Simplifying Distributed Neural Network Training on Massive Graphs: Randomized Partitions Improve Model Aggregation

Jiong Zhu [1 2 *] **Aishwarya Reganti** [2] **Edward Huang** [2] **Charles Dickens** [3 *] **Nikhil Rao** [4] **Karthik Subbian** [2]
Danai Koutra [1 2]

Abstract

Conventional distributed Graph Neural Network (GNN) training relies either on inter-instance communication or periodic fallback to centralized training, both of which create overhead and constrain their scalability. In this work, we propose a streamlined framework for distributed GNN training that eliminates these costly operations, yielding improved scalability, convergence speed, and performance over state-of-the-art approaches. Our framework (1) comprises independent trainers that *asynchronously* learn local models from locally-available parts of the training graph, and (2) synchronize these local models only through periodic (time-based) model aggregation. Contrary to prevailing belief, our theoretical analysis shows that it is not essential to maximize the recovery of cross-instance node dependencies to achieve performance parity with centralized training. Instead, our framework leverages randomized assignment of nodes or super-nodes (i.e., collections of original nodes) to partition the training graph to enhance data uniformity and minimize discrepancies in gradient and loss function across instances. Experiments on social and e-commerce networks with up to 1.3 billion edges show that our proposed framework achieves state-of-the-art performance and 2.31x speedup compared to the fastest baseline, despite using less training data.

1. Introduction

Graph neural networks (GNNs) achieve state-of-the-art performance on a variety of graph-based machine learning tasks with applications to recommendation systems (van den Berg et al., 2017; Ying et al., 2018; Fan et al., 2019), fraud detection (Wang et al., 2019; 2018; Dou et al., 2020), social network analysis (Qiu et al., 2018; Breuer et al., 2020; Cao et al., 2020), and more. As applications scale to massive social and other networks with billions of edges (Zhu et al., 2019), they pose scalability challenges to typical multi-layer GNN models (e.g. GCN (Kipf & Welling, 2017)), which require a Message Flow Graph (MFG) based on each node's multi-hop neighborhood. These MFGs quickly exceed the storage and computational capacity of modern systems even under moderate batch sizes and number of GNN layers. This issue has motivated a productive line of work on scalable centralized GNN training on a single instance (Hamilton et al., 2017; Chen et al., 2018; Zeng et al., 2019; Chiang et al., 2019; Zeng et al., 2021; Fey et al., 2021; Narayanan et al., 2021), but the size of the graphs that can be trained on a single machine is ultimately limited by its available computational resources.

Distributed training overcomes the resource limitation of a single machine by leveraging parallelism on multiple machines. By partitioning training samples across multiple trainers and coordinating distributed updates to model weights on each trainer (Narayanan et al., 2019), data parallelism approaches have facilitated the training of computer vision (Krizhevsky et al., 2017; Goyal et al., 2017; Yu et al., 2019) and language models (McMahan et al., 2017) on massive-scale datasets. However, *graph* datasets pose additional unique challenges for data parallelism due to *cross-instance node dependencies* (i.e., graph connections that reach across instance boundaries) when the data is partitioned and distributed to multiple trainer instances. Different strategies have been proposed to address these challenges.

One popular strategy, adopted by DistDGL and other frameworks (Jiang & Rumi, 2021; Zheng et al., 2020; 2021), is to respect the cross-instance dependencies and implement communication mechanisms that allow embeddings to traverse through instance boundaries. To reduce the communication overhead, these approaches often distribute the training data by leveraging min-cut based graph partitioning algorithms (e.g., METIS (Karypis & Kumar, 1998)) and data replica-

[*]Work conducted during the authors' internship at Amazon. [1]University of Michigan, Ann Arbor, MI, USA. [2]Amazon, Palo Alto, CA, USA. [3]University of California, Santa Cruz, CA, USA. [4]Microsoft, Redmond, WA, USA. Correspondence to: Jiong Zhu <jiongzhu@{umich.edu,amazon.com}>, Danai Koutra <dkoutra@{umich.edu,amazon.com}>.

In *ICML Workshop on Localized Learning (LLW)*, Honolulu, Hawaii, USA. 2023.

tion. This strategy provides equivalency of a distributed training setup to a centralized one, but its reliance on excessive communication to enable unrestricted graph access across instances creates a bottleneck for further improving the speed, scalability and robustness to failures.

Another strategy is to initially ignore the cross-instance dependencies by restricting the graph access per trainer to only local graph data assigned to it, and later compensating for the lost data with methods like periodic centralized training (Ramezani et al., 2021). This strategy is usually coupled with a *model aggregation* mechanism, which periodically replaces the local model weights per trainer with aggregated weights (e.g., average) from all trainers (Stich, 2018; Ramezani et al., 2021). While it overcomes the overhead of excessively communicating node representations across machines, different implementations handle the incurred data loss and its presumed negative impact on performance by intermittently resorting to centralized training (Ramezani et al., 2021) or replicating nodes across trainers (Angerd et al., 2020). These solutions impose new bottlenecks and additional overhead on distributed training frameworks.

This work. In this study, we reexamine the prevalent assumptions of distributed GNN training. We propose a streamlined framework that removes these bottlenecks by fully discarding cross-instance dependencies, focusing instead on the graph partition schemes and model aggregation mechanism. We summarize our contributions as follows:

- **Simplified time-based model aggregation training framework:** We present a simplified model aggregation training framework (§3.1) that (1) assembles independent trainers, each of which asynchronously learns a local model on locally-available parts of the training graph only, and (2) synchronizes the local models by only conducting periodic, time-based model aggregation that accommodates imbalanced loads and speeds among trainers.
- **Randomized graph partition schemes with theoretical justifications:** Contrary to the prevailing belief that minimizing cross-instance edges is vital to bridge the performance gap between model aggregation and centralized training, we provide both theoretical (§3.2.1) and empirical (§4.2) evidence that min-cut partitioning algorithms adversely impact training performance by inducing discrepancies in data distributions among different partitions. Consequently, we propose improved partition strategies (§3.2.2) that employ a randomized assignment of nodes, thus evading the overhead of graph clustering, or supernodes (i.e., mini-clusters of original nodes) to trainers.
- **Extensive empirical analysis**[1]: Our extensive experiments, involving over 4,600 GPU hours across 3 machines, confirm the scalability of our framework for link prediction on large-scale social and e-commerce networks with up to 1.3 billion edges (§4.1). Despite utilizing less training data than the baselines, our proposed methods—RandomTMA and SuperTMA—outperform state-of-the-art approaches with a 2.31x speedup in convergence time over the fastest baseline (§4.2), and offer enhanced robustness to trainer failures (§D.1).

[1]Code is available on GitHub: amazon-science/random-tma.

2. Related Work

We focus on works for distributed settings here, and defer works for scalable GNN training on a single trainer to §A.

Distributed GNN Training. The majority of distributed GNN training research employs a data parallelism paradigm. Approaches under this paradigm can be scrutinized from three perspectives: the graph access scope per trainer, the data partition schemes and assignments to trainers, and the model synchronization mechanism across trainers. DistDGL (Zheng et al., 2020; 2021) and DistGNN (Md et al., 2021) enable unrestricted access to the full training graph for each trainer, adopts min-cut based graph partitioning algorithms (e.g., METIS (Karypis & Kumar, 1998)) to partition training graph, and utilizes fully synchronous Stochastic Gradient Descent (SGD) to update local model weights after each training step; they also incorporate extensive optimization on training pipeline. To further reduce the communication cost under a similar setup, Tripathy et al. (2020) optimizes matrix multiplication operations of GNNs, and Jiang & Rumi (2021) adopts skewed sampling of MFGs to bias towards local neighbors of each node. On the other hand, Parallel SGD with Periodic Averaging (PSGD-PA) (Ramezani et al., 2021) restricts the graph access to local data only per trainer, adopts METIS to minimize cross-partition edges ignored in training, and conducts periodic averaging to synchronize local model weights on the trainers. To recover more ignored cross-partition edges under this setup, LLCG (Ramezani et al., 2021) further employs fallbacks to centralized training during the averaging process, while Angerd et al. (2020), per partition, replicates nodes from other partitions through breadth first search.

Our proposed data parallel approach, similar to PSGD-PA, restricts the graph access to local data per trainer. However, it adopts randomized partitioning to reduce data discrepancy across trainers and enhance performance. It also employs time-based (instead of step-based) intervals for model aggregation, accommodating heterogeneity in instance load and training speed. These designs largely eliminate communication wait time between trainers, which, according to a previous study (Gandhi & Iyer, 2021), accounts for about 80% of the training time for DistDGL. As a result, our data parallel approach is as efficient as the more complex hybrid (data and model) parallel approaches like P^3 (Gandhi & Iyer, 2021). We compare our approaches to existing frameworks in detail in §B.

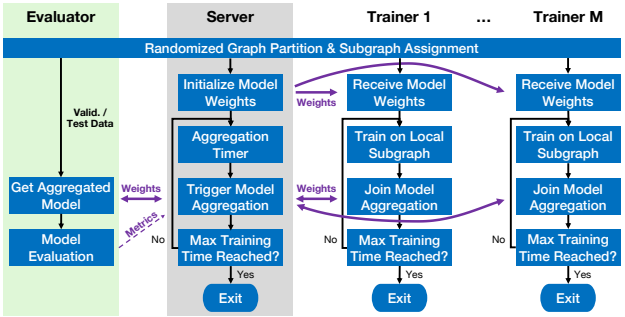


Figure 1: Architecture of our Time-based Model Aggregation (TMA) framework. We provide the pseudo-code for the server and trainer in Alg. 1 and 2, respectively. We use solid purple arrows to represent synchronous communications, and dashed ones for asynchronous communications.

Model Averaging & Federated Learning. Model Averaging, a technique that averages parameters of models with identical architecture and initialization, yet trained on different data subsets, has been proven to enhance performance over the individual models being combined (Matena & Raffel, 2022). This approach is central to federated learning (McMahan et al., 2017) and the recently proposed model soup paradigm (Wortsman et al., 2022). While several works have attempted to provide theoretical underpinnings behind the success of model averaging from the perspective of convex optimization and estimation theory (Polyak & Juditsky, 1992; Li et al., 2014; Yu et al., 2019; Matena & Raffel, 2022; Wortsman et al., 2022), this phenomenon largely remains an empirical observation. Existing research on model averaging is primarily focused on vision and language models, with only few studies examining its application to GNNs (Angerd et al., 2020; Ramezani et al., 2021). In this work, we delve deeper into the potential of model averaging for GNN training on datasets comprising billions of edges, enhancing performance and efficiency via asynchronous model aggregation and randomized partitions. While our proposed methods can be applied within federated learning, as they accommodate independent trainers with diverse computational capacities, our primary focus in this work is to address GNN scalability challenges. We leave a comprehensive evaluation of our approach for federated learning for future research.

3. Time-based Model Aggregation & Randomized Partition Schemes

In this section, we first give key notation, and then present our proposed time-based model aggregation training framework (§3.1) and two randomized partition schemes (§3.2). We defer our theoretical analysis on why partition schemes that minimize cross-partition edges negatively impact the performance of model aggregation training to App. §3.2.1. We discuss related works in details in App. §2 and provide an in-depth comparison of our proposed framework to prior distributed GNN training frameworks in §B.

Preliminaries. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a simple graph with node set $\mathcal{V}$, edge set $\mathcal{E}$, adjacency matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$, and node feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}|\times\mathcal{F}}$. Let $\mathcal{G}' = (\mathcal{V}', \mathcal{E}') \subset \mathcal{G}$ be a subgraph with $\mathcal{V}' \subset \mathcal{V}$ and $\mathcal{E}' \subset \mathcal{E}$. Given a node partition $\alpha : \mathcal{V} \to \mathcal{I}$ on graph $\mathcal{G}$ and its inverse $\alpha^{-1}(i) = \{v \mid v \in \mathcal{V} \wedge \alpha(v) = i\}$, we define the node-induced subgraph $\mathcal{G}^{(i)} = (\mathcal{V}^{(i)}, \mathcal{E}^{(i)})$ of partition $i \in \mathcal{I}$ as $\mathcal{V}^{(i)} = \alpha^{-1}(i)$ and $\mathcal{E}^{(i)} = \{(u, v) \mid (u, v) \in \mathcal{E} \wedge u, v \in \alpha^{-1}(i)\}$. For a graph with node class labels $y_v \in \mathcal{Y}$, we define its homophily ratio as the fraction of homophilic edges linking same-class nodes (Zhu et al., 2020): $h = |\{(u, v) \mid (u, v) \in \mathcal{E} \wedge y_u = y_v\}|/|\mathcal{E}|$. We refer to a graph with $h \geq 0.5$ as homophilic.

3.1. TMA: Proposed Time-based Model Aggregation Framework

We propose a streamlined distributed framework for GNN training that leverages the idea of Time-based Model Aggregation (TMA). Figure 1 illustrates the architecture of our TMA framework, which consists of M trainer processes, a server process, and one or more evaluation processes. These processes may run on a cluster of machines or a single machine based on the scale of the dataset and availability of resources. The design of the server and trainer processes are formally presented in Algorithms 1 and 2, respectively.

Trainer. Each *trainer* process $i \in \{1, \cdots, M\}$ loads a part of the training graph $\mathcal{G}^{(i)}_{\text{train}} \subset \mathcal{G}_{\text{train}}$, and conducts stochastic gradient descent on mini-batches sampled *solely* from the local training subgraph $\mathcal{G}^{(i)}_{\text{train}}$ assigned to it via partition function α. We discuss partition options and propose improved approaches for partitioning and assigning the local training subgraphs in §3.2.

Server & Evaluator. On the *server* side, our TMA framework periodically executes a model aggregation operation ϕ to synchronize the learned model parameters $\mathbf{W}_i$ across trainers. This procedure is triggered on a time-based interval, supporting asynchronous training across heterogeneous trainers; this is critical for a scalable and efficient framework as we empirically observe (in §4.3) that the number of training steps finished on the slowest trainer can be up to 28.8% less than that on the fastest trainer. For the choice of aggregation operator, we find that simply averaging the model parameters of the trainers provides better performance over more complex model aggregation operators that consider the losses on different trainers. We use a separate *evaluator* process to evaluate the aggregated model.

3.2. Improving Time-based Model Aggregation with Randomized Partition Schemes

Partitioning and assignment of graphs are standard preprocessing steps in distributed GNN training: the full train-

ing graph $\mathcal{G}_{\text{train}}$ is first partitioned into smaller subgraphs $\mathcal{G}^{(i)}_{\text{train}} \subset \mathcal{G}_{\text{train}}$, which are then assigned to distinct trainers $i \in \{1, \cdots, M\}$. Existing frameworks such as DistDGL, PSGD-PA, and LLCG strive to maximize the coverage of cross-instance node dependencies in their partition schemes. However, our theoretical analysis uncovers that partitions minimizing cross-machine edges contribute to increased disparities in training data across localized trainers. This discrepancy further leads to disparities in gradients and training losses that hinder the convergence of model aggregation training (§3.2.1).[2] To alleviate the disparity between partitions, we propose randomized partition schemes at the node or super-node level (§3.2.2), which achieve improved performance and convergence speed despite using less training data due to discarding the cross-instance edges (§4.2).

3.2.1. Minimizing cross-partition edges harms model aggregation training

The residual error of the loss function and its gradients caused by the local-access constraint is considered the key behind the performance gap between model aggregation training and centralized training (Ramezani et al., 2021); in other words, the mismatch of the loss values and their gradients on different distributed trainers and to those of a centralized trainer hurts the performance of model aggregation training. Here we provide a theoretical analysis about how the popular approach in existing distributed frameworks (Zheng et al., 2020; 2021; Md et al., 2021; Ramezani et al., 2021; Angerd et al., 2020) of one-to-one mapping of METIS partitions to trainers, which minimizes the cross-partition edges, contributes to the residual error in the gradient descent process of model aggregation training on homophilic graphs.

We analyze a case of a homophilic graph, where the disparity of partitions is measured by the difference of the feature distributions, which correlate with two class labels. In Lem. 1, we show that partitions minimizing the number of cross-partition edges amplify the differences of feature distributions among partitions, which in the case we assume leads to complete separation of nodes from different classes.

Lemma 1. *Assume a homophilic graph with two equally-sized classes and edges modeled by a class compatibility matrix* $\mathbf{H}$ *(Zhu et al., 2020) as follows: the probability* p_{ji} *of node* j *linking to node* i *satisfies*

$$p_{ji} \propto \mathbf{H}(y_i, y_j) = \begin{cases} h \geq 0.5, & \text{for } y_i = y_j \\ 1 - h & \text{otherwise,} \end{cases}$$

where $y_i, y_j \in \{0, 1\}$ *are the class labels of nodes* i, j*. Let* $\mathbf{x}_v = \text{onehot}(y_v)$ *be features of node* $v \in \mathcal{V}$*, and* $\alpha : \mathcal{V} \to \{1, 2\}$ *be a function that assigns the nodes into two equally-sized partitions. Then, the smallest expected edge-cut is reached when each partition has same-class nodes with the same features:* $\alpha(i) = \alpha(j)$ *iff* $y_i = y_j$ *or* $\mathbf{x}_i = \mathbf{x}_j$*.*

In Thm. 2, we demonstrate the effects of disparity between partitions by showing that it leads to discrepancy between (initial) gradients and loss derived on different trainers, which is the key factor affecting the performance under model aggregation training when only local data is used (Ramezani et al., 2021).

Theorem 2. *Given the same homophilic graph with two classes* $y \in \{0, 1\}$ *and partition function* $\alpha : \mathcal{V} \to \{1, 2\}$ *as in Lem. 1, suppose the feature distribution of each partition is* $\mathbf{C}_1, \mathbf{C}_2 \in [0, 1]^2$*, respectively. Consider a 1-layer GNN formulated as* $\mathbf{z} = f(\mathbf{A}, \mathbf{X}) = \sigma(\bar{\mathbf{A}}\mathbf{X}\mathbf{W})$ *for node classification, with row-normalized adjacency matrix* $\bar{\mathbf{A}}$*, sigmoid function* σ*, node features* $\mathbf{x}_v = \text{onehot}(y_v)$*, and a L2-loss function* $\mathcal{L}(y, z) = \frac{1}{2}\|\mathbf{y} - \mathbf{z}\|^2$ *for training. Then, we have:*

1. *When initializing* $\mathbf{W} = \mathbf{0}$*, the discrepancies among the expected initial local gradients* $\mathrm{E}[\nabla \mathcal{L}_i^{local}], i \in \{1, 2\}$ *on each instance, without considering cross-partition edges, and the expected initial gradient* $\mathrm{E}[\nabla \mathcal{L}^{global}]$ *for centralized training increase with the differences of the group distributions* $\|\mathbf{C}_2 - \mathbf{C}_1\|$*.*
2. *For arbitrary learned model weights* $\mathbf{W}$*, the expected loss values* $\mathrm{E}[\mathcal{L}_i^{local}(\mathbf{W})]$ *on each instance* $i \in \{1, 2\}$*, without considering cross-partition edges, is equal if and only if* $\mathbf{C}_1 = \mathbf{C}_2$*.*

We give the proofs of both Lem. 1 and Thm. 2 in App. §E. While our theoretical analysis holds under specific assumptions, we discuss the empirical observations on the discrepancy of loss functions among different trainers under more generalized settings on real-world datasets in §4.2.

3.2.2. Proposed Randomized Partition Schemes

Based on our analysis that disparity of training graph partitions stalls the convergence under model aggregation training, we propose two simple yet effective randomized partition schemes that reduce this disparity in model aggregation training, and combine them with our time-based training framework: RandomTMA employs a randomized partition of nodes, and SuperTMA utilizes a randomized partition of super-nodes (i.e., mini-clusters of nodes (Liu et al., 2018)).

RandomTMA: Randomized Node Partition-based TMA. The idea of randomized node partition is simple: each node is randomly and independently assigned to one of the graph partitions, and the node-induced subgraph $\mathcal{G}^{(i)}$ of each partition i is assigned to the trainers through a one-to-one map-

[2] Although METIS supports balancing nodes with different labels across partitions, this is incompatible with our focus on link prediction as (1) this task does not use node labels during training; and (2) obtaining accurate node labels can be costly for web-scale applications. Even in node classification, only a small portion of labels is available during training.

ping. Since the assignment of each node is considered independently, this partition scheme does not bias towards minimizing the cross-partition edges: the probability of each edge that does not connect nodes in different partitions is $\frac{1}{M}$, where M is the number of trainers. Despite having less data available for model aggregation training than clustering-based frameworks, this partition scheme eliminates the time and cost of graph clustering (c.f. Table 7), and the expected disparity of training data on different partitions. We formalize the latter next:

Corollary 3. *Given the same homophilic graph with two class labels $y \in \{0, 1\}$ and partition function $\alpha : \mathcal{V} \rightarrow \{1, 2\}$ as in Lem. 1, when the nodes are randomly assigned to each partition under independent and identical distributions, the following hold:*

1. $\mathrm{E}[\mathbf{C}_1 - \mathbf{C}_0] = \mathbf{0}$.
2. *For training the GNN described in Thm. 2, the expected loss values $\mathrm{E}[\mathcal{L}_i^{local}(\mathbf{W})]$ and gradients $\mathrm{E}[\nabla\mathcal{L}_i^{local}]$ are equal across trainers $i \in \{0, 1\}$ for arbitrary model weights $\mathbf{W}$.*

We demonstrate the generalizability of this corollary on real-world datasets and different learning tasks in §4.2. Specifically, we observe that RandomTMA reduces the differences in loss functions across different trainers, achieves comparable or better performance than existing distributed training approaches, and has faster convergence speed despite using significantly less training data than frameworks that rely on min-cut partitioning.

SuperTMA: Randomized Super-Node Partition-based TMA. This partition scheme combines (1) the ability of node-level randomized partition in RandomTMA to handle the data disparity issue with (2) the better training data availability and robustness to overfitting of clustering-based partitions (as in PSGD-PA and LLCG (Ramezani et al., 2021)). At a high level, it randomly assigns super-nodes or mini-clusters[3] generated by clustering algorithms to each partition. Specifically, we first use an efficient clustering algorithm like METIS to generate $N \gg M$ mini-clusters for training on M instances. Each mini-cluster is treated as a super-node and is randomly assigned to a graph partition similar to RandomTMA. Then, training subgraph $\mathcal{G}_{train}^{(i)}$ is derived as the subgraph induced by all the collections of *nodes* assigned to partition i (i.e., the union of the nodes in all its assigned super-nodes).

The use of super-nodes generated by clustering algorithms reduces the loss of cross-partition edges compared to RandomTMA, which mitigates the issue of overfitting on smaller datasets or smaller graph partitions when using a large number of trainers. In both cases, SuperTMA shows better performance than RandomTMA and benefits more from an increased number of trainers (§4.2, §4.4).

[3]Similar to our work, ClusterGCN (Chiang et al., 2019) also leverages mini-clusters but it does so in order to form mini-batches for scalable single-instance training; on the other hand, we use mini-clusters to partition the graph for distributed training.

Table 1: Dataset statistics.

Dataset	#Nodes $\lvert\mathcal{V}\rvert$	#Edges $\lvert\mathcal{E}\rvert$	#Feat. F	#Val. / Test Edges
Reddit	232,965	114,615,892	602	114,615 / 114,617
ogbl-citation2	2,927,963	30,561,187	128	86,956 / 86,956
MAG240M-P	121,751,666	1,297,748,926	768	122,088 / 129,781
E-comm	33,886,911	207,157,590	300	1,232,708 / 123,270,705

4. Empirical Analysis

In this section, we seek to address the following research question: **(Q1)** How does the convergence speed and performance of the proposed approaches, RandomTMA and SuperTMA, compare with other training approaches? **(Q2)** What factors contribute to the improved convergence speed and performance of RandomTMA and SuperTMA over the baselines? **(Q3)** How robust are RandomTMA and SuperTMA to different hyperparameters, such as aggregation interval and number of trainers? In App. §D.1, we address **(Q4)**: Are the performance and convergence time of RandomTMA and SuperTMA robust to possible failure of distributed trainers?

4.1. Experimental Setup

In this section, we briefly describe the datasets and training approaches that we study in our experiments. We give more details on our experiment setups in App. §C.

Dataset and evaluation setup. We consider four large-scale networks for our experiments: (1) Reddit (Hamilton et al., 2017), (2) ogbl-citation2 (Hu et al., 2020), (3) MAG240M-P, the paper citation network extracted from MAG240M (Hu et al., 2021), and (4) E-comm, a proprietary heterogeneous dataset of queries and items, which are sampled from anonymized logs of four different market locales of an e-commerce store. We list the statistics of these datasets in Table 1. For link prediction performance, we report the Mean Reciprocal Rank (MRR) of the predicted score of each positive candidate in validation/test splits over 1,000 randomly selected negative candidates.

Training Approaches. We compare the convergence speed and performance for two variants of our proposed training approach, RandomTMA and SuperTMA (with number of super-nodes $N = 15,000$), along with the following baselines: (1) PSGD-PA (Ramezani et al., 2021), which we implemented as a special case of SuperTMA with the number of super-nodes N equal to the number of trainers M to minimize the cross-machine edges; while its original design conducts synchronization on a step-based interval, we enhance it with our time-based model aggregation mechanism

Table 2: Comparison of different training approaches on link prediction: ratio r of training samples (i.e., edges in the training graph) available to each approach, performance (Test MRR), convergence time (in minutes). We report the performance for each training approach as the test MRR obtained on the best encoder (more details in Table 7 and 8), and the convergence time as the time to reach within 1% interval relative to its maximum validation MRR. The average rank is calculated as the average rankings of MRR and convergence time for each approach across all datasets.

Training Approach	#Parts (N)	Reddit ($\|\mathcal{E}\|$ = 114M)			citation2 ($\|\mathcal{E}\|$ = 30.5M)			MAG240M-P ($\|\mathcal{E}\|$ = 1.30B)			E-comm ($\|\mathcal{E}\|$ = 207M)			Average Rank	
		Ratio r	MRR (%)	Time (min)	Ratio r	MRR (%)	Time (min)	Ratio r	MRR (%)	Time (min)	Ratio r	MRR (%)	Time (min)	MRR	Time
RandomTMA	$\|\mathcal{V}\|$	0.33	47.78 ±0.21	67.4 ±7.1	0.33	83.28 ±0.24	56.4 ±14.3	0.33	85.77 ±0.09	169.3 ±27.6	0.33	84.12 ±0.02	52.5 ±20.0	2.0	**1.5**
SuperTMA	15,000	0.35	48.68 ±0.64	154.4 ±6.9	0.58	83.75 ±0.43	126.8 ±39.6	0.64	85.27 ±0.36	189.5 ±0.1	0.76	84.44 ±0.45	126.3 ±12.8	**1.2**	3.5
PSGD-PA	$M = 3$	0.88	46.02 ±0.35	37.2 ±10.0	0.95	82.40 ±0.28	130.2 ±18.6	0.93	84.13 ±0.29	211.8 ±16.6	0.96	83.51 ±0.27	121.4 ±39.9	3.8	3.0
LLCG	$M = 3$	0.88	47.87 ±0.31	229.0 ±15.5	0.95	81.88 ±0.02	134.5 ±10.7	0.93	84.43 ±0.10	184.8 ±2.1	0.96	83.14 ±0.40	91.4 ±13.4	3.5	3.5
GGS	-	1.00	46.63 ±0.11	47.5 ±4.3	1.00	81.95 ±0.20	173.4 ±0.8	1.00	79.52 ±0.24	240.0 ±0.0	1.00	82.13 ±0.42	87.4 ±0.3	4.5	3.5

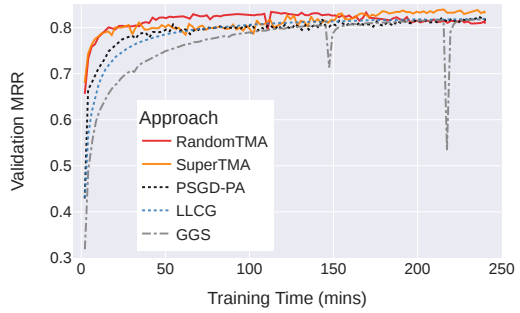


Figure 2: Validation MRR vs. training time for different training approaches on the best-performing GNN on ogbl-citation2. Table 2 gives the test MRR and convergence time.

and focus our analysis on the effects of its partition scheme. (2) Learn locally, correct globally (LLCG) (Ramezani et al., 2021), which behaves similarly to PSGD-PA (we also enhance it with our time-based model aggregation mechanism in our experiments), but has an additional step of global model correction on the server in the model aggregation process; (3) Global Graph Sampling (GGS) (Ramezani et al., 2021; Zheng et al., 2021; Md et al., 2021), where each trainer has unrestricted access to the full training graph, with local models on trainers updated through synchronous SGD to synchronize the gradients among trainers after each training step.

4.2. (Q1) Performance and Convergence Speed

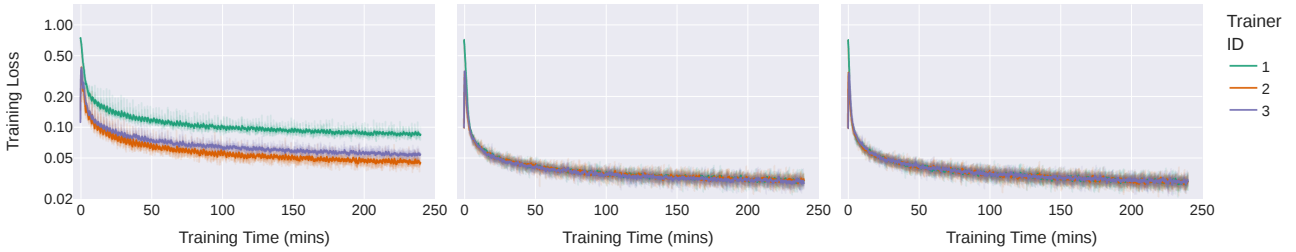
Setup. We compare the link prediction performance and convergence speed of the proposed RandomTMA and SuperTMA approaches with other baselines on benchmark datasets. In Table 2, we list the best performance achieved by each approach with the best GNN encoder. (Table 7 and 8 in the App. §D.2 provides the full results, and the graph partitioning runtime per approach, if applicable.) For the convergence speed of each approach, we report the training time that each approach takes to reach within 1% interval of its maximum validation MRR. We also list the ratio r of the edges in the training graph that are available to each method. We plot the change of validation MRR with respect to the training time on ogbl-citation2 in Fig. 2.

Observations. Despite having less training samples available due to increased cross-partition edges, RandomTMA and SuperTMA perform against the expectation of the previous work (Ramezani et al., 2021) and achieve the best performance on each dataset and the highest average rankings in MRR. Moreover, the faster variant RandomTMA has the best convergence speed overall (it has the highest average ranking in convergence time) and is up to 2.31x faster than the fastest baseline, while still achieving comparable performance to SuperTMA. Overall, we find that RandomTMA strikes the best balance between performance and convergence speed, while SuperTMA may be preferred in applications were the best possible task performance is critical. The superior performance and convergence speed of RandomTMA and SuperTMA also demonstrate the effectiveness of our proposed partition schemes.

4.3. (Q2) Advantages over Baselines

To further dive into the reason behind the improved performance by our proposed approaches, we summarize and discuss two advantages of SuperTMA and RandomTMA over existing approaches.

Reduced Discrepancy among Trainers with Randomized Partitions – Empirical Validation of Theory. We empirically validate our theoretical analysis by comparing the discrepancy of training losses among different trainers for PSGD-PA, SuperTMA and RandomTMA, and show the plots in Fig. 3. The usual $N = M$ partition scheme, which is adopted by baseline approaches like PSGD-PA, leads to significant discrepancies among different trainers in the converged loss values, as shown in Fig. 3a, despite having the least cross-partition edges ignored in the training. In comparison, both the super-node assignment ($N = 15000$, Fig. 3b) and random assignment ($N = |\mathcal{V}|$, Fig. 3c), adopted respectively by SuperTMA and RandomTMA, show better consistency of the converged loss values across multiple trainers; they also converge to smaller loss values compared



(a) PSGD-PA ($N = M$) (b) SuperTMA ($N = 15000$) (c) RandomTMA ($N = |\mathcal{V}|$)

Figure 3: Training loss per trainer vs. training time for PSGD-PA, SuperTMA and RandomTMA for SAGE on MAG240M-P. We show the curves smoothed by exponential moving average ($\alpha = 0.1$), with raw curves dimmed in the background.

to the classical PSGD-PA partition scheme. We observe similar trends for other GNN models and datasets. While smaller loss values on the training split do not always correspond to better performance on the validation and test splits, as the issue of overfitting can occur, the improved consistency of loss convergence across trainers explains the significant improvement in performance of the proposed approaches over PSGD-PA and LLCG and further validates Thm. 2 on real-world datasets.

Improved Efficiency with TMA. We compare the efficiency of the proposed approaches to the baselines by measuring their GPU memory usage and the number of training steps finished on distributed trainers, and present the results on MAG240M-P dataset in Table 3. These results demonstrate the improved efficiency of our Time-based Model Aggregation (TMA) mechanism: all approaches with TMA (including PSGD-PA and LLCG that we enhanced) finish 2.69x to 6.45x more training steps on their *slowest* trainer compared to GGS, which conducts synchronous SGD after each training step. Though the reduced size of the training graph on TMA approaches also contributes to reduced time per training step, the ratio of throughput improvement far exceeds the ratio of reduced graph size (which is reflected by the sampling ratio r and GPU memory usage). Thus, we attribute the significantly improved efficiency of TMA-based approaches to the reduced overhead of synchronization among trainers enabled by the TMA mechanism: by eliminating the need for synchronization after each training step, TMA better accommodates the speed difference among trainers, which is up to 28.8% as we show in Table 3. In comparison, the slowest trainer controls the training speed of the distributed system in GGS (and also in the original design of step-based aggregation interval in PSGD-PA and LLCG), which results in significantly fewer completed training steps. We also note that our proposed approaches, RandomTMA and SuperTMA, by having the least GPU memory usage among all approaches due to the reduced training graph size, enable better scalability to large datasets. Overall, these results show the improved efficiency of our proposed approaches over the baselines (i.e., larger number of completed steps), which also contributes to improved performance and convergence speed.

Table 3: Efficiency of training approaches: GPU memory usage, convergence time (min), and the range of the amount of training steps (in thousands) finished on distributed trainers. Results with the best efficiency are highlighted in green. As discussed in Sec. 4.1, we enhanced PSGD-PA and LLCG with our time-based model aggregation; GGS uses synchronous SGD after each training step.

Dataset ($\lvert\mathcal{E}\rvert$, GNN)	Train Approach	Ratio (r)	GPU RAM (GB)	Conv. Time (min)	Step Finished (10^3)		
					Min	Max	Diff
MAG 240M-P (1.30B, SAGE)	RandomTMA	0.33	7.98 ±0.03	169.3 ±27.6	6.64 ±0.39	7.07 ±0.16	6.1%
	SuperTMA	0.64	9.32 ±0.01	189.5 ±0.1	4.74 ±0.05	5.70 ±0.16	16.9%
	PSGD-PA	0.93	11.25 ±0.00	211.8 ±16.6	3.80 ±0.02	5.33 ±0.33	28.8%
	LLCG	0.93	11.30 ±0.04	184.8 ±2.1	4.08 ±0.01	5.32 ±0.01	23.4%
	GGS	1.00	12.12 ±0.01	240.0 ±0.0	1.03 ±0.07	1.03 ±0.07	0.0%

4.4. (Q3) Robustness to Hyperparameters

Ablation on Aggregation Interval. For approaches that leverage model aggregation (i.e., RandomTMA, SuperTMA, PSGD-PA and LLCG), we examine the effect of the aggregation interval ρ by varying it as 2 (default setting), 8 and 30 minutes. In Table 4, we report the performance and convergence time under these scenarios, where we select the best-performing base model for each training approach and dataset (cf. Table 7 for the best models).

RandomTMA and SuperTMA show consistent prediction performance regardless of the choice of the interval: the differences in test MRR is less than 1% and 0.2% on Reddit and MAG240M-P, respectively. On the other hand, the baseline approaches PSGD-PA and LLCG show significant sensitivity to the interval: as the aggregation interval increases, the test MRR drops for both methods by up to 7.88% and 1.66% on Reddit and MAG240M-P, respectively. These observations show that RandomTMA and SuperTMA, thanks to our proposed partition schemes, do not require frequent aggregations like PSGD-PA and LLCG to achieve their peak performance and convergence speed, which enables further reduction of the communication overhead with longer intervals.

Table 4: Varying aggregation interval ρ: Comparison of link prediction performance (MRR) and convergence time (min). Within each row, we highlight the aggregation interval with the best MRR in blue and the least convergence time in green.

Dataset ($\lvert\mathcal{E}\rvert$, GNN)	Train Approach	Test MRR (%)			Conv. Time (min)		
		$\rho = 2$	$\rho = 8$	$\rho = 30$	$\rho = 2$	$\rho = 8$	$\rho = 30$
Reddit (114M, GCN)	RandomTMA	47.78±0.21	47.38±0.60	46.86±0.47	67.4±7.1	40.1±11.3	60.0±0.0
	SuperTMA	48.68±0.64	48.51±0.09	47.77±0.09	154.4±6.9	76.1±51.0	75.0±21.2
	PSGD-PA	46.02±0.35	43.78±0.34	40.21±0.13	37.2±10.0	188.3±51.0	165.1±63.7
	LLCG	47.87±0.31	44.54±0.19	39.99±0.61	229.0±15.5	48.2±11.3	165.2±63.7
MAG 240M-P (1.30B, SAGE)	RandomTMA	85.77±0.09	85.79±0.17	85.82±0.26	169.3±27.6	164.9±5.6	180.2±0.0
	SuperTMA	85.27±0.36	85.38±0.25	85.22±0.10	189.5±0.1	193.2±11.4	210.3±0.0
	PSGD-PA	84.13±0.29	83.44±0.25	82.47±0.39	211.8±16.6	201.4±45.7	210.3±0.0
	LLCG	84.43±0.10	84.27±0.40	82.95±0.12	184.8±2.1	191.9±6.0	211.3±0.2

Table 5: Varying number of trainers M: Comparison of ratio r of training samples available, link prediction performance (MRR), and convergence time (min). Within each row, we highlight the number of trainers with the best MRR in blue and the least convergence time in green. "OOM" denotes that experiments run out of memory.

Dataset ($\lvert\mathcal{E}\rvert$, GNN)	Train Approach	Ratio (r)			Test MRR (%)			Conv. Time (min)		
		M=3	M=5	M=23	M=3	M=5	M=23	M=3	M=5	M=23
MAG 240M-P (1.30B, SAGE)	RandomTMA	0.33	0.20	0.04	85.77±0.09	85.97±0.32	84.94±0.23	169.3±27.6	158.4±4.1	125.0±5.2
	SuperTMA	0.64	0.56	0.48	85.27±0.36	86.02±0.27	86.21±0.53	189.5±0.1	181.8±9.8	190.6±16.3
	PSGD-PA	0.93	0.90	0.78	84.13±0.29	84.35±0.07	82.13±0.12	211.8±16.6	206.5±0.1	208.8±17.0
	LLCG	0.93	0.90	0.90	84.43±0.10	83.87±0.39	(OOM)	184.8±2.1	194.6±16.6	(OOM)
E-comm (207M, GCN)	RandomTMA	0.33	0.20	0.04	84.12±0.02	84.95±0.41	80.73±0.06	52.5±20.0	67.0±23.0	18.4±14.5
	SuperTMA	0.76	0.71	0.65	84.44±0.45	84.95±0.27	85.48±0.37	126.3±12.8	129.0±19.1	130.5±2.3
	PSGD-PA	0.96	0.96	0.92	83.51±0.27	83.55±0.29	83.40±0.01	121.4±39.9	124.0±11.7	107.6±0.1
	LLCG	0.96	0.96	0.92	83.14±0.40	83.46±0.37	83.93±0.55	91.4±13.4	124.1±32.2	111.1±10.2

Ablation on Number of Trainers. To understand the effect of increased number of trainers for model aggregation approaches, we compare the performance, convergence time and ratio of training samples available in the cases of $M = 3$ (the default setting), $M = 5$, and a very large number of $M = 23$ trainers[4] in Table 5. We run this experiment on the largest MAG240M-P dataset and the proprietary large E-comm dataset, and select the best-performing GNN (i.e., GraphSAGE for MAG240M-P, and GCN with MLP decoder for E-comm) for all training approaches as the base model.

We observe that the amount of available training samples decreases for all approaches as the number of trainers increases, due to the increase of cross-partition edges. RandomTMA has a sweet spot for edge ratio r and the number of trainers M: compared to $M = 3$, it shows slightly improved performance for $M = 5$, but worse performance for $M = 23$, especially for the smaller E-comm dataset. We attribute this to the trade-off between increased data throughput and decreased amount of training samples for an increased number of trainers. SuperTMA, on the other hand, effectively mitigates the side-effect of data loss under increased number of trainers, has significantly more training samples compared to RandomTMA, and shows consistently improved performance; this demonstrates the effectiveness of conducting randomized partitions on mini-clusters. Despite leveraging the most training edges under all cases, PSGD-PA and LLCG consistently perform worse than SuperTMA (and RandomTMA in most cases), which highlights the importance of data uniformity over the amount of available training samples.

[4]Number of trainers $M = 23$ maps to the maximum number of trainers we can set up with 24 GPUs, as we reserve one GPU for model evaluation on the server process.

5. Conclusion

Reexamining prior assumptions linking distributed GNN training performance with cross-instance node dependency coverage, we find, both theoretically and empirically, that min-cut partitioning algorithms negatively impact training by causing data distribution discrepancies across trainers. Consequently, we introduce the Time-based Model Aggregation (TMA) framework for distributed GNN training, along with randomized partition schemes of nodes or supernodes to minimize data discrepancy across localized trainers. Despite utilizing significantly fewer edges in training, our proposed methods, RandomTMA and SuperTMA, deliver state-of-the-art link prediction performance and rapid convergence. Future evaluations will encompass a broader array of graph learning tasks such as node classification.

References

Angerd, A., Balasubramanian, K., and Annavaram, M. Distributed training of graph convolutional networks using subgraph approximation. *arXiv preprint arXiv:2012.04930*, 2020.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Breuer, A., Eilat, R., and Weinsberg, U. Friend or faux: Graph-based early detection of fake accounts on social networks. In *Proceedings of the 2020 World Wide Web Conference*, pp. 1287–1297, 2020.

Cao, Q., Shen, H., Gao, J., Wei, B., and Cheng, X. Popularity prediction on social platforms with coupled graph neural networks. In *WSDM*, 2020.

Chen, J., Zhu, J., and Song, L. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pp. 942–950. PMLR, 2018.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. 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.

Dou, Y., Liu, Z., Sun, L., Deng, Y., Peng, H., and Yu, P. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *CIKM*, 2020.

Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., and Yin, D. Graph neural networks for social recommendation. In *Proceedings of the 2019 World Wide Web Conference*, pp. 417–426, 2019.

Fey, M., Lenssen, J. E., Weichert, F., and Leskovec, J. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *International Conference on Machine Learning*, pp. 3294–3304. PMLR, 2021.

Frasca, F., Rossi, E., Eynard, D., Chamberlain, B., Bronstein, M., and Monti, F. Sign: Scalable inception graph neural networks. In *Workshop of Graph Representation Learning and Beyond (GRL+)*, 2020.

Gandhi, S. and Iyer, A. P. P3: Distributed deep graph learning at scale. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, pp. 551–568, 2021.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Hu, W., Fey, M., Ren, H., Nakata, M., Dong, Y., and Leskovec, J. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

Jiang, P. and Rumi, M. A. Communication-efficient sampling for distributed training of graph convolutional networks. *arXiv preprint arXiv:2101.07706*, 2021.

Karypis, G. and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Li, J., Shomer, H., Ding, J., Wang, Y., Ma, Y., Shah, N., Tang, J., and Yin, D. Are graph neural networks really helpful for knowledge graph completion? *arXiv preprint arXiv:2205.10652*, 2022.

Li, M., Andersen, D. G., Smola, A. J., and Yu, K. Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems*, 27, 2014.

Liu, Y., Safavi, T., Dighe, A., and Koutra, D. Graph summarization methods and applications: A survey. *ACM Comput. Surv.*, 51(3):62:1–62:34, 2018.

Matena, M. S. and Raffel, C. A. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 35:17703–17716, 2022.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Md, V., Misra, S., Ma, G., Mohanty, R., Georganas, E., Heinecke, A., Kalamkar, D., Ahmed, N. K., and Avancha, S. Distgnn: Scalable distributed training for large-scale graph neural networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14, 2021.

Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., and Zaharia, M. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pp. 1–15, 2019.

Narayanan, S. D., Sinha, A., Jain, P., Kar, P., and SELLAMANICKAM, S. Iglu: Efficient gcn training via lazy updates. In *International Conference on Learning Representations*, 2021.

Polyak, B. T. and Juditsky, A. B. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

Qiu, J., Tang, J., Ma, H., Dong, Y., Wang, K., and Tang, J. Deepinf: Social influence prediction with deep learning. In *KDD*, pp. 2110–2119, 2018.

Ramezani, M., Cong, W., Mahdavi, M., Kandemir, M., and Sivasubramaniam, A. Learn locally, correct globally: A distributed algorithm for training graph neural networks. In *International Conference on Learning Representations*, 2021.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*, pp. 593–607. Springer, 2018.

Stich, S. U. Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

Tripathy, A., Yelick, K., and Buluç, A. Reducing communication in graph neural network training. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14. IEEE, 2020.

van den Berg, R., Kipf, T., and Welling, M. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

Wang, J., Wen, R., Wu, C., Huang, Y., and Xiong, J. Fdgars: Fraudster detection via graph convolutional networks in online app review system. In *Proceedings of the 2019 World Wide Web Conference*, pp. 310–316, 2019.

Wang, X., He, X., Wang, M., Feng, F., and Chua, T.-S. Neural graph collaborative filtering. In *SIGIR*, pp. 165–174, 2018.

Wortsman, M., Ilharco, G., Gadre, S. Y., Roelofs, R., Gontijo-Lopes, R., Morcos, A. S., Namkoong, H., Farhadi, A., Carmon, Y., Kornblith, S., et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, pp. 23965–23998. PMLR, 2022.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *ICML*, 2019.

Yang, B., Yih, S. W.-t., He, X., Gao, J., and Deng, L. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the International Conference on Learning Representations (ICLR) 2015*, 2015.

Ying, R., He, R., Chen, K., Eskombatchai, P., Hamilton, W., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, pp. 974–983, 2018.

You, J., Ying, Z., and Leskovec, J. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33:17009–17021, 2020.

Yu, H., Yang, S., and Zhu, S. Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5693–5700, 2019.

Yu, L., Shen, J., Li, J., and Lerer, A. Scalable graph neural networks for heterogeneous graphs. *arXiv preprint arXiv:2011.09679*, 2020.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2019.

Zeng, H., Zhang, M., Xia, Y., Srivastava, A., Malevich, A., Kannan, R., Prasanna, V., Jin, L., and Chen, R. Decoupling the depth and scope of graph neural networks. *Advances in Neural Information Processing Systems*, 34: 19665–19679, 2021.

Zhang, Z., Wang, J., Ye, J., and Wu, F. Rethinking graph convolutional networks in knowledge graph completion. In *Proceedings of the ACM Web Conference 2022*, pp. 798–807, 2022.

Zheng, D., Ma, C., Wang, M., Zhou, J., Su, Q., Song, X., Gan, Q., Zhang, Z., and Karypis, G. Distdgl: distributed graph neural network training for billion-scale graphs. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pp. 36–44. IEEE, 2020.

Zheng, D., Song, X., Yang, C., LaSalle, D., and Karypis, G. Distributed hybrid cpu and gpu training for graph neural networks on billion-scale graphs. *arXiv preprint arXiv:2112.15345*, 2021.

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. Beyond homophily in graph neural networks: Current limitations and effective designs. In *NeurIPS*, 2020.

Zhu, R., Zhao, K., Yang, H., Lin, W., Zhou, C., Ai, B., Li, Y., and Zhou, J. Aligraph: a comprehensive graph neural network platform. *arXiv preprint arXiv:1902.08730*, 2019.

Appendix

A. Additional Related Works

Scalable GNN Training on Single Instance. Scalable single-instance training approaches can be grouped into three categories:

• *Sampling of the Message Flow Graph* (MFG): This approach is popular for reducing the complexity of message passing. For example, GraphSAGE (Hamilton et al., 2017) and VR-GCN (Chen et al., 2018) aggregate embeddings from a subset of neighbors for each node encountered in a training step to cap the size of MFG, while ClusterGCN (Chiang et al., 2019), GraphSAINT (Zeng et al., 2019) and shaDow-GNN (Zeng et al., 2021) sample a subgraph per training step and confine the training of GNN on the sampled subgraph.

• *Message passing pre-computation*: This approach relies on pre-computing aggregated features in the neighborhood of each node and using them to learn embeddings for each node independently (e.g., SIGN (Frasca et al., 2020), NARS (Yu et al., 2020)). However, it requires models like SGC (Wu et al., 2019) that are capable of decoupling feature aggregations from (usually linear) transformations, which restricts the GNN expressiveness.

• *Caching and lazy updates of stale representations or gradients*: Methods in this category aim to limit the expansion of MFG. For example, IGLU (Narayanan et al., 2021) uses these techniques on backward propagation; GNNAutoScale (Fey et al., 2021) stores the historical node embeddings per layer, and only updates the stored embeddings for nodes in the mini-batch, while using the historical embeddings for the other nodes.

Our approach on model aggregation training is orthogonal to these efforts as we focus on distributed settings with multiple trainers. Any of the approaches mentioned above can be adopted in our framework to further speed up each individual trainer.

B. Comparison of TMA with Existing Frameworks

In this section, we formally present (in addition to Fig. 1) the design of the server and trainer processes in Algorithm 1 and 2, respectively, and provide an in-depth comparison of our proposed TMA framework to prior distributed GNN training frameworks.

TMA vs. DistDGL. DistDGL (Zheng et al., 2021) assumes that each trainer (or mini-batch sampler) has access to the full training graph $\mathcal{G}_{\text{train}}$; whereas our Time-based Model Aggregation (TMA) framework only allows each trainer i to access its local training subgraph $\mathcal{G}^{(i)}_{\text{train}} \subset \mathcal{G}_{\text{train}}$. The more restrictive access to the training data in the TMA framework reduces the amount of available training samples and is widely believed to result in inferior performance in previous works (Ramezani et al., 2021). However, we show in §3.2 that with our proposed partition schemes, which minimize the discrepancy of gradient and loss function across trainers, the TMA framework can achieve better or comparable performance to DistDGL with improved convergence speed. In addition, DistDGL synchronizes the gradients and SGD of all trainers after each training step; TMA only periodically synchronizes the model weights (instead of gradients) among trainers, which significantly reduces the number of synchronizations and allows asynchronous training steps before time-based model aggregation.

TMA vs. PSGD-PA and LLCG. While the PSGD-PA and LLCG approaches (Ramezani et al., 2021) are also designed upon the model aggregation mechanism, they adopt a different approach to mitigate the performance gap compared to global-access and fully-synchronous approach like DistDGL: PSGD-PA uses one-to-one mapping of METIS clusters to trainers to minimize the number of cross-partition edges, and LLCG further employs periodical fallbacks to centralized training to recover more cross-instance edges. In contrast, our TMA framework discards the cross-instance dependencies (resulting in significantly fewer training edges) and leverages randomized partition schemes that reduce the disparity of training data among trainers. Also, the design of PSGD-PA and LLCG requires more synchronization of the training progress across different trainers, as averaging is triggered after a certain number of training steps per trainer, while our framework utilizes time-based aggregation intervals to accommodate different speeds across instances.

C. Additional Details on Experiment Setups

In this section, we give additional details on the experimental setups presented in §4.1.

Details in dataset and evaluation setup. We list the statistics of all datasets in Table 1. To our knowledge, MAG240M-P is the largest publicly-available homogeneous benchmark network with 768-dimensional node features (175 GB in storage) and over 1.29 billion edges. For ogbl-citation2, we use the train / validation / test splits provided with the dataset; for Reddit and MAG240M-P which are originally proposed as node classification benchmarks, we create the validation / test splits by randomly selecting and removing one outgoing edge per node in the validation / test splits of node classification; for E-comm, we use all item

Algorithm 1: Time-based Model Aggregation Server

Input: Total training time ΔT_{train}, model aggregation interval ΔT_{int} & operator ϕ, validation & test sample splits and subgraphs $\mathcal{G}_{\text{val}}$ & $\mathcal{G}_{\text{test}}$, trainer IDs $\{1, \cdots, M\}$ and network addresses, `optimizer` function, model configurations & hyperparameters.

Establish communication with trainers and distributed Key-Value Store `KV`; broadcast `optimizer` function, model configurations & hyperparameters.
Setup model and initialize model weights $\mathbf{W}_{\text{global}}[0]$.
Wait until `all`([`KV[ready][i]` for $i \in \{1, \cdots, M\}$]).
`KV[agg]` = `KV[stop]` = `False`
Broadcast initialized model weights $\mathbf{W}_{\text{global}}[0]$ to trainers.
$T_{\text{start}} = T_{\text{agg}} =$ `current_time()`; $t = 0$.
while *not* `KV[stop]` **do**
  if `current_time()` $- T_{\text{agg}} \geq \Delta T_{\text{int}}$ **then**
    `KV[agg]` = `True`
    Receive weights $\mathbf{W}_i$[t] from trainer $i \in \{1, \cdots, M\}$.
    `KV[agg]` = `False`
    $\mathbf{W}_{\text{global}}[t+1] = \phi(\mathbf{W}_1[t], \cdots, \mathbf{W}_M[t])$.
    Broadcast global weights $\mathbf{W}_{\text{global}}[t+1]$ to trainers.
    Invoke `metrics`[t+1] = `eval`($\mathbf{W}_{\text{global}}[t+1]$, $\mathcal{G}_{\text{val}}$) on an evaluation process.
    $t = t + 1$
  if `current_time()` $- T_{\text{start}} > \Delta T_{\text{train}}$ **then**
    `KV[stop]` = `True`
Wait until `metrics`[t] is ready; $t_* =$ `arg_best`(`metrics`).
`metrics`[t_*] = `eval`($\mathbf{W}_{\text{global}}[t_*]$, $\mathcal{G}_{\text{test}}$).

Output: Best model weights $\mathbf{W}_{\text{global}}[t_*]$, `metrics` and t_*.

Algorithm 2: Time-based Model Aggregation Trainer

Input: Trainer ID $i \in 1, \cdots, M$; assigned training subgraph $\mathcal{G}^{(i)}_{\text{train}} \subset \mathcal{G}_{\text{train}}$.

Establish communication with the server and distributed Key-Value Store `KV`; receive `optimizer` function, model configurations & hyperparameters.
Initialize `KV[ready][i]` = `False`.
Load $\mathcal{G}^{(i)}_{\text{train}}$; prepare data for training; set up GNN model.
`KV[ready][i]` = `True`
Receive initialized model weights $\mathbf{W}_{\text{global}}[0]$ from server.
$t = 0$
while *not* `KV[stop]` **do**
  Construct mini-batch $\xi_i^{(t)}$ on local subgraph $\mathcal{G}^{(i)}_{\text{train}}$.
  $\mathbf{W}_i[t] =$ `optimizer`$(\xi_i^{(t)}, \mathbf{W}_i[t])$
  if `KV[agg]` **then**
    Send local model weights $\mathbf{W}_i$[t] to the server.
    Get global model weights $\mathbf{W}_{\text{global}}[t+1]$ from server.
    Overwrite local weights $\mathbf{W}_i$[t]$\leftarrow$ $\mathbf{W}_{\text{global}}[t+1]$.
    $t = t + 1$

correlations and 3 months of query-item associations for training, and use the next month of query-item associations for model evaluation. We do not use neighborhood sampling in the evaluation process as it introduces additional randomness to the test results. The E-comm dataset has two types of edges: (1) edges connecting two items that are related, and (2) edges connecting items that are related to one or more queries. The node features E-comm dataset is generated by a fine-tuned BERT model.

GNN Encoders. We consider two GNN choices for encoders on homogeneous graphs: GCN (Kipf & Welling, 2017) and GraphSAGE (Hamilton et al., 2017). In addition, we adopt MLP as an additional baseline, as previous works have revealed that GNNs are not guaranteed to perform better than a graph-agnostic baseline (Zhu et al., 2020). For the heterogeneous E-comm dataset, we test GCN (Kipf & Welling, 2017) and RGCN (Schlichtkrull et al., 2018) as encoders. For all models, we follow Chen et al. (2018) and You et al. (2020) and use PReLU as non-linear activation function, and Layer Normalization (Ba et al., 2016) before activation to improve performance of all encoders. We list more hyperparameters for encoder in App. §C.

Link Prediction Decoder. On homogeneous graphs, we use an MLP decoder to predict the link probability between a pair of nodes: we find in our experiments that multi-layer MLP with non-linearity significantly improves the link prediction performance over a vanilla dot product decoder. We elaborate on the formulation of MLP decoder in App. §C. We additionally test DistMult (Yang et al., 2015) as a decoder for heterogeneous graphs on E-comm dataset. To ensure a fair comparison between different training approaches and encoders, we fix on each dataset the number of layers and the sizes of hidden states for the decoder; we list these parameters in App. §C.

Mini-batch and Negative Sampling. For all trainings, we randomly select edges in the training set to form the mini-batches, and use GraphSAGE sampler (Hamilton et al., 2017) to reduce the size of Message Flow Graph (MFG). For each positive edge sample (u, v) in the mini-batches, we randomly sample one edge (u, v') with a different tail $v' \in \mathcal{V}$ as the negative sample.

Hardware Specifications. We spend ∼4,600 GPU hours on a maximum of three AWS EC2 p3.16xlarge instances for our experiments, with each instance featuring 64 CPU cores, 488 GB RAM, and 8 NVIDIA Tesla V100 GPU with 16 GB Memory per GPU.

Details on Trainer Setup. To keep the empirical analysis resource- and cost-efficient, we run experiments with $M = 3$ training processes on Reddit and ogbl-citation2 on a single physical instance; for MAG240M-P, we run $M = 3$ training processes on 2 physical instances by default. In §4.4, we further report the results of $M = 5$ and the maximum $M = 23$ training processes using all 3 physical instances.

We give the distribution of the trainers on physical instances as follows: For the largest dataset, MAG240M-P, with $M = 3$ trainers, we run the TMA server and one trainer on physical instance #1 and the other two trainers on instance #2. For $M = 5$ trainers, we run the two additional trainers on instance #3. For the smaller datasets (Reddit, ogbl-citation2 and E-comm), we run the TMA server and $M = 3$ or $M = 5$ trainers on a single physical instance. For all experiments with $M = 23$ trainers, we use all 24 GPUs on three physical instances, with one GPU reserved for model evaluation on the server process.

Hyperparameter Choices. We tune and select the best-performing hyperparameters on the GGS baseline, and adopt the same hyperparameters for distributed training approaches for a fair comparison.

- For Reddit and ogbl-citation2, we use 2-layer models with the size of hidden representations as 256 for both the encoder and decoder;
- For the larger MAG240M-P dataset, we use 2-layer models with the size of hidden representations as 64.
- For E-comm, we use 2-layer GCN or RGCN models as the encoder, and DistMult or 2-layer MLP as the decoder. For GCN, we set the dimension of hidden representations as 128. To reduce the memory usage of RGCN, we adopt basis decomposition (Schlichtkrull et al., 2018) with 4 bases (equal to the total number of forward and inverse relations), each with 128 dimension, and added an MLP layer before RGCN input to reduce the dimension of input representations to 128. For DistMult decoder, we set the dimension of each relational embedding as 128. For MLP decoder, we use 2 layers with the size of hidden representations as 128.

We set the learning rate $\texttt{lr} = 0.001$ in all the experiments, since we find that it significantly improves the performance compared to $\texttt{lr} = 0.01$. For all experiments, we allocate 4-hour training time; in most cases, this is sufficient time for models to reach convergence (as shown in Fig. 2), while not incurring excessive time and monetary cost.

Implementation of Training Approaches. We implement GGS using the MultiGPU training functionality, where each trainer runs on a separate GPU of the physical machine; though DistDGL (Zheng et al., 2021) is not directly compatible with our implementation, our implementation emulates its training pipeline and represents an ideal version of DistDGL without the communication overhead of accessing node embeddings remotely. For all approaches, we create a separate process for model evaluation as in Fig. 1, and adopt the same interval for model evaluation to ensure a fair comparison.

Table 6: Robustness to trainer failures: Comparison of link prediction performance (MRR) and convergence time (min) when one of the $M = 3$ trainers fails to start. For $F = 1$, we run M experiments per random seed by dropping a different subgraph at a time, and report the average metrics.

Dataset ($\lvert\mathcal{E}\rvert$, GNN)	Train Approach	Test MRR (%)		Conv. Time (min)	
		$F = 1$	$F = 0$	$F = 1$	$F = 0$
MAG 240M-P (1.30B, SAGE)	RandomTMA	85.54 ±0.08	85.77 ±0.09	161.8 ±13.6	169.3 ±27.6
	SuperTMA	85.17 ±0.11	85.27 ±0.36	191.7 ±12.5	189.5 ±0.1
	PSGD-PA	82.09 ±4.09	84.13 ±0.29	199.0 ±16.0	211.8 ±16.6
	LLCG	82.20 ±3.45	84.43 ±0.10	203.2 ±22.3	184.8 ±2.1

Formulation of MLP Decoder. Given the embeddings $\mathbf{r}_u$ and $\mathbf{r}_v$ for nodes u, v by GNN encoder, respectively, the k-th layer of the MLP decoder is formulated as $\mathbf{e}_{u,v}^{(k+1)} = \sigma(\mathbf{e}_{u,v}^{(k)}\mathbf{\Theta}^{(k)})$, where $\mathbf{\Theta}^{(k)}$ is the learnable weight matrix, and $\mathbf{e}_{u,v}^{(0)} = \mathbf{r}_u \odot \mathbf{r}_v$ is the element-wise product of $\mathbf{r}_u$ and $\mathbf{r}_v$; the predicted link probability $\hat{y}_{u,v} = \mathbf{e}_{u,v}^{(K)}$ equals to the output scalar for a K-layer decoder. We adopt PReLU as the activation function σ, as we do for the encoders.

D. Additional Experiment Results

D.1. (Q4) Robustness to Trainer Failures

Distributed systems can suffer from failure of workers as a result of unexpected faults or issues with the communication network. Fortunately, model aggregation training allows the frameworks to be robust to partial failures (e.g., when some trainers go offline), as the training can continue with only the remaining trainers. However, the subgraphs assigned to failed instances will be unavailable in the remaining of the training process, unless the server reassigns these subgraphs to any available back-up training instances.

Setup. Here we emulate a simple scenario of failure where $F = 1$ of the $M = 3$ trainers in previous experiments fail to start, with no back-up trainer in place; in this case, we complete the training with the remaining graph information on $M - 1$ trainers. Our goal is to understand the robustness of RandomTMA and SuperTMA to trainer failures in comparison with model aggregation baselines.

Table 6 reports the performance and convergence time of the training approaches when a worker fails to start ($F = 1$), compared with the case where all workers proceed normally ($F = 0$). For the $F = 1$ case, we run the M experiments per random seed by dropping a different partition at a time to emulate failure of different trainers under the same assignment, and report the average results.

Observations. We observe that the performance and convergence speed of RandomTMA and SuperTMA are more robust to the failure of the trainers compared to PSGD-PA and LLCG: the test MRR decreases by less than 0.3% for RandomTMA and SuperTMA as a result of the single trainer failure; in comparison, the test MRR of PSGD-PA and LLCG decreases more than 2.0% in the case of failure. These results demonstrate the improved robustness of RandomTMA and SuperTMA: less discrepancy among data assigned to different trainers minimizes the information loss in the case of failures.

D.2. Ablation on Base Models

We list in Table 7 the performance and convergence time of the training approaches on different base models for homogeneous datasets (i.e., GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017) and MLP encoders; all with MLP decoder), with results on base models in Table 8 for E-comm dataset (i.e., GCN (Kipf & Welling, 2017) and RGCN (Schlichtkrull et al., 2018) with MLP or DistMult (Yang et al., 2015) decoder); in Table 2 of the main paper, we report the results for the best-performing base model per approach and dataset. We did not test MLP for LLCG, as MLP is graph-agnostic and does not benefit from the LLCG global model correction process for recovering cross-partition edges (Ramezani et al., 2021).

On homogeneous datasets (Table 7), we observe that GCN and GraphSAGE are the best-performing base models for all approaches on Reddit and MAG240M-P, respectively; on ogbl-citation2, the best-performing base models vary for different training approaches.

On the heterogeneous E-comm dataset (Table 8), we surprisingly observe that GCN, which ignores the heterogeneous edge types in the dataset, outperforms RGCN designed for heterogeneous graphs by a large margin. Prior works have also observed that modeling heterogeneous relations in GNN models may not be as crucial as widely presumed (Zhang et al., 2022; Li et al., 2022), and we leave for the future works for further investigation of this finding.

E. Proofs of Theorems

Proof for Lemma 1. Following the assumption, the probability p_{ji} of node j to connect to node i can be written as

$$p_{ji} = \frac{\mathbf{H}(y_i, y_j)}{\sum_{l\in\mathcal{V}} \mathbf{H}(y_l, y_j)} = \frac{1}{C}\mathbf{H}(y_i, y_j) = \begin{cases} h/C, & \text{if } y_i = y_j \\ (1-h)/C, & \text{if } y_i \neq y_j \end{cases}$$

Now assume that the ratio of nodes v with label $y_v = 0$ in partition 1 as β_1, and in partition 2 as β_2, then we have the feature distributions $\mathbf{C}_1$ and $\mathbf{C}_2$ in each partition, which follow the distributions for class labels 0 and 1 under onehot-encoded node features $\mathbf{x}_v$, as $\mathbf{C}_1 = [\beta_1, 1-\beta_1]$ and $\mathbf{C}_2 = [\beta_2, 1-\beta_2]$. Since we assume that the two partitions have equal sizes η, and two class labels with equal sizes, we have $\beta_1\eta + \beta_2\eta = (1-\beta_1)\eta + (1-\beta_2)\eta$ and $\beta_2 = 1-\beta_1$. Thus, we denote $\beta_1 = \beta \in [0, 1]$ and simplify $\mathbf{C}_1$ and $\mathbf{C}_2$ as $\mathbf{C}_1 = [\beta, 1-\beta]$ and $\mathbf{C}_2 = [1-\beta, \beta]$; without loss of generality, we assume $\beta \geq 0.5$.

We denote the random variable $A_{ij} = 1$ if an edge exists between node i and j, and $A_{ij} = 0$ otherwise. Then we have $\mathrm{E}[A_{ij}] = p_{ji}$. The expected number of edge cuts between the two partitions λ is

$$\lambda = \sum_{\substack{i\in\alpha^{-1}(1)\\ j\in\alpha^{-1}(2)}} \mathrm{E}[A_{ij}] = \sum_{\substack{i\in\alpha^{-1}(1),y_i=0\\ j\in\alpha^{-1}(2)}} \mathrm{E}[A_{ij}] + \sum_{\substack{i\in\alpha^{-1}(1),y_i=1\\ j\in\alpha^{-1}(2)}} \mathrm{E}[A_{ij}] \coloneqq (\lambda_0 + \lambda_1)/C, \tag{1}$$

and it is straightforward to show that $\lambda_0 = \beta\eta((1-\beta)\eta h + \beta\eta(1-h))$ and $\lambda_1 = (1-\beta)\eta((1-\beta)\eta(1-h) + \beta\eta h)$. As a result, we have

$$\lambda = \left(1 - 2(1-\beta)\beta - (2\beta-1)^2 h\right)\frac{\eta^2}{C} \tag{2}$$

For $\beta \in [0.5, 1]$ and homophilic graph with $h \geq 0.5$, it is easy to show that λ reaches the minimal value when $\beta = 1$, and $\mathbf{C}_1 = [1, 0]$ and $\mathbf{C}_2 = [0, 1]$. Therefore, we show that the minimal expected edge cut is reached when each partition contains only the node from a single class with the same features. □

Proof for Theorem 2. We note that under the L2-loss function $\mathcal{L}(y, z) = \frac{1}{2}\|\mathbf{y} - \mathbf{z}\|^2$, the loss value (or gradient) for a training batch with multiple nodes is the sum of the loss value (or gradient) calculated individually on each node; thus, we can simplify our discussion by only examining the loss value and gradient for a single node. For an arbitrary node $v \in \mathcal{V}$ for training, we have

$$z_v = \sigma\left(\sum_{u\in\mathcal{N}_\mathcal{G}(v)} \frac{1}{d_v}\mathbf{x}_u^\mathsf{T}\mathbf{w}\right) \coloneqq \sigma(g(\mathbf{w})) = \frac{1}{1+e^{-g(\mathbf{w})}} \tag{3}$$

Without loss of generality, we assume the class label of node v as $y_v = 1$. In this case, we have the loss function

Table 7: Comparison of link prediction performance (MRR) and convergence time (minutes) under different base models (i.e., GCN, GraphSAGE and MLP). The convergence time is reported as the time to reach within 1% interval of the maximum validation MRR. We also report for each dataset and approach the ratio r of the edges in the training graph that are available, and the preprocessing time of METIS (if needed). We highlight within each row the GNN model (excluding MLP) with the best MRR in blue, and the least convergence time in green. MLP is graph-agnostic and thus not tested for LLCG.

Dataset ($\|\mathcal{E}\|$)	Train Approach	#Parts (N)	Ratio (r)	Prep. Time (mins)	Test MRR (%)			Conv. Time (min)		
					GCN	SAGE	MLP	GCN	SAGE	MLP
Reddit (114M)	RandomTMA	$\|\mathcal{V}\|$	0.33	0	47.78±0.21	43.65±0.49	22.92±0.02	67.4±7.1	175.9±22.7	166.9±31.2
	SuperTMA	15,000	0.35	6.5	48.68±0.64	43.93±0.03	22.89±0.15	154.4±6.9	188.9±4.2	194.1±9.9
	PSGD-PA	$M=3$	0.88	0.7	46.02±0.35	43.42±2.08	16.55±0.38	37.2±10.0	240.0±0.0	240.0±0.0
	LLCG	$M=3$	0.88	0.7	47.87±0.31	44.61±0.14	-	229.0±15.5	185.3±77.4	-
	GGS	-	1.00	0	46.63±0.11	43.85±0.22	24.31±0.09	47.5±4.3	209.5±18.0	129.5±14.6
ogbl-citation2 (30.5M)	RandomTMA	$\|\mathcal{V}\|$	0.33	0	83.28±0.24	80.96±0.00	40.69±0.01	56.4±14.3	101.9±12.9	57.3±12.8
	SuperTMA	15,000	0.58	1.6	83.75±0.43	80.90±0.01	41.36±0.08	126.8±39.6	100.2±18.6	86.4±11.4
	PSGD-PA	$M=3$	0.95	0.7	82.40±0.28	81.64±0.00	39.43±0.16	130.2±18.6	146.0±8.8	176.1±24.3
	LLCG	$M=3$	0.95	0.7	81.62±0.47	81.88±0.02	-	142.6±48.2	134.5±10.7	-
	GGS	-	1.00	0	81.64±0.17	81.95±0.20	41.71±0.03	178.5±43.8	173.4±0.8	90.1±9.9
MAG 240M-Papers (1.30B)	RandomTMA	$\|\mathcal{V}\|$	0.33	0	85.08±0.30	85.77±0.09	48.54±0.27	213.8±23.2	169.3±27.6	156.1±10.9
	SuperTMA	15,000	0.64	153.9	82.21±0.04	85.27±0.36	49.14±0.15	214.4±36.2	189.5±0.1	164.8±5.7
	PSGD-PA	$M=3$	0.93	84.4	80.90±0.09	84.13±0.29	48.30±0.15	240.0±0.0	211.8±16.6	195.7±17.6
	LLCG	$M=3$	0.93	84.4	78.61±1.23	84.43±0.10	-	238.4±2.3	184.8±2.1	-
	GGS	-	1.00	0	77.75±0.83	79.52±0.24	47.97±0.09	236.1±5.5	240.0±0.0	177.0±4.2

Table 8: Comparison of link prediction performance (MRR) and convergence time (minutes) under different base models (i.e., GCN, RGCN) and link prediction decoders (i.e., MLP, DistMult). The convergence time is reported as the time to reach within 1% interval of the maximum validation MRR. We also report for each dataset and approach the ratio r of the edges in the training graph that are available, and the preprocessing time of METIS (if needed). We highlight within each row the GNN model with the best MRR in blue, and the least convergence time in green. "OOM" denotes that experiments run out of memory.

Dataset ($\|\mathcal{E}\|$)	Train Approach	#Parts (N)	Ratio (r)	Prep. Time (mins)	Test MRR (%)				Conv. Time (min)			
					GCN-M	GCN-D	RGCN-M	RGCN-D	GCN-M	GCN-D	RGCN-M	RGCN-D
Ecomm (207M)	RandomTMA	$\|\mathcal{V}\|$	0.33	0	84.12 ±0.02	79.94 ±0.34	33.17 ±0.95	50.73 ±2.13	52.5 ±20.0	41.4 ±4.3	66.5 ±11.4	106.9 ±11.5
	SuperTMA	15,000	0.76	6.1	84.44 ±0.45	81.53 ±0.08	36.22 ±2.79	52.84 ±3.32	126.3 ±12.8	105.2 ±17.5	33.4 ±1.2	132.3 ±12.8
	PSGD-PA	$M=3$	0.96	4.8	83.51 ±0.27	81.42 ±0.46	38.60 ±3.94	55.23 ±2.82	121.4 ±39.9	91.0 ±2.9	188.0 ±11.6	155.7 ±11.4
	LLCG	$M=3$	0.96	4.8	83.14 ±0.40	80.60 ±0.14	(OOM)	(OOM)	91.4 ±13.4	88.3 ±8.4	(OOM)	(OOM)
	GGS	-	1.00	0	82.13 ±0.42	81.35 ±1.33	(OOM)	(OOM)	87.4 ±0.3	199.3 ±30.8	(OOM)	(OOM)

$\mathcal{L}(y_v, z_v)$ as

$$\mathcal{L}(y_v, z_v) = \tfrac{1}{2}(\sigma(g(\mathbf{w})) - 1)^2. \tag{4}$$

Discrepancies Among Expected Initial Gradients. The gradient of the model weights $\mathbf{w}$ for training node v is

$$\nabla\mathcal{L} = \frac{\partial L}{\partial \mathbf{w}} = (\sigma(g(\mathbf{w})) - 1) \cdot \frac{\partial \sigma(g(\mathbf{w}))}{\partial g(\mathbf{w})} \cdot \frac{\partial g(\mathbf{w})}{\partial \mathbf{w}}. \tag{5}$$

As σ is the sigmoid function, we have

$$\frac{\partial \sigma(g(\mathbf{w}))}{\partial g(\mathbf{w})} = \sigma(g(\mathbf{w}))(1 - \sigma(g(\mathbf{w}))) \tag{6}$$

$$\frac{\partial g(\mathbf{w})}{\partial \mathbf{w}} = \sum_{u \in \mathcal{N}_{\mathcal{G}}(v)} \frac{1}{d_v} \mathbf{x}_u \tag{7}$$

Now we look into how the gradients change when we ignore the cross-partition edges under model aggregation training, which changes the *effective* neighborhood $\mathcal{N}'_{\mathcal{G}}(v)$ of node v in Eq. (3) and (7). Following the analyses in Proof 1, we can assume the feature distribution $\mathbf{C}_1$ and $\mathbf{C}_2$ in each partition as $\mathbf{C}_1 = [\beta, 1-\beta]$ and $\mathbf{C}_2 = [1-\beta, \beta]$, where $\beta \in [0, 1]$; the difference of the group distributions $\|\mathbf{C}_2 - \mathbf{C}_1\| = \sqrt{2}|1 - 2\beta|$.

① For *centralized training*, the effective neighborhood $\mathcal{N}'_{\mathcal{G}}(v)$ of node v is equal to its actual neighborhood $\mathcal{N}_{\mathcal{G}}(v)$. Thus, for $y_v = 1$, and the assumed node features $\mathbf{x}_v = \text{onehot}(y_v)$, we have

$$\mathrm{E}\left[\sum_{u \in \mathcal{N}_{\mathcal{G}}(v)} \frac{1}{d_v} \mathbf{x}_u\right] = \frac{1}{d_v}\begin{bmatrix}(1-h)d_v & hd_v\end{bmatrix} = \begin{bmatrix}(1-h) & h\end{bmatrix} \tag{8}$$

When initializing $\mathbf{w} = 0$, we have $g(\mathbf{w}) = 0$ and $\sigma(g(\mathbf{w})) = 0.5$. Combining Eq. (5)-(8), we have the expected initial gradient $\mathrm{E}[\nabla\mathcal{L}^{global}]$ for centralized training

as

$$\mathrm{E}[\nabla\mathcal{L}^{global}] = -\tfrac{1}{8}\begin{bmatrix}(1-h) & h\end{bmatrix} \tag{9}$$

② When v is on *instance 1* with class distribution $\mathbf{C}_1 = [\beta, 1-\beta]$, the effective neighborhood $\mathcal{N}'_{\mathcal{G}}(v)$ of node v is changed compare to its actual neighborhood. In this case, we have for $y_v = 1$

$$\mathrm{E}\left[\frac{\partial g(\mathbf{w})}{\partial \mathbf{w}}\right] = \frac{1}{d_v((1-h)\beta + h(1-\beta))} \cdot \begin{bmatrix}(1-h)\beta d_v & h(1-\beta)d_v\end{bmatrix}, \tag{10}$$

and when initializing $\mathbf{w} = 0$, we have the expected initial local gradient $\mathrm{E}[\nabla\mathcal{L}_1^{local}]$ on instance 1 as

$$\mathrm{E}[\nabla\mathcal{L}_1^{local}] = -\frac{1}{8((1-h)\beta + h(1-\beta))} \cdot \begin{bmatrix}(1-h)\beta & h(1-\beta)\end{bmatrix}. \tag{11}$$

③ When v is on *instance 2* with class distribution $\mathbf{C}_1 = [1-\beta, \beta]$, we have for $y_v = 1$

$$\mathrm{E}\left[\frac{\partial g(\mathbf{w})}{\partial \mathbf{w}}\right] = \frac{1}{d_v((1-h)(1-\beta) + h\beta)} \cdot \begin{bmatrix}(1-h)(1-\beta)d_v & h\beta d_v\end{bmatrix}, \tag{12}$$

and when initializing $\mathbf{w} = 0$, we have the expected initial local gradient $\mathrm{E}[\nabla\mathcal{L}_2^{local}]$ on instance 2 as

$$\mathrm{E}[\nabla\mathcal{L}_2^{local}] = -\frac{1}{8((1-h)(1-\beta) + h\beta)} \cdot \begin{bmatrix}(1-h)(1-\beta)d_v & h\beta d_v\end{bmatrix}. \tag{13}$$

Based on Eq. (9), (11), (13), we have the discrepancies measured under l^2-norm between these expected initial gradients as

$$\|\mathrm{E}[\nabla\mathcal{L}^{global}] - \mathrm{E}[\nabla\mathcal{L}_1^{local}]\|_2 = \frac{\sqrt{2}}{8}\left|\frac{(1-2\beta)(h-1)h}{\beta - 2\beta h + h}\right|$$

$$\|\mathrm{E}[\nabla\mathcal{L}^{global}] - \mathrm{E}[\nabla\mathcal{L}_2^{local}]\|_2 = \frac{\sqrt{2}}{8}\left|\frac{(2\beta-1)(h-1)h}{1-\beta+(2\beta-1)h}\right|$$

$$\|\mathrm{E}[\nabla\mathcal{L}_1^{local}] - \mathrm{E}[\nabla\mathcal{L}_2^{local}]\|_2 = \left|\begin{matrix}\frac{1}{4\sqrt{2}}(2\beta-1) \\ \frac{(h-1)h}{(\beta-2\beta h+h-1)} \\ (\beta-2\beta h+h)\end{matrix}\right|$$

Given that the difference of the group distributions $\|\mathbf{C}_2 - \mathbf{C}_1\| = \sqrt{2}|1-2\beta|$, it is straightforward to see from the above equations that (1) there is no discrepancy among all initial gradients when $\beta = 0.5$, and (2) the discrepancies increase with the increase of $\|\mathbf{C}_2 - \mathbf{C}_1\| = \sqrt{2}|1-2\beta|$ when $h \geq 0.5$.

Discrepancies Among Expected Loss Values. We only show the proof for node v with class label $y_v = 1$ here; the case of $y_v = 0$ can be proved similarly. Assume the model weight $\mathbf{w} = [w_0, w_1]$; based on Eq. (3), (4), (7), (10), and (12), we have for instance 1 and instance 2, when not considering cross-partition edges,

$$\mathrm{E}[\mathcal{L}_1^{local}(\mathbf{W})] = \left(1 + \exp\left(\frac{\beta(h-1)w_0 + (\beta-1)hw_1}{(2\beta-1)h - \beta}\right)\right)^{-2},$$

$$\mathrm{E}[\mathcal{L}_2^{local}(\mathbf{W})] = \left(1 + \exp\left(\frac{(\beta-1)(h-1)w_0 + \beta h w_1}{-\beta + (2\beta-1)h + 1}\right)\right)^{-2}.$$

Note that function $(1+\exp(x))^{-2}$ monotonically decreases with variable x, therefore $\mathrm{E}[\mathcal{L}_1^{local}(\mathbf{W})] = \mathrm{E}[\mathcal{L}_2^{local}(\mathbf{W})]$ if and only if

$$\frac{\beta(h-1)w_0 + (\beta-1)hw_1}{(2\beta-1)h - \beta} = \frac{(\beta-1)(h-1)w_0 + \beta h w_1}{-\beta + (2\beta-1)h + 1}. \tag{14}$$

Eq. (14) holds if and only if $\beta = 0.5$, which means $\|\mathbf{C}_2 - \mathbf{C}_1\| = \sqrt{2}|1-2\beta| = 0$. Therefore, the expected loss values $\mathrm{E}[\mathcal{L}_i^{local}(\mathbf{W})]$ on each instance $i \in \{1, 2\}$, without considering cross-partition edges, is equal if and only if $\mathbf{C}_1 = \mathbf{C}_2$. □