DAHA: Accelerating GNN Training with Data and Hardware Aware Execution Planning

Zhiyuan Li
The Hong Kong University of Science and Technology
zlicw@cse.ust.hk

Xun Jian*
The Hong Kong University of Science and Technology
xjian@cse.ust.hk

Yue Wang*
Shenzhen Institute of Computing Sciences
yuewang@sics.ac.cn

Yingxia Shao
Beijing University of Posts and Telecommunications
shaoyx@bupt.edu.cn

Lei Chen
Data Science and Analytics Thrust, The Hong Kong University of Science and Technology (Guangzhou)
leichen@cse.ust.hk

ABSTRACT

Graph neural networks (GNNs) have been gaining a reputation for effective modeling of graph data. Yet, it is challenging to train GNNs efficiently. Many frameworks have been proposed but most of them suffer from high batch preparation cost and data transfer cost for mini-batch training. In addition, existing works have limitations on the device utilization pattern, which results in fewer opportunities for pipeline parallelism. In this paper, we present DAHA, a GNN training framework with data and hardware aware execution planning to accelerate end-to-end GNN training. We first propose a data and hardware aware cost model that is lightweight and gives accurate estimates on per-operation time cost for arbitrary input and hardware settings. Based on the cost model, we further explore the optimal execution plan for the data and hardware with three optimization strategies with pipeline parallelism: (1) group-based in-turn pipelining of batch preparation neural training to explore more optimization opportunities and prevent batch preparation bottlenecks; (2) data and hardware aware rewriting for intra-batch execution planning to improve computation efficiency and create more opportunities for pipeline parallelism; and (3) inter-batch scheduling to further boost the training efficiency. Extensive experiments demonstrate that DAHA can consistently and significantly accelerate end-to-end GNN training and generalize to different message-passing GNN models.

PVLDB Reference Format:
Zhiyuan Li, Xun Jian, Yue Wang, Yingxia Shao, and Lei Chen. DAHA: Accelerating GNN Training with Data and Hardware Aware Execution Planning. PVLDB, 17(6): 1364 - 1376, 2024.
doi:10.14778/3648160.3648176

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/fr8nkL/DAHA.

*Xun Jian and Yue Wang are the corresponding authors.

Proceedings of the VLDB Endowment, Vol. 17, No. 6 ISSN 2150-8097.
doi:10.14778/3648160.3648176

1 INTRODUCTION

Graph Neural Networks (GNNs) [19] have gained great popularity due to their success in solving problems in graph-structured data like network embedding [22] and recommendation [31]. Training GNNs efficiently has long been a challenge and hot research topic. The most popular approach is the class of mini-batch sampling GNNs [4–7, 12, 15, 33, 34]. Many dedicated frameworks like PyG [8] and DGL [28] have been developed for training them on GPUs because traditional deep learning systems cannot efficiently process the graph operations in GNNs.

Most of the GNN frameworks share the same device utilization pattern where CPU is responsible for producing sampled batches and slicing out the feature vectors while GPU receives the batch data and performs GNN training. Under such setting, however, a slow production of sampled batches on CPU can easily block GPU training [37]. In addition, due to the unique nature of GNNs and graphs, the CPU-GPU data transfer cost for GNNs is significant [10] considering the possible neighbor explosion [34] and the large feature dimension. Many works [10, 16, 21, 24, 26] have also reported that (1) **batch preparation** (including sampling) and (2) **CPU-GPU data transfer** are the two bottlenecks in GNN training. They can lead to poor GPU utilization because GPU depends on the batch preparation and CPU-GPU data transfer to perform its work and these two bottlenecks block the dependent operation.

To address this, DistDGLv2 [38] and SALIENT [16] propose novel pipelining strategies to overlap batch preparation and data transfer with computation to hide the cost. Despite their efforts, without changing the classic device utilization pattern of CPU-sample and GPU-train, a slow sampler or a long CPU-GPU data transfer operation will always block the following dependent GPU operation. Recent works begin to explore alternative device utilization patterns. ByteGNN [37] and GNNLab [32] propose dedicated optimizations for CPUs and GPUs, respectively. But their utilization patterns of the other hardware resource are limited, leaving room for further improvement. Kim et al. [18] propose to perform partial aggregation of on-device data on both CPU and GPU. However, it might encounter CPU bottlenecks and it is still limited to CPU-only sampling. The smallest unit to schedule in most literature is the entire computation or communication of one batch. They seldom consider breaking down batch processing into fine-grained

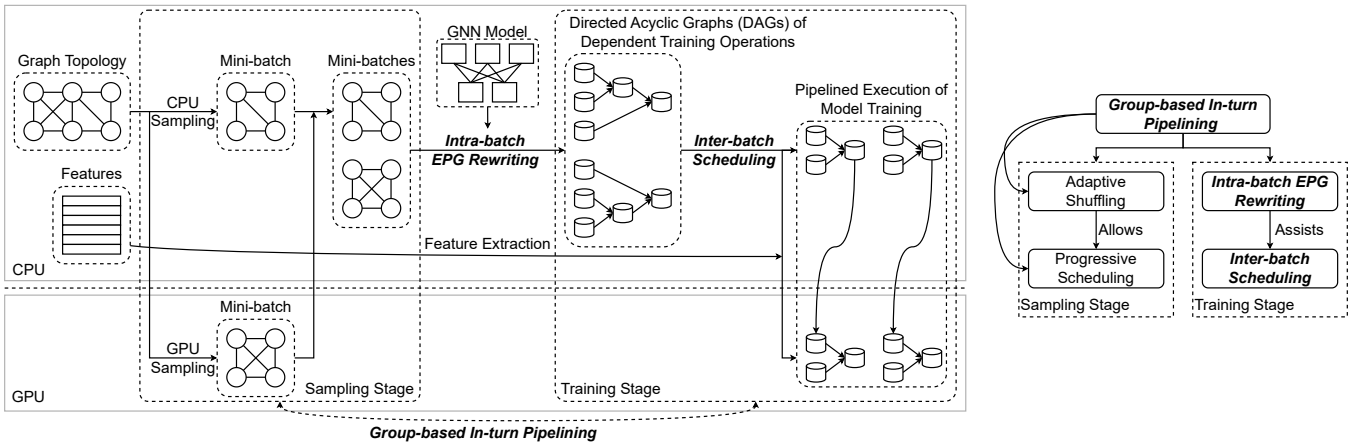


Figure 1: The overview of DAHA with three novel modules: (1) group-based in-turn pipelining; (2) intra-batch execution plan graph (EPG) rewriting; and (3) inter-batch scheduling. Module 1 sets the overall pipeline style of DAHA with interleaving sampling and training stages. Module 2 decides the search space for module 3 to optimize for accelerating the training stage.

operations. Hence, they miss optimization opportunities for more fine-grained pipeline parallelism.

To conclude, we identify the research gap of failure to explore all possible combinations of device utilization patterns on fine-grained sets of operations, hence missing opportunities for the combinatorial optimization problem of finding the optimal execution plan. We aim to fill the gap by expanding the efficiency optimization search space to fully hybrid CPU-GPU, where each fine-grained operation of a batch can reside on any feasible device, to find the optimal execution plan based on the input data and hardware like [30].

To begin with, we need to answer a fundamental question, what operation and data are suitable for what device? Although it is often assumed that GPU training is faster than CPU, we find GPUs might be worse when the GPU speedup of computation cannot offset the additional data transfer cost. To answer the question, a quantitative cost model is needed to estimate the time of an operation on arbitrary input data and devices. We build a data and hardware aware cost model for both data transfer and major GNN operations, which takes the input of the graph property and model specifications to output the expected training efficiency based on the available hardware.

Based on the cost model, we present DAHA, a GNN training framework with data and hardware aware execution planning to accelerate end-to-end GNN training. The overview of DAHA is illustrated in Figure 1. DAHA consists of three modules all guided by the data and hardware aware cost model. They are designed for fully hybrid CPU-GPU pipeline parallelism to leverage the hardware resources efficiently. The three modules are: (1) group-based in-turn pipelining; (2) intra-batch execution plan graph (EPG) rewriting; and (3) inter-batch scheduling. When GNN training starts, we first apply the group-based in-turn pipelining strategy, where DAHA in turn pipelines the sampling stage first and then switches to the model training stage. The idea is to group tasks with similar patterns of data access and device utilization to increase data and device affinity, amortizing the cost of memory access and kernel launching. It prevents slow samplers from blocking trainers because samplers are pipelined together followed by the pipeline of trainers. With group-based in-turn pipelining, DAHA goes fully hybrid CPU-GPU on sampling and training stages, diversifying the device utilization pattern for more optimization opportunities. DAHA also performs adaptive shuffling and progressive scheduling to efficiently pipeline the sampling stage. After the sampling stage, DAHA applies the intra-batch EPG rewriting and inter-batch scheduling optimizations to accelerate the training stage at a fine-grained operation level with more optimization opportunities. The cost model helps estimate the makespan of each operation to find the optimal execution plan. To this end, we present the comparison of DAHA to literature in Table 1 to illustrate how DAHA identifies and fills the research gap.

In summary, we make the following contributions in this paper: (1) **Problem exploration**. We identify the gap of device utilization pattern and hardware selection problem in GNN training and provide a new perspective to accelerate GNN training with fully hybrid CPU-GPU pipeline parallelism. (2) **Data and hardware aware cost model**. We provide a data and hardware aware cost model to estimate the time cost of an operation to automatically help schedule the input (sub)graphs on the available hardware. (3) **GNN training framework with novel execution planning**. We separate the sampling stage and neural training stage and pipeline them independently to increase data and device affinity, amortizing the cost of memory access and kernel launching and preventing sampling from blocking training. Based on this design which opens up more pipeline parallelism opportunities, we propose a series of novel intra-batch and inter-batch execution planning strategies to accelerate GNN training in the hybrid CPU-GPU setting. (4) **Remarkable performance.** We perform extensive experiments to evaluate DAHA to show its significant speedup and ability to generalize to different data and message-passing GNNs.

The outline of the paper is as follows. Problem statement is in Section 2. Section 3 covers the detailed description of DAHA and its modules. We present the experimental results in Section 4 and related works in Section 5. Section 6 concludes the paper.

Table 1: Comparison of DAHA and literature

Work	CPU util	GPU util	Result
DGL [28]	Sample	Train	Sampling & data transfer become bottlenecks
DistDGLv2 [38]	Async Sample	Train	Possible sampling bottleneck, coarse-grained pipeline
SALIENT	Fast sample	Train	Possible sampling bottleneck, coarse-grained pipeline
ByteGNN [37]	Sample & train	N.A.	Coarse-grained inter-batch pipelining, no GPU speedup
GNNLab [32]	Train	Sample & train	Coarse-grained inter-batch pipelining, limited CPU utilization
Kim et al. [18]	Sample & train	Train	Possible CPU bottleneck, miss pipeline opportunity
DAHA	Sample & train	Sample & train	Fine-grained intra- & inter-batch pipelining, no blocking bottleneck

2 PRELIMINARIES

We introduce related concepts and give our problem statement.

Graphs. Let $G = (V, E, X, Y)$ be an undirected, featured and labelled graph, where V is the set of nodes, $E \subseteq V \times V$ is the set of edges, $X \in \mathbb{R}^{n \times f}$ is the set of node features and Y is the set of node labels. We also refer to a graph as $G = (V, E)$.

Graph neural networks. Let $A \in \mathbb{R}^{n \times n}$ be the adjacency matrix of a graph or a subgraph as a mini-batch $G = (V, E, X, Y)$, I be the $n \times n$ identity matrix, D be the diagonal matrix of modified vertex degrees, and $\sigma(\cdot)$ be the activation function. Let $\hat{A} = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}$. Then the formula for one forward layer of GCN [19] is

$$H^l = \sigma(\hat{A}^T H^{l-1} W^l), \tag{1}$$

where H^l is the dense embedding matrix of the nodes and W^l is the dense weight matrix at the l-th layer. $H^0 = X \in \mathbb{R}^{n \times f}$. The final representation matrix is H^L where L is the number of GNN layers. For simplicity, we refer to $\hat{A}$ as A as well.

Problem Statement. In this work, we focus on devising novel execution planning strategies aware of the nature of the underlying graph data and hardware to accelerating message-passing GNN training. We formulate the problem as below:

Given a graph G, a GNN model $\mathcal{A}$, a set of batches (jobs) $\mathcal{B} = \{\mathcal{B}_1, ..., \mathcal{B}_n\}$ and available hardware resources (machines) $M = \{M_1, ..., M_m\}$, determine the optimal execution plan of training $\mathcal{A}$ with input $\mathcal{B}$ on devices M that minimized the makespan.

Full-graph training GNNs also fit in the formulation since the full graph can be regarded as a batch. We further model the problem in two stages: the sampling stage and the training stage. In the sampling stage, our problem can be modeled as a uniform-machines scheduling problem [3]. In the training stage, we have two levels: intra-batch and inter-batch. At the intra-batch level, by considering the dependent graph and neural operations of each batch as a sequence of directed acyclic graphs (DAG), the problem can be modeled as an EPG rewriting problem [17]. At the inter-batch level, we need to arrange the pipeline of multiple mini-batches, which can be modeled as a flexible job-shop scheduling problem [3].

3 THE DATA AND HARDWARE AWARE EXECUTION PLANNING

We first elaborate on our data and hardware aware cost model in Section 3.1. Section 3.2 discusses the three modules of DAHA.

3.1 A Data and Hardware aware Cost Model

DAHA learns a data and hardware aware cost model to measure (1) the CPU and GPU neural computation time and (2) CPU-GPU data transfer time of various input data and underlying hardware.

For the neural computation time, since major neural operations involved in GNNs can typically be classified as dense matrix multiplications (mm) and sparse matrix-matrix multiplications (spmm), it suffices to build the computation cost model with these two neural operations. Following the abstraction by DGL [28], we regard each GNN layer as a message flow graph (MFG), which can be viewed as a special bipartite graph where each directed edge from the source node to the destination node represents one propagation in this GNN layer. Then, one batch computation can be regarded as a list of MFGs, each representing the computation process in one GNN layer. Based on the abstraction, we identify two vital variable factors from input data: (1) the number of distinct nodes in the original graph involved in MFG_i denoted by n_i; and (2) the number of edges in MFG_i denoted by m_i. Since the major neural operations involved are spmm and mm whose time complexity is at most cubic, DAHA abstracts the computation cost model as a multivariate cubic regression on n_i and m_i, that is,

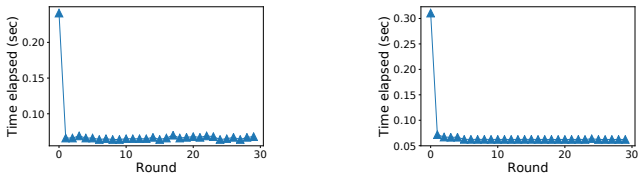
$$\begin{aligned} compCost(MFG_i) = \beta_0 + \beta_1 n_i + \beta_2 m_i + \beta_{11} {n_i}^2 + \beta_{22} {m_i}^2 + \\ \beta_{12} n_i m_i + \beta_{111} {n_i}^3 + \beta_{222} {m_i}^3 + \beta_{112} {n_i}^2 m_i + \beta_{122} n_i {m_i}^2. \end{aligned} \tag{2}$$

Here in Eq.3, the regression parameters $\{\beta_j\}$ reflect the GNN model parameters including the hidden dimension and the underlying hardware performance. This data and hardware aware formulation allows lightweight instantiation of the cost model and swift adaptation to various input data and underlying hardware. To estimate the per-operation computation cost, we just need to learn another set of regression parameters.

The per-MFG cost model can be generalized to batch-level. The total computation time of one batch $\mathcal{B}_j$ consisting of a list of MFGs is predicted by the sum of the time cost of these MFGs, that is,

$$compCost(\mathcal{B}_j) = \sum_{MFG_i \in \mathcal{B}_j} compCost(MFG_i). \tag{3}$$

For the data transfer time, DAHA follows the similar approach with some special amendments. We find that the first CPU-GPU data transfer operation typically takes a magnitude larger amount of time than the following rounds, which means there exists a remarkable fixed initialization cost for this communication operation. This is evidenced by Figure 2 which reports the CPU-GPU transfer time



(a) M1. (b) M2.

Figure 2: Communication overhead is significantly larger in the first round.

of the feature matrix of Reddit for each round (details of machines M1 and M2 are in Section 4). To highlight this fact, we formulate the communication cost model in two parts, fixed and variable cost. The fixed cost is subjected to the hardware environment which can be captured by launching artificial CPU-GPU data transfer operations. The variable cost modeling follows the approach in the computation cost model. It mainly consists of two parts as well, the dense and sparse matrices. For the dense node and edge features, we use n_j, m_j, f_n and f_m as input variables to the cost model, where n_j and m_j are the number of nodes and edges in $\mathcal{B}_i$, and f_n and f_m are the dimensions of node and edge features, respectively. For the sparse adjacency matrix, n_i and m_i suffice to capture the communication volume. Hence, DAHA models the variable communication time cost as a polynomial function on $\{n_j, m_j, f_n, f_m\}$ and the amortized communication cost of one MFG is abstracted as

$$\begin{aligned} commCost(\mathcal{B}_j) &= \frac{\alpha}{N} + Poly_d(n_j, m_j, f_n, f_m) + Poly_s(n_j, m_j) \\ &= \frac{\alpha}{N} + Poly(n_j, m_j, f_n, f_m), \end{aligned} \tag{4}$$

where α is the fixed initialization cost and N is the total number of batches in the training process. $Poly_d$ and $Poly_s$ are the polynomial functions for estimating the variable cost of dense matrices and sparse matrices, respectively.

With the computation and communication cost models, by learning a dedicated set of regression parameters for an arbitrary operation on an arbitrary device, DAHA can estimate the cost of an arbitrary sequence of operations on arbitrary input data of any level. It guides the execution planning for pipeline parallelism in the training stage given the batch statistics.

3.2 Hybrid CPU-GPU Optimizations

If the expected GPU execution is faster than CPU, then the GPU speed gain could compensate for the data transfer cost, thus GPU should be involved in the optimal execution plan. DAHA presents three major aspects for hybrid CPU-GPU optimizations.

3.2.1 Group-based in-turn pipelining of the batch preparation stage and neural training stage. The traditional approach to pipelining mini-batch GNN training mixes batch preparation stage and neural training stage by assigning CPU to sampling and GPU to learning, respectively. DAHA proposes a novel perspective for GNN pipelining by in turn grouping multiple batch preparation stages with all available hardware resources and performing neural training on these batch groups, as illustrated in Figure 3. Batches across different epochs can also be grouped together in DAHA. Take Figure 3 as an example. DAHA adopts a group size of two epochs and schedules the batch preparation for the entire graph for two epochs with all available CPUs and GPUs followed by the neural training of these 2-epoch batches. Note that DAHA only stores the subgraph structural information of the batches at the batch preparation stage so that the feature information is not duplicated to avoid redundant memory usage. By grouping tasks with the same nature, more tasks sharing similar patterns of data access and device utilization are processed together. This can increase data and device affinity and amortize the cost of memory access and kernel launching. As a result, by processing more tasks of the same nature together, DAHA reduces the per-batch cost due to the amortization. This is an analogy to economies of scale which refers to the unit production cost advantage experienced by a company when it increases its production level of output. Despite the amortization effect, when too many tasks are grouped together, which means too many batches are produced together, it will increase the time cost to schedule the execution and impose pressure on device capacity of both storage and computation power, lowering the benefit of the grouping strategy. This is an analogy to diseconomies of scale when a company grows so giant that the per-unit cost increases with the production level.

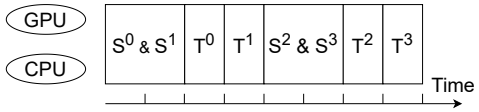


Figure 3: Group-based in-turn pipeline. S^i and T^i mean the sampling and training stages of i-th epoch, respectively.

In order to effectively leverage all the available hardware to schedule both the batch preparation and neural training stages, DAHA needs to estimate the batch preparation (sampling) time and the statistics of the involved MFGs of an arbitrary batch to distribute the workloads to different devices. An accurate and efficient estimation scheme is not trivial due to the randomness brought by different batch sampling algorithms, the complexity of overlapping neighborhoods of different target nodes, and the uncertainty of target nodes in a batch rising from shuffling. To address these challenges, DAHA proposes two strategies: (1) adaptive shuffling and (2) progressive scheduling.

Adaptive shuffling. Though seldom explored in literature, we find in experiments that shuffling has a substantial influence on both the effectiveness of the model and the efficiency of the training process. Briefly speaking, enabling shuffling could strengthen the generalization of the model, typically resulting in higher test accuracy of more than 1%. Meanwhile, it can slow down the end-to-end training speed by up to 10%. To strike a balance between effectiveness and efficiency and pave the way for a reliable estimation scheme of the batch preparation time and statistics of the involved MFGs, DAHA proposes an adaptive shuffling strategy. Similar to the idea of transfer learning, DAHA disables shuffling in the initial epochs, which serves as a pre-training step with better efficiency. Since shuffling is disabled, the uncertainty of target nodes in a batch is eliminated so that DAHA can apply forecasting methods

for estimation, as covered in our progressive scheduling strategy. When the accuracy gain becomes insignificant, DAHA concludes that a decent "pre-trained model" is retrieved and then switches to enable shuffling to inherit the stronger generalization ability as a fine-tuning step. With shuffling, however, it is difficult to saturate the sampling pipeline with static scheduling due to the uncertainty of shuffling. As a remedy, DAHA dynamically fills gaps in the sampling pipeline with neural operations whose input data enjoys fine locality to the idle devices.

To find a proper switching point of the shuffling mode, DAHA monitors the model convergence plot and automatically detects the elbow point of the plot, which serves as the switching point. It strikes a balance between the effectiveness of the "pretrained model" and the end-to-end efficiency since there will not be significant accuracy gain after the elbow point. After the switching point, it just requires a few more epochs to converge with the "pre-trained model" and "shuffle on" mode.

Progressive scheduling. With "shuffle off" mode, DAHA formulates the estimation of the batch sampling time and statistics of the involved MFGs as a forecasting problem. Based on the formulation, DAHA proposes a progressive scheduling framework. At the beginning of GNN training, DAHA splits the sampling requests evenly on the available devices to obtain the initial observations of the sampling time and statistics of the involved MFGs for forecasting in the following runs. Since the targeted variables typically vary alongside a median value and no obvious seasonal trend is observed in our experiments, DAHA uses the moving median method to ensure effective and efficient estimation.

In the first round to obtain the initial observations, since GPU typically enjoys faster speed and finishes earlier than CPU, DAHA again dynamically fills in some proper neural operations as in the "shuffle on" mode to saturate the possible idle devices to achieve a higher utilization rate.

In later rounds, with the progressively refined estimations obtained from the forecasting method, DAHA performs the scheduling of the sampling workloads on the CPU and GPU workers. We define the problem as follows. We are given n batches to sample to be scheduled on either CPU or GPU and their processing time is given by the forecasting estimates. Assume that GPU sampling speed is uniformly faster than CPU, that is, there exists a speed factor s such that for an arbitrary batch $\mathcal{B}_i$, the sampling time of $\mathcal{B}_i$ on GPU equals its sampling time on CPU divided by s. Our goal is to determine the optimal execution plan that minimizes the makespan. This problem can be modeled as a uniform-machines scheduling problem. It has a FPTAS running in $O(n/\epsilon^2)$ time which yields at most $1+\epsilon$ times the optimal makespan [13]. Since DAHA leverages all available devices, the objective of minimum makespan also ensures balanced workload and high utilization rates of all devices because otherwise the makespan can be further shortened by assigning some additional tasks to the idle devices. Also, because we have static estimations of the sampling time of the batches, the inputs to the scheduling problem remain the same. Hence, we just need to solve it once for all following runs.

3.2.2 Data and hardware aware rewriting rules for intra-batch execution plan graphs. On top of the novel group-based in-turn pipelining strategy, DAHA proposes both intra-batch and inter-batch neural training optimizations to better tailor the hardware to the input data. Since the entire statistics of the batch are available before the neural training stage due to the group-based in-turn pipeline, with the data and hardware aware cost models in Eq.2-4, DAHA can perform effective and efficient execution planning for both intra-batch level and inter-batch level scheduling for every epoch. We first present the intra-batch strategy.

Abstraction as execution plan graph (EPG). As in Section 1, existing works have some limitations in the roles of hardware with a tendency to constrain CPUs to graph operations and GPUs to neural operations. To explore more optimization opportunities by considering both CPU and GPU as a general-purpose processing unit, DAHA views the GNN training as matrix multiplications and a typical forward propagation can be abstracted by

$$Z^{l+1} = AH^lW^l,\ H^{l+1} = \sigma(Z^{l+1}), \tag{5}$$

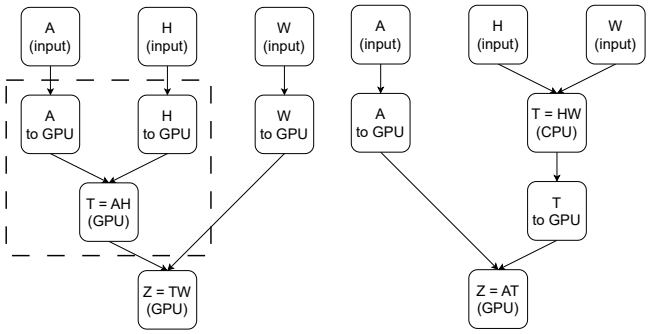
where A is the sparse modified adjacency matrix, H^l is the dense node embedding matrix and W^l is the dense weight matrix at the l-th layer. DAHA then abstracts the execution of the matrix multiplication as an execution plan graph (EPG) which is a directed acyclic graph (DAG) capturing the dependency of the intra-batch computation. In an EPG, each node represents an operation and each edge captures the dependency of the operations.

For example, the EPG of the classic pure GPU execution is shown in Figure 4a. For simplicity of illustration, only the execution plan of $Z^{l+1} = AH^lW^l$ is shown. The three nodes at the top whose in-degree is zero are the roots of the DAG. They represent the inputs for execution. They are then transferred to GPU for further processing, represented by the "to GPU" nodes. Since the actual inputs are needed before the data transfer, there exists an edge from each input node to the corresponding "to GPU" node, representing the dependency. Similarly, the matrix A and H need to be transferred to GPU before their multiplication on GPU can be executed, represented by the area of the dashed rectangle. Finally, the desired output is retrieved by the bottom node.

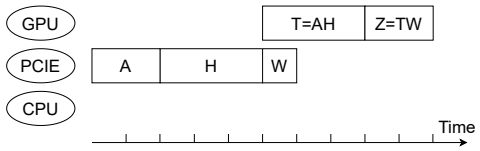
To better illustrate the device utilization of the classic pure-GPU execution plan, Figure 4c gives a per-device execution plot. Since the spmm of $T = AH$ is the first dependent GPU operation, the PCIE bandwidth for CPU-GPU data transfer places a higher priority on the matrix A and H than W. After A and H are ready in GPU, the spmm of $T = AH$ can start along with transferring W. After both finish, GPU can then perform the mm of $Z = TW$. As shown in Figure 4c, such an execution plan has poor efficiency on intra-batch device utilization since only the relatively lightweight operation of transferring W can be pipelined with the spmm of $T = AH$.

Neural training with hybrid processing units (HybridPU). By rewriting the EPG, various candidate execution plans with hybrid utilization patterns of both CPU and GPU can be explored for optimization. Based on the EPG abstraction, the available batch statistics as well as our data and hardware aware cost model, DAHA proposes a novel HybridPU strategy that rewrites the pure-GPU EPG to retrieve an efficient execution plan of the neural training operations with a hybrid CPU-GPU setting and renders more opportunities for pipeline parallelism.

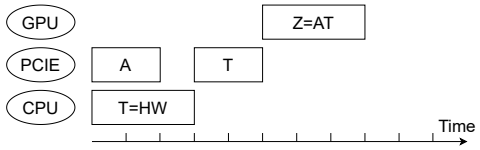
Take the execution of $Z^{l+1} = AH^lW^l$ as example again, Figure 4b and 4d show a candidate rewriting rule. It performs the mm of $T = HW$ on CPU pipelined with the data transferring of A. After



(a) EPG: pure GPU. (b) EPG: HybridPU.



(c) Device: pure GPU.



(d) Device: HybridPU.

Figure 4: Pure GPU vs HybridPU.

the mm finishes, it transfers the mm output to GPU for the spmm of $Z = AT$ to follow. Since the neural transformation involving the weight matrix W is performed on CPU ahead of others, we call it the **pre-transformation** rewriting rule. The advantages of such a HybridPU strategy are twofold. Firstly, it renders more opportunity for pipeline parallelism. As shown in Figure 4d, although the execution of the mm operation is slower on CPU than GPU, it can be well pipelined with CPU-GPU data transfer of A, hiding the communication cost behind the computation cost. Secondly, not only the number of operations but also the communication volume is reduced. The reason is that the pre-transformation on CPU just needs to transfer the mm output to GPU instead of the two input matrices and the size of the mm output $T = H^0W^0$ is often smaller than the input feature matrix H^0 since the hidden dimension is typically smaller than the feature dimension. As a result, the pre-transformation rewriting rule serves as a quality candidate. The pre-transformation is especially effective for forward propagation. Hence, it also helps accelerate inference.

Another candidate performs the spmm of $T = AH$ on CPU followed by the mm $Z = TW$ on GPU, which we call the **pre-aggregation** rewriting rule. Pre-aggregation is typically outperformed by pre-transformation if we only consider the execution of a single batch or sequential execution of multiple batches. The reason is that pre-aggregation performs the more computation-intensive spmm operation on CPU while GPU speedup for spmm is typically much more significant than mm, as shown later in Table 6. However, when we consider scheduling and pipelining multiple batches later, various rewritten EPGs can be useful in different cases because different resources can be idle at different periods. For example, in Figure 4d, after the mm $T = HW$ on CPU of batch 1 finishes, batch 2 can use pre-aggregation so the spmm $T = AH$ of batch 2 can continue on CPU without waiting for gradient update.

To decide whether a rewriting rule should be adopted and how much acceleration is expected, DAHA leverages the batch statistics and our data and hardware aware cost model to estimate the time cost. Notably, this intra-batch level optimization can be applied to full-graph training like GCN since the full-graph could be regarded as a batch. This concludes the data and hardware aware intra-batch EPG rewriting module of DAHA.

3.2.3 Inter-batch scheduling optimizations. DAHA combines its intra-batch optimization with inter-batch scheduling to boost device utilization and improve end-to-end training efficiency.

Problem formulation. Given a set of batches, available CPU workers, GPU workers, and PCIE bandwidth for CPU-GPU data transfer. Each batch consists of a series of dependent operations that can be processed in any order if the dependency is not broken. Each computation operation in a batch can be processed on any CPU or GPU worker. Each communication operation consumes the PCIE bandwidth. Given the per-operation estimates from the cost model, our goal is to determine the schedule of the operations on the available hardware that yields the minimum makespan.

This formulation takes into consideration the intra-batch EPG rewriting as well because the intra-batch operations can take place in an arbitrary order that does not break the dependency. Thus, the inter-batch scheduling optimizations of DAHA naturally inherit the benefits of its intra-batch strategies. Besides, there is few risk of GPU waiting due to synchronization because DAHA can automatically allocate the idle GPU resources to suitable operations and bounded staleness is applied for more pipeline opportunities.

Hardness of the problem. DAHA's scheduling problem can be reduced from a variant of the flexible job-shop scheduling problem (FJSP) which is NP-Hard [2]. The inputs to FJSP are a set of jobs and a set of machines. Each job consists of a sequence of operations processed in such an order that one precedes another. Each operation can be processed on any machine of a given type and the duration is known. The goal of the problem is to determine the schedule that gives the minimum makespan. It is well known that FJSP is not only NP-Hard but also intractable as it is considered one of the most computationally stubborn problems [2] and existing approximation solutions might also fail to have a logarithmic performance guarantee [3, 11]. The possibly many intra-batch EPGs further add to the hardness of the problem.

Proof of hardness. We prove that FJSP is reducible to our problem. Consider a FJSP with three input machines namely CPU, PCIE, and GPU. Each job consists of three operations, CPU computation, PCIE communication and GPU computation. These three operations should be processed in the above order on the stated machine. Then if there exists an solution to DAHA's inter-batch scheduling problem, then we can just fix the intra-batch EPG as CPU-PCIE-GPU and give the scheduling plan. This will minimize the FJSP's makespan as well. This finishes the proof.

Algorithm 1 Bi-level EM-based Scheduling Algorithm.

Input : $\mathcal{M} = \{\mathcal{M}_1, ..., \mathcal{M}_m\}$: available processing units; $\mathcal{B} = \{\mathcal{B}_1, ..., \mathcal{B}_n\}$: batches to schedule; $\{\mathcal{R}_1, ..., \mathcal{R}_r\}$: EPG rewriting rules; $compCost_{M_i}(\cdot)$ and $commCost(\cdot)$: computation and communication cost models; μ^*: a parameter in $(0, 1)$ controlling the grouping quality; α: a rate controlling the increase of grouping quality; β: maximum size of a batch group; τ: maximum iterations.

Output : An execution plan of the batches on the processing units.

Initiate a grouping $\mathcal{G} = \{\mathcal{G}_i\}$ randomly that partitions $\mathcal{B}$;
$f \leftarrow$ a list of zeros with length $|\mathcal{G}|$;
$\mu, iterCount \leftarrow 0$;
while $iterCount < \tau$ *and* $\mu < \mu^*$ **do**
  $\mu \leftarrow \mu + \alpha$; $iterCount \leftarrow iterCount + 1$;
  foreach $\mathcal{G}_i \in \mathcal{G}$ **do**
    $f[i] \leftarrow 0$;
    Apply the candidate EPG rewriting rules $\{\mathcal{R}_1, ..., \mathcal{R}_r\}$ to $\mathcal{G}_i$ to list the feasible execution plans of the group;
    Apply $compCost_{M_i}(\cdot)$ and $commCost(\cdot)$ to estimate the makespan of each execution plan;
    Determine the execution plan $\mathcal{P}_i$ that minimizes the makespan and the minimum makespan C_i;
    if $C_i > (1 - \mu) \cdot \sum_{\mathcal{B}_j \in \mathcal{G}_i} cost_{\mathcal{P}_i}(\mathcal{B}_j)$ **then**
      $f[i] \leftarrow 1$;
  $\Delta\mathcal{B} \leftarrow \{\mathcal{B}_j : \mathcal{B}_j \in \mathcal{G}_i \text{ and } f[i] = 1\}$;
  foreach $\mathcal{B}_j \in \Delta\mathcal{B}$ **do**
    $k \leftarrow \arg\min_i cost_{R_i}(\mathcal{B}_j)$;
    $y_j \leftarrow \frac{compCost_{R_k}(\mathcal{B}_j)}{commCost_{R_k}(\mathcal{B}_j)}$;
  $\Delta\mathcal{G}^* \leftarrow \arg\min_{\Delta\mathcal{G}} \max_{\Delta\mathcal{G}_i} |\log(\prod_{\mathcal{B}_j \in \Delta\mathcal{G}_i} y_j)|$ under the constraint that $|\Delta\mathcal{G}_i| \le \beta$ for $\forall \Delta\mathcal{G}_i \in \Delta\mathcal{G}$, where $\Delta\mathcal{G} = \{\Delta\mathcal{G}_i\}$ is a grouping that partitions $\Delta\mathcal{B}$ and $|\Delta\mathcal{G}_i|$ is the number of batches in the group $\Delta\mathcal{G}_i$;
  $\mathcal{G} \leftarrow \{\mathcal{G}_i : f[i] = 0\} \cup \Delta\mathcal{G}^*$;
Determine the execution plan $\mathcal{P}_i$ that minimizes the makespan for each $\mathcal{G}_i$;
return $\mathcal{G} = \{\mathcal{G}_i\}$ and $\{\mathcal{P}_i\}$;

Algorithm overview. To plan the intra-batch and inter-batch execution efficiently online, DAHA proposes in Algorithm 1 a heuristic bi-level EM-based strategy leveraging the grouping idea [9]. The idea behind is that, due to resource limitation and task switching cost, only a couple batches might be scheduled simultaneously and interleaving. Hence, we can first focus on planning the execution of a relatively small number of batches according to the number of available workers, which is regarded as a batch group, then switch to another batch group in sequence. This can effectively reduce the search space. With the motivation, DAHA formulates the bi-level strategy as below. Denote a set $\mathcal{G} = \{\mathcal{G}_i\}$ that partitions the batches $\mathcal{B}$ as a grouping of $\mathcal{B}$. For example, $\{\{a, b\}, \{c\}\}$ is a grouping of $\{a, b, c\}$. At the lower level, since a grouping $\mathcal{G}$ is given and the search space is well reduced, DAHA optimizes the execution planning of each batch group by leveraging valid EPG rewriting rules inherited from its intra-batch strategies. At the upper level, DAHA then evaluates the pipeline efficiency of the optimal execution plan for each group and optimizes the grouping of the batches in the inefficient groups. The regrouped batches are then sent to lower-level optimization, which makes the entire optimization process bi-level and recursive like an Expectation-Maximization (EM) algorithm. The details are in Algorithm 1.

Algorithm detail. DAHA first randomly initiates a grouping that partitions the batches to be scheduled. It then performs the lower-level optimization with the help of the candidate EPG rewriting rules and the cost models to estimate the end-to-end makespan of each execution plan and choose the optimal intra-batch EPGs considered together with the inter-batch scheduling. With each batch group optimized, DAHA identifies the groups that fail to meet the performance expectation. The criteria for a failed group is that its optimal makespan is greater than $1 - \mu$ times the sum of the optimal makespan of all its belonging batches being pipelined alone. The criteria justifies that pipelining the batches together cannot harvest enough efficiency gain compared to pipelining them separately. Therefore, the grouping of these batches needs to be re-considered. Note that with a larger value of μ, DAHA expects to achieve a better grouping result since the criteria become stricter. Therefore, DAHA uses an increasing μ to progressively improve the grouping result. Another benefit of the adaptive μ is that it could help break some local optima by recursively changing the grouping. Further, since the criteria progressively become stricter, the expected number of batch groups to be modified is relatively small, which means better efficiency for the upper level optimization. Now with the target batches to be re-grouped, DAHA aims to update their grouping with the objective to balance the computation cost and communication cost of each group under the constraint that the makespan of each group is under control. The update objective is to identify batches that have a high potential to be pipelined efficiently with their computation cost and communication cost well overlapped. Such batches are expected to form a quality batch group. The additional constraint is to control the makespan and size of the batch group for better efficiency in determining the optimal execution plan at the lower level. To achieve the update objective under the constraint, DAHA maps each batch to y, its ratio of communication cost against computation cost under the optimal pipelined execution plan. Since we want a batch group with well-overlapped computation and communication, the desired condition is that the cumulative product of the y values of the batches in the group should be as close as to 1. However, simply using $z = |1 - \prod_{\mathcal{B}_j \in \Delta\mathcal{G}_i} y_j|$ could result in a bias between over-computation and over-communication because the range of z in terms of over-computation is $(0, +\infty)$ while that of over-communication is $(0, 1)$. Therefore, DAHA adopts the objective function of $z = |\log(\prod_{\mathcal{B}_j \in \Delta\mathcal{G}_i} y_j)|$, which has a balanced range for both cases hence no bias towards any side. To perform the update, we use a greedy approach for approximation. We maintain a sorted array of the batches in terms of y. We then calculate the z values of the batches with the maximum and minimum y values. Then we find the batch with y value closest to $\frac{1}{z_{max}}$ and combine these two batches into a batch group. Regard the batch group as a new batch, we calculate its y value and insert it back to the sorted array. When the number of batches in a batch group exceeds β, we remove it from the sorted array. The process terminates when the

Table 2: Dataset statistics. "#" means the number of. Vol_G and Vol_F are the size of the sparse adjacency matrix and dense feature matrix in MB, respectively.

Dataset	#Nodes	#Edges	#Feat.	#Class	Vol_G	Vol_F
Cora	2708	10556	1433	7	0.403	14.803
Citeseer	3327	9104	3703	6	0.350	46.997
Pubmed	19717	88648	500	3	3.382	38.000
CS	18333	163788	6805	15	3.125	476.000
Physics	34493	495924	8415	5	9.459	1108.000
Arxiv	169343	1166243	128	40	44.898	82.687
Products	2449029	247436304	100	47	4720.000	934.230
Reddit	232965	229231784	602	41	4372.450	534.992

sorted array is empty or the remaining array has y values all larger or smaller than 1. With the upper level optimized, DAHA then goes to the lower level again, which initiates the recursive EM process. Finally, when DAHA meets the required pipeline quality or the maximum limitation on the number of iterations, the optimization process is terminated and the current batch grouping together with the execution plan of each group is returned as the output. With the batch grouping and per batch group execution plan, DAHA further saturates the pipeline by eagerly starting the next batch group as early as possible instead of waiting for the previous one to stop entirely. For example, while the previous batch group finishes its use of PCIE bandwidth, DAHA can begin transferring the data of the next batch group from CPU to GPU. DAHA then executes the batch groups guided by the per-group optimal execution plans and the eager cross-group optimization.

Algorithm Complexity. For each iteration, the time complexity for the upper-level optimization is dominated by the grouping update. The array sort for y value of n batches takes $O(n \log n)$ time. For each batch, it will be processed at most β times because after that, it will be removed from the sorted array. The search for the batch with the closest y value to the current group's reciprocal takes $O(\log n)$ time. Therefore, the time complexity for each batch is at most $O(\beta \log n)$. Note that this is true for all the batches in the group with size at most β. Therefore, the average time complexity for each beach is $O(\log n)$ and the total time complexity is $o(n \log n)$ for the grouping update. This shows the time complexity for the upper level is $O(n \log n)$. For the lower level, since we restrict the batch group size with β, the per-group complexity is upper bounded by $O(\beta!)$ and there are at most n groups. Hence, the lower level complexity is $O(\beta! \cdot n)$. Therefore, the total time complexity for one iteration is $O(n \log n + \beta! \cdot n)$. Normally, β should be at the magnitude of available workers. Otherwise the group size becomes too large and resource contention could occur. In our settings, typically $\beta < 10$ and $n < 100$. Hence, the complexity is justified and manageable.

This concludes the inter-batch scheduling optimizations of DAHA.

4 EXPERIMENTS

We evaluate the effectiveness and efficiency of DAHA in this section. The experiment setups are as below.

Environments. The experiments are by default deployed on a machine **"M1"** equipped with an Intel i9-12900KF CPU, 32 GB DDR5 main memory, and an NVIDIA GeForce RTX 3080 Ti GPU with 12 GB memory. PyTorch v1.12, DGL v0.9 and CUDA v11.6 are used. We also include **"M2"** with an Intel(R) Xeon(R) Gold 6240 CPU, a GeForce RTX 2080 Ti GPU and CUDA v10.1 to show the data and hardware aware ability of DAHA.

Implementation. We implement DAHA based on DGL [28] using PyTorch [23] backend as the deep learning framework. We modify the DGL samplers and dataloaders to defer the feature fetching process in the message flow graph (MFG) of DGL to enable lightweight storage of only the topology of mini-batches produced for multiple epochs. To unleash the potential of pipelining and mitigate the convergence issue, we implement bounded staleness for gradient updates in DAHA following prior work [10, 24, 26].

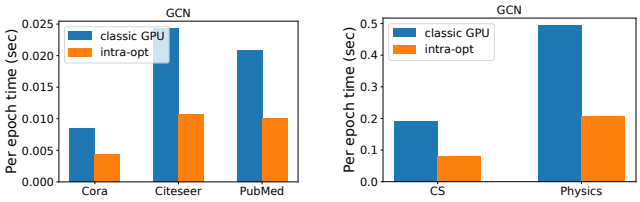
Datasets. We evaluate DAHA on the commonly-used [1, 25] benchmark datasets listed in Table 2. Cora, Citeseer, Pubmed, and Arxiv (ogbn-arxiv) are citation networks. CS and Physics are coauthor networks. Products (ogbn-products) is a product co-purchase network and Reddit is a post-to-post graph. These datasets are all publicly available on DGL [28], OGB [14], and PyG [8]. We aim to select graphs with different sizes, densities, and feature dimensions to evaluate whether DAHA can tailor different kinds of input data to consistently accelerate the end-to-end training process.

GNN models. We use three popular message-passing GNN models for evaluation: (1) GCN [19] with full-graph training, (2) GraphSAGE [12] with node-level sampling for mini-batch training, and (3) GraphSAINT [34] with subgraph-level sampling for mini-batch training. 100 epochs are used for all models by default. For mini-batch training, we use a batch size of 8192. A budget of 16384 is used for GraphSAINT with node sampler.

Baselines. We compare DAHA with three popular frameworks and systems, PyTorch [23], DGL [28], DGL+, ByteGNN [37] and SALIENT [16]. Since we implement DAHA based on PyTorch, it is natural to measure how much speedup gain is achieved against PyTorch. DGL is a popular dedicated mini-batch GNN training framework. We report its classic CPU-sample-GPU-train pattern as DGL and its pure-GPU execution with GPU sampling and GPU training as DGL+. Since original ByteGNN uses pure-CPU execution, we extend it by running its original version on CPU and DGL+ on GPU simultaneously for fair comparison with others since they share the same available hardware resources. SALIENT is an up-to-date mini-batch GNN training system work with a pipeline parallel strategy and a fast sampling implementation. For fair comparison with DGL, we use the DGL sampler for all the baselines. We test the GCN model on the small datasets Cora, Citeseer, Pubmed, CS, and Physics for performance evaluation on full-graph training while the GraphSAGE and GraphSAINT models on the medium to large datasets Arxiv, Products, and Reddit for mini-batch training. Since our focus is on execution planning and pipelining while GNNLab focuses on caching policy and situations where GPU memory cannot hold both graph topology and feature data, we do not compare with it. For DistDGLv2 [38] and the work by Kim et al. [18], their execution pattern is represented and covered by SALIENT and ByteGNN, respectively. Hence, we do not compare them to DAHA.

4.1 Overall Performance

We first compare the overall performance of the competitors. We report the average time elapsed for GNN training (excluding the



(a) DGL datasets. (b) PyG datasets.

Figure 5: Overall performance for GCN.

Table 3: GraphSAGE accuracy with various shuffling mode.

Dataset	Shuffle on	Shuffle off	Adaptive shuffle
Arxiv	69.58	68.11	69.21
Products	76.76	76.40	76.50
Reddit	96.29	96.22	96.27

model evaluation time) per epoch over a 100-epoch run. For the GCN model on Cora, Citeseer, Pubmed, and CS, we compare DAHA with PyTorch to illustrate the computation speedup gained from the re-written EPGs of DAHA since no batch preparation operation is involved. For the GraphSAGE and GraphSAINT models on Physics, Arxiv, Products, and Reddit, we compare DAHA with others to evaluate the end-to-end pipeline parallel strategies of DAHA. We also report the accuracy of mini-batch training to show that DAHA can achieve significant speedup at a negligible cost of accuracy.

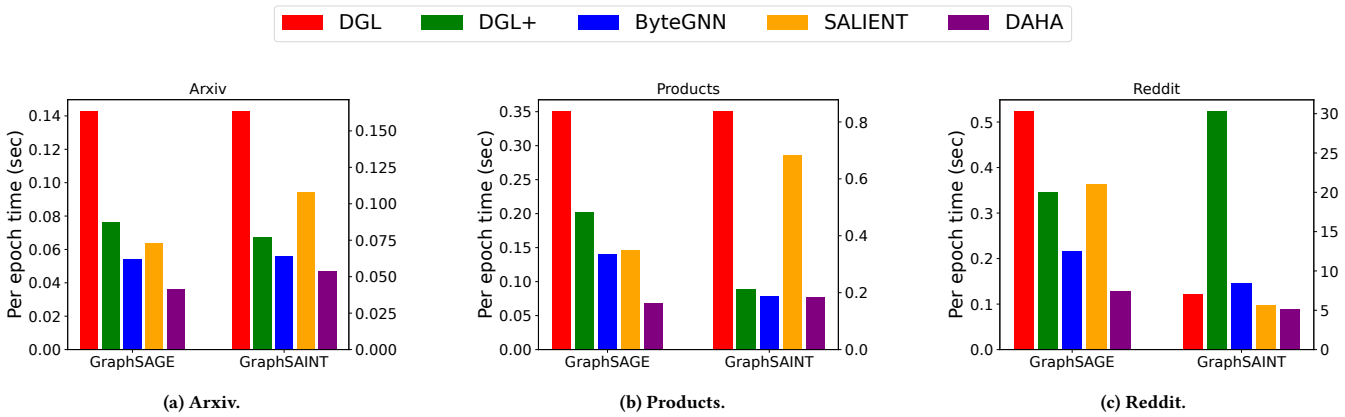
4.1.1 GCN speedup. Figure 5 shows the per-epoch training time of GCN on both DGL and PyG datasets. Each of the two adjacent bars records the performance of PyTorch (left) and DAHA (right) on one dataset. DAHA achieves 48.36%, 55.92%, 51.53%, 59.13%, and 58.01% less end-to-end training time compared with the PyTorch implemented with classic pure GPU execution on the five datasets, respectively. Because full-graph GCN training does not involve any batch preparation cost like sampling, this is the pure computation speedup of DAHA, attributed to the EPG abstraction of DAHA and the corresponding intra-batch level optimizations like the pre-transformation in Section 3.2.2. This proves that by going from pure GPU computation to hybrid CPU-GPU computation, DAHA explores more optimization opportunities by effectively utilizing the otherwise idle hardware resources to reduce the training time. Additionally, the five involved datasets have various sizes, densities, and feature dimensions, while DAHA can consistently achieve a similar yet high speedup regardless of the different settings. This shows that DAHA can adapt to various kinds of input data to produce a suitable hybrid CPU-GPU execution plan that effectively utilizes the underlying hardware.

4.1.2 Mini-batch training speedup. Figure 6 reports the per-epoch training time of GraphSAGE and GraphSAINT on the three datasets, Arxiv, Products, and Reddit. Each subplot reports a dataset. The left y-axis of each subplot is the per-epoch time for GraphSAGE while the right y-axis is the per-epoch time for GraphSAINT. Table 6 shows that DAHA consistently accelerates mini-batch training for various kinds of input data. The consistent performance on the two representative GNN models shows the generalization ability of DAHA to design the execution plan according to the model specifications. Compared with the default baseline DGL, DAHA achieves 74.94% and 67.23% less training time on Arxiv with GraphSAGE and GraphSAINT, respectively. The efficiency gain is 80.83% and 78.00% on Products, and 75.37% and 27.80% on Reddit. Compared with SALIENT, DAHA achieves an efficiency gain of 43.66% and 50.42% on Arxiv, 54.13% and 72.97% on Products, and 64.61% and 9.66% on Reddit, for GraphSAGE and GraphSAINT, respectively. SALIENT and DAHA achieve better efficiency than DGL due to their adopted pipeline parallelism. DAHA further beats SALIENT because (1) the group-based in-turn pipeline strategy achieves a faster effective sampling time per epoch; (2) the inter-batch level execution planning further exploits the optimization opportunities made available by the group-based in-turn pipeline and achieves a faster effective computation speed. Details will be covered in Section 4.2. DAHA also consistently outperforms DGL+ with pure-GPU execution. Since we extend ByteGNN by running its dedicated CPU scheduling on CPU and DGL+ on GPU simultaneously, ByteGNN now utilizes both CPU and GPU as both samplers and trainers. But the process on CPU is independent to the process on GPU. Thus, it misses inter-device optimization opportunities, which is why DAHA outperforms it. One exception is the workload of GraphSAINT on Products where ByteGNN achieves similar performance yet is still slower than DAHA. The reason could be that sampling accounts for the majority of end-to-end time because the sampled subgraphs are too sparse. This relatively reduces the DAHA's effectiveness in inter-batch optimization for training. In addition, CPU sampling is typically slower than GPU sampling with the exception of GraphSAINT on Reddit as DGL outperforms DGL+ in this workload. The reason could be the extremely dense connections in Reddit and the sampling mechanism of GraphSAINT. Thus, SALIENT beats ByteGNN in this workload due to its faster CPU sampling while ByteGNN beats SALIENT in other workloads. Despite this, DAHA always achieves the fastest end-to-end time, showcasing its ability to adapt to various scenarios.

4.1.3 Mini-batch accuracy. Table 3 reports the mini-batch training accuracy with various shuffling mode. Shuffle on is the common practice where each epoch shuffles the target nodes to be sampled. Shuffle off means no shuffling. Adaptive shuffle is DAHA's strategy in Section 3.2.1. It shows that shuffle on enjoys the best accuracy while shuffle off the worst. Shuffle off suffers from an accuracy drop of nearly 1.5% on Arxiv. Although they converge to a similar evaluation accuracy, shuffle off typically has poorer generalization ability which results in its worse test accuracy. This demonstrates the potential drawback of directly using shuffle off in DAHA. By leveraging the adaptive shuffling strategy where shuffle off is first used to get a "pretrained" model and shuffle off is then used to obtain a fine-grained result, DAHA achieves better test accuracy than shuffle off and is comparable with shuffle on.

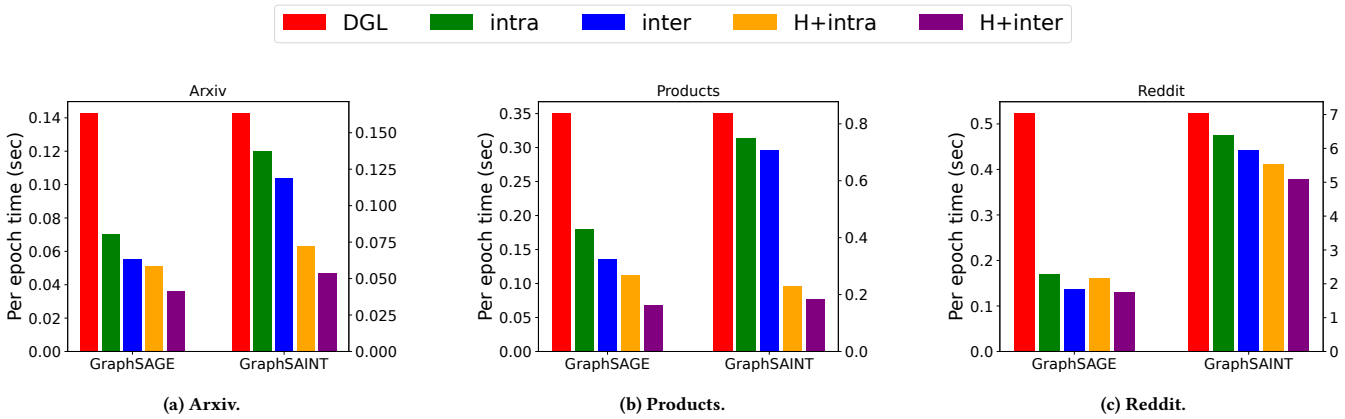
4.2 Breakdown Analysis

We then analyze the effectiveness of each of the three modules in DAHA: (1) group-based in-turn pipelining; (2) intra-batch EPG rewriting; and (3) inter-batch scheduling. We also examine the scalability of DAHA and demonstrate the efficiency and effectiveness



(a) Arxiv. (b) Products. (c) Reddit.

Figure 6: Overall performance for mini-batch. The left y-axis is for GraphSAGE and the right y-axis is for GraphSAINT.



(a) Arxiv. (b) Products. (c) Reddit.

Figure 7: Ablation study for mini-batch. Left y-axis is for GraphSAGE and right y-axis is for GraphSAINT.

of our data and hardware aware cost model in this section. First, we introduce four variants of DAHA.

- "intra": DGL CPU sampler with intra-batch optimizations;
- "inter": DGL CPU sampler with both intra- and inter-batch optimizations;
- "H+intra": group-based in-turn pipelining with intra-batch optimizations;
- "H+inter": group-based in-turn pipelining with both intra- and inter-batch optimizations (DAHA's default variant).

Table 4 lists the training time statistics for the workloads of training GraphSAGE and GraphSAINT on Arxiv, Products and Reddit. Table 4 and Figure 7 also serve as an ablation study of DAHA as they show (1) intra-batch EPG rewriting accelerates the training stage over DGL as intra outperforms DGL with the same sampling stage; (2) inter-batch scheduling builds on and improves intra-batch optimizations as inter outperforms intra; and (3) group-based in-turn pipelining can further boost the efficiency as H+intra outperforms intra and H+inter outperforms inter.

4.2.1 Intra-batch EPG Rewriting. We first analyze the efficiency gain from the intra-batch EPG rewriting module which introduces pipeline parallelism to the training stage alone. The effectiveness of our intra-batch EPG rewriting has already been partially illustrated in the GCN speedup in Figure 5 since full-graph GCN training can be regarded as one batch. In Table 4, we further report the average computation time (excluding the sampling stage) of all batches for the mini-batch training setting in the column "intraT". It can be regarded as the standalone time for the neural training stage. Similar to the full-graph GCN case, DAHA achieves significant speedup for all the tested datasets and sampling-based mini-batch GNN models. Although intra does not pipeline the sampling stage as for DGL, intra can achieve much faster end-to-to training time than DGL. Since the underlying sampler is both DGL's version, the speedup against DGL all comes from the computation. This proves that intra can leverage the otherwise idle CPU to effectively accelerate the neural computation and hide the communication cost. This serves as an ablation study on the intra-batch EPG rewriting module of DAHA. Notably, intra even outperforms SALIENT which pipelines both sampling and training stages when training GraphSAGE on Reddit. The reason could be that this workload is extremely computation-intensive (sampling-light), so the computation speedup of intra has a great impact on the entire end-to-end training process. For other workloads, however, since intra wastes hardware resources at the sampling stage, it still fails to beat SALIENT.

Table 4: Breakdown of DAHA modules and comparison with others on mini-batch training. The records here are per-100-epoch. "sample" means the sampling time given the default DGL sampler. "intraT" and "interT" mean the computation time (training stage only, excluding sampling stage) using DAHA's intra-batch and inter-batch optimizations, respectively. "DGL" reports CPU-sample-GPU-train time and "DGL+" reports pure-GPU time of DGL. The last four columns are variants of DAHA.

Dataset	GNN	sample	intraT	interT	DGL	DGL+	ByteGNN	SALIENT	intra	inter	H+intra	H+inter
Arxiv	GraphSAGE	4.40	2.63	1.10	14.26	7.60	5.43	6.34	7.03	5.50	5.10	3.57
Arxiv	GraphSAINT	10.24	3.47	1.61	16.32	7.68	6.36	10.79	13.71	11.85	7.21	5.35
Products	GraphSAGE	10.38	7.63	3.15	35.00	20.11	14.03	14.63	18.00	13.52	11.19	6.71
Products	GraphSAINT	67.38	7.45	3.15	83.60	20.95	18.59	68.06	74.83	70.52	22.70	18.39
Reddit	GraphSAGE	8.14	8.82	5.57	52.26	34.41	21.66	36.37	16.96	13.71	16.13	12.87
Reddit	GraphSAINT	551.59	85.81	40.81	702.74	3024.34	846.21	561.64	637.40	592.40	552.36	507.36

4.2.2 Inter-batch Execution Planning. We then analyze the inter-batch execution planning module, which is unique for the mini-batch training setting. Since DAHA considers intra-batch EPG rewriting as well in its inter-batch execution planning, interT should be faster than intraT and inter should outperform intra, as evidenced by the "interT" column in Table 4 which records the average computation time with DAHA's inter-batch optimizations. This serves as an ablation study on the inter-batch scheduling module of DAHA. Despite the remarkable efficiency gain of the intra-batch EPG rewriting module, the inter-batch execution planning further boosts the computation speed against intra by 58.14%, 53.59%, 58.75%, 57.78%, 36.88%, 52.44% for the six workloads in Table 4, respectively. This shows that with intra-batch optimization alone, a substantial amount of hardware resources are idle. With the help of DAHA's inter-batch execution planning, the unsaturated computation pipeline is filled by effectively utilizing the otherwise idle hardware resources. As a result, the "inter" variant of DAHA even outperforms SALIENT on three GraphSAGE workloads out of the total six workloads listed in Table 4. Since we have exhausted the optimization space with inter-batch scheduling, it is vital to emphasize pipelining of the sampling stage as well.

4.2.3 Group-based In-turn Pipelining. Recall that the group-based in-turn pipelining lays the foundation for DAHA since it provides novel optimization opportunities for intra-batch and inter-batch scheduling. The DAHA variants intra and inter also follow the group-based in-turn pipeline to get accurate time estimates for scheduling with the exception that they do not perform pipeline parallelism at the sampling stage. To unleash the full power of DAHA, pipelining of the sampling stage with all available hardware resources is vital. As a result, Table 4 shows that the DAHA variant H+intra outperforms its base version intra by 27.40%, 47.42%, 37.84%, 69.66%, 4.92%, 13.34% for the six tested workloads, respectively. The DAHA variant H+inter also outperforms its base version inter by 35.03%, 54.85%, 50.38%, 73.92%, 6.08%, 14.36%. This serves as an ablation study on the group-based in-turn pipelining module of DAHA. The speedup for training GraphSAGE on Reddit is limited while for training GraphSAINT on Products is significant. The reason is that the former workload is extremely computation-intensive while the latter is extremely sampling-intensive. The different bottlenecks in the pipeline result in different end-to-end speedups. Despite this, DAHA consistently accelerates the end-to-end training time with a remarkable average speedup by enabling pipelining in the sampling stage. Now DAHA outperforms SALIENT on all the six workloads even with H+intra. This showcases that the group-based in-turn pipelining strategy of DAHA not only prevents slow production of sampled batches from blocking fast consumption but also opens up a diverse space for optimization opportunities.

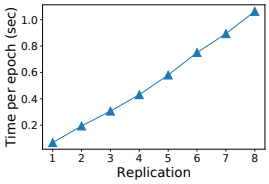


Figure 8: Scalability of DAHA on replicated Products dataset.

Table 5: Preprocessing time in seconds for cost model.

Machine	Data collection	Regression
M1	37.8564	0.0205
M2	64.2065	0.2793

4.2.4 Scalability. To evaluate the effect of data size, we replicate the Products dataset from $n = 2$ to 8. We extend DAHA to treat each replication as a different subgraph to mimic the condition where the GPU memory cannot hold the entire graph adjacency (assume only one replication can reside in GPU), so frequent swaps in and out of subgraphs are needed. Figure 8 reports the per-epoch time. Note $n = 1$ means the original result. It shows that DAHA hides the CPU-GPU transfer cost of the graph adjacency behind other operations with its pipeline execution planning and achieves a near-linear growth of end-to-end time.

4.2.5 Data and hardware aware cost model. We also demonstrate the efficiency and effectiveness of our data and hardware aware cost model. Table 5 reports the data collection and regression time to retrieve the cost model. We use 10 data points to fit the regression model and each comes from a random subgraph of Pubmed. Table 6 lists the true and predicted time cost of various operations on various datasets, devices and machines. The operation of data transfer consists of transferring both the sparse adjacency matrix and the dense feature matrix. DAHA consistently gives accurate estimates

Table 6: Comparison of true and predicted time cost (in seconds). The statistics are per 100 epoch. Data transfer involves both the sparse adjacency matrix and the dense feature matrix.

Machine	Operation	spmm						mm						Data transfer		
	Dataset	Pubmed		CS		Physics		Pubmed		CS		Physics		Pubmed	CS	Pyhsics
	Device	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	PCIE	PCIE	PCIE
M1	Truth	5.8000	0.5000	68.8993	6.2001	243.9285	17.3027	0.3001	0.1000	1.0000	0.1001	2.4999	0.2000	0.6000	6.0110	13.8314
	Prediction	5.7397	0.4994	69.2511	5.5836	235.2394	17.6340	0.3061	0.0945	0.9029	0.0988	2.4201	0.2044	0.5434	5.9200	13.6227
M2	Truth	4.8111	0.2213	57.3462	1.8583	231.3737	6.1514	0.0209	0.0147	0.8864	0.2030	2.2079	0.3689	0.9971	10.9417	25.2243
	Prediction	4.8857	0.2224	58.1973	1.7921	276.5741	6.3681	0.0216	0.0142	0.9073	0.1890	2.2909	0.3719	0.9793	11.0075	22.1922

on the three datasets of various sizes, densities and feature dimensions for different devices. Notably, the GPU speedup for the spmm operation is typically much more significant than the mm operation. This also justifies the design of the pre-transformation strategy in our intra-batch EPG rewriting module. We just need to preprocess each machine once for the cost model in advance of model training and its performance is satisfying for various datasets. This shows the hardware and data awareness of DAHA to adapt to different hardware and input data. Since the preprocessing is once-for-all and fast, we exclude it from the end-to-end training time. The cost model takes 0.005 seconds for 100 predictions, thus, the validation overhead during training is trivial.

Summary of experiments. (1) DAHA consistently outperforms the tested baselines in terms of end-to-end training efficiency; (2) The group-based in-turn pipelining strategy of DAHA diversifies the optimization opportunities to accelerate end-to-end training; (3) The intra-batch EPG rewriting module and the inter-batch execution planning module of DAHA consistently and significantly accelerates the GNN computation.

5 RELATED WORK

We examine the related work and identify a research gap here.

The efficiency issue of GNNs attracts many research interests to apply DB4AI techniques to accelerate GNN training and many works have been published on top database conferences [20, 24, 27, 29, 35–37, 39]. The CPU-sample-GPU-train execution in the popular GNN frameworks like DGL [28] faces high batch preparation cost and CPU-GPU data transfer cost [16], which can result in device under-utilization and poor efficiency. There are mainly two classes of optimizations that could help.

Pipeline parallelism. To address the data transfer bottleneck, Dorylus [26] breaks down the forward and backward passes into fine-grained tasks to identify pipeline parallelism opportunities. However, the system is designed for the serverless scenario, which differs greatly from the common CPU-GPU machines. SALIENT [16] leverages a faster sampling approach and novel pipelining strategies to offset the influence of batch preparation and data transfer cost. DistDGLv2 [38] proposes an asynchronous mini-batch preparation pipeline for the distributed setting to overlap batch preparation and data transfer with batch computation. However, limited by their device utilization pattern, slow sampling could still block the following operations and the pipeline might not be effective. In addition, the smallest unit in the pipeline is the computation or communication of one entire batch, ignoring operation-level optimization. The limited hardware utilization pattern and coarse-grained pipeline limits the optimization opportunities.

Device utilization pattern. GNNLab [32] suggests a new computing paradigm better leveraging the GPUs. It aims to alleviate the GPU underutilization problem by proposing a factored system for a single-machine multi-GPU setup where some GPUs are dedicated to sampling and others to training. Despite the speed gain on sampling and the high GPU utilization rate, it involves few CPU workloads, leaving room for improvement by diversifying the role of CPU cores. Kim et al. [18] perform partial aggregation of on-device data on both CPU and GPU. But it might encounter CPU bottleneck when CPUs are overloaded. CPU is limited to sampling alone and the task division is still coarse-grained as there is no further breakdown of the aggregation operation.

Despite their efforts, they do not explore all possible device utilization patterns, hence missing opportunities for optimizing the execution plan. We aim to fill the gap by expanding the search space to full combinations of different operations on different devices.

6 CONCLUSION

In this paper, we present DAHA, a data and hardware aware framework that optimizes GNN execution planning at a fine-grained operation-level. DAHA devises a novel pipeline strategy that in turn processes the sampling and training stages with all hardware resources. Along with intra-batch execution plan graph rewriting, DAHA allows all fine-grained operations in GNN computation to be executed on all valid hardware. Hence, DAHA explores more opportunities for optimizing the execution plan. DAHA further proposes an efficient solution to the optimization problem of inter-batch scheduling. Extensive experiments show DAHA can effectively accelerate GNN training. An interesting further direction is to incorporate caching in DAHA.

ACKNOWLEDGMENTS

Yue Wang is partially supported by China NSFC(No.62002235). Yingxia Shao's work is supported by the National Natural Science Foundation of China (Nos. 62272054, 62192784), Beijing Nova Program (No. 20230484319), and Xiaomi Young Talents Program. Lei Chen's work is partially supported by National Science Foundation of China (NSFC) under Grant No. U22B2060, the Hong Kong RGC GRF Project 16213620, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, CRF Project C2004-21G, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Microsoft Research Asia Collaborative Research Grant and HKUST-Webank joint research lab grants.

REFERENCES

[1] Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, and Eduard Alarcón. 2021. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys (CSUR)* (2021).

[2] David Applegate and William Cook. 1991. A computational study of the job-shop scheduling problem. *ORSA Journal on computing* 3, 2 (1991), 149–156.

[3] Bo Chen, Chris N Potts, and Gerhard J Woeginger. 1998. A review of machine scheduling: Complexity, algorithms and approximability. *Handbook of Combinatorial Optimization: Volume1–3* (1998), 1493–1641.

[4] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*.

[5] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *ICML*.

[6] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *SIGKDD*.

[7] Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. 2020. Minimal Variance Sampling with Provable Guarantees for Fast Training of Graph Neural Networks. In *SIGKDD*.

[8] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. http://arxiv.org/abs/1903.02428

[9] Aleksei V Fishkin, Klaus Jansen, and Monaldo Mastrolilli. 2008. Grouping techniques for scheduling problems: Simpler and faster. *Algorithmica* 51, 2 (2008), 183–199.

[10] Swapnil Gandhi and Anand Padmanabha Iyer. 2021. P3: Distributed Deep Graph Learning at Scale. In *OSDI*.

[11] Leslie Ann Goldberg, Mike Paterson, Aravind Srinivasan, and Elizabeth Sweedyk. 2001. Better approximation guarantees for job-shop scheduling. *SIAM Journal on Discrete Mathematics* 14, 1 (2001), 67–92.

[12] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*.

[13] Ellis Horowitz and Sartaj Sahni. 1976. Exact and Approximate Algorithms for Scheduling Nonidentical Processors. *J. ACM* 23, 2 (apr 1976), 317–327. https://doi.org/10.1145/321941.321951

[14] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. In *NIPS*.

[15] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive sampling towards fast graph representation learning. In *NIPS*.

[16] Tim Kaler, Nickolas Stathas, Anne Ouyang, Alexandros-Stavros Iliopoulos, Tao Schardl, Charles E Leiserson, and Jie Chen. 2022. Accelerating training and inference of graph neural networks with fast sampling and pipelining. In *MLSys*.

[17] Qifa Ke, Michael Isard, and Yuan Yu. 2013. Optimus: a dynamic rewriting framework for data-parallel execution plans. In *EuroSys*.

[18] Taehyun Kim, Changho Hwang, KyoungSoo Park, Zhiqi Lin, Peng Cheng, Youshan Miao, Lingxiao Ma, and Yongqiang Xiong. 2021. Accelerating gnn training with locality-aware partial execution. In *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems*. 34–41.

[19] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[20] Yiming Li, Yanyan Shen, Lei Chen, and Mingxuan Yuan. 2023. Zebra: When Temporal Graph Neural Networks Meet Temporal Personalized PageRank. In *PVLDB*.

[21] Zhiqi Lin, Cheng Li, Youshan Miao, Yunxin Liu, and Yinlong Xu. 2020. Pagraph: Scaling gnn training on large graphs via computation-aware caching. In *SoCC*.

[22] Zekun Lu, Qiancheng Yu, Xia Li, Xiaoning Li, and Qinwen Yang. 2023. Learning Weight Signed Network Embedding with Graph Neural Networks. *Data Science and Engineering* 8, 1 (2023), 36–46.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NIPS*.

[24] Jingshu Peng, Zhao Chen, Yingxia Shao, Yanyan Shen, Lei Chen, and Jiannong Cao. 2022. Sancus: sta le n ess-aware c omm u nication-avoiding full-graph decentralized training in large-scale graph neural networks. In *PVLDB*.

[25] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).

[26] John Thorpe, Yifan Qiao, Jonathan Eyolfson, Shen Teng, Guanzhou Hu, Zhihao Jia, Jinliang Wei, Keval Vora, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2021. Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads. In *OSDI*.

[27] Xinchen Wan, Kaiqiang Xu, Xudong Liao, Yilun Jin, Kai Chen, and Xin Jin. 2023. Scalable and Efficient Full-Graph GNN Training for Large Graphs. In *SIGMOD*.

[28] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315* (2019).

[29] Qiange Wang, Yanfeng Zhang, Hao Wang, Chaoyi Chen, Xiaodong Zhang, and Ge Yu. 2022. NeutronStar: Distributed GNN Training with Hybrid Dependency Management. In *SIGMOD*.

[30] Yongji Wu, Matthew Lentz, Danyang Zhuo, and Yao Lu. 2022. Serving and Optimizing Machine Learning Workflows on Heterogeneous Infrastructures. In *PVLDB*.

[31] Shuo Xiao, Dongqing Zhu, Chaogang Tang, and Zhenzhen Huang. 2023. Combining Graph Contrastive Embedding and Multi-head Cross-Attention Transfer for Cross-Domain Recommendation. *Data Science and Engineering* 8, 3 (2023), 247–262.

[32] Jianbang Yang, Dahai Tang, Xiaoniu Song, Lei Wang, Qiang Yin, Rong Chen, Wenyuan Yu, and Jingren Zhou. 2022. GNNLab: A Factored System for Sample-Based GNN Training over GPUs. In *EuroSys*.

[33] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*.

[34] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *ICLR*.

[35] Dalong Zhang, Xin Huang, Ziqi Liu, Jun Zhou, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Lin Wang, Zhiqiang Zhang, and Yuan Qi. 2020. AGL: A Scalable System for Industrial-Purpose Graph Machine Learning. In *PVLDB*.

[36] Xin Zhang, Yanyan Shen, Yingxia Shao, and Lei Chen. 2023. DUCATI: A Dual-Cache Training System for Graph Neural Networks on Giant Graphs with the GPU. In *SIGMOD*.

[37] Chenguang Zheng, Hongzhi Chen, Yuxuan Cheng, Zhezheng Song, Yifan Wu, Changji Li, James Cheng, Hao Yang, and Shuai Zhang. 2022. ByteGNN: efficient graph neural network training at large scale. In *PVLDB*.

[38] Da Zheng, Xiang Song, Chengru Yang, Dominique LaSalle, and George Karypis. 2022. Distributed hybrid CPU and GPU training for graph neural networks on billion-scale heterogeneous graphs. In *SIGKDD*.

[39] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. 2019. AliGraph: A Comprehensive Graph Neural Network Platform. In *PVLDB*.