of smaller collectives increases the latency-sensitive component predicted by classical latency–bandwidth models (Hockney, 1994; Culler et al., 1993; Alexandrov et al., 1997). This is the reason for optimizing $\widehat{U}(C)$ rather than overlap ratio.

D EXPERIMENTAL PROTOCOL AND EXTENDED RESULTS

D.1 MEASUREMENT SCOPES

Preparation timing includes TARGETGROUP formation, SUPERBLOCK packing, workload assignment, device-local edge collection, index construction, buffer allocation, and metadata materialization. It excludes forward/backward propagation, optimizer updates, and first-epoch framework warm-up. Each synchronized observation is the maximum wall-clock time over devices and is averaged over repeated trials.

Recurrent timing begins after the graph organization is fixed. Blocking reduction is the reference; nonblocking destination-range schedules use the same graph, parameters, seed, and number of epochs. Hidden communication is the portion of collective execution concurrent with useful local aggregation. Exposed communication is the remainder on the synchronized path, and overlap ratio is $T_{\text{hidden}}/(T_{\text{hidden}} + T_{\text{exposed}})$.

D.2 PREPARATION-MODEL METRICS

Table 3: Held-out Day-5 preparation-model results.

Category	Metric	Result
Prediction	MAE	2.329 s
Prediction	RMSE	3.352 s
Prediction	NRMSE	2.823%
Diagnosis	R^2	-0.257
Selection	measured regret	0.709%

For empirical preparation time T_{prep} and selected and optimal candidate budgets $\widehat{B}$ and B^*, selection regret is

$$\text{Regret}_{\text{prep}} = \frac{T_{\text{prep}}(\widehat{B}) - T_{\text{prep}}(B^*)}{T_{\text{prep}}(B^*)}. \tag{8}$$

D.3 COMMUNICATION MEASUREMENTS

Table 4: Measured communication behavior for blocking and destination-range reduction.

Mode	Hidden (s)	Exposed (s)	Overlap
Blocking	0.0000	0.5013	0.00%
$C = 2$	0.0306	0.4288	14.57%
$C = 3$	0.0470	0.3350	21.86%
$C = 4$	0.0554	0.5075	22.59%
$C = 5$	0.0587	0.3714	25.61%
$C = 6$	0.0673	0.3072	30.07%
$C = 7$	0.0733	0.3667	31.36%
$C = 8$	0.0755	0.3019	33.01%

E SCOPE, LIMITATIONS, AND REQUIRED ADDITIONS

The current study uses node replication and is therefore bounded by the memory needed to store node representations on every device. It is most appropriate when edge structure and edge computation dominate, as in the evaluated graphs. It does not address clusters with partitioned node features, inter-node networking, elastic device counts, graph sampling, or asynchronous model updates.

The five snapshots form a consecutive sequence with comparable graph scale and structure. The held-out result supports short-horizon temporal configuration transfer inside this regime, not cross-domain generalization. The communication model is calibrated to a fixed workload and hardware environment; $C = 5$ is an operating point, not a universal constant. The source manuscript also lacks the following evidence needed for a strong ICLR systems submission:

1. full hardware/software/model specifications and repeated-run confidence intervals;
2. public graph benchmarks and at least one additional heterogeneous graph family;
3. competitive mutli-GPU full-graph baselines under matched semantics;
4. scaling across device counts and, if claimed, graph sizes;
5. component ablations and profiler-based compute/communication/utilization breakdowns;
6. numerical-equivalence and downstream activity-prediction/screening-quality results;
7. an anonymous reproducibility package or a precise artifact-availability statement.