Graph Coarsening via Convolution Matching for Scalable Graph Neural Network Training

Charles Dickens*
cadicken@ucsc.edu
University of California Santa Cruz
Santa Cruz, CA, U.S.A

Eddie Huang
ewhuang@amazon.com
Amazon
U.S.A

Aishwarya Reganti
areganti@amazon.com
Amazon
U.S.A

Jiong Zhu*
jiongzhu@umich.edu
University of Michigan
Ann Arbor, MI, U.S.A

Karthik Subbian
ksubbian@amazon.com
Amazon
U.S.A

Danai Koutra
dkoutra@amazon.com
Amazon
U.S.A

ABSTRACT

Graph summarization as a preprocessing step is an effective and complementary technique for scalable graph neural network (GNN) training. In this work, we propose the Coarsening Via Convolution Matching (ConvMatch) algorithm and a highly scalable variant, A-ConvMatch, for creating summarized graphs that preserve the output of graph convolution. We evaluate ConvMatch on six real-world link prediction and node classification graph datasets, and show it is efficient and preserves prediction performance while significantly reducing the graph size. Notably, ConvMatch achieves up to 95% of the prediction performance of GNNs on node classification while trained on graphs summarized down to 1% the size of the original graph. Furthermore, on link prediction tasks, ConvMatch consistently outperforms all baselines, achieving up to a 2× improvement.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Neural networks**; **Machine learning algorithms**.

KEYWORDS

Graph Neural Networks, Graph Summarization, Graph Convolutions

ACM Reference Format:
Charles Dickens, Eddie Huang, Aishwarya Reganti, Jiong Zhu, Karthik Subbian, and Danai Koutra. 2024. Graph Coarsening via Convolution Matching for Scalable Graph Neural Network Training. In *Companion Proceedings of the ACM Web Conference 2024 (WWW '24 Companion), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3589335.3651920

*Work done while the authors were on an internship at Amazon.

This work is licensed under a Creative Commons Attribution International 4.0 License.

WWW '24 Companion, May 13–17, 2024, Singapore, Singapore
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0172-6/24/05
https://doi.org/10.1145/3589335.3651920

1 INTRODUCTION

Graph neural networks (GNNs) have achieved state-of-the-art performance on various tasks ranging from recommendation to predicting drug interactions [13]. However, a drawback of a GNN's modeling capacity is a computationally expensive inference process with a complexity that scales with the size of the graph. Modern techniques for scaling the training of deep models, such as leveraging the parallel structure of GPUs for processing large blocks of data, have been successfully adopted by the GNN community [9, 36]. However, graph datasets encountered in real-world applications are on the order of tens of billions of edges [44] and quickly exceed the costly and limited memory capacity of today's GPUs. Techniques for partitioning and distributing the training graph across computational resources and integrating graph sampling into the training pipeline have been proposed [16, 30, 43, 45]. Nonetheless, training GNNs is still a highly expensive process, which limits applicability and prohibits large-scale architecture searches. Furthermore, distributed training and sampling techniques introduce their own difficulties. For example, distributed training faces communication overhead across machines [10], while sampling techniques bring additional hyperparameters that affect model performance [30, 45].

A promising new direction of scalable GNN training is to perform summarization, i.e., create a smaller graph with fewer nodes and edges, as a preprocessing step. These methods either sample nodes and edges from the original training graph [11, 26, 32, 34, 38], coarsen by clustering nodes into supernodes [15], or create synthetic connections and node features [17, 42]. To be applicable for scalable GNN training, the summarization process should be faster thanfi tting a GNN on the original graph. Additionally, the summarized graph should share properties with the original graph such that a GNN can befi t for various downstream tasks with good performance. Existing approaches to summarization typically do not satisfy at least one of the mentioned desirable properties (Table 1).

In this work, we introduce Coarsening Via Convolution Matching (ConvMatch), a scalable coarsening algorithm. ConvMatch merges nodes that minimize a cost quantifying the change in the output of graph convolution. Notably, ConvMatch merges nodes that are structurally similar, allowing the algorithm to identify and summarize redundant nodes that are distant, or even disconnected, in the original graph. Our primary contributions are:

- **New Approach**: We introduce ConvMatch, a coarsening algorithm that preserves the output of graph convolutions.

- **Highly-scalable Variant**: We propose a principled approximation to computing costs in the ConvMatch algorithm, A-ConvMatch, which allows it to scale to large graphs.
- **Extensive Empirical Analysis**: We perform an extensive empirical analysis demonstrating our method's ability to summarize large-scale graphs and preserve prediction performance. On link prediction tasks, ConvMatch achieves up to a 2× prediction performance improvement over the best baseline. In node classification, it achieves up to 95% of the prediction performance of a GNN on a graph that is 1% the size of the original.

2 RELATED WORK

We give a qualitative comparison of methods for graph summarization for scalable GNN training in Table 1.

Coreset Selection. Coreset methods aim tofi nd a subset of training examples such that a model trained on the subset will perform similarly to a model trained on the complete dataset [11, 26, 32, 34, 38]. *Herding*, proposed by Welling (2009) [38] is a coreset technique in which training examples arefi rst mapped to an embedding space and then clustered by class. Examples closest to the cluster center in the embedding space are selected. The *KCenter* algorithm [32] similarly embeds training data and then incrementally selects examples with the largest minimum distance to the growing cluster subset.

Graph Condensation. Distillation and condensation techniques search for a small synthetic dataset such that model parametersfit on the synthetic dataset are approximate minimizers of the training objective on the original dataset [2, 17, 18, 28, 33, 37, 41, 42]. Recently, Jin et al. (2022b) [18] extended the dataset condensation via gradient matching scheme proposed by Zhao et al. (2021) [42] with the *GCond* algorithm, which synthesizes graph data for training GNNs, and later with *DosCond*, which performs one-step gradient matching tofi nd the synthesized graph [17]. Alternatively, Liu et al. (2022) [23] propose a condensation method for creating a synthetic graph that aims to match statistics of the receptivefield of the original graph nodes.

Graph Coarsening. Coarsening is a graph summarization [24] technique in which nodes and/or edges from an original graph are merged to form a supergraph. Graph coarsening methods are widely applied and studied for problems ranging from influence analysis [29], visualization [6, 12, 35], combinatorial optimization [7, 27], and, recently, scaling graph embeddings [1, 5, 8, 15, 20, 22, 40]. Moreover, coarsening methods typically have the practically advantageous property of producing multi-level summaries, i.e., producing summaries at multiple level of granularity. Huang et al. (2021) [15] specifically proposed coarsening to overcome scalability issues of GNN training. The authors coarsen the graph used for training the GNN, with algorithms by Loukas (2019) [25].

ConvMatch is a graph coarsening algorithm that aims to preserve the graph convolution operations that are fundamental to spectral-based GNNs. ConvMatch advances existing coarsening approaches for scalable GNN training by merging structurally similar nodes instead of neighbors.

3 BACKGROUND AND PRELIMINARIES

We start with key notations and the necessary background for describing our proposed approach.

Graph Notations. Let $G = (\mathcal{V}, \mathcal{E})$ denote a graph with a node attribute matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, where $n = |\mathcal{V}|$ and $d > 0$. Let $\mathbf{A} \in \{0,1\}^{n\times n}$ be the adjacency matrix corresponding to the graph G, and $\mathbf{D}$ be the diagonal degree matrix.

Graph Coarsening. A *coarse graph* is defined from a partitioning of the nodes into $n' \leq n$ clusters: $\mathcal{P} = \{C_1, C_2, \cdots, C_{n'}\}$. Each partition, $C_i \in \mathcal{P}$, is referred to as a *supernode*. The partitioning is represented by a partition matrix

$$\mathbf{P} \in \left\{ \mathbf{P}' \in \{0,1\}^{n\times n'} \,\middle|\, \sum_j \mathbf{P}'_{i,j} = 1,\ \forall i \right\} \triangleq \mathbb{P}(n,n') \tag{1}$$

, where entry $\mathbf{P}_{i,j} = 1$ if and only if $v_i \in C_j$. Given the partition matrix, the coarse graph $G' = (\mathcal{V}', \mathcal{E}')$ is constructed with an adjacency matrix $\mathbf{A}' \triangleq \mathbf{P}^T\mathbf{A}\mathbf{P}$ and degree matrix $\mathbf{D}' \triangleq \mathbf{P}^T\mathbf{D}\mathbf{P}$. We define the supernode size matrix of the coarse graph as $\mathbf{C} \triangleq \mathrm{diag}([|C_1|, |C_2|, \cdots, |C_{n'}|])$. Then, the coarse node attribute matrix is given as $\mathbf{X}' \triangleq \mathbf{C}^{-1}\mathbf{P}^T\mathbf{X}$.

Spectral Graph Convolutions. *Spectral-based GNNs* are a prominent class of models rooted in graph Fourier analysis [3, 4, 19, 21, 39]. These methods generally assume graphs to be undirected and rely on the graph Laplacian: $\mathbf{\Delta} \triangleq \mathbf{D} - \mathbf{A}$, and its eigendecomposition: $\mathbf{\Delta} = \mathbf{U}\Lambda\mathbf{U}^T$, where $\mathbf{U} \in \mathbb{R}^{n\times n}$ is an orthonormal matrix comprising the eigenvectors of $\mathbf{\Delta}$, and $\Lambda = \mathrm{diag}(\lambda_1, \cdots, \lambda_n)$ is the diagonal matrix of eigenvalues. The *graph Fourier transform* of a signal $\mathbf{x} \in \mathbb{R}^n$ over the graph G is defined as $\mathcal{F}_G(\mathbf{x}) \triangleq \mathbf{U}^T\mathbf{x}$. As $\mathbf{U}$ is an orthonormal matrix, the inverse graph Fourier transform is thus $\mathcal{F}_G^{-1}(\mathbf{x}) \triangleq \mathbf{U}\mathbf{x}$. The *graph convolution* of a signal $\mathbf{x} \in \mathbb{R}^n$ and a signal, orfi lter, $\mathbf{g} \in \mathbb{R}^n$, is the inverse transform of the element-wise product ($\odot$) of the signals in the transformed domain:

$$\mathbf{g} \star_G \mathbf{x} \triangleq \mathcal{F}_G^{-1}(\mathcal{F}_G(\mathbf{x}) \odot \mathcal{F}_G(\mathbf{g})) = \mathbf{U}(\mathbf{U}^T\mathbf{x} \odot \mathbf{U}^T\mathbf{g}). \tag{2}$$

Spectral-based GNNs use this definition to motivate architectures that approximate graph convolutions and parameterize thefi lter. In this work, we use the principled approximation of graph convolution proposed by Kipf and Welling (2017)[19]. Specifically, let $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{g}_\theta$ be a signal andfi lter parameterized by a scalar θ, respectively. The graph convolution of $\mathbf{x}$ and $\mathbf{g}_\theta$ is approximately:

$$\mathbf{g}_\theta \star_G \mathbf{x} \approx \theta(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})\mathbf{x}, \tag{3}$$

where $\tilde{\mathbf{A}} = \mathbf{A}+\mathbf{I}$ (graph with self-loops) and $\tilde{\mathbf{D}}$ is the degree matrix of $\tilde{\mathbf{A}}$. Graph convolutions are generalized to multi-dimensional signals to define graph convolutional networks (GCN). Specifically, let $\mathbf{X}$ be an $(n\times f)$ matrix, then a K-layer GCN parameterized by a matrix $\mathbf{\Theta}^{(K)}$ is a recursive application of the convolution approximation and an activation function, $\sigma(\cdot)$:

$$\mathbf{H}^{(K)} \triangleq \begin{cases} \mathbf{X} & K = 0 \\ \sigma((\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})\mathbf{H}^{(K-1)}\mathbf{\Theta}^{(K)}) & \text{o.w.} \end{cases} \tag{4}$$

Further, defining $\tilde{\mathbf{H}}^{(K)} \triangleq (\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})\mathbf{H}^{(k-1)}$ and $\mathbf{H}^{(0)} \triangleq \mathbf{X}$, we have the equivalent compact form: $\mathbf{H}^{(K)} \triangleq \sigma(\tilde{\mathbf{H}}^{(K)}\mathbf{\Theta}^{(K)})$. A notable instantiation of the GCN architecture proposed by Wu et al. (2019)[39] is the simplified GCN (SGC), which uses the identity

Table 1: Qualitative comparison of graph summarization methods. 'Summary': the type of summarized graph produced; 'NC/LP': summarized graph can be used to train a model for node classification or link prediction, resp.; 'No GNN on full graph': does not requirefi tting a GNN on the full graph; 'Multi-level': multiple levels of summarization produced; 'Merge Strategy': if applicable, the strategy for selecting nodes to merge.

Core Technique	Summary	Method	NC	LP	No GNN on Full Graph	Multi-Level	Merge Strategy
Coreset	Sampled graph	RS	✓	✓	✓	✓	-
		KCenter [32]	✓	×	×	✓	
		Herding [38]	✓	×	×	✓	
Condensation	Synthetic graph	GCond [18]	✓	✓	×	×	-
		DosCond [17]	✓	✓	×	×	
Coarsening	Supergraph	VN [25]	✓	✓	✓	✓	Neighbors
		ConvMatch (ours)	✓	✓	✓	✓	Structurally similar
		A-ConvMatch (ours)	✓	✓	✓	✓	Structurally similar

operator as the activation. K recursive applications of SGC layers is equivalent to a single linear operator acting on $(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})^K\mathbf{X}$:

$$\mathbf{H}_{SGC}^{(K)} \triangleq (\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})^K\mathbf{X}\mathbf{\Theta}. \tag{5}$$

This expression illustrates the primary benefits of the SGC architecture; the result of $(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})^K\mathbf{X}$ is cached so future inferences do not require computation of the intermediate representations of nodes. Moreover, the parameter space reduces to a single matrix $\mathbf{\Theta}$.

Coarse Graph Convolutions. Huang et al. (2021)[15] propose *coarse graph convolution layers*. Setting $\tilde{\mathbf{A}}' \triangleq \mathbf{A}'+\mathbf{C}$ and $\tilde{\mathbf{D}}' \triangleq \mathbf{D}'+\mathbf{C}$, a coarse graph convolution is recursively defined as

$$\mathbf{H}'^{(K)} \triangleq \begin{cases} \mathbf{X}' & K = 0 \\ \sigma((\tilde{\mathbf{D}}'^{-\frac{1}{2}}\tilde{\mathbf{A}}'\tilde{\mathbf{D}}'^{-\frac{1}{2}})\mathbf{H}'^{(K-1)}\mathbf{\Theta}^{(K)}) & \text{o.w.} \end{cases} \tag{6}$$

Similar to GCN convolutions, defining $\mathbf{H}'^{(0)} \triangleq \mathbf{X}'$ and $\tilde{\mathbf{H}}'^{(K)} \triangleq (\tilde{\mathbf{D}}'^{-\frac{1}{2}}\tilde{\mathbf{A}}'\tilde{\mathbf{D}}'^{-\frac{1}{2}})\mathbf{H}'^{(K-1)}$ we have the compact expression: $\mathbf{H}'^{(K)} \triangleq \sigma(\tilde{\mathbf{H}}'^{(K)}\mathbf{\Theta}^{(K)})$. Note that the dimensions of the parameter matrix, $\mathbf{\Theta}^{(K)}$, of a coarse graph convolution layer is not dependent on the partition $\mathcal{P}$, but rather on the dimensions of the original node attribute matrix $\mathbf{X}$ and can thus be applied to Eq. (4). In other words, the parameters $\mathbf{\Theta}^{(K)}$ learned on a coarse graph can also be used for inference on the original graph.

4 CONVMATCH: COARSENING VIA CONVOLUTION MATCHING

A coarsening algorithm designed for scalable GNN training should: (1) produce a small coarsened graph, and (2) a GNNfi t on the coarsened graph should have a similar prediction performance to a GNN fit on the original graph. We hypothesize, and empirically verify in Section 5, that preserving the output of graph convolutions by minimizing the difference in the intermediate node representations computed for a GCN layer and a coarse graph convolution layer produces good coarsenings for scalable training of spectral-based GNNs. In this section, we formalize the notion of preserving the output of graph convolutions with a combinatorial optimization problem. We then introduce two coarsening methods: Coarsening Via Convolution Matching (ConvMatch) and a highly scalable variant, A-ConvMatch, both approximately solving the proposed optimization problem.

4.1 Convolution Matching Objective

Preserving the output of GCN graph convolution for a given graph signal $\mathbf{x}$ and parameterizedfilter $\mathbf{g}_\theta$ is formalized by the following problem:

$$\underset{\mathbf{P}\in\mathbb{P}(n,n')}{\arg\min} \|\theta\mathbf{P}(\tilde{\mathbf{D}}'^{-\frac{1}{2}}\tilde{\mathbf{A}}'\tilde{\mathbf{D}}'^{-\frac{1}{2}})\mathbf{x}' - \theta(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})\mathbf{x}\|_1^1. \tag{7}$$

We aim tofi nd a partition matrix that minimizes the sum of the L_1 distances between the node representations obtained via the output of a single graph convolution on the original and coarsened graph. The parameter θ acts as a positive scalar multiple in our objective, thus, minimizing the difference to the unscaled GCN convolution is equivalent.

We generalize the objective in Eq. (7) to multi-dimensional graph signals by formulating a multi-objective problem. Specifically, we equally weigh the difference in the GCN convolution operation for each component of the graph signal to define a linear scalarization of the multi-objective problem:

$$\underset{\mathbf{P}\in\mathbb{P}(n,n')}{\arg\min} \sum_{i=1}^{f} \|\mathbf{P}(\tilde{\mathbf{D}}'^{-\frac{1}{2}}\tilde{\mathbf{A}}'\tilde{\mathbf{D}}'^{-\frac{1}{2}})\mathbf{x}'_i - (\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})\mathbf{x}_i\|_1^1, \tag{8}$$

where $\mathbf{x}_i$ and $\mathbf{x}'_i$ are the $i'th$ components of the f-dimensional graph signals $\mathbf{X}$ and $\mathbf{X}'$. The resulting objective ensures that whenfitting GNN parameters using the coarsened graph, the parameters are trained to operate on a matrix that is close to the original. It is important to note that the coarsening problem formulated in Eq. (8) does not restrict the partitioning to preserve connections in the original graph, i.e., two nodes that are distant or even existing in disconnected components of the original graph may be merged into a single supernode.

4.2 ConvMatch

A brute-force approach solving Eq. (8) by computing the cost of all partitionings of the nodes is an intractable procedure as the number of partitionings grows combinatorially with the number

of nodes. Therefore, we take a bottom-up hierarchical agglomerative clustering approach with ConvMatch tofi nd an approximate solution. ConvMatch is outlined in Algorithm 1 (and Algs. 2-3 in the Appendix) and an illustration is provided in Figure 2 in the Appendix.

Algorithm 1: ConvMatch

input : Graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, Ratio r, Merges per level k
output: Coarsened graph $G' = (\mathcal{V}', \mathcal{E}', \mathbf{X}')$

1 $G' = (\mathcal{V}', \mathcal{E}', \mathbf{X}') \leftarrow G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$;
2 $\mathbf{P} \leftarrow \mathbf{I}$; $\tilde{\mathbf{H}}' \leftarrow (\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})\mathbf{X}$;
3 `candidates` $\leftarrow$ `CandidateSupernodes`(G');
4 `supernode_costs` $\leftarrow$ `ComputeCosts`$(G', \tilde{\mathbf{H}}'$, `candidates`$)$;
5 **while** $|\mathcal{V}'| > r \cdot |\mathcal{V}|$ **do**
6 $\quad G', \mathbf{P}, \tilde{\mathbf{H}}'$, `supernodes` $\leftarrow$ `Merge`$(G'$, `TopKNonOverlap`(`supernode_costs`, k)$)$;
7 $\quad$ `supernode_costs` $\leftarrow$ `ComputeCosts`$(G', \tilde{\mathbf{H}}'$, `Neighborhood`(`supernodes`)$)$;

In short, ConvMatch proceeds byfi rst, in `Step` 1, computing the intermediate node representation obtained via a coarse graph convolution (Eq. (3)) and creating an initial set of candidate node pairs, or supernodes. Then, `Step` 2 computes a cost for each pair measuring the change in the objective value of Eq. (8) caused by creating the supernode, i.e., the change in the GCN convolution output. Finally, `Step` 3 of ConvMatch finds a number of lowest-cost node pairs, merges them, andfi nds new node pair candidates and costs. This process is repeated until the desired coarsening ratio is reached. As a hierarchical approach, ConvMatch produces multiple levels of coarsening, i.e., we refer to a the graph after ℓ passes of the ConvMatch algorithms a level-ℓ coarsened graph. In the following subsections, we describe the processes for generating candidate supernodes, computing supernode costs, andfi nally merging nodes.

4.2.1 `Step` 1*: Candidate Supernodes.* Considering all $\sim n^2$ node pairs as candidate supernodes is infeasible for large-scale graphs with millions of nodes and edges. Therefore, we only look at a subset of all possible pairs that capture attribute and structural similarities between nodes. Specifically, to generate the initial set of candidate supernodes ConvMatch pairs nearest neighbors in the embedding space of a trivially parameterized ($\mathbf{\Theta} = \mathbf{I}$) K-layer SGC network (Eq. (5)). This embedding is the output of K recursive applications of a GCN convolution Eq. (3), the very operation we are aiming to preserve. The supernode candidate set defines the *merge-graph*: $G_{merge} = (\mathcal{V}', \mathcal{E}_{merge})$, where, initially, $\mathcal{V}' = \mathcal{V}$ and $\mathcal{E}_{merge}$ is the set of edges connecting the generated node pairs. The embedding step has a computational time complexity of $O(K \cdot d_{avg} \cdot |\mathcal{V}|)$, where K is the depth of the SGC network being used and d_{avg} is the average degree of nodes in the graph. Note that computing the embedding is embarrassingly parallelizable. See Appendix B.1 for a more detailed description and algorithm.

4.2.2 `Step` 2*: Supernode Cost Computation.* Each edge connecting two supernodes, $u,v \in \mathcal{V}'$, in the merge-graph is associated with a cost quantifying the objective value in Eq. (8) for a partitioning that merges the incident supernodes. Let $\mathbf{P}_{(u,v)}$ be the partition matrix merging supernodes u and v. Moreover, let $\tilde{\mathbf{H}}^{(1)}_{(l)}$ and $\tilde{\mathbf{H}}^{(1)}_{(l,\mathbf{P}_{(u,v)})}$ represent the coarse graph convolution node representations obtained before and after applying the partitioning $\mathbf{P}$ at level l, respectively. Then, the cost of merging two supernodes is:

$$cost(u,v) \triangleq \|\mathbf{P}_{(u,v)}\tilde{\mathbf{H}}^{(1)}_{(l,\mathbf{P}_{(u,v)})} - \tilde{\mathbf{H}}^{(1)}_{(l)}\|_1^1. \tag{9}$$

A scalable algorithm and an illustration of an instance of the supernode cost computation is provided in Appendix B.2. Computing the cost of merging two nodes, u and v, exactly as it is defined in Eq. (9) has a time complexity of $O(d_u + d_v)$, where d_u and d_v are the degrees of u and v, respectively. This is because merging nodes u and v affects the representation of each neighbor of u and v. Caching techniques for scaling the evaluation of Eq. (9) are in Appendix B.4.

A-ConvMatch. Motivated by the following result, we propose A-ConvMatch, an approximation of the supernode cost computation that yields significant improvements in graph summarization time.

Theorem1. *The following is a tight upper bound on Eq. (9)*

$$\begin{aligned} cost(u,v) \leq & \|\tilde{\mathbf{H}}^{(1)}_{(l)}[u] - \mathbf{P}_{(u,v)}\tilde{\mathbf{H}}^{(1)}_{(l,\mathbf{P}_{(u,v)})}[(u,v)]\|_1^1 \\ & + \|\tilde{\mathbf{H}}^{(1)}_{(l)}[v] - \mathbf{P}_{(u,v)}\tilde{\mathbf{H}}^{(1)}_{(l,\mathbf{P}_{(u,v)})}[(u,v)]\|_1^1 \\ & + \|\tilde{\mathbf{x}}_{(u,v)} - \tilde{\mathbf{x}}_u\|_1^1 \sum_{i \in \mathcal{N}(\{u\})} \frac{a_{ui}}{\sqrt{(d_i + |C_i|)}} \\ & + \|\tilde{\mathbf{x}}_{(u,v)} - \tilde{\mathbf{x}}_v\|_1^1 \sum_{i \in \mathcal{N}(\{v\})} \frac{a_{vi}}{\sqrt{(d_i + |C_i|)}}, \end{aligned} \tag{10}$$

where $\tilde{\mathbf{x}}_i \triangleq \frac{\mathbf{x}_i}{\sqrt{(d_i+|C_i|)}}$ *are the normalized features for supernode* i*. The bound is satisfied with equality if* $\mathcal{N}(u) \cap \mathcal{N}(v) = \emptyset$*.*

The proof of Theorem 1 is provided in Appendix B.5. We use this bound as an approximation of the cost of merging two nodes in A-ConvMatch. This approximation allows the cost of merging two nodes to be a function of properties local to the two nodes being considered, making the cost computation fast and highly scalable. More formally, the time complexity of computing the approximate cost of merging two nodes, u and v, is a constant, $O(1)$, operation.

4.2.3 `Step` 3*: Node Merging.* At every level of coarsening, ConvMatch simultaneously merges the top-k non-overlapping lowest-cost candidate supernodes. For the coarsened graph and merge-graph, when supernodes u and v are merged to create a new supernode, the new supernode is connected to every neighbor of u and v. Furthermore, the edges connecting supernodes in the resulting coarsened graph are weighted by the number of edges connecting nodes in the two incident supernodes. Moreover, the features of the supernodes are a weighted average of the features of the nodes being merged, and in node classification settings, the node label used for training is the majority label of nodes in a supernode. More formally, in this step the partition matrix $\mathbf{P}$ and the coarsened graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ are updated to account for the merge. In addition, the cost of a subset of node pairs connected by an edge in the merge-graph must be updated after a merge. More formally, the time complexity of merging two nodes u and v is roughly $O(d^{merge}_{avg} \cdot (d_u + d_v))$, where d^{merge}_{avg} is the average degree of nodes

in the merge-graph. This process is also highly parallelizable. Details on the exact updates to the coarse graph and a highly scalable merging procedure are provided in Appendix B.3. Additionally, we propose a scalable cost computation and update in Appendix B.5. An empirical study on the effect of the number of nodes simultaneously merged at this step on the coarsening time and performance is provided in the following section.

5 EXPERIMENTS

We perform experiments to answer the following research questions:

- **RQ1**: At varying coarsening ratios, how do our ConvMatch variants compare to the baselines in terms of summarization time, as well as GNN training runtime and memory requirements?
- **RQ2**: How effective are the GCNs trained on graphs summarized by our ConvMatch variants (vs. baselines) in downstream node classification and link prediction tasks?
- **RQ3**: What is the effect of the number of merges k at each level of coarsening in ConvMatch on the summarization time and downstream task performance?

All reported results are fully reproducible, with code and data available at: github.com/amazon-science/convolution-matching.

Datasets. In our experiments, we use six datasets summarized in Table 2. Citeseer and Cora are citation networks which we use for both node classification (NC) and link prediction (LP) tasks [31]. Additionally, we use four datasets from the Open Graph Benchmark (OGB) [14]. OGBNArxiv (Arxiv) and OGBLCitation2 (Cit2) are also citation networks and are curated for testing NC and LP performance, respectively. OGBLCollab (Coll) is a collaboration network with the LP task of ranking true collaborations higher than false collaborations. Finally, OGBNProducts (Prod) is a product co-purchasing network with the NC task of predicting product categories. Prediction performance for NC tasks is measured using accuracy. The prediction performance for LP tasks is measured using AUC on Citeseer and Cora, Hits@50 on Coll, and MRR on Cit2.

Table 2: Table of dataset statistics and task (NC: node classification; LP: link prediction).

Dataset	Task	Nodes	Edges	Features
Citeseer	NC / LP	3,327	4,732	3,703
Cora	NC / LP	2,708	5,429	1,433
OGBNArxiv	NC	169,343	1,166,243	128
OGBLCollab	LP	235,868	1,285,465	128
OGBLCitation2	LP	2,927,963	30,561,187	128
OGBNProducts	NC	2,449,029	61,859,140	100

Baselines. We additionally evaluate the performance of six baselines: (a) three coreset methods: *Random Node Sampling* (RS), *Herding* [38], and *KCenter* [32]; (b) two graph condensation methods: *graph condensdation* (GCond) [18] and *one-step gradient matching* (DosCond) [17]; and (c) one coarsening method: *Variation Neighborhoods* (VN) [15, 25]. For implementation and hyperparameter details see Appendix C.

GCN Architectures and Hyperparameters. All experiments are performed using a GCN model. We give the details of the GCN architectures and hyperparameters for summarization baselines and ConvMatch and A-ConvMatch in Appendix C. The merge batch sizes of our algorithm arefi xed for each dataset for experiments in Section 5.1 and Section 5.2 and an ablation study along withfinal hyperparameter settings are provided in Appendix C.

5.1 (RQ1) Runtime and Memory Efficiency

First, we evaluate the efficiency of our proposed ConvMatch variants and the baseline graph summarization algorithms, as well as the training time and memory efficiency of the GNNs trained on the resultant graph summaries.

Graph Summarization Time. We compare the graph summarization time of ConvMatch and A-ConvMatch to baselines at varying coarsening ratios for each dataset and task. The average time across 5 rounds of summarization for all datasets are shown in Figure 1. First, we observe for Cora A-ConvMatch is over 5× faster than ConvMatch. For this reason we chose to only run A-ConvMatch on the larger OGB datasets. A-ConvMatch is consistently faster than all other baseline graph summarization methods on the larger OGB datasets. The VN baseline is faster than A-ConvMatch in Cora, however we empirically demonstrate this method struggles to scale to larger graphs (e.g., it timed out after 24 hours on the OGBNProducts dataset). A-ConvMatch is faster than the condensation and coreset baselines as it does not compute gradients of a GNN model with the full graph during summarization. Finally, we note that the coarsening methods have the additional advantage of being bottom-up multi-level approaches, and thus the time required to reach the coarsening ratio $r = 0.1\%$ includes the time required to reach the ratio $r = 1\%$ and $r = 10\%$ and so on. On the other hand, creating the synthetic graphs with the condensation methods for two different ratios is two separate procedures and the work to reach one ratio is not obviously usable to reach another. This property could be leveraged in GNN learning curriculums or hyperparameter exploration.

GNN Training Runtime and Memory Efficiency. We examine the amount of GPU memory used and the time required to complete afi xed number of training epochs for each dataset at varying coarsening ratios. Table 3 shows the average total time and maximum amount of GPU memory used across 5 rounds of training on a graph coarsened using A-ConvMatch. Table 3 shows there is a significant decrease in the amount of GPU memory and time required to complete training on a coarsened graph. The results are most notable on the largest datasets: OGBLCitation2 and OGBNProducts, where the amount of memory required to compute the batch gradient for the GCN exceeded the $120GB$ of GPU memory available on our machine.

5.2 (RQ2) Downstream Task Prediction Performance

We now compare the prediction performance of GCNs trained on graphs summarized using ConvMatch, A-ConvMatch, and baseline summarization methods at varying coarsening ratios. Link prediction and node classification performance of the trained GCNs are reported in Table 4a and Table 4b, respectively. ConvMatch or

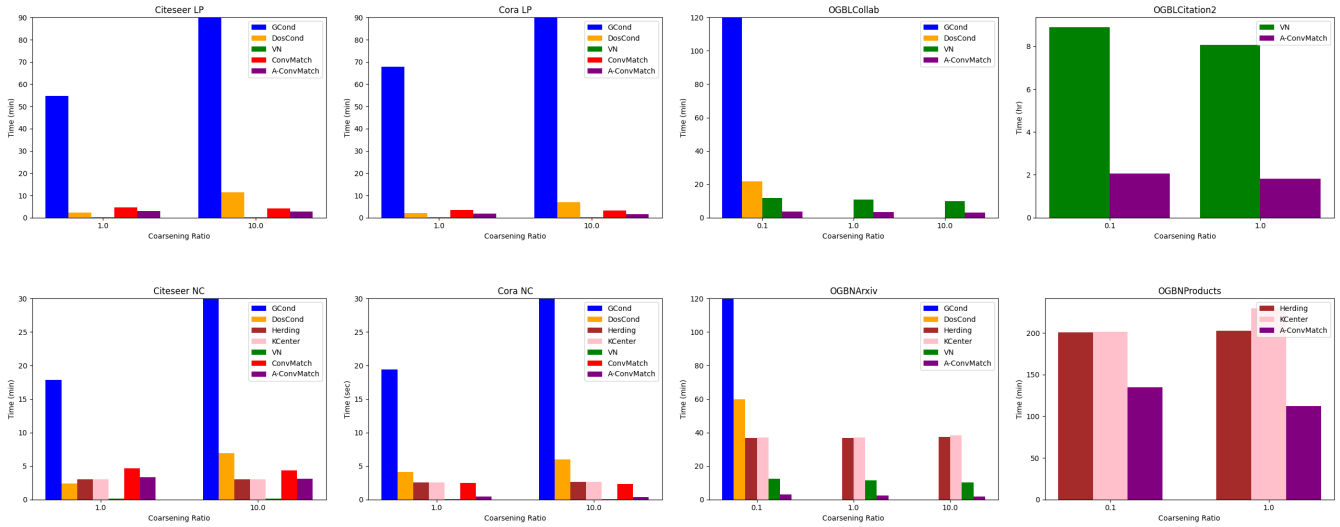

Figure 1: Plots of graph summarization times at multiple coarsening ratios for all datasets and tasks. ConvMatch and A-ConvMatch are fast summarization algorithms when compared to baselines.

Table 3: Average time, rounded to the nearest minute, and GPU memory, rounded to the nearest GB, required to complete all training epochs at varying coarsening ratios.

(a) Link Prediction

Dataset	Ratio	Time (min)	Memory (GB)
Citeseer	1	3	2
	10	3	3
	100	10	12
Cora	1	4	1
	10	4	1
	100	12	7
Coll	0.1	6	9
	1	15	9
	10	40	12
	100	210	70
Cit2	0.1	60	22
	1	660	40
	100	> 1440	> 120

(b) Node Classification

Dataset	Ratio	Time (min)	Memory (GB)
Citeseer	1	4	1
	10	5	1
	100	5	1
Cora	1	2	1
	10	4	1
	100	4	1
Arxiv	0.1	5	7
	1	10	7
	10	20	8
	100	170	20
Prod	0.1	60	38
	1	150	45
	100	> 1440	> 120

A-ConvMatch is consistently among the top three performing summarization methods for both tasks. Furthermore, A-ConvMatch achieves the best overall performance, as indicated by the lowest average rank.

Table 4a shows ConvMatch and A-ConvMatch are significantly better at creating summarized graphs for training a GCN for link prediction compared to alternative summarization methods. Notably, GCN's trained on ConvMatch and A-ConvMatch summarized graphs achieve up to 90% of the link prediction performance at a coarsening ratio of $r = 1\%$ in Citeseer and Cora, respectively. Moreover, A-ConvMatch achieves a nearly $2\times$ improvement over the best performing baseline in Cit2 at $r = 0.1\%$ and over a 20% point improvement at $r = 1\%$. A possible explanation for this is that ConvMatch and A-ConvMatch merge nodes that are equivalent or similar with respect to the GCN convolution operation, which captures both nodes attributes and structural properties. For this reason the summarized graph contains supernodes that can be used to create good positive and negative link training examples.

Table 4b shows Herding and KCenter methods perform well for the node classification task on the larger OGB datasets. These methods do however have a trade off as they initiallyfi t a GNN

Table 4: Link prediction and node classification performance at varying coarsening levels. The top three performing scores are highlighted as: First, Second, Third. Average ranks are reported for methods that were ran on all datasets and coarsening ratios. Our A-ConvMatch approach performs the best across datasets and coarsening ratios, as indicated by the lowest average rank in both link prediction and node classification tasks (1.4 and 2.5, resp.).

(a) Link Prediction

		Sumarizer					
	Ratio	**RS**	**GCond**	**DosCond**	**VN**	**ConvMatch** (ours)	**A-ConvMatch** (ours)
Citeseer	1%	67.41 ± 1.26	65.01 ± 1.09	63.54 ± 6.36	80.26 ± 4.51	**87.72 ± 1.38**	87.68 ± 1.24
	10%	69.11 ± 2.09	72.65 ± 6.78	53.06 ± 12.28	88.55 ± 0.77	**90.43 ± 0.49**	88.41 ± 0.97
	100%	**91.57 ± 0.55**					
Cora	1%	70.91 ± 3.67	67.08 ± 0.64	71.60 ± 0.35	75.19 ± 3.01	66.33 ± 2.35	**78.63 ± 1.55**
	10%	72.84 ± 2.15	66.19 ± 0.39	60.21 ± 0.52	81.81 ± 2.06	**83.51 ± 1.01**	83.30 ± 1.45
	100%	**85.04 ± 0.68**					
Coll	.1%	7.22 ± 0.97	7.61 ± 0.41	1.15 ± 0.88	6.60 ± 1.47	-	**9.14 ± 2.07**
	1%	11.41 ± 0.35	OOM	OOM	5.45 ± 1.95	-	**26.25 ± 1.84**
	10%	15.80 ± 3.11	OOM	OOM	24.33 ± 1.61	-	**35.94 ± 1.86**
	100%	**44.08 ± 0.94**					
Cit2	.1%	9.86 ± 0.01	OOM	OOM	10.43 ± 2.71	-	**30.42 ± 0.63**
	1%	9.87 ± 0.01	OOM	OOM	38.95 ± 4.83	-	**60.34 ± 1.15**
	100%	**84.74 ± 0.00**					
Avg. Rank		3.4	–	–	2.6	–	**1.4**

(b) Node Classification

		Sumarizer							
	Ratio	**RS**	**KCenter**	**Herding**	**GCond**	**DosCond**	**VN**	**ConvMatch** (ours)	**A-ConvMatch** (ours)
Citeseer	1%	19.22 ± 6.85	59.02 ± 2.25	62.80 ± 1.33	68.22 ± 2.08	**71.36 ± 1.27**	35.92 ± 3.76	65.40 ± 4.12	67.68 ± 2.87
	10%	28.00 ± 9.15	54.70 ± 3.62	53.5 ± 1.68	**70.10 ± 3.43**	68.76 ± 0.97	53.88 ± 7.36	69.84 ± 1.58	68.46 ± 1.61
	100%	**71.40 ± 0.35**							
Cora	1%	18.76 ± 9.67	59.92 ± 1.67	63.74 ± 2.24	77.44 ± 1.90	78.40 ± 0.83	31.96 ± 11.02	72.60 ± 3.11	72.30 ± 2.90
	10%	27.52 ± 9.89	56.12 ± 7.46	64.10 ± 2.12	80.02 ± 0.92	78.64 ± 1.87	59.22 ± 4.90	79.82 ± 0.60	80.12 ± 1.09
	100%	**81.02 ± 0.19**							
Arxiv	0.1%	42.45 ± 3.62	49.79 ± 2.87	**58.79 ± 0.90**	48.10 ± 3.27	21.69 ± 4.30	28.90 ± 5.79	-	54.82 ± 2.67
	1%	59.97 ± 1.58	61.99 ± 1.24	61.42 ± 1.34	OOM	OOM	53.61 ± 2.76	-	**64.08 ± 0.39**
	10%	67.50 ± 0.63	**68.25 ± 0.25**	67.75 ± 0.58	OOM	OOM	67.12 ± 1.26	-	67.51 ± 0.47
	100%	**72.14 ± 0.18**							
Prod	0.1%	48.20 ± 1.57	60.88 ± 1.31	**66.14 ± 0.71**	OOM	OOM	TIMEOUT	-	62.36 ± 0.84
	1%	66.57 ± 1.47	69.66 ± 0.31	**71.79 ± 0.45**	OOM	OOM	TIMEOUT	-	68.20 ± 0.36
	100%	**75.64 ± 0.00***							
Avg. Rank		5.9	3.8	3.2	-	-	-	-	**2.5**

using the complete graph to obtain node embeddings resulting in a slower summarization time, as shown in the previous section. Condensation methods perform extremely well on Citeseer and Cora, however they have difficulty creating larger summarized graphs hitting out of memory errors on the OGB datasets because they compute a full gradient using the original graph. Furthermore, the results wefi nd for GCond on OGBNArxiv differ from those reported in Jin et al. (2022a) [17] as the GCN architecture in the summarizer and the GCN being trained are not exactly matched. The authors mention this behavior in their appendix section C.5.

Finally, we observe A-ConvMatch is the most reliable summarizer for node classification with the best average rank of 2.3. The next best method in terms of average rank is Herding at 3.3.

5.3 Ablation for ConvMatch Merge Batch Size

First, we analyze the effect the number of node pairs simultaneously merged at each level of coarsening in ConvMatch has on the summarization time and prediction performance of the proposed approach. We summarize the training graph for each of the 6 datasets to the coarsening ratio $r = 1.0\%$ and train a GCN using the

Table 5: ConvMatch graph summarization time in seconds and prediction performance at varying merge batch sizes and a coarsening ratio $r = 1.0\%$.

(a) Link Prediction

Dataset	Batch Size	Time (sec)	Perf. Valid.	Perf. Test
Citeseer	1	178.03	85.42	87.68
	10	37.95	86.78	86.44
	100	8.46	88.19	88.25
Cora	1	105.46	74.21	74.93
	10	25.73	76.20	78.44
	100	4.72	69.98	67.53
Coll	100	777.77	22.59	26.23
	1,000	259.11	22.78	24.63
	10,000	205.53	23.70	26.25
Cit2	10,000	6,555	60.05	60.05
	100,000	4,740	55.21	55.24

(b) Node Classification

Dataset	Batch Size	Time (sec)	Perf. Valid.	Perf. Test.
Citeseer	1	198.84	66.92	66.08
	10	36.78	72.40	68.90
	100	8.03	65.20	62.30
Cora	1	101.74	74.40	74.70
	10	27.16	70.20	72.30
	100	4.89	71.40	71.20
Arxiv	100	624.91	63.94	63.46
	1,000	194.73	62.25	61.07
	10,000	149.91	64.05	64.12
Prod	10,000	11,285	85.95	68.57
	100,000	6,729	86.18	68.20

summarized graph. The summarization time in seconds and prediction performance are reported in Table 5 for both link prediction and node classification. Wefi nd increasing the merge batch size has a limited effect on the prediction performance across all datasets and for both NC and LP tasks. However, the summarization time improves considerably.

6 CONCLUSION AND FUTURE WORK

We introduced the ConvMatch graph summarization algorithm and a principled approximation, A-ConvMatch, which preserve the output of graph convolution. Our methods were empirically proven to produce summarized graphs that can be used tofi t GCN model parameters with significantly lower memory consumption, faster training times, and good prediction performance on both node classification and link prediction tasks, afi rst for summarization for scalable GNN training. Notably, our model is consistently a top-performing summarization method and achieves up to 20% point improvements on link prediction tasks. There are exciting next steps to this research, including extending the idea of convolution matching to heterogeneous graphs and developing GNN training algorithms that leverage multiple levels of a graph coarsening. Moreover, although in this work we focus on motivating our approach and providing a comprehensive evaluation for GCN's, no change to our coarsening algorithm is necessary to apply it to different GNN architectures. Future research on applying the same algorithmic framework of ConvMatch but specialized to preserving the operations applied in another GNN model is promising.

REFERENCES

[1] Esra Akbas and Mehmet Emin Aktas. 2019. Network Embedding: On Compression and Learning. In *IEEE Transactions on Big Data*.

[2] Ondrej Bohdal, Yongxin Yang, and Timothy Hospedales. 2020. Flexible Dataset Distillation: Learn Labels Instead of Images. In *Arxiv preprint*.

[3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. 2014. Spectral networks and Locally Connected Networks on Graphs. In *International Conference on Learning Representations (ICLR)*.

[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems (NIPS)*.

[5] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. 2020. GraphZoom: A Multi-Level Spectral Approach for Accurate and Scalable Graph Embedding. In *International Conference on Learning Representations (ICLR)*.

[6] Cody Dunne and Ben Schneiderman. 2013. Motif Simplification: Improving Network Visualization Readability with Fan, Connector, and Clique Glyphs. In *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*.

[7] Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Racke, Inbal Talgam-Cohen, and Kunal Talwar. 2014. Vertex Sparsifiers: New Results From Old Techniques. *SIAM J. Comput.* 43, 4 (2014), 1239–1262.

[8] Matthew Fahrbach, Gramoz Goranci, Richard Peng, Sushant Sachdeva, and Chi Wang. 2020. Faster Graph Embeddings Via Coarsening. In *International Conference on Machine Learning (ICML)*.

[9] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[10] Swapnil Gandhi and Anand Padmanabha Iyer. 2021. P3: Distributed deep graph learning at scale. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 551–568.

[11] Sariel Har-Peled and Akash Kushal. 2005. Smaller Coresets for K-Mediam and K-Means Clustering. In *ACM Annual Symposium on Computational Geometry (ACG)*.

[12] David Harel and Yehuda Koren. 2002. A Fast Multi-scale Method for Drawing Large Graphs. *Journal of Graph Algorithms and Applications* 6, 3 (2002), 179–202.

[13] 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. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[14] Weihua Hu, Matthias Fey, Marinka Zitnk, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *Conference on Neural Information Processing Systems (NeurIPS)*.

[15] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. 2021. Scaling Up Graph Neural Networks Via Graph Coarsening. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*.

[16] Peng Jiang and Masuma Akter Rumi. 2021. Communication-efficient sampling for distributed training of graph convolutional networks. *arXiv preprint arXiv:2101.07706* (2021).

[17] Wei Jin, Xianfeng Tang, Haoming Jiang, Zheng Li, Danqing Zhang, Jiliang Tang, and Bing Yin. 2022. Condensing Graphs via One-Step Gradient Matching. In

ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD).

[18] Wei Jin, Lingxiao Zhao, Schichang Zhang, Yozen Liu, Jiliang Tang, and Neil Shah. 2022. Graph Condensation for Graph Neural Networks. In *International Conference on Learning Representations (ICLR).*

[19] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR).*

[20] Manoj Kumar, Anurag Sharma, Shashwat Saxena, and Sandeep Kumar. 2023. Featured Graph Coarsening with Similarity Guarantees. In *International Conference on Machine Learning (ICML).*

[21] Ron Levie, Federico Monti, Xavier Bresson, and Michael M. Bronstein. 2017. CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters. *IEEE Transactions on Signal Processing* 67, 1 (2017), 97–109.

[22] Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. 2021. MILE: A Multi-Level Framework for Scalable Graph Embedding. In *International AAAI Conference on Web and Social Media (ICWSM).*

[23] Mengyang Liu, Shanchuan Li, Xinshi Chen, and Le Song. 2022. Graph Condensation via Receptive Field Distribution Matching. In *Arxiv preprint.*

[24] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. 2019. Graph Summarization Methods and Applications: A Survey. *Comput. Surveys* 51, 62 (2019).

[25] Andreas Loukas. 2019. Graph Reduction with Spectral and Cut Guarantees. *Journal of Machine Learning Research* 20, 116 (2019), 1–42.

[26] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. 2020. Coreset for Data-efficient Training of Machine Learning Models. In *International Conference on Machine Learning (ICML).*

[27] Ankur Moitra. 2009. Approximation Algorithms for Multicommodity-type Problems with Guarantees Independent of the Graph Size. In *IEEE Symposium on Foundation of Computer Science.*

[28] Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. 2021. Dataset Meta-Learning from Kernel Ridge-Regression. In *International Conference on Learning Representations (ICLR).*

[29] Manish Purohit, B. Aditya Prakash, Chanhyun Kang, Yao Zhang, and V. S. Subrahmanian. 2014. Fast Influence-based Coarsening for Large Networks. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD).*

[30] Morteza Ramezani, Weilin Cong, Mehrdad Mahdavi, Mahmut Kandemir, and Anand Sivasubramaniam. 2021. Learn Locally, Correct Globally: A Distributed Algorithm for Training Graph Neural Networks. In *International Conference on Learning Representations.*

[31] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. *AI Magazine* 29, 3 (2008).

[32] Ozan Sener and Silvio Savarses. 2018. Active Learning for Convolutional Neural Networks: A Core-Set Approach. In *International Conference on Learning Representations (ICLR).*

[33] Ilia Sucholutsky and Matthias Schonlau. 2020. Soft-Label Dataset Distillation and Text Dataset Distillation. In *Arxiv preprint.*

[34] Ivor W. Tsang, James T. Kwok, and Pak-Min Cheung. 2005. Core Vector Machines: Fast SVM Training on Very Large Data Sets. *Journal of Machine Learning Research* 6 (2005), 363–392.

[35] Chris Walshaw. 2006. A Multilevel Algorithm for Force-Directed Graph Drawing. *Journal of Graph Algorithms and Applications* 7, 3 (2006), 253–285.

[36] 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).

[37] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. 2018. Dataset Distillation. In *Arxiv preprint.*

[38] Max Welling. 2009. Herding Dynamical Weights to Learn. In *International Conference on Machine Learning (ICML).*

[39] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr., Christopher Fifty, Tao Yu, and Kilian Q. Wienberger. 2019. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning (ICML).*

[40] Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Maosong Sun, Zhichong Fang, Bo Zhang, and Leyu Lin. 2020. COSINE: Compressive Network Embedding on Large-Scale Information Networks. In *IEEE Transactions on Knowledge and Data Engineering.*

[41] Bo Zhao and Hakan Bilen. 2021. Dataset Condensation with Differentiable Siamese Augmentation. In *International Conference on Machine Learning (ICML).*

[42] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. 2021. Dataset Condensation with Gradient Matching. In *International Conference on Learning Representations (ICLR).*

[43] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. 2020. Distdgl: distributed graph neural network training for billion-scale graphs. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3).* IEEE, 36–44.

[44] Da Zheng, Xiang Song, Chengru Yang, Dominique LaSalle, and George Karypis. 2021. Distributed Hybrid CPU and GPU training for Graph Neural Networks on Billion-Scale Graphs. *arXiv preprint arXiv:2112.15345* (2021).

[45] Jiong Zhu, Aishwarya Naresh Reganti, Edward W Huang, Charles Andrew Dickens, Nikhil Rao, Karthik Subbian, and Danai Koutra. 2023. Simplifying Distributed Neural Network Training on Massive Graphs: Randomized Partitions Improve Model Aggregation. In *ICML Workshop on Localized Learning (LLW).*