Towards Efficient Training of Graph Neural Networks: A Multiscale Approach

Eshed Gal *eshedg@post.bgu.ac.il*
Department of Computer Science
Ben-Gurion University of the Negev

Moshe Moshe Eliasof *me532@cam.ac.uk*
Department of Applied Mathematics and Theoretical Physics
University of Cambridge

Carola-Bibiane Schönlieb *cbs31@cam.ac.uk*
Department of Applied Mathematics and Theoretical Physics
University of Cambridge

Eldad Haber *ehaber@eoas.ubc.ca*
Department of Earth, Ocean and Atmospheric Sciences
University of British Columbia

Eran Treister *erant@bgu.ac.il*
Department of Computer Science
Ben-Gurion University of the Negev

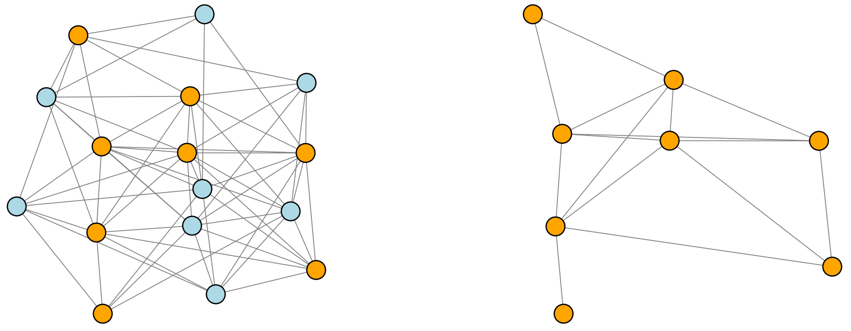
Abstract

Graph Neural Networks (GNNs) have emerged as a powerful tool for learning and inferring from graph-structured data, and are widely used in a variety of applications, often considering large amounts of data and large graphs. However, training on such data requires large memory and extensive computations. In this paper, we introduce a novel framework for efficient multiscale training of GNNs, designed to integrate information across multiscale representations of a graph. Our approach leverages a hierarchical graph representation, taking advantage of coarse graph scales in the training process, where each coarse scale graph has fewer nodes and edges. Based on this approach, we propose a suite of GNN training methods: such as coarse-to-fine, sub-to-full, and multiscale gradient computation. We demonstrate the effectiveness of our methods on various datasets and learning tasks.

1 Introduction

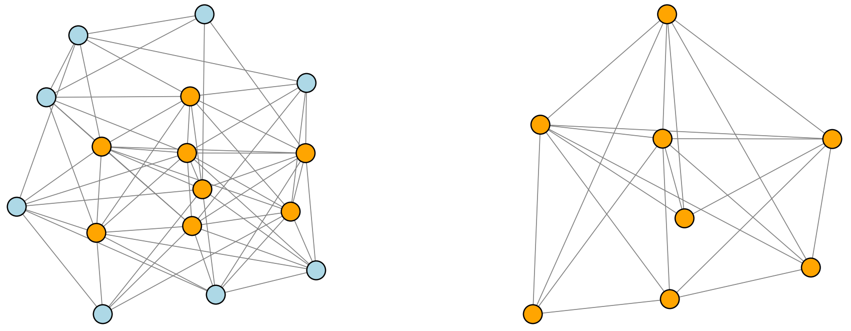
Large-scale graph networks arise in diverse fields such as social network analysis Kipf & Welling (2016); Defferrard et al. (2016), recommendation systems Tang et al. (2018), and bioinformatics Jumper et al. (2021), where relationships between entities are crucial to understanding system behavior Abadal et al. (2021). These problems involve the processing of large graph data sets necessitating scalable solutions. Some examples of those problems include identifying community structures in social networks Hamilton et al. (2017), optimizing traffic flow in smart cities Geng et al. (2019), or studying protein-protein interactions in biology Xu et al. (2024). In all these problems, efficient processing of graph data is required for practical reasons.

In recent years, many Graph Neural Networks (GNNs) algorithms have been proposed to solve large graph problems, such as GraphSage Hamilton et al. (2017). However, despite their success, when the involved graph is of a large scale, with many nodes and edges, it takes a long time and many resources to train the network Galmés et al. (2021).



(a) Fine original graph (b) Coarse graph

Figure 1: Coarse graph using **random** pooling. The Orange nodes in the fine graph are selected to the coarse graph.



(a) Fine original graph (b) Coarse graph

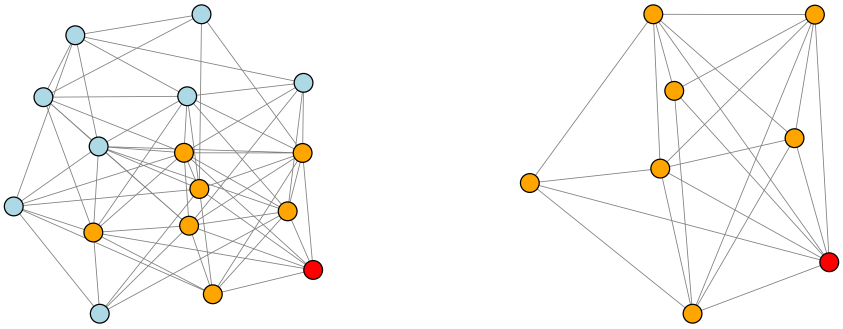
Figure 2: Coarse graph using **Topk** pooling, meaning nodes with highest degrees are chosen. Orange nodes in the fine graph are the nodes selected to the coarse graph.

In this paper, we study and develop multiscale training methodologies, to improve the training efficiency of GNNs. To achieve that, we focus on the standard approach for implementing GNNs, that is, the Message-Passing (MP) scheme Gilmer et al. (2017). Mathematically, this approach is equivalent to the multiplication of the graph adjacency matrix $\mathbf{A}$ with node feature maps $\mathbf{X}^{(l)}$, which is then multiplied by a learned channel-mixing weight matrix $\mathbf{W}^{(l)}$, followed by an application of an element-wise nonlinear activation function σ, as follows:

$$\mathbf{X}^{(l+1)} = \sigma(\mathbf{A}\mathbf{X}^{(l)}\mathbf{W}^{(l)}). \tag{1}$$

Assuming that the adjacency matrix is sparse with an average sparsity of k neighbors per node, and a channel-mixing matrix of shape $c \times c$, we now focus on the the total multiplications in the computation of Equation (1), to which we refer to as *'work'*. In this case, the work required in a single MP layer is of order $\mathcal{O}(n \cdot k \cdot c^2)$, where n is the number of the nodes in the graph, and c is the number of channels. For large or dense graphs, i.e., n or k are large, the MP in Equation (1) requires a high computational cost. Any algorithm that works with the entire graph at once is therefore bound to face the cost of this matrix-vector product.

A possible solution to this high computational cost involves the utilization of a *multiscale framework*, which requires the generation of surrogate problems to be solved (Kolaczyk & Nowak, 2003). If the solution of the surrogate problem approximates the solution of the fine (original) scale problem, then it can be used as an initial starting point, thus reducing the work required to solve the optimization problem. In the context


(a) Fine original graph

(b) Coarse graph

Figure 3: Coarse graph using **subgraph** pooling. Orange nodes in the fine graph are the nodes selected to the coarse graph, and the red node is the root node for the ego-network.

of graphs, such a multiscale representation of the original problem can be achieved using *graph coarsening*. Graph coarsening algorithms are techniques that reduce the size of graphs while preserving their essential properties (Cai et al., 2021). These methods aim to create smaller, more computationally manageable graphs by choosing a subset of the nodes or the edges, based on certain policies, or by using a subgraph of the original graph, and may be implemented in various ways.

We demonstrate the generation of a coarse graph using random pooling in Figure 1 and Topk pooling (which is based on the nodes with high degrees in the graph) in Figure 2. As for choosing a subgraph, this can be implemented using node-based selection methods, such as ego-networks (Gupta et al., 2014) or node marking and deletion (Frasca et al., 2022), which offer straightforward ways to extract subgraphs based on specific nodes of interest. We show an example of subgraph coarsening in Figure 3.

Our approach. While coarsening and subgraph methods have been in use for various purposes, from improved architectures Gao & Ji (2019), to enhanced expressiveness Bevilacqua et al. (2022; 2024), to the best of our knowledge, they were not considered in the context of efficient training of GNNs. Therefore, in this work, we explore three novel training mechanisms to design surrogate processes that approximate the original training problem with a fraction of the computational cost, using a multiscale approach. All the methods attempt to use the full graph with N nodes as little as possible, and focus on training the network using coarse-scale graphs, thus reducing computational cost. The three algorithms, detailed in Section 4, are as follows:

- **Coarse-to-Fine.** A coarse-to-fine algorithm approximates the optimization problem on smaller, pooled graphs, reducing costs. If the coarse-scale solution matches the fine-scale one, it can serve as an effective initialization for the fine-scale optimization. Since gradient-based methods converge faster when initialized near the solution (Arora et al., 2018; Boyd & Vandenberghe, 2004), this approach reduces the overall number of iterations required.
- **Sub-to-Full.** This approach leverages a sequence of subgraphs that progressively grow to the original fine-scale graph. Like coarsening, each subgraph is significantly smaller, enabling faster optimization. Subgraphs can be generated based on node distances (if available) or ego-network methods Gupta et al. (2014).
- **Multiscale Gradients Computation.** Building on the previous methods and inspired by Multiscale Monte Carlo Giles (2001), which is widely used for solving stochastic differential equations, we introduce a mechanism to approximate fine-scale gradients at a fraction of the computational cost, achieved by combining gradients across scales to efficiently approximate the fine-scale gradient.

Contributions. The main contribution of this paper is the design of *novel graph multiscale optimization algorithms* to obtain computationally efficient GNN training methods that retain the performance of standard gradient-based training of GNN. To demonstrate their effectiveness, we benchmark the obtained performance and reduce computational costs on multiple graph learning tasks, from transductive node classification to inductive tasks like shape segmentation.

2 Related Work

We discuss related topics to our paper, from hierarchical graph pooling to multiscale methods and their learning.

Graph Pooling. Graph pooling can be performed in various ways, as reviewed in Liu et al. (2022). Popular pooling techniques include DiffPool (Ying et al., 2018) MaxCutPool (Abate & Bianchi, 2024), graph wavelet compression Eliasof et al. (2023), as well as Independent-Set Pooling (Stanovic et al., 2024), all aiming to produce a coarsened a graph that bears similar properties to the original graph. Wang et al. (2020) and Wang et al. (2024) introduce the concept of subgraph pooling to GNNs. Bianchi & Lachi (2023) study the expressive power of pooling in GNNs. Here, we use graph pooling mechanisms, as illustrated in Figure 1–Figure 3, as core components for efficient training.

General Multiscale Methods. The idea of Coarse-to-Fine for the solution of optimization problems has been proposed in Brandt (1984), while a more detailed analysis of the idea in the context of optimization problems was introduced in Nash (2000). Multiscale approaches are also useful in the context of Partial Differential Equations (Guo & Li, 2024; Li et al., 2024b), as well as for various applications including flood modeling (Bentivoglio et al., 2024). Lastly, Monte-Carlo methods (Hastings, 1970) have also been adapted to a multiscale framework (Giles, 2001), which can be used for various applications such as electrolytes and medical simulations (Liang et al., 2015; Klapproth et al., 2021). In this paper, we draw inspiration from multiscale methods to introduce a novel and efficient GNN training framework.

Multiscale Learning. In the context of graphs, the recent work in Shi et al. (2022) has shown that a matching problem can be solved using a coarse-to-fine strategy, and Cai et al. (2021) proposed different coarsening strategies. Other innovations include the recent work of Li et al. (2024a), whose goal is to mitigate the lack of labels in semi-supervised and unsupervised settings. We note that it is important to distinguish between *algorithms that use a multiscale structure as a part of the network*, such as Unet (Ronneberger et al., 2015) and Graph-Unet (Gao & Ji, 2019), and *multiscale training that utilizes multiscale for training.* In particular, the latter, which is the focus of this paper, can be potentially applied to any GNN architecture.

3 Preliminaries and Notations

We now provide related graph-learning notations and preliminaries that we will use throughout this paper. Let us consider a graph defined by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of n nodes and $\mathcal{E}$ is a set of m edges. Alternatively, a graph can be represented by its adjacency matrix $\mathbf{A}$, where the (i, j)-th entry is 1 if an edge between nodes i and j exists, and 0 otherwise. We assume that each node is associated with a feature vector with c channels as $\mathbf{x}_i \in \mathbb{R}^c$, and $\mathbf{X} = [\mathbf{x}_0 \dots, \mathbf{x}_{n-1}] \in \mathbb{R}^{n \times c}$ as the node features matrix. We denote by $\mathbf{y}_i \in \mathbb{R}^d$ the i-th node label, and $\mathbf{Y} = [\mathbf{y}_0, \dots, \mathbf{y}_{n-1}] \in \mathbb{R}^{n \times d}$ the label matrix.

In a typical graph learning problem, we are given graph data $(\mathcal{G}, \mathbf{X}, \mathbf{Y})$, and our goal is to learn a function $f(\mathbf{X}, \mathcal{G}; \boldsymbol{\theta})$, with learnable parameters $\boldsymbol{\theta}$ such that

$$\mathbf{Y} \approx f(\mathbf{X}, \mathcal{G}; \boldsymbol{\theta}). \tag{2}$$

For instance, in the well-known Graph Convolution Network (GCN) (Kipf & Welling, 2016), each layer is defined using

Equation (1), such that $\mathbf{X}^{(l+1)}$ is the output of the l-th layer and serves as the input for the $(l + 1)$-th layer. For a network with L layers, we denote by $\boldsymbol{\theta} = \{\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}\}$ the collection of all learnable weight matrices within the network.

It is important to note that the matrix $\mathbf{W}^{(l)}$ is independent of the size of the adjacency and node feature matrices, and its dimensions only correspond to the feature space dimensions of the input and output layer. In terms of computational costs, a the l-th GCN layer requires the multiplication of the three matrices $\mathbf{A}, \mathbf{X}^{(l)}$ and $\mathbf{W}^{(l)}$.

We note that the number of multiplications computed in the l-th GCN layer is directly affected by the size of the node features matrix $\mathbf{X}^{(l)}$, and the graph adjacency matrix $\mathbf{A}$. This means that for a very large graph $\mathcal{G}$, each layer requires significant computational resources, both in terms of time and memory usage. These computational costs come into effect both during the training process and at the inference stage. In this paper, we directly address these associated costs by leveraging GNN layers with smaller matrices $\mathbf{A}_r$ and $\mathbf{X}_r^{(l)}$, yielding trained networks whose downstream performance is similar to those that would have been achieved had we used the original input graph and features, while significantly reducing the computational costs.

Lastly, we note that, as shown in Section 6, our approach is general and applicable to various GNN architectures.

4 Efficient Training Framework for GNNs

Overarching goal. We consider the graph-learning problem in Equation (2), defined on a graph with n nodes. To reduce the cost of the learning process, we envision a sequence of R surrogate problems, defined by the set of data tuples $\{(\mathcal{G}_r, \mathbf{X}_r, \mathbf{Y}_r)\}_{r=1}^{R}$ and problems of the form

$$\mathbf{Y}_r \approx f(\mathbf{X}_r, \mathcal{G}_r; \boldsymbol{\theta}), \quad r = 1, \ldots, R, \tag{3}$$

where f is a GNN and $\mathcal{G}_r$ is a graph with n_r nodes, where $n = n_1 > n_2 > \ldots > n_R$. Each graph $\mathcal{G}_r$ is obtained by pooling the fine graph $\mathcal{G}$, using a smaller set of nodes compared to the previous (finer) graph $\mathcal{G}_{r-1}$.

$\mathbf{X}_r$ is the node features matrix of the graph $\mathcal{G}_r$, and $\mathbf{Y}_r$ is the corresponding label matrix. In what follows, we explore different ways to obtain $\mathcal{G}_r, \mathbf{X}_r$ and $\mathbf{Y}_r$ from the original resolution variables.

An important feature of GNNs is that the weights $\boldsymbol{\theta}$ of the network are independent of the input and output size. Each weight matrix $\mathbf{W}^{(l)}$ is of dimensions $c_{in} \times c_{out}$, for a $\mathbf{X}^{(l)} \in \mathbb{R}^{n \times c_{in}}$ node feature matrix of the layer l, and $\mathbf{X}^{(l+1)} \in \mathbb{R}^{n \times c_{out}}$ for the corresponding dimensions of the next layer $l+1$. This means that changing the number of nodes and edges in the graph will not affect the weight matrix dimensions while reducing the number of computational operations conducted in the learning process. That is, the set of parameters $\boldsymbol{\theta}$ can be used on any of the R problems in Equation (3). Let us reformulate the learning problem on the r-th level as follows:

$$\widehat{\boldsymbol{\theta}}_r = \operatorname{argmin}_{\boldsymbol{\theta}} \mathbb{E}[\mathcal{L}\left(f(\mathbf{X}_r, \mathcal{G}_r; \boldsymbol{\theta}), \mathbf{Y}_r\right)], \tag{4}$$

where $\mathcal{L}$ is the loss function that measures the error in Equation (3), and the expectation is on the data $(\mathbf{X}_r, \mathbf{Y}_r)$. Our goal is to use the optimization problem in Equation (4) as a cheap surrogate problem to obtain an approximate solution (i.e., weights $\boldsymbol{\theta}$) of the original problem in Equation (2). We describe three methodologies to achieve this goal.

4.1 Coarse-to-Fine Training

One seemingly simple idea is to solve the r-th surrogate problem in Equation (4) and use the obtained weights to solve the problem on the $(r-1)$-th graph, that is $\mathcal{G}_{r-1}$, which is larger. This approach can be beneficial only if $\widehat{\boldsymbol{\theta}}_{r+1} \approx \widehat{\boldsymbol{\theta}}_r$, that is, the obtained optimal weights are similar under different scales, and if the total amount of work on the coarse scale is smaller than on the fine one, such that it facilitates computational cost reduction. The multiscale framework is summarized in Algorithm 1.

While Algorithm 1 is relatively simple, it requires obtaining the r-th surrogate problem by coarsening the graph, as well as stopping a criterion for the r-th optimization problem. Generally, in our coarse-to-fine method, we use the following graph coarsening scheme:

$$\mathbf{A}_r = \mathbf{P}_r^T \mathbf{A}^p \mathbf{P}_r, \tag{5}$$

Algorithm 1 Multiscale algorithm

```
Initialize weights $\boldsymbol{\theta}$ and network $f$.
for $r = R, R-1, \dots, 1$ do
    Compute the coarse graph $\mathcal{G}_r$ that has $n_r$ nodes.
    Compute the coarse node features & labels $(\mathbf{X}_r, \mathbf{Y}_r)$.
    Solve the optimization problem in Equation (4).
    Update the weights $\boldsymbol{\theta}$.
end for
```

where $\mathbf{P}_r \in \mathbb{R}^{n \times n_r}$ is a binary injection matrix that maps the fine graph with n nodes to a smaller graph with n_r nodes. We use $\mathbf{X}_r = \mathbf{P}_r^T \mathbf{X}$ as the coarse graph node features matrix and $\mathbf{Y}_r = \mathbf{P}_r^T \mathbf{Y}$ as the label matrix of the coarse graph.

Choosing the non-zero entries of the matrix $\mathbf{P}_r$, which are the nodes that remain in the graph after pooling, can be implemented in various ways. For example, $\mathbf{P}_r$ can be obtained by sampling random nodes of the fine graph $\mathcal{G}$, as illustrated in Figure 1, or with some desired pooling logic, such as choosing nodes of the highest degree in the graph, which is a method often described as $Topk$ pooling (Huang et al., 2015) and we present an illustration of it in Figure 2. The power p in Equation (5) controls the connectivity of the graph, where $p = 1$ is the original graph adjacency matrix, and using a value $p > 1$ will enhance the connectivity of the graph.

In Algorithm 1, we apply GNN layers such as in Equation (1), using the coarsened feature matrix $\mathbf{X}_r$ and its corresponding adjacency matrix $\mathbf{A}_r$, that is based on a desired pooling ratio ρ. We note that there is no re-initialization of the weight matrix $\mathbf{W}$ when transferring from coarse grids to finer ones, allowing a continuous learning process along the steps of the algorithm. As we show later in our experiments in Section 6, our coarse-to-fine training approach, summarized in Algorithm 1, is not restricted to a specific GNN architecture or a pooling technique.

4.2 Sub-to-Full Training

We now describe an additional method for efficient training on GNNs, that relies on utilizing subgraphs, called Sub-to-Full. At its core, the method is similar to Coarse-to-Fine, as summarized in Algorithm 1, using the same coarsening in Equation (5). However, its main difference is that we now choose the indices of $\mathbf{P}_r$ such that the nodes $\mathbf{X}_r$ form a subgraph of the original graph $\mathcal{G}$.

A subgraph can be defined in a few possible manners. One is by choosing a root node j, and generating a subgraph centered around that node. If the graph $\mathcal{G}$ has some geometric features, meaning each input node corresponds to a point in the Euclidean space $\mathbb{R}^d$, one might consider constructing a subgraph with n_r nodes to be the n_r nodes that are closest to the chosen center node in terms of Euclidean distance norm. Otherwise, we can consider a subgraph created using an ego-network Gupta et al. (2014) of $k - hop$ steps, such that we consider all neighbors that are k steps away from the central node. An illustration of this method is presented in Figure 3. The number of hops, k, is a hyperparameter of the algorithm process, and we increase it as we get from coarse levels to fine ones within the series of graph $\{\mathcal{G}_r\}_{r=1}^{R}$.

4.3 Multiscale Gradients Computation

In Section 4.1–Section 4.2 we presented two possible methods to facilitate the efficient training of GNNs via a multiscale approach. In what follows, we present a complementary method that builds on these ideas, as well as draws inspiration from Monte-Carlo optimization techniques (Giles, 2001), to further reduce the cost of GNN training.

We start with the observation that optimization in the context of deep learning is dominated by stochastic gradient-based methods (Kingma & Ba, 2014; Ruder, 2016). At the base of such methods, stands the gradient descent method, where at optimization step s we have:

$$\boldsymbol{\theta}_{s+1} = \boldsymbol{\theta}_s - \mu_s \mathbb{E}(\nabla \mathcal{L}\left(f(\mathbf{X}_1, \mathcal{G}_1; \boldsymbol{\theta}_s)\right). \tag{6}$$

However, when large datasets are considered, it becomes computationally infeasible to take the expectation of the gradient with respect to the complete dataset. This leads to the *stochastic* gradient descent (SGD) method, where the gradient is approximated using a smaller batch of examples of size B, obtaining the following weight update equation:

$$\boldsymbol{\theta}_{s+1} = \boldsymbol{\theta}_s - \frac{\mu_s}{B} \sum_{b=1}^{B} \nabla \mathcal{L}\left(f(\mathbf{X}_1^{(b)}, \mathcal{G}_1^{(b)}; \boldsymbol{\theta})\right), \tag{7}$$

where $\mathbf{X}_1^{(b)}, \mathcal{G}_1^{(b)}$ denote the node features and graph structure of the b-th element in the batch.

To compute the update in Equation (7), one is required to evaluate the loss $\mathcal{L}$ for B times on the finest graph $\mathcal{G}_1 = \mathcal{G}$, which can be computationally demanding. To reduce this computational cost, we consider the following telescopic sum identity:

$$\mathbb{E}\,\mathcal{L}_s^{(1)}(\boldsymbol{\theta}) = \mathbb{E}\,\mathcal{L}_s^{(R)}(\boldsymbol{\theta}) + \mathbb{E}\,\left(\mathcal{L}_s^{(R-1)}(\boldsymbol{\theta}) - \mathcal{L}_s^{(R)}(\boldsymbol{\theta})\right) + \ldots + \mathbb{E}\,\left(\mathcal{L}_s^{(1)}(\boldsymbol{\theta}) - \mathcal{L}_s^{(2)}(\boldsymbol{\theta})\right), \tag{8}$$

where $\mathcal{L}_s^{(r)} = \mathcal{L}\left(f(\mathbf{X}_r, \mathcal{G}_r; \boldsymbol{\theta}), \mathbf{Y}_r\right)$ represents the loss of the r-th scale at the s-th optimization step. Equation (8) induces a *hierarchical stochastic gradient descent* computation based on the set of multiscale graph $\{\mathcal{G}_r\}_{r=1}^{R}$.

The main idea of the hierarchical stochastic gradient descent method in Equation (8) is that each expectation in the telescopic sum can use a different number of realizations for its approximation. The assumption is that we can choose hierarchies such that

$$\left|\mathcal{L}_s^{(r-1)}(\boldsymbol{\theta}) - \mathcal{L}_s^{(r)}(\boldsymbol{\theta})\right| \leq \gamma_r \cdot \mathcal{L}_s^{(r-1)}(\boldsymbol{\theta}) \tag{9}$$

for some $1 > \gamma_r > \gamma_{r-1} > \ldots > \gamma_1 = 0$. To understand why this concept is beneficial, let us consider the discretization of the fine scale problem (i.e., for $\mathcal{G}_1$) in Equation (4) using M samples. Suppose that the error in approximating the expected loss at the first, finest scale $\mathbb{E}[\mathcal{L}_s^{(1)}]$ by the samples behaves as $e^{(1)} = \frac{C}{\sqrt{M}}$. Now, let us consider a 2-scale process, with M_1 samples on the finest resolution graph. The error in evaluating the loss on the second scale $\mathbb{E}[\mathcal{L}_s^{(2)}]$ with M_2 examples is given by $e^{(2)} = \frac{C}{\sqrt{M_2}}$ while the error in evaluating $\mathbb{E}[(\mathcal{L}_s^{(1)} - \mathcal{L}_s^{(2)})]$ is $e^{(1,2)} = \frac{C\gamma_1}{\sqrt{M_1}}$. Since the assumption is that $\gamma < 1$, we are able to choose M_2 and M_1 such that $e^{(1)} \approx e^{(2)} + e^{(1,2)}$. Hence, rather than using M finest graph resolution observations, we use only $M_1 < M$ such observations and sample the coarser problems instead.

More broadly, we can generalize the case of a 2-scale optimization process to an R multiscale process, such that an approximation of the fine-scale loss $\mathcal{L}_s^{(1)}$ is obtained by:

$$\mathbb{E}\,\mathcal{L}_s^{(1)}(\boldsymbol{\theta}) \approx \tfrac{1}{M_R}\sum \mathcal{L}_j^{(R)}(\boldsymbol{\theta}) + \tfrac{1}{M_{R-1}}\sum \left(\mathcal{L}_s^{(R-1)}(\boldsymbol{\theta}) - \mathcal{L}_s^{(R)}(\boldsymbol{\theta})\right) + \ldots + \tfrac{1}{M_1}\sum \left(\mathcal{L}_s^{(1)}(\boldsymbol{\theta}) - \mathcal{L}_s^{(2)}(\boldsymbol{\theta})\right), \tag{10}$$

where $M_R > M_{R-1} > M_1$. This implies that for the computation of the loss on the fine resolution $\mathcal{G}_1 = \mathcal{G}$, only a few fine graph approximations are needed, and most of the computations are done on coarse, small graphs. Our method is summarized in Algorithm 2, and naturally suits inductive learning tasks, where multiple graphs are available.

4.4 A Simple Model Problem

For multiscale training methods to adhere to the original learning problem, the condition in Equation (9) needs to be satisfied. This condition is dependent on the chosen pooling strategy. To study the impact of different pooling strategies, we explore a simplified model problem. We construct a dataset, illustrated in

Algorithm 2 Multiscale Gradients Computation

```
Initialize $\mathcal{L} = 0$
for $r = 2, \ldots, R$ do
    Sample $M_r$ node subsets
    Compute $\mathbf{X}_r, \mathbf{X}_{r-1}, \mathbf{Y}_r, \mathbf{Y}_{r-1}$
    Compute the graphs $\mathcal{G}_r$ and $\mathcal{G}_{r-1}$
    Update $\mathcal{L} \leftarrow \mathcal{L} + \frac{1}{M_r}(\mathcal{L}^{(r-1)} - \mathcal{L}^{(r)})$
end for
Compute $\mathcal{L}^{(R)}$ and update $\mathcal{L} \leftarrow \mathcal{L} + \frac{1}{M_R}\mathcal{L}^{(R)}$
```

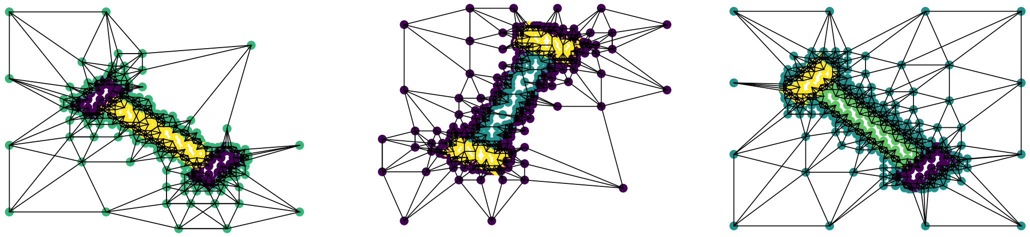


Figure 4: The Q-tips data set. The data contains three types of a "q-tip" and the goal is to label the stick as type 1,2 or 3.

Figure 4, that consists of structured synthetic graphs representing rods with three distinct types: (1) rods with both endpoints colored blue, (2) rods with both endpoints colored yellow, and (3) rods with mismatched endpoint colors. We wish to classify each node as either part of a rod or background and to identify the rod type.

A key challenge in this task is ensuring that the model can effectively capture global structure. To accurately classify a rod, a graph neural network (GNN) must aggregate information from both ends of the structure. A shallow GNN, such as one with a single or two-layer message-passing scheme, is insufficient, as its receptive field does not extend across the entire rod. Additionally, an effective coarsening strategy must preserve key structural information—retaining nodes from both rod endpoints, central regions, and the background—to maintain an informative representation.

To explore this further, we train a four-layer GCN on a dataset of 100 rods in varying orientations and evaluate different coarsening methods for constructing a reduced graph. Table 1 presents the loss values for various coarsening strategies along with the corresponding edge counts. The results highlight a crucial aspect of our approach: methods that fail to increase connectivity prior to pooling (i.e., with $p = 1$) lead to significant variance in loss across different coarsened graphs. For this dataset, introducing additional connectivity before pooling successfully mitigates this issue while still reducing the graph size in terms of both nodes and edges.

Coarsening type	# edges	Δloss (initial)	Δloss (final)
No coarsening	41,358	0	0
Random(p=1)	3,987	7.72e-01	3.45e-01
Random(p=2)	6,673	5.62e-02	1.25e-02
Topk(p=1)	8,731	1.41e-01	2.53e-02
Topk(p=2)	20,432	3.12e-03	8.64e-03
Subgraph	9,494	1.25e-01	1.93e-01

Table 1: Experiments for different coarsening strategies, creating coarse graph with $\frac{1}{4}$ of the original nodes. Loss discrepancy between the loss computed on the original graph, and the pooled graph (Δloss) is calculated using γ_s in Equation (9), and p denotes the power in Equation (5).

5 Computational Complexity

As noted in Section 3, the computational cost of GNN message-passing layers is directly influenced by the size of both the node feature matrix and the adjacency matrix. This observation suggests that reducing the number of nodes and edges can lead to improved computational efficiency during training.

To illustrate this, consider Graph Convolutional Networks (GCNs) Kipf & Welling (2016), where each layer follows the update rule defined in Equation 1. When analyzing floating-point operations (FLOPs), we account for both the input and output dimensions of each layer, as well as the sparsity of the adjacency matrix, meaning that only nonzero entries contribute to computation. The total FLOPs required for a single GCN layer is given by:

$$2 \cdot |\mathcal{E}| \cdot c_{in} + |\mathcal{V}| \cdot c_{in} \cdot c_{out}, \tag{11}$$

where $\mathcal{E}$ represents the set of edges, $\mathcal{V}$ the set of nodes, c_{in} the input feature dimension, and c_{out} the output feature dimension of the layer. From this expression, it follows that reducing the number of nodes and edges leads to a decrease in computational cost. For instance, if the number of nodes is halved, i.e., $|\mathcal{V}_{\mathcal{C}}| = \frac{1}{2}|\mathcal{V}|$, and the number of edges satisfies $|\mathcal{E}_{\mathcal{C}}| < |\mathcal{E}|$ for any pooling strategy, then the FLOPs per layer are reduced accordingly. This reduction translates into a significant theoretical improvement in efficiency.

Importantly, this effect is not limited to GCNs. A similar computational advantage is observed for other GNN architectures. For instance, in Graph Isomorphism Networks (GIN) Xu et al. (2018), where the per-layer complexity is:

$$|\mathcal{E}| \cdot c_{in} + 2 \cdot |\mathcal{V}| \cdot c_{in} \cdot c_{out}. \tag{12}$$

Similarly, in Graph Attention Networks (GAT) Velickovic et al. (2017), the per-layer complexity is:

$$2 \cdot |\mathcal{E}| \cdot c_{in} \cdot h + 2 \cdot |\mathcal{V}| \cdot c_{in} \cdot c_{out} \cdot h, \tag{13}$$

with h denoting the number of attention heads. In all cases, reducing the graph size results in fewer computations per layer, demonstrating a generalizable advantage across different GNN architectures.

6 Experiments

Objectives. We demonstrate that our approach achieves comparable or superior downstream performance to standard GNN training on the original graph while reducing computational costs across the training techniques outlined in Section 4 and the pooling strategies discussed below.

Pooling Strategies. Our method incorporates various coarsening techniques to enhance GNN training efficiency:

(i) **Random Coarsening:** Selects n_r random nodes from the graph and constructs the new adjacency matrix $\mathbf{A}_r$ via Equation (5).

(ii) **Topk Coarsening:** Selects n_r nodes with the highest degrees before computing $\mathbf{A}_r$ using Equation (5). This approach may be computationally expensive for large graphs as it requires full degree information.

(iii) **Subgraph Coarsening:** Samples a random node j and selects its nearest n_r neighbors. If geometric positions are available, the selection is based on Euclidean distance; otherwise, a k-hop ego network is used. This method captures local graph structures but may overlook global properties.

(iv) **Hybrid Coarsening:** Alternates between Random and Subgraph Coarsening to leverage both high- and low-frequency components of the graph.

Experimental Settings. We evaluate multiscale training with 2, 3, and 4 levels of coarsening, reducing the graph size by a factor of 2 at each level. Training epochs are doubled at each coarsening step, while fine-grid epochs remain fewer than in standard training. For multiscale gradient computation, we coarsen the graph to retain 75% of the nodes, perform half of the training on the coarsened graph, and then transition to fine-grid training. We validate our methods on node-level tasks, demonstrating that incorporating coarse levels during training enhances inference performance.

	Method ↓ / Dataset →	Cora		CiteSeer		PubMed	
		FLOPs	Test acc	FLOPs	Test acc	FLOPs	Test acc
GCN	Fine grid	9.91×10^2	82.9% (1.1%)	2.69×10^3	69.6% (1.0%)	3.55×10^3	77.5% (0.4%)
	Random(p=1)	2.40×10^2	82.0% (0.5%)	6.58×10^2	70.4% (0.2%)	8.51×10^2	77.5% (0.6%)
	Random(p=2)	2.63×10^2	83.0% (0.5%)	6.82×10^2	66.0% (0.8%)	9.99×10^2	73.8% (0.4%)
	Topk(p=1)	2.49×10^2	79.9% (4.4%)	6.83×10^2	68.1% (0.1%)	9.33×10^2	**78.2% (0.3%)**
	Topk(p=2)	3.09×10^2	79.6% (2.4%)	8.08×10^2	63.6% (2.7%)	1.87×10^3	74.5% (0.4%)
	Subgraphs(p=1)	2.43×10^2	**83.5% (0.4%)**	6.56×10^2	**71.2% (0.4%)**	8.63×10^2	78.0% (0.3%)
	Subgraphs(p=2)	3.26×10^2	83.1% (0.5%)	6.82×10^2	70.9% (0.2%)	1.19×10^3	76.1% (0.3%)
	Random and subgraphs(p=1)	2.41×10^2	81.5% (0.3%)	6.57×10^2	70.0% (0.0%)	8.57×10^2	77.8% (0.8%)
	Random and subgraphs(p=2)	2.92×10^2	82.6% (0.5%)	6.83×10^2	70.9% (0.2%)	1.07×10^3	73.9% (0.4%)
GIN	Fine grid	2.37×10^3	76.4% (1.6%)	6.79×10^3	63.8% (2.6%)	7.75×10^3	75.4% (0.4%)
	Random(p=1)	5.89×10^2	76.2% (2.7%)	1.69×10^3	66.5% (2.2%)	1.92×10^3	75.7% (2.0%)
	Random(p=2)	6.00×10^2	77.2% (0.6%)	1.70×10^3	60.5% (0.8%)	1.99×10^3	75.1% (0.9%)
	Topk(p=1)	5.94×10^2	77.6% (5.6%)	1.70×10^3	64.8% (3.9%)	1.96×10^3	75.6% (1.6%)
	Topk(p=2)	6.22×10^2	74.6% (3.0%)	1.76×10^3	60.5% (2.0%)	2.40×10^3	76.5% (1.5%)
	Subgraphs(p=1)	5.91×10^2	**78.7% (1.1%)**	1.69×10^3	**68.8% (1.6%)**	1.93×10^3	**77.2% (1.2%)**
	Subgraphs(p=2)	6.31×10^2	76.9% (1.72%)	1.70×10^3	65.0% (1.8%)	2.08×10^3	75.2% (1.5%)
	Random and subgraphs(p=1)	5.90×10^2	76.9% (1.0%)	1.69×10^3	65.3% (1.2%)	1.92×10^3	76.1% (2.8%)
	Random and subgraphs(p=2)	6.14×10^2	75.9% (0.8%)	1.70×10^3	66.0% (0.2%)	2.02×10^3	75.9% (0.5%)
GAT	Fine grid	1.11×10^3	79.6% (2.05%)	3.35×10^3	67.2% (6.5%)	3.08×10^3	75.5% (0.5%)
	Random(p=1)	2.65×10^2	80.3% (0.2%)	8.11×10^2	**72.9% (1.5%)**	7.29×10^2	77.5% (1.5%)
	Random(p=2)	3.01×10^2	79.5% (0.6%)	8.55×10^2	70.7% (1.0%)	9.01×10^2	77.4% (1.2%)
	Topk(p=1)	2.79×10^2	79.8% (0.6%)	8.56×10^2	66.8% (0.6%)	8.24×10^2	76.1% (0.9%)
	Topk(p=2)	3.72×10^2	76.8% (6.7%)	1.08×10^3	65.2% (0.7%)	1.92×10^3	74.9% (0.7%)
	Subgraphs(p=1)	2.70×10^2	78.5% (0.8%)	8.08×10^2	70.2% (0.6%)	7.43×10^2	75.4% (4.6%)
	Subgraphs(p=2)	3.99×10^2	**80.7% (1.2%)**	8.54×10^2	68.6% (0.3%)	1.12×10^3	77.3% (1.1%)
	Random and subgraphs(p=1)	2.67×10^2	79.9% (1.4%)	8.10×10^2	71.2% (0.8%)	7.36×10^2	76.8% (0.5%)
	Random and subgraphs(p=2)	3.46×10^2	80.2% (1.1%)	8.57×10^2	70.8% (0.4%)	9.87×10^2	**77.5% (1.1%)**

Table 2: Comparison of different methods for Cora, CiteSeer, and PubMed datasets. We present the test accuracy for multiscale training with 3 levels, as well as FLOPs analysis, in 10^6 units. The cheapest FLOPs count for each network and dataset appears in red. Results for fine grid appear in blue.

6.1 Transductive Learning

We evaluate our method on the Cora, Citeseer, and Pubmed datasets Sen et al. (2008) using the standard split from Yang et al. (2016), with 20 nodes per class for training, 500 validation nodes, and 1,000 testing nodes. Our experiments cover standard architectures—GCN Kipf & Welling (2016), GIN Xu et al. (2018), and GAT Velickovic et al. (2017)—and are summarized in Table 2, where we achieve performance that is either superior to or on par with the fine-grid baseline, which trains directly on the original graphs.

We explore two pooling strategies—random and Top-k—along with subgraph generation via k-hop neighborhoods and a hybrid approach combining both. Each method is tested under two connectivity settings: the original adjacency ($p = 1$) and an enhanced version ($p = 2$), where we square the adjacency matrix. We report test accuracy across three runs with different random seeds and include standard deviation to assess robustness. Our results indicate that enhanced connectivity improves accuracy in some settings. Table 2 further presents three levels of multiscale training alongside theoretical FLOP analyses, demonstrating the efficiency of our approach. The details of the FLOPs analysis are provided in 5, and additional results for two and four levels of multiscale training are provided in Appendix A.

We evaluate our approach on large-scale graphs using the OGBN-Arxiv dataset Hu et al. (2020) with GCN Kipf & Welling (2016), GIN Xu et al. (2018), and GAT Velickovic et al. (2017). This dataset consists of 169,343 nodes and 1,166,243 edges. Using the same experimental setup as before, we present our results in Table 3, demonstrating that our multiscale framework achieves comparable or superior performance relative to standard training. Notably, random pooling proves to be the most effective approach, offering both efficiency

Model	Method	2 levels			3 levels			4 levels		
		FLOPs	Time	Test Acc	FLOPs	Time	Test Acc	FLOPs	Time	Test Acc
GCN	Fine grid			1 level:	1.96×10^4	39.70	71.76% (0.1%)			
	Random	9.37×10^3	26.65	**72.07% (0.1%)**	4.59×10^3	18.52	**71.88% (0.0%)**	2.27×10^3	15.53	**72.03% (0.0%)**
	Topk	1.01×10^4	31.96	71.68% (0.0%)	5.16×10^3	22.01	71.44% (0.2%)	2.58×10^3	17.01	70.99% (0.2%)
	Subgraphs	1.61×10^3	53.71	71.06% (0.6%)	8.06×10^2	53.27	71.18% (0.2%)	4.05×10^2	54.46	71.13% (0.0%)
	Random and subgraphs	5.49×10^3	37.44	70.71% (0.2%)	2.70×10^3	34.50	70.93% (0.0%)	1.34×10^3	33.77	70.53% (0.2%)
GIN	Fine grid			1 level:	3.75×10^4	61.66	68.39% (0.6%)			
	Random	1.86×10^4	28.96	**69.21% (1.0%)**	9.24×10^3	17.21	**69.19% (4.8%)**	4.61×10^3	12.23	**69.30% (0.5%)**
	Topk	1.89×10^4	32.53	67.82% (3.2%)	9.50×10^3	19.84	68.10% (1.6%)	4.75×10^3	13.68	67.90% (1.0%)
	Subgraphs	2.52×10^3	47.47	67.13% (1.3%)	1.26×10^3	47.10	66.46% (1.6%)	6.30×10^2	47.69	66.03% (6.4%)
	Random and subgraphs	1.06×10^4	39.56	65.87% (6.6%)	5.25×10^3	31.16	64.79% (3.0%)	2.62×10^3	32.36	67.46% (1.2%)
GAT	Fine grid			1 level:	1.13×10^4	54.71	71.07% (0.2%)			
	Random	5.32×10^3	30.32	**71.70% (0.5%)**	2.59×10^3	19.39	**71.78% (0.0%)**	1.28×10^3	16.48	**71.12% (0.0%)**
	Topk	5.86×10^3	43.90	71.34% (0.3%)	3.01×10^3	29.72	70.68% (0.1%)	1.50×10^3	21.26	70.63% (0.1%)
	Subgraphs	5.04×10^3	52.96	70.24% (1.0%)	2.52×10^3	50.72	70.10% (0.6%)	1.26×10^3	55.15	69.82% (0.7%)
	Random and subgraphs	5.18×10^3	43.05	70.09% (0.2%)	2.56×10^3	35.79	69.62% (0.2%)	1.27×10^3	35.33	70.06% (0.2%)

Table 3: Comparison of different methods and coarsening levels for OGBN-Arxiv dataset. Time represents average epoch time in the observed levels of coarsening, in *ms* units. The best time for a given network and level appear in orange. FLOP values represent computational cost at the achieved level, in 10^6 units. The cheapest FLOPs count for each network and dataset appears in red. Results for fine grid appear in blue.

Method	1st Coarsening	2nd Coarsening	3rd Coarsening
No Coarsening	1 level:	1,166,243	
Random	280,009	76,559	17,978
Topk	815,023	480,404	238,368

Table 4: Edge number for coarsening strategies for the coarse-to-fine method, with the OGBN-Arxiv dataset. Fine grid edge number is in blue .

and strong performance. Given the large number of edges in this dataset, we use $p = 1$ and report the corresponding edge counts in Table 4. The results highlight the substantial reduction in edges at each pooling level, significantly improving training efficiency. In Table 3, we also provide a timing analysis, including both theoretical FLOP estimates and empirical runtime measurements, confirming that our multiscale approach enhances efficiency while maintaining strong predictive performance.

Additionally, consider the Flickr Zeng et al. (2019), WikiCS Mernyei & Cangea (2020), DBLP Bojchevski & Günnemanng (2017), the transductive versions of Facebook, BlogCatalog, and PPI Yang et al. (2020) datasets, detailed in Appendix B. We present results for Flickr and PPI, along with timing evaluations, as well as full results for all those datasets and additional experimental details, in Appendix B.

6.2 Inductive Learning

We evaluate our method on the ShapeNet dataset Chang et al. (2015), a point cloud dataset with multiple categories, for a node segmentation task. Specifically, we present results on the 'Airplane' category, which contains 2,690 point clouds, each consisting of 2,048 points. Nodes are classified into four categories: fuselage, wings, tail, or engine. For this dataset, we use the Dynamic Graph Convolutional Neural Network (DGCNN) Wang et al. (2018), which dynamically constructs graphs at each layer using the k-nearest neighbors algorithm, with k as a tunable hyperparameter. We test our multiscale approach with random pooling, subgraph pooling, and a combination of both. In subgraph pooling, given that the data resides in $\mathbb{R}^3$, we construct subgraphs by selecting a random node and its N_j nearest neighbors. Experiments are conducted using three random seeds, and we report the mean Intersection-over-Union (IoU) scores along with standard deviations

	Training using Multiscale Training			
	Fine Grid	Random	Subgraphs	Random and Subgraphs
k=6	78.36% (0.28%)	79.86% (0.87%)	79.31% (0.20%)	80.01% (0.21%)
k=10	79.71% (0.23%)	80.41% (0.14%)	80.65% (0.15%)	80.97% (0.10%)
k=20	81.06% (0.08%)	81.37% (0.05%)	81.18% (0.32%)	81.89% (0.58%)

	Training using Multiscale Gradient Computation Method			
	Fine Grid	Random	Subgraphs	Random and Subgraphs
k=6	78.36% (0.28%)	79.42% (1.40%)	79.35% (0.12%)	79.62% (0.93%)
k=10	79.71% (0.23%)	80.00% (0.33%)	80.15% (0.71%)	80.19% (1.63%)
k=20	81.06% (0.08%)	81.30% (0.41%)	81.17% (0.25%)	81.12% (0.24%)

Table 5: Results on Shapenet 'Airplane' dataset using DGCNN. Results represent Mean test IoU.

in the top section of Table 5. Our results show that for all tested values of k, applying either random or subgraph pooling improves performance compared to standard training while reducing computational cost. Furthermore, combining both pooling techniques consistently yields the best results across connectivity settings, suggesting that this approach captures both high- and low-frequency components of the graph while maintaining training efficiency. Additional results for multiscale training on the full ShapeNet dataset are provided in Appendix C.

We apply the Multiscale Gradients Computation method to the ShapeNet 'Airplane' category, evaluating various pooling techniques using Algorithm 2 with two levels. As shown in the bottom section of Table 5, our approach reduces training costs while achieving performance comparable to fine-grid training. These results show that the early stages of optimization can be efficiently performed using an approximate loss, as we propose, with fine-grid calculations needed only once the loss function nears its minimum. Additional results, timing analyses and experimental details for the ShapeNet dataset are provided in Appendix C.

We further evaluate our Multiscale Gradients Computation method on the protein-protein interaction (PPI) dataset Zitnik & Leskovec (2017), consisting of graphs representing different human tissues. The results, with additional details, are provided in Appendix D.

7 Conclusions

In this work, we propose an efficient approach for training graph neural networks and evaluate its effectiveness across multiple datasets, architectures, and pooling strategies. We demonstrate that all proposed approaches, Coarse-to-Fine, Sub-to-Full, and Multiscale Gradient Computation, improve performance while reducing the training costs.

Our method is adaptable to any architecture, dataset, or pooling technique, making it a versatile tool for GNN training. Our experiments reveal that no single setting is universally optimal; performance varies depending on the dataset and architecture. However, our approach consistently yields improvements across all considered settings.

References

Abadal, S., Jain, A., Guirado, R., López-Alonso, J., and Alarcón, E. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys (CSUR)*, 54(9):1–38, 2021.

Abate, C. and Bianchi, F. M. Maxcutpool: Differentiable feature-aware maxcut for pooling in graph neural networks. In *arXiv.org*, 2024. URL `https://arxiv.org/pdf/2409.05100`.

Arora, S., Cohen, N., Golowich, N., and Hu, W. A convergence analysis of gradient descent for deep linear neural networks. *arXiv preprint arXiv:1810.02281*, 2018. URL `https://arxiv.org/abs/1810.02281`.

Bentivoglio, R., Isufi, E., Jonkman, S., and Taormina, R. Multi-scale hydraulic graph neural networks for flood modelling. In *EGUsphere*, 2024. URL `https://egusphere.copernicus.org/preprints/2024/egusphere-2024-2621/egusphere-2024-2621.pdf`.

Bevilacqua, B., Frasca, F., Lim, D., Srinivasan, B., Cai, C., Balamurugan, G., Bronstein, M. M., and Maron, H. Equivariant subgraph aggregation networks. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=dFbKQaRk15w`.

Bevilacqua, B., Eliasof, M., Meirom, E., Ribeiro, B., and Maron, H. Efficient subgraph GNNs by learning effective selection policies. In *The Twelfth International Conference on Learning Representations*, 2024.

Bianchi, F. and Lachi, V. The expressive power of pooling in graph neural networks. In *Neural Information Processing Systems*, 2023.

Bojchevski, A. and Günnemanng, S. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, 2017. URL `https://arxiv.org/pdf/1707.03815`.

Boyd, S. and Vandenberghe, L. *Convex optimization*. Cambridge University press, 2004.

Brandt, A. Multigrid techniques: 1984 Guide with applications to fluid Dynamics. The Weizmann Institute of Science, Rehovot, Israel, 1984.

Cai, C., Wang, D., and Wang, Y. Graph coarsening with neural networks. In *International Conference on Learning Representations (ICLR)*, 2021. URL `https://arxiv.org/abs/2102.01350`.

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q.-X., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. Shapenet: An information-rich 3d model repository. In *arXiv.org*, 2015.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, 2020. URL `https://arxiv.org/pdf/2007.02133`.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Neural Information Processing Systems*, 2016.

Eliasof, M., Bodner, B. J., and Treister, E. Haar wavelet feature compression for quantized graph convolutional networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

Frasca, F., Bevilacqua, B., Bronstein, M. M., and Maron, H. Understanding and extending subgraph gnns by rethinking their symmetries. In *Neural Information Processing Systems*, 2022. URL `https://arxiv.org/pdf/2206.11140`.

Galmés, M. F., Suárez-Varela, J., Rusek, K., Barlet-Ros, P., and Cabellos-Aparicio, A. Scaling graph-based deep learning models to larger networks. In *arXiv.org*, 2021.

Gao and Ji. Graph u-nets. In *IEEE*, 2019.

Geng, X., Li, Y., Wang, L., Zhang, L., Yang, Q., Ye, J., and Liu, Y. Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting. In *AAAI*, 2019.

Giles, M. B. Multilevel monte carlo methods. In *Large-Scale Scientific Computing*, 2001.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272. JMLR. org, 2017.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010.

Guo, Z.-H. and Li, H.-B. Mgfno: Multi-grid architecture fourier neural operator for parametric partial differential equations. In *arXiv.org*, 2024. URL `https://arxiv.org/pdf/2407.08615`.

Gupta, S., Yan, X., and Lerman, K. Structural properties of ego networks. In *Social Computing, Behavioral Modeling, and Prediction*, 2014.

Hamilton, W. L., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Neural Information Processing Systems*, 2017. URL `https://arxiv.org/pdf/1706.02216`.

Hastings, W. Monte carlo sampling methods using markov chains and their applications. In *Biometrika*, 1970. URL `https://appliedmaths.sun.ac.za/~htouchette/markovcourse/1970_monte%20carlo%20sampling%20methods%20using0.pdf`.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. In *Neural Information Processing Systems*, 2020.

Huang, X., Cheng, H., Li, R.-H., Qin, L., and Yu, J. X. Top-k structural diversity search in large networks. *The VLDB Journal*, 24(3):319–343, 2015.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014. URL `https://arxiv.org/pdf/1412.6980`.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2016.

Klapproth, A., Schuemann, J., Stangl, S., Xie, T., Li, W. B., and Multhoff, G. Multi-scale monte carlo simulations of gold nanoparticle-induced dna damages for kilovoltage x-ray irradiation in a xenograft mouse model using topas-nbio. In *Cancer nanotechnology*, 2021.

Kolaczyk, E. and Nowak, R. Multiscale statistical models. In Denison, D., Hansen, M., Holmes, C., Mallick, B., and Yu, B. (eds.), *Nonlinear Estimation and Classification*, volume 171 of *Lecture Notes in Statistics*, pp. 293–308. Springer, New York, NY, 2003. doi: 10.1007/978-0-387-21579-2_14.

Li, S., Zhang, B., Song, J., Ruan, G., Wang, C., and Xie, J. Graph neural networks with coarse-and fine-grained division for mitigating label sparsity and noise. *arXiv preprint arXiv:2411.03744*, 2024a.

Li, Z., Song, H., Xiao, D., Lai, Z., and Wang, W. Harnessing scale and physics: A multi-graph neural operator framework for pdes on arbitrary geometries. In *arXiv.org*, 2024b. URL `https://arxiv.org/pdf/2411.15178`.

Liang, Y., Xu, Z., and Xing, X. A multi-scale monte carlo method for electrolytes. In *New Journal of Physics*, 2015. URL `https://arxiv.org/pdf/1504.00454`.

Liu, Zhan, and et al. Graph pooling for graph neural networks: Progress, challenges, and opportunities. In *International Joint Conference on Artificial Intelligence*, 2022. URL `https://arxiv.org/pdf/2204.07321`.

Mernyei, P. and Cangea, C. Wiki-cs: A wikipedia-based benchmark for graph neural networks. In *arXiv.org*, 2020. URL `https://arxiv.org/pdf/2007.02901`.

Nash, S. A multigrid approach to discretized optimization problems. *Optimization Methods and Software*, 14: 99–116, 2000.

Ronneberger, Fischer, and Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2015.

Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B., and Eliassi-Rad, T. Collective classification in network data. In *Computer Science*, 2008.

Shi, Y., Cai, J.-X., Shavit, Y., Mu, T.-J., Feng, W., and Zhang, K. Clustergnn: Cluster-based coarse-to-fine graph neural network for efficient feature matching. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12517–12526, 2022.

Stanovic, S., Gaüzère, B., and Brun, L. Graph neural networks with maximal independent set-based pooling: Mitigating over-smoothing and over-squashing. In *Pattern Recognition Letters*, 2024.

Tang, S. W. Y., Zhu, Y., Wang, L., Xie, X., and Tan, T. Session-based recommendation with graph neural networks. In *AAAI*, 2018. URL `https://arxiv.org/pdf/1811.00855`.

Velickovic, Cucurull, Casanova, Romero, Lio, and Bengio. Graph attention networks. In *ICLR*, 2017.

Wang, Y., Sun, Y., Liu, Z., Sarma, S., Bronstein, M., and Solomon, J. Dynamic graph cnn for learning on point clouds. In *ACM Transactions on Graphics*, 2018.

Wang, Y., Wang, W., Liang, Y., Cai, Y., and Hooi, B. Graphcrop: Subgraph cropping for graph classification. In *arXiv.org*, 2020. URL `https://arxiv.org/pdf/2009.10564`.

Wang, Z., Zhang, Z., Zhang, C., and Ye, Y. Subgraph pooling: Tackling negative transfer on graphs. In *International Joint Conference on Artificial Intelligence*, 2024. URL `https://arxiv.org/pdf/2402.08907v2`.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *ICLR*, 2018.

Xu, M., Qian, P., Zhao, Z., Zeng, Z., Chen, J., Liu, W., and Yang, X. Graph neural networks for protein-protein interactions - a short survey. In *arXiv.org*, 2024.

Yang, R., Shi, J., Xiao, X., Yang, Y., Bhowmick, S., and Liu, J. Pane: scalable and effective attributed network embedding. In *The VLDB journal*, 2020. URL `https://arxiv.org/pdf/2009.00826`.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

Ying, You, Morris, Ren, Hamilton, and Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Neural Information Processing Systems*, 2018. URL `https://arxiv.org/pdf/1806.08804`.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2019. URL `https://arxiv.org/pdf/1907.04931`.

Zitnik, M. and Leskovec, J. Predicting multicellular function through multi-layer tissue networks. In *Bioinform*, 2017. URL `https://arxiv.org/pdf/1707.04638`.

A Additional Results for Planetoid Datasets

We provide additional results for Cora, Citeseer, and Pubmed datasets (Sen et al., 2008), for 2, 3, and 4 levels of multiscale training, using datasets details provided in 6.1. We test our methods using GCN (Kipf & Welling, 2016), GIN (Xu et al., 2018), and GAT (Velickovic et al., 2017). Results are presented in Table 6, where we show that across various levels of multiscale training and different methods, we achieve consistently better or comparable results to those of the fine grid original training while using a much cheaper training process.

	Method ↓ / Dataset →	Cora			CiteSeer			PubMed		
		2 levels	3 levels	4 levels	2 levels	3 levels	4 levels	2 levels	3 levels	4 levels
GCN	Fine grid	1 level:	82.9% (1.1%)		1 level:	69.6% (1.0%)		1 level:	77.5% (0.4%)	
	Random(p=1)	81.7% (0.6%)	82.0% (0.5%)	81.7% (0.4%)	70.5% (0.3%)	70.4% (0.2%)	70.7% (0.2%)	77.6% (0.8%)	77.5% (0.6%)	77.5% (0.4%)
	Random(p=2)	**83.2% (0.6%)**	83.0% (0.5%)	81.5% (0.3%)	66.6% (2.6%)	66.0% (0.8%)	66.2% (2.2%)	74.2% (0.4%)	73.8% (0.4%)	74.2% (0.7%)
	Topk(p=1)	79.5% (0.7%)	79.9% (4.4%)	79.8% (2.2%)	68.5% (1.1%)	68.1% (0.1%)	67.4% (0.3%)	**77.8% (0.9%)**	**78.2% (0.3%)**	78.2% (0.6%)
	Topk(p=2)	80.5% (2.5%)	79.6% (2.4%)	80.8% (0.8%)	66.4% (0.8%)	63.6% (2.7%)	65.8% (2.1%)	75.4% (0.4%)	74.5% (0.4%)	74.2% (0.6%)
	Subgraphs(p=1)	82.7% (0.7%)	**83.5% (0.4%)**	81.3% (0.1%)	**71.6% (0.5%)**	**71.2% (0.4%)**	**71.3% (0.4%)**	77.7% (0.2%)	78.0% (0.3%)	**78.7% (1.3%)**
	Subgraphs(p=2)	82.8% (0.4%)	83.1% (0.5%)	**81.8% (0.7%)**	71.5% (0.5%)	70.9% (0.2%)	70.9% (0.4%)	74.8% (1.2%)	76.1% (0.3%)	76.5% (1.1%)
	Random and subgraphs(p=1)	81.7% (0.4%)	81.5% (0.3%)	81.5% (0.3%)	71.4% (0.5%)	70.0% (0.0%)	70.7% (0.3%)	77.4% (0.5%)	77.8% (0.8%)	76.7% (0.3%)
	Random and subgraphs(p=2)	82.7% (0.3%)	82.6% (0.5%)	80.8% (0.1%)	70.8% (0.3%)	70.9% (0.2%)	70.7% (0.1%)	74.1% (0.7%)	73.9% (0.4%)	75.6% (0.9%)
GIN	Fine grid	1 level:	76.4% (1.6%)		1 level:	63.8% (2.6%)		1 level:	75.4% (0.4%)	
	Random(p=1)	76.6% (1.5%)	76.2% (2.7%)	77.6% (2.2%)	66.3% (7.0%)	66.5% (2.2%)	66.4% (1.1%)	75.6% (1.7%)	75.7% (2.0%)	76.2% (1.3%)
	Random(p=2)	73.1% (0.3%)	77.2% (0.6%)	71.6% (1.6%)	60.6% (2.6%)	60.5% (0.8%)	62.8% (1.3%)	75.6% (2.9%)	75.1% (0.9%)	74.3% (1.2%)
	Topk(p=1)	77.5% (1.3%)	77.6% (5.6%)	**79.4% (1.9%)**	67.3% (1.3%)	64.8% (3.9%)	66.5% (1.5%)	76.6% (0.8%)	75.6% (1.6%)	75.5% (1.6%)
	Topk(p=2)	76.2% (1.6%)	74.6% (3.0%)	75.6% (2.3%)	61.1% (1.1%)	60.5% (2.0%)	60.0% (2.9%)	76.0% (1.8%)	76.5% (1.5%)	66.1% (6.4%)
	Subgraphs(p=1)	77.5% (0.9%)	**78.7% (1.1%)**	78.2% (1.4%)	**68.2% (0.5%)**	**68.8% (1.6%)**	**68.7% (1.4%)**	75.8% (1.2%)	**77.2% (1.2%)**	**77.2% (0.7%)**
	Subgraphs(p=2)	76.2% (8.57%)	76.9% (1.72%)	77.3% (8.55%)	60.1% (0.29%)	65.0% (1.8%)	66.6% (1.0%)	75.6% (2.8%)	75.2% (1.5%)	55.4% (4.7%)
	Random and subgraphs(p=1)	**80.2% (2.5%)**	76.9% (1.0%)	76.8% (2.1%)	66.0% (0.9%)	65.3% (1.2%)	65.5% (1.1%)	75.2% (0.8%)	76.1% (2.8%)	75.0% (0.7%)
	Random and subgraphs(p=2)	75.5% (0.7%)	75.9% (0.8%)	74.6% (22.5%)	64.1% (2.2%)	66.0% (0.2%)	66.7% (1.0%)	**77.8% (5.6%)**	75.9% (0.5%)	76.4% (2.4%)
GAT	Fine grid	1 level:	79.6% (2.05%)		1 level:	67.2% (6.5%)		1 level:	75.5% (0.5%)	
	Random(p=1)	80.1% (0.8%)	80.3% (0.2%)	79.8% (0.8%)	71.1% (0.4%)	**72.9% (1.5%)**	**71.4% (0.3%)**	76.5% (0.3%)	77.5% (1.5%)	76.4% (0.8%)
	Random(p=2)	**82.0% (0.5%)**	79.5% (0.6%)	79.5% (1.9%)	**71.6% (1.1%)**	70.7% (1.0%)	70.6% (1.5%)	75.2% (1.0%)	77.4% (1.2%)	**78.6% (2.5%)**
	Topk(p=1)	77.3% (1.8%)	79.8% (0.6%)	77.5% (2.9%)	67.2% (1.5%)	66.8% (0.6%)	70.8% (1.3%)	76.1% (0.2%)	76.1% (0.9%)	75.6% (2.5%)
	Topk(p=2)	79.6% (2.3%)	76.8% (6.7%)	79.0% (1.6%)	67.4% (2.2%)	65.2% (0.7%)	67.0% (0.2%)	74.2% (1.0%)	74.9% (0.7%)	73.3% (1.0%)
	Subgraphs(p=1)	80.2% (2.6%)	78.5% (0.8%)	79.5% (0.8%)	69.1% (0.6%)	70.2% (0.6%)	68.4% (2.7%)	77.0% (1.0%)	75.4% (4.6%)	75.8% (1.6%)
	Subgraphs(p=2)	80.5% (1.5%)	**80.7% (1.2%)**	79.9% (0.9%)	70.1% (0.5%)	68.6% (0.3%)	66.0% (0.2%)	75.1% (0.85%)	77.3% (1.1%)	78.2% (3.0%)
	Random and subgraphs(p=1)	80.6% (0.2%)	79.9% (1.4%)	**80.3% (0.9%)**	70.8% (0.1%)	71.2% (0.8%)	71.2% (0.3%)	**77.3% (0.8%)**	76.8% (0.5%)	76.0% (1.9%)
	Random and subgraphs(p=2)	81.1% (1.1%)	80.2% (1.1%)	78.7% (0.4%)	70.3% (0.4%)	70.8% (0.4%)	71.1% (0.3%)	75.7% (1.8%)	**77.5% (1.1%)**	76.4% (1.9%)

Table 6: Comparison of different methods and coarsening levels for Cora, CiteSeer, and PubMed datasets.

B Additional Results for Transductive Learning

We further test our methods using additional datasets. We provide results for Flickr Zeng et al. (2019) and PPI (transductive) Yang et al. (2020), showing both timing analysis, in Table 7, and full multiscale training results in Table 8. Our results show that while achieving comparable accuracy to the original fine grid, our methods show significant timing improvement both in a theoretical manner as well as in FLOPs analysis, which we calculate as explained in Appendix 5.

In addition, we test our methods and show results for Facebook, BlogCatalog (Yang et al., 2020), DBLP (Bojchevski & Günnemanng, 2017) and WikiCS (Mernyei & Cangea, 2020), in Table 9 and Table 10. This further illustrates that our methods are not restricted to a specific dataset or network architecture, as our results are consistently comparable to the original training baseline results. Details of our experiments are provided in Table 11, and we list additional details of the datasets in Table 12.

C Additional Results for Shapenet Dataset

We test out methods on the full Shapenet dataset Chang et al. (2015) using DGCNN Wang et al. (2018) and demonstrate the results of our multiscale training process in Table 13. We observe that our method is not limited to the 'Airplane' category of this dataset, and we achieved comparable results across various connectivity values and pooling methods while using a significantly cheaper training process.

To emphasize the efficiency of our methods, we further provide timing analysis of Shapanet 'Airplanes' data computations. Timing was calculated using GPU computations, and we averaged across 5 epochs after the

	Method ↓ / Dataset →	Flickr			PPI (transductive)		
		FLOPs	Time	Test acc	FLOPs	Time	Test acc
GCN	Fine grid	1.72×10^4	68.90	54.77% (0.2%)	8.09×10^3	64.66	70.74% (0.2%)
	Random	3.94×10^3	27.73	55.30% (1.6%)	1.65×10^3	25.60	71.10% (0.2%)
	Topk	4.28×10^3	36.31	55.14% (0.1%)	2.46×10^3	46.14	71.37% (0.4%)
	Subgraphs	4.92×10^3	73.16	54.61% (0.8%)	1.61×10^3	113.90	**71.63% (0.3%)**
	Random and subgraphs	4.44×10^3	49.74	**55.85% (1.6%)**	1.63×10^3	68.06	71.50% (0.0%)
GIN	Fine grid	3.58×10^4	76.29	52.05% (1.0%)	1.34×10^4	48.76	85.67% (7.2%)
	Random	8.77×10^3	27.93	52.16% (3.9%)	3.17×10^3	21.47	89.24% (0.4%)
	Topk	8.93×10^3	34.47	51.99% (5.1%)	3.54×10^3	33.98	84.92% (0.8%)
	Subgraphs	9.23×10^3	75.88	**53.41% (1.0%)**	3.15×10^3	117.70	87.40% (8.1%)
	Random and subgraphs	9.01×10^3	53.51	52.00% (1.8%)	3.16×10^3	68.52	**90.83% (12.3%)**
GAT	Fine grid	1.53×10^4	103.54	52.71% (0.9%)	4.57×10^3	192.20	82.37% (3.1%)
	Random	3.40×10^3	29.57	52.91% (0.3%)	9.32×10^2	32.30	82.93% (1.3%)
	Topk	3.80×10^3	44.35	52.79% (0.4%)	1.39×10^3	99.79	73.13% (1.6%)
	Subgraphs	4.55×10^3	94.21	**53.37% (0.1%)**	9.10×10^2	132.67	**87.71% (1.8%)**
	Random and subgraphs	3.99×10^3	62.38	53.26% (0.2%)	9.21×10^2	79.21	87.03% (0.8%)

Table 7: Comparison of different methods for Flickr and PPI (transductive) dataset. The test accuracy is calculated for 3 levels of multiscale training. Time represents average epoch time in the observed levels of coarsening, in *ms* units. The best time for a given network appears in orange. FLOP values represent computational cost at the achieved level, in 10^6 units. The cheapest FLOPs count for each network and dataset appears in red. Results for fine grid appear in blue.

	Method ↓ / Dataset →	Flickr			PPI (transductive)		
		2 levels	3 levels	4 levels	2 levels	3 levels	4 levels
GCN	Fine grid	1 level:	54.77% (0.2%)		1 level:	70.74% (0.2%)	
	Random	55.01% (0.6%)	55.30% (1.6%)	54.47% (0.5%)	**70.97% (0.2%)**	71.10% (0.2%)	71.44% (0.1%)
	Topk	55.28% (0.2%)	55.14% (0.1%)	**55.15% (0.1%)**	70.62% (0.0%)	71.37% (0.4%)	71.09% (0.3%)
	Subgraphs	55.25% (0.1%)	54.61% (0.8%)	55.47% (0.3%)	70.96% (0.1%)	**71.63% (0.3%)**	**73.07% (0.4%)**
	Random and subgraphs	**55.65% (0.1%)**	**55.85% (1.6%)**	55.08% (0.2%)	70.88% (0.1%)	71.50% (0.0%)	72.89% (0.1%)
GIN	Fine grid	1 level:	52.05% (1.0%)		1 level:	85.67% (7.2%)	
	Random	52.24% (0.6%)	52.16% (3.9%)	**53.74% (4.2%)**	84.85% (2.6%)	89.24% (0.4%)	89.43% (0.6%)
	Topk	**53.14% (3.5%)**	51.99% (5.1%)	52.19% (5.3%)	81.74% (1.2%)	84.92% (0.8%)	83.70% (1.2%)
	Subgraphs	53.09% (1.4%)	**53.41% (1.0%)**	53.33% (2.5%)	**91.27% (8.8%)**	87.40% (8.1%)	88.58% (0.2%)
	Random and subgraphs	52.74% (1.0%)	52.00% (1.8%)	52.50% (2.0%)	84.25% (4.1%)	**90.83% (12.3%)**	**93.31% (12.1%)**
GAT	Fine grid	1 level:	52.71% (0.9%)		1 level:	82.37% (3.1%)	
	Random	52.98% (0.3%)	52.91% (0.3%)	52.18% (1.5%)	81.23% (1.4%)	82.93% (1.3%)	82.97% (3.0%)
	Topk	**52.98% (0.2%)**	52.79% (0.4%)	52.72% (0.2%)	77.39% (1.4%)	73.13% (1.6%)	70.37% (0.4%)
	Subgraphs	52.96% (0.3%)	**53.37% (0.1%)**	**53.33% (0.3%)**	**84.99% (0.9%)**	**87.71% (1.8%)**	86.26% (1.9%)
	Random and subgraphs	52.70% (0.0%)	53.26% (0.2%)	52.97% (0.2%)	84.48% (1.0%)	87.03% (0.8%)	**87.15% (1.3%)**

Table 8: Comparison of different methods and coarsening levels for Flickr and PPI (transductive) datasets.

training loss was stabilized. Results appear in Table 14 for multiscale training and in Table 15 for multiscale gradient computations, where we show that our training process was indeed faster than the original process which only took into consideration the fine grid.

	Method ↓ / Dataset →	DBLP			Facebook		
		2 levels	3 levels	4 levels	2 levels	3 levels	4 levels
GCN	Fine grid	1 level:	86.48% (0.4%)		1 level:	74.88% (0.6%)	
	Random(p=1)	**87.13% (0.1%)**	**87.05% (0.1%)**	**87.08% (0.1%)**	**75.87% (0.4%)**	75.25% (0.3%)	**75.25% (0.7%)**
	Random(p=2)	85.58% (0.0%)	85.30% (0.3%)	85.47% (0.5%)	75.50% (0.4%)	74.38% (0.0%)	73.76% (0.3%)
	Topk(p=1)	86.79% (0.2%)	86.82% (0.1%)	86.03% (0.2%)	75.25% (0.5%)	74.75% (0.4%)	74.75% (0.4%)
	Topk(p=2)	85.75% (0.1%)	85.78% (0.1%)	85.86% (0.0%)	74.38% (0.3%)	74.63% (0.2%)	74.88% (0.8%)
	Subgraphs(p=1)	86.63% (0.1%)	86.88% (0.3%)	86.77% (0.0%)	74.63% (0.6%)	74.75% (0.3%)	74.88% (0.3%)
	Subgraphs(p=2)	85.86% (0.2%)	85.75% (0.1%)	85.78% (0.1%)	74.88% (0.3%)	75.25% (0.6%)	75.12% (0.3%)
	Random and subgraphs(p=1)	86.68% (0.1%)	86.71% (0.2%)	86.77% (0.2%)	74.63% (0.3%)	**75.37% (0.4%)**	75.00% (0.3%)
	Random and subgraphs(p=2)	85.64% (0.2%)	85.50% (0.2%)	85.84% (0.2%)	74.75% (0.2%)	74.75% (0.5%)	75.12% (0.3%)
GIN	Fine grid	1 level:	84.82% (0.4%)		1 level:	73.89% (0.3%)	
	Random(p=1)	84.65% (0.3%)	84.20% (0.2%)	84.68% (0.3%)	**75.62% (0.3%)**	74.38% (3.8%)	74.88% (2.7%)
	Random(p=2)	**85.27% (0.3%)**	85.16% (0.3%)	**85.55% (0.3%)**	74.88% (0.4%)	75.62% (0.8%)	**76.11% (1.0%)**
	Topk(p=1)	84.99% (0.7%)	84.71% (0.1%)	84.09% (0.3%)	74.75% (0.7%)	74.01% (3.3%)	73.89% (6.0%)
	Topk(p=2)	84.26% (0.1%)	84.99% (0.4%)	85.47% (0.1%)	74.50% (0.7%)	74.13% (20.3%)	71.78% (16.9%)
	Subgraphs(p=1)	84.85% (0.2%)	84.00% (0.2%)	83.80% (0.1%)	74.88% (0.7%)	74.75% (0.3%)	75.25% (1.0%)
	Subgraphs(p=2)	83.04% (0.6%)	**85.21% (1.2%)**	84.88% (1.0%)	74.13% (0.3%)	73.14% (0.4%)	72.77% (1.0%)
	Random and subgraphs(p=1)	84.73% (0.5%)	84.45% (0.4%)	83.86% (0.1%)	75.25% (0.5%)	**75.87% (0.7%)**	75.62% (1.5%)
	Random and subgraphs(p=2)	84.51% (1.1%)	85.16% (0.1%)	84.48% (0.1%)	74.75% (0.7%)	74.63% (0.2%)	70.79% (19.8%)
GAT	Fine grid	1 level:	85.16% (0.2%)		1 level:	72.65% (0.7%)	
	Random(p=1)	84.62% (0.2%)	84.71% (0.3%)	84.59% (0.3%)	74.38% (0.4%)	**73.76% (0.7%)**	73.02% (0.5%)
	Random(p=2)	85.58% (0.3%)	85.05% (0.3%)	85.21% (0.5%)	73.64% (0.6%)	71.91% (0.4%)	71.53% (0.4%)
	Topk(p=1)	85.61% (0.3%)	85.07% (0.4%)	**85.78% (0.7%)**	73.27% (1.3%)	72.15% (0.8%)	72.15% (1.1%)
	Topk(p=2)	85.86% (0.4%)	85.05% (0.2%)	84.42% (0.5%)	72.40% (0.7%)	73.14% (1.8%)	72.90% (1.2%)
	Subgraphs(p=1)	84.76% (0.1%)	84.71% (0.2%)	84.59% (0.4%)	71.91% (3.1%)	71.16% (0.8%)	71.66% (0.7%)
	Subgraphs(p=2)	85.84% (0.3%)	85.16% (0.2%)	85.30% (0.1%)	72.15% (2.2%)	71.16% (1.9%)	71.66% (2.0%)
	Random and subgraphs(p=1)	85.86% (0.3%)	85.30% (0.2%)	85.47% (0.4%)	**74.38% (1.3%)**	72.65% (0.2%)	**73.51% (0.7%)**
	Random and subgraphs(p=2)	**85.86% (0.3%)**	**85.30% (0.2%)**	85.47% (0.4%)	72.28% (1.6%)	72.40% (0.1%)	70.54% (1.0%)

Table 9: Comparison of different methods and coarsening levels for DBLP and Facebook datasets.

	Method ↓ / Dataset →	BlogCatalog			WikiCS		
		2 levels	3 levels	4 levels	2 levels	3 levels	4 levels
GCN	Fine grid	1 level:	75.87% (0.4%)		1 level:	78.07% (0.0%)	
	Random	76.54% (0.3%)	76.63% (0.2%)	76.54% (0.3%)	**79.55% (0.1%)**	78.35% (0.4%)	**78.96% (0.1%)**
	Topk	76.44% (0.1%)	76.25% (0.3%)	75.96% (0.3%)	77.95% (0.4%)	78.06% (0.2%)	77.83% (0.2%)
	Subgraphs	**76.73% (0.1%)**	**77.88% (0.3%)**	**77.40% (0.3%)**	78.36% (0.3%)	78.28% (0.1%)	78.42% (0.1%)
	Random and subgraphs	76.44% (0.1%)	77.12% (0.3%)	76.92% (0.8%)	79.29% (0.1%)	**78.59% (0.5%)**	78.28% (0.6%)
GIN	Fine grid	1 level:	80.58% (4.0%)		1 level:	71.73% (0.4%)	
	Random	80.19% (5.4%)	81.25% (11.6%)	81.83% (1.6%)	**75.01% (1.0%)**	**76.96% (1.1%)**	**76.18% (0.9%)**
	Topk	**80.96% (8.8%)**	**83.08% (23.9%)**	80.38% (15.5%)	72.62% (1.7%)	71.49% (1.1%)	71.83% (0.2%)
	Subgraphs	80.19% (27.6%)	82.88% (24.5%)	**82.88% (7.7%)**	71.71% (1.2%)	73.03% (0.7%)	73.01% (1.7%)
	Random and subgraphs	80.58% (14.7%)	82.12% (3.8%)	81.35% (3.0%)	72.64% (0.4%)	73.71% (0.8%)	74.16% (1.3%)
GAT	Fine grid	1 level:	71.44% (3.9%)		1 level:	79.09% (1.1%)	
	Random	**79.04% (3.9%)**	**77.98% (2.1%)**	72.98% (3.0%)	**79.10% (0.5%)**	**80.28% (0.8%)**	79.19% (0.4%)
	Topk	71.83% (1.2%)	72.31% (1.5%)	72.31% (0.1%)	79.00% (0.4%)	77.90% (0.9%)	76.09% (0.9%)
	Subgraphs	70.67% (2.0%)	69.52% (0.6%)	71.73% (3.0%)	78.45% (2.3%)	77.94% (0.6%)	76.81% (0.8%)
	Random and subgraphs	72.98% (3.9%)	74.13% (1.9%)	**76.15% (3.1%)**	78.93% (0.6%)	79.20% (0.2%)	**79.60% (0.9%)**

Table 10: Comparison of different methods and coarsening levels for BlogCatalog and WikiCS datasets.

D Additional Details and Results for PPI Dataset

The PPI dataset Zitnik & Leskovec (2017) includes 20 graphs for training, 2 for validation, and 2 for testing, with an average of 2,372 nodes per graph. Each node has 50 features and is assigned to one or more of 121 possible labels. We use the preprocessed data from Hamilton et al. (2017) and present our results in Table 17. In this experiment, we use the Graph Convolutional Network with Initial Residual and Identity Mapping

Component	**Details**
Network architecture	GCN Kipf & Welling (2016) with 4 layers and 192 hidden channels. GIN Xu et al. (2018) with 3 layers and 256 hidden channels. GAT Velickovic et al. (2017) with 3 layers and 64 hidden channels, using 2 heads
Loss function	Negative Log Likelihood Loss except for PPI (transductive) and Facebook Yang et al. (2020) which use Cross Entropy Loss
Optimizer	Adam Kingma & Ba (2014)
Learning rate	1×10^{-3}
Baseline training	2000 epochs
Mulstiscale gradients level	2, 3, and 4 levels
Coarse-to-fine strategy	Each coarsening reduces the number of nodes by half compared to the previous level
Sub-to-Full strategy	Using ego-networks Gupta et al. (2014). 6 hops on level 2, 4 on level 3, and 2 on level 4
Multiscale training strategy	Using [1000, 2000] epochs for 2 levels, [800, 1600, 3200] epochs for 3 levels, and [600, 1200, 2400, 4800] epochs for 4 levels (the first number is the fine grid epoch number)
Evaluation metric	Accuracy
Timing computation	GPU performance. An average of 100 epochs after training was stabled

Table 11: Experimental setup for transductive learning datasets.

Dataset	**Nodes**	**Edges**	**Features**	**Classes**
Cora Sen et al. (2008)	2,708	10,556	1,433	7
Citeseer Sen et al. (2008)	3,327	9,104	3,703	6
Pubmed Sen et al. (2008)	19,717	88,648	500	3
Flickr Zeng et al. (2019)	89,250	899,756	500	7
WikiCS Mernyei & Cangea (2020)	11,701	216,123	300	10
DBLP Bojchevski & Günnemanng (2017)	17,716	105,734	1,639	4
Facebook Yang et al. (2020)	4,039	88,234	1,283	193
BlogCatalog Yang et al. (2020)	5,196	343,486	8,189	6
PPI (transductive) Yang et al. (2020)	56,944	1,612,348	50	121

Table 12: Datasets statistics.

(GCNII) Chen et al. (2020). The details of the network architecture and the experiment are available in Table 18. Our results demonstrate that the Multiscale Gradients Computation method achieves comparable accuracy across various pooling techniques while significantly reducing training costs.

	Fine Grid	Random	Subgraphs	Random and Subgraphs
k=6	77.86% (0.07%)	79.14% (0.17%)	77.92% (0.41%)	77.96% (0.07%)
k=10	79.70% (0.44%)	79.78% (0.32%)	79.41% (0.12%)	79.42% (0.25%)
k=20	80.16% (0.30%)	80.75% (0.18%)	80.51% (0.23%)	80.67% (0.22%)

Table 13: Results on full Shapenet dataset using DGCNN. Results represent Mean test IoU, using **multiscale training**.

	Fine Grid	Random	Subgraphs	Random and Subgraphs
k=6	18,596	13,981	15,551	14,760
k=10	20,729	15,409	17,015	16,288
k=20	29,979	21,081	22,646	21,891

Table 14: Average epoch time for Shapenet 'Airplane' dataset using DGCNN, using **multiscale training**. Time is calculated in $mili - seconds$.

	Fine Grid	Random	Subgraphs	Random and Subgraphs
k=6	44,849	28,666	30,265	29,465
k=10	54,064	34,528	36,094	35,299
k=20	89,923	58,840	60,660	59,814

Table 15: Average epoch time for Shapenet 'Airplane' dataset using DGCNN, using **multiscale gradient computation training**. Time is calculated in $mili - seconds$.

Component	**Details**
Dataset	ShapeNet Chang et al. (2015), 'Airplanes' category
Network architecture	DGCNN Wang et al. (2018) with 3 layers and 64 hidden channels
Loss function	Negative Log Likelihood Loss
Optimizer	Adam Kingma & Ba (2014)
Learning rate	1×10^{-3}
Batch size	16 for multiscale training, 12 for multiscale gradient computation
Multiscale levels	3, each time reducing graph size by half
Epochs per level	Fine training used 100 epochs. Multiscale training used [40, 80, 160] epochs for the 3 levels.
Mulstiscale gradients level	2, coarse graph reduces graph size by 0.75%
Multiscale gradients strategy	50 epochs using multilevels gradients, 50 epochs using original fine data
Evaluation metric	Mean IoU (Intersection over Union)

Table 16: Experimental setup for ShapeNet dataset.

Method	F1 Score
Fine Grid	99.27% (0.11%)
Random	99.18% (0.02%)
Topk	99.23% (0.04%)
Subgraph	99.21% (1.40%)
Random and Subgraph	99.20% (0.01%)

Table 17: Results on PPI (inductive) dataset using GCNII and multiscale gradient computation method.

Component	**Details**
Dataset	PPI Zitnik & Leskovec (2017)
Network architecture	GCNII Chen et al. (2020) with 9 layers and 2048 hidden channels
Loss function	Binary Cross Entropy with logits
Optimizer	Adam Kingma & Ba (2014)
Learning rate	5×10^{-4}
Wight initialization	Xavier Glorot & Bengio (2010)
Batch size	1
Baseline training	1000 epochs
Mulstiscale gradients level	2, coarse graph reduces graph size by 0.75%
Multiscale gradients strategy	500 epochs using multilevels gradients, 500 epochs using original fine data
Evaluation metric	F1

Table 18: Experimental setup for PPI dataset.