FIT-GNN: Faster Inference Time for GNNs Using Coarsening

Shubhajit Roy[1,†], **Hrriday Ruparel**[1,†], **Kishan Ved**[1,†] and **Anirban Dasgupta**[1]
royshubhajit@iitgn.ac.in, hrriday.ruparel@iitgn.ac.in, kishan.ved@iitgn.ac.in, anirbandg@iitgn.ac.in
[1]Indian Institute of Technology Gandhinagar

Abstract

Scalability of Graph Neural Networks (GNNs) remains a significant challenge, particularly when dealing with large-scale graphs. To tackle this, coarsening-based methods are used to reduce the graph into a smaller graph, resulting in faster computation. Nonetheless, prior research has not adequately addressed the computational costs during the inference phase. This paper presents a novel approach to improve the scalability of GNNs by reducing computational burden during both training and inference phases. We demonstrate two different methods (Extra-Nodes and Cluster-Nodes). Our study also proposes a unique application of the coarsening algorithm for graph-level tasks, including graph classification and graph regression, which have not yet been explored. We conduct extensive experiments on multiple benchmark datasets in the order of $100K$ nodes to evaluate the performance of our approach. The results demonstrate that our method achieves competitive performance in tasks involving classification and regression on nodes and graphs, compared to traditional GNNs, while having single-node inference times that are orders of magnitude faster. Furthermore, our approach significantly reduces memory consumption, allowing training and inference on low-resource devices where traditional methods struggle.

1 Introduction

Graph Neural Networks (GNNs) have proven to be a tool of surprising versatility and modeling capacity. However, there are still a few issues that limit their general applicability, the chief among these being the scalability of the GNN models. This scalability issue is problematic for both the training and the inference phases. Previous research has attempted to address this by reducing the graph into a smaller graph in order to reduce the computational cost. One such broad technique is graph coarsening methods, which judiciously remove nodes and edges from the graph. However, there are still certain points of weakness of these algorithms. For one, coarsening methods can potentially remove important node information. Secondly, and more pertinent to the current discussion, coarsening methods only reduce the training computation; they do not address the inference time. The next obvious question would be to ask why we cannot use coarsening methods for inference as well. Given a graph G, coarsening algorithms create a coarsened graph G'. While G' is smaller than G, the nodes in G' are combinations of nodes in G, which can be both train and test nodes. As a result, the prediction per test node is not available. In another scenario, suppose a node $v' \in G'$ is a combination of $v_i, v_j, v_k \in G$ where v_i, v_j, v_k are test nodes. Also, let us assume that the classes of v_i, v_j, v_k are $0, 0, 2$ respectively. The coarsening algorithm will take the majority label and assign class 0 to v'. Hence, the model will be trained to predict only the class 0, leading to discarding the quantification of the model's performance on predicting less represented nodes. This is the research gap that we seek to address in this work. We use coarsening algorithms to partition the graph into a set of subgraphs $\mathcal{G}_s = \{G_1, G_2, \ldots, G_k\}$. We show methods to train on these subgraphs that show equivalent results to non-coarsened baselines and sometimes perform better. This comes hand in hand with a significant reduction in inference time and memory consumption.

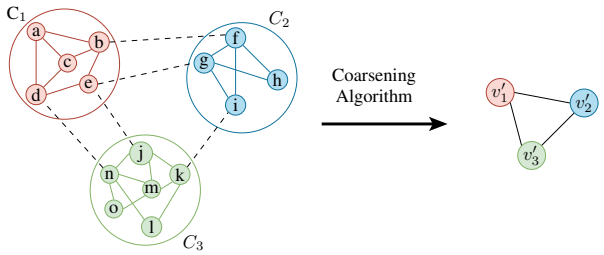

Figure 1: The figure shows how the coarsening algorithm reduces a graph G with 15 nodes to a coarsened graph G' with 3 nodes. It also shows which nodes belong to which partition.

We demonstrate two different methods to append additional nodes in the subgraphs— **Extra Nodes** and **Cluster Nodes** that respectively append additional nodes to each subgraph, either by adding neighboring nodes, or by adding rep-

[†]These authors contributed equally to this work

resentational nodes corresponding a neighboring cluster. We theoretically characterize the upper bound of the coarsening ratio for which the time complexity of our approach is better than the naive uncoarsened approach. In node classification and node regression tasks, the baselines typically require the entire dataset during inference to get a prediction for a single node. In contrast, our method only needs to pass the subgraph to which the node belongs. Our results show up to $\mathbf{100}\times$ **reduction in inference time** and up to $\mathbf{70}\times$ **less memory** than the baseline while achieving equivalent or better performance.

2 Preliminaries

2.1 Graph Notations

Given an undirected graph $G = (V, E, X, W)$, V is the vertex set, E is the edge set, $X \in \mathbb{R}^{n\times d}$ is the feature matrix, and W is the edge weight matrix. The feature of each node v_i in the graph is the ith row vector in X represented as $x_i \in \mathbb{R}^d$. Let $|V| = n$ be the number of nodes and $|E| = m$ be the number of edges. Let us assume $A \in \{0, 1\}^{n\times n}$ be the adjacency matrix of G, d_i to be the degree of the node v_i. $D \in \mathbb{R}^{n\times n}$ is a diagonal degree matrix with ith diagonal element as d_i. We denote $\mathcal{N}_j(v_i)$ as the j-hop neighbourhood of node $v_i \in V$.

2.2 Graph Neural Network

Graph Neural Network is a neural network designed to work for graph-structured data. The representation of each node in the graph is updated recursively by aggregation and transforming node representations of its neighbors. [Kipf and Welling, 2017] introduced Graph Convolutional Network (GCN) as follows:

$$X_{l+1} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X_l\mathcal{W}_l) \tag{1}$$

where X_l is the node representation after l layers of GCN, $\tilde{A} = A + I$, $\tilde{D} = D + I$, I is an identity matrix of same dimension as A. $\mathcal{W}_l$ is the learnable parameter and σ is a non-linear activation function.

2.3 Graph Coarsening and Graph Partitioning

[Loukas, 2018] discussed multiple coarsening algorithms which create a coarsened graph $G' = (V', E', X', W')$ from a given graph $G = (V, E, X, W)$. We refer the vertex set $V' = \{v'_1, v'_2, \ldots, v'_k\}$ of G' as *coarsened nodes*. Given a coarsening ratio $r \in [0, 1]$, we have $k = n \times r$. We can interpret it as creating k disjoint partitions, $C_1, C_2, \ldots, C_k$, from a graph of n nodes. Mathematically, we create a partition matrix $P \in \{0, 1\}^{n\times k}$, where $P_{ij} = 1$ if and only if node $v_i \in C_j$.

For the coarsened graph, the adjacency matrix $A' = P^TAP$. Similarly, the corresponding degree matrix $D' = P^TDP$. [Huang *et al.*, 2021] used normalized partition matrix $\mathcal{P} = PC^{-\frac{1}{2}}$, where C is defined as a diagonal matrix with diagonal entries $C_{j,j} = |C_j|, j = 1, 2, \ldots, k$.

Along with a coarsened graph, we create a set of disjoint subgraphs $\mathcal{G}_s = \{G_1, G_2, \ldots, G_k\}$ corresponding to the partitions $C_1, C_2, \ldots, C_k$. The number of nodes in each partition C_i is denoted by n_i. Each subgraph G_i is the induced subgraph of G formed by the nodes in C_i. A_i and D_i are the adjacency matrix and degree matrix of G_i respectively.

3 Related Work

The goal of reducing computation costs has been tackled via different methodologies. Research like [Chiang *et al.*, 2019] and [Zeng *et al.*, 2020] use a subgraph sampling-based approach. They sample a small subgraph at each training iteration and train the GNN. Other techniques like layer-wise sampling along with mini-batch training have been studied by [Chen *et al.*, 2018; Chen and Zhu, 2018; Cong *et al.*, 2020; Zou *et al.*, 2019]. [Fahrbach *et al.*, 2020] introduced a new coarsening algorithm based on Schur complements to generate embeddings for vertices in a large graph. [Huang *et al.*, 2021] introduced using GNN for a coarsened graph. The motivation was to utilize graph coarsening algorithms to reduce the number of nodes and edges in the graph, resulting in less training time. This approach helps to train GNN for graphs with a large number of nodes. However, inference time is not reduced because the nodes cannot be removed during inference. This training process also caused the removal of important node-level information. Later, [Xue *et al.*, 2022] explored shifting the training process from graph-level training to subgraph-level. Coarsening algorithms were used to split the graph into $k-$subgraphs, resulting in the removal of a few edges. Further, to speed up the training process, they propose a multi-device training approach where subgraphs are sent to different devices. This results in a faster training process. [Kumar *et al.*, 2023] further proposed the usage of feature-based coarsening algorithms.

4 Methodology

In [Huang *et al.*, 2021], a graph $G = (V, E, X, W)$ is reduced to $G' = (V', E', X', W')$ using a partition matrix P generated from the coarsening algorithm. Here $X' = P^TX$. The labels for the coarsened graph are $Y' = \arg\max(P^TY)$, where Y is the label matrix of G storing the one-hot encoding of the label of each node for the node classification task. While using $\arg\max$, some essential information is lost. Therefore, we follow a subgraph-level training process where we don't remove these label information. To explore another node-level task, such as node regression, functions such as the mean of regression targets wouldn't be appropriate because they lose the variance of the targets. Hence, our goal is to use subgraphs for training and inference purposes.

To tackle the issue of loss of information corresponding to the nodes (and their edges) present at the periphery of the subgraphs after partitioning the graph, we append nodes in two ways:

• **Extra Nodes:** Adding the 1-hop neighbouring nodes in each subgraph G_i denoted by $\mathcal{E}_{G_i}$.

$$\mathcal{E}_{G_i} = \bigcup_{v\in G_i} \{u \in V : u \in \mathcal{N}_1(v) \wedge u \notin G_i\}$$

However, we don't perform backpropagation of the prediction on $\mathcal{E}_{G_i}$. [Xue *et al.*, 2022] introduced this approach to reduce information loss after partitioning. An edge between two Extra Nodes in $\mathcal{E}_{G_i}$ is added if they were connected in G.

• **Cluster Nodes**: Although adding 1-hop neighboring nodes in the subgraph helps recover lost information after

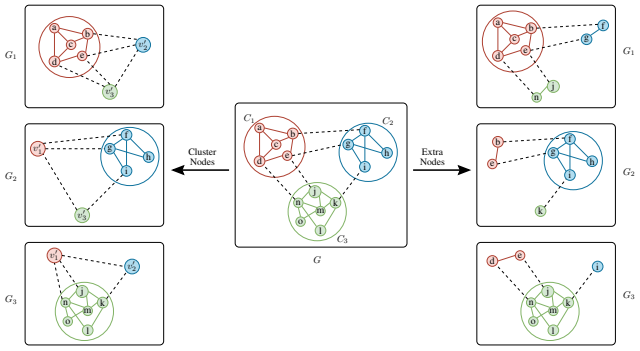

Figure 2: This figure shows the methods of appending additional nodes in G_1, G_2, G_3.

Algorithm 1 Node-Model(L, A, D, X)

Require: Number of Layers L; A; D; $X_0 = X$;
for $i = 1$ to L **do**
  $X_i = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X_{i-1}\mathcal{W}_{i-1})$ # Equation 1
end for
$Z = X_L\mathcal{W}_L$
return Z

partition, for GNN models with a large number of layers, information loss will still persist. Instead of adding the neighboring nodes, we will add a node, namely Cluster Nodes, representing the neighboring cluster, denoted by $\mathcal{C}_{G_i}$.

$$\mathcal{C}_{G_i} = \bigcup_{v \in \mathcal{E}_{G_i}} \{t : P_{v,t} \neq 0\}$$

[Liu *et al.*, 2024] introduced this method to add nodes in the graph instead of adding to separate subgraphs. They also added cross-cluster edges among these Cluster Nodes and proposed a subgraph sampling-based method after adding these Cluster Nodes and cross-cluster edges. In our work, we add cross-cluster edges; however, we do not sample from the constructed set of subgraphs.

Figure 2 illustrates appending these additional nodes using these approaches. The dashed line shows which nodes are added as part of these additional nodes. Predictions on these newly appended nodes will not contribute to loss calculation and weight update.

Lemma 1. *Models with* 1 *layer of GNN cannot distinguish between* G *and* $\mathcal{G}_s$ *when* ***Extra Nodes*** *method is used.*

Proof. There are 2 sets of nodes in G_i as follows:

- The set S_1 of nodes with 1-hop neighbours in G_i.
- The set S_2 of nodes with not all 1-hop neighbors in G_i

Let $\mathcal{I}_i^1$ be the number of nodes whose information does not get passed on after 1 layer of GNN for G_i.

$$\mathcal{I}_i^1 = \left| \bigcup_{v \in S_2} \mathcal{N}_1(v) - V(G_i) \right|$$

Algorithm 2 Training GNN (Train on $\mathcal{G}_s$)

Require: $G = (V, E, X)$; Labels Y; Model M; Loss ℓ;
Apply a coarsening algorithm on G, and output a normalized partition matrix P.
Construct $\mathcal{G}_s = \{G_1, G_2, \ldots, G_k\}$ using P;
Construct mask for each subgraph $Mask = \{mask_1, mask_2, \ldots, mask_k\}$. Masks denote the nodes that are used for training;
Construct feature matrix $\{X_1, X_2, \ldots, X_k\}$ for $\mathcal{G}_s$;
Construct label matrix $\{Y_1, Y_2, \ldots, Y_k\}$ for $\mathcal{G}_s$;
$O \leftarrow []$; $Y \leftarrow []$;
for each subgraph G_i in $\mathcal{G}_s$ **do**
  $O_i = M(A_i, D_i, X_i)$;
  $O.append(O_i[mask_i])$;
  $Y.append(Y_i[mask_i])$;
end for
$Loss = \ell(O, Y)$;
Train M to minimize $Loss$;

Now, $\left| \bigcup_{v \in S_1} \{u \in V : u \in \mathcal{N}_1(v) \wedge u \notin G_i\} \right| = 0$

Also, $\mathcal{E}_{G_i} = \bigcup_{v \in S_2} \{u \in V : u \in \mathcal{N}_1(v) \wedge u \notin G_i\}$

$$\Rightarrow |\mathcal{E}_{G_i}| = \mathcal{I}_i^1$$

Hence when **Extra Nodes** is used, 1 Layer GNN model cannot distinguish G and $\mathcal{G}_s$. □

Using **Cluster Nodes** approach to append additional nodes to subgraphs is better for two reasons:

• The number of nodes added to a subgraph via Cluster Nodes is less than or equal to the number of nodes added via the Extra Nodes approach. This is because, using Extra Nodes, we add all the neighboring nodes. However, we only add one representative node for each neighboring cluster using Cluster Nodes. As the GNN propagation depends on the number of nodes, the time taken to train and infer these modified subgraphs with Cluster Nodes is less than or equal to that with Extra Nodes.

• As mentioned before, the Extra Nodes approach reduces the information loss due to partitioning. However, with multiple layers of GNN, longer dependencies are not captured. In contrast, the Cluster Nodes approach overcomes this by computing a single Cluster Nodes's representation as a combination of features of all the nodes present in the corresponding cluster. This results in sharing further node information in 1-hop. Additionally, the transfer of information from one cluster to another is captured.

4.1 Node-level Task

We use Algorithm 1 with L number of layers to create a standard GNN model for node-level tasks. Training the model on G' is similar to the [Huang *et al.*, 2021] training algorithm. We design Algorithm 2 to train the model on $\mathcal{G}_s$.

Algorithm 3 Graph-Model-$G'(L, A', D', X')$

Require: Number of Layers L; A'; D'; $X_0 = X'$

for $i = 1$ to L **do**
  $X_i = \sigma(\tilde{D'}^{-\frac{1}{2}} \tilde{A'} \tilde{D'}^{-\frac{1}{2}} X_{i-1} \mathcal{W}_{i-1})$ #Equation 1
end for
$\bar{X} = \text{MaxPooling}(X_L)$
$Z = \bar{X}\mathcal{W}_L$
return Z

Algorithm 4 Graph-Model-$\mathcal{G}_s(\mathcal{G}_s, L, mask)$

Require: $\mathcal{G}_s = \{G_1, G_2, \ldots, G_k\}$; Number of Layers L; $mask = \{mask_1, mask_2, \ldots, mask_k\}$;

$X \leftarrow []$
for each subgraph $G_i \in \mathcal{G}_s$ **do**
  for $j = 1$ to L **do**
    $X_j = \sigma(\tilde{D_i}^{-\frac{1}{2}} \tilde{A}_i \tilde{D_i}^{-\frac{1}{2}} X_{j-1} \mathcal{W}_{j-1})$
  end for
  $X.append(X_L[mask_i])$
end for
$\bar{X} = \text{MaxPooling}(X)$
$Z = \bar{X}\mathcal{W}_L$
return Z

For the node classification task, $\arg\max$ is used for creating labels for G' and CrossEntropy as a loss function. For the node regression task, we do not create any coarsened graph. Mean Absolute Error (MAE) is used as the loss function.

4.2 Graph-level Task

Unlike node-level tasks, the number of dataset samples is not decreasing. For a given graph $G \in \mathcal{D}$, we create G' and $\mathcal{G}_s = \{G_1, G_2, \ldots, G_k\}$. Each subgraph contains overlapping nodes when we use Extra Nodes or Cluster Nodes. Therefore, a boolean array $mask_i$ is created for each subgraph G_i such that $True$ is set for nodes that actually belong to the subgraph G_i (not as an Extra Nodes or Cluster Nodes); otherwise, $False$.

Unlike node-level tasks, we cannot train and infer on G' and $\mathcal{G}_s$ using the same model. We use Algorithm 3 to create a model to train on G'. We design Algorithm 4 to train the model on $\mathcal{G}_s$. We use the CrossEntropy loss function for the graph classification task and the Mean Absolute Error(MAE) for the graph regression task. We use Algorithm 2 without the backpropagation for inference.

4.3 Time and Space Complexity

The dimensions of the matrices in Equation 1 are $(n \times n)$ for $\tilde{A}$, $(n \times d)$ for X_l and $(d \times d)$ for $\mathcal{W}_l$. Time complexity for one layer of GNN computation is $\mathcal{O}(n^2 d + nd^2)$. If we take L layers, then the total time is $\mathcal{O}(Ln^2 d + Lnd^2)$. The space complexity is $\mathcal{O}(n^2 + Lnd + Ld^2)$. When we compute on a sparse graph, the time complexity is $\mathcal{O}(m + Lnd^2)$ and space complexity is $\mathcal{O}(m + Lnd + Ld^2)$.

[Huang *et al.*, 2021] improved the time and space complexity (mentioned in Table 1 and Table 2) for training the network by reducing the number of nodes from n to k. However, the inference time and space complexity remain the same.

	Train	Test
Classical	$nd^2 + n^2 d$	$nd^2 + n^2 d$
Coarsening	$kd^2 + k^2 d$	$nd^2 + n^2 d$
FIT-GNN	$kd^2 + k^2 d$ $+ \sum_{i=1}^{k} [\bar{n}_i^2 d + \bar{n}_i d^2]$	$\sum_{i=1}^{k} [\bar{n}_i^2 d + \bar{n}_i d^2]$

Table 1: This table shows the training and inference time complexity of FIT-GNN as compared to classical and coarsening approaches.

	Train	Test
Classical	$n^2 + nd + d^2$	$n^2 + nd + d^2$
Coarsening	$k^2 + kd + d^2$	$n^2 + nd + d^2$
FIT-GNN	$k^2 + kd + d^2$ $+ \max_{i=1}[\bar{n}_i^2 + \bar{n}_i d]$	d^2 $+ \max_{i=1}[\bar{n}_i^2 + \bar{n}_i d]$

Table 2: This table shows the training and inference space complexity of FIT-GNN compared to classical and coarsening approaches.

Let us compare three different models. One is the classical model, where no coarsening or partitioning is done; the second is the [Huang *et al.*, 2021] approach; third is our approach where from the graph G we create G' and $\mathcal{G}_s$.

The inference time complexity of our model is $\sum_{i=1}^{k} [\bar{n}_i^2 d + \bar{n}_i d^2]$, where $\bar{n}_i = n_i + \phi_{\max}$ where $\phi_{\max} = \max_i(\mathcal{E}_{G_i})$ for Extra Nodes or $\phi_{\max} = \max_i(\mathcal{C}_{G_i})$ for Cluster Nodes is the maximum number of additional nodes added per subgraph. Table 1 and Table 2 show the comparison between different approaches.

Lemma 2. *The inference time complexity is at most* $\sum_{i=1}^{k} [(n_i + \phi_{\max})^2 d + (n_i + \phi_{\max}) d^2] \leq n^2 d + nd^2$ *for* $r \leq \frac{d-2}{d+\phi_{\max}}$ *and* $\phi_{\max} \leq \frac{n^2 - \sum_{i=1}^{k} n_i^2}{nd}$

Proof.

$$\sum_{i=1}^{k} [(n_i + \phi_{\max})^2 d + (n_i + \phi_{\max}) d^2]$$

$$= \sum_{i=1}^{k} [n_i^2 d + \phi_{\max}^2 d + 2 n_i \phi_{\max} d + n_i d^2 + \phi_{\max} d^2]$$

$$= k\phi_{\max}^2 d + 2n\phi_{\max} d + nd^2 + k\phi_{\max} d^2 + \sum_{i=1}^{k} n_i^2 d$$

$$= nr\phi_{\max}^2 d + 2n\phi_{\max} d + nd^2 + nr\phi_{\max} d^2 + \sum_{i=1}^{k} n_i^2 d$$

$$(\text{where } k = nr)$$

$$= nd^2 + n\phi_{\max} d(r\phi_{\max} + 2 + rd) + \sum_{i=1}^{k} n_i^2 d$$

Now we know, $r \leq \frac{d-2}{d+\phi_{\max}} \Rightarrow r\phi_{\max} + 2 + rd \leq d$ and $\phi_{\max} \leq \frac{n^2 - \sum_{i=1}^{k} n_i^2}{nd} \Rightarrow n\phi_{\max} d \leq n^2 - \sum_{i=1}^{k} n_i^2$.
Using this, we can say,

$$
\begin{aligned}
& nd^2 + n\phi_{\max} d(r\phi_{\max} + 2 + rd) + \sum_{i=1}^{k} n_i^2 d \\
\leq & nd^2 + (n^2 - \sum_{i=1}^{k} n_i^2)d + \sum_{i=1}^{k} n_i^2 d \\
= & nd^2 + n^2 d
\end{aligned}
$$

Hence, $\sum_{i=1}^{k}[(n_i + \phi_{\max})^2 d + (n_i + \phi_{\max})d^2] \leq n^2 d + nd^2$. □

For a given graph, if the above condition is satisfied, our approach has better inference time and space complexity than other approaches.

5 Experiments

Once we have constructed G' and $\mathcal{G}_s$, we train FIT-GNN in 4 different setups:

• **Gc-train-to-Gs-train**: Train the GNN model on G', then use the learned weight as an initialization for training and final inference on $\mathcal{G}_s$.

• **Gc-train-to-Gs-infer**: Train the GNN model on G', then infer on $\mathcal{G}_s$.

• **Gs-train-to-Gs-infer**: Train and infer only on $\mathcal{G}_s$.

• **Gc-train-to-Gc-infer**: Unlike node-level tasks, we can use G' to infer for graph-level tasks. Therefore, in this setup, we train and infer on G'.

For tasks like node regression, we can only perform **Gs-train-to-Gs-infer** because a coarsened graph is not created.

5.1 Experimental Setup

Dataset descriptions are in Section A of the appendix, and Section C of the appendix covers the hyperparameters, device configuration, and key packages used for the experiments. Source Code present at https://github.com/Roy-Shubhajit/FIT-GNN.

Node regression

We use three real-world network datasets for the node regression task: Chameleon, Crocodile, and Squirrel [Rozemberczki *et al.*, 2019]. The nodes are divided into train (30%), validation (20%), and test (50%) sets. We perform 20 runs and report the average and standard deviation of the smallest 10 normalized MAE, with normalization achieved by dividing the loss by the standard deviation of the target values.

Node classification

We conducted node classification experiments on six real-world network datasets: Cora [McCallum *et al.*, 2000], Citeseer [Giles *et al.*, 1998], Pubmed [Sen *et al.*, 2008], Coauthor Physics [Shchur *et al.*, 2018], DBLP [Tang *et al.*, 2008], and a subset of OGBN-Products [Hu *et al.*, 2020] extracted using Leiden algorithm [Traag *et al.*, 2019]. We used three types of splits for the datasets: we applied the public split from [Yang *et al.*, 2016] for Cora, Citeseer, and PubMed. In the "few" split, we set aside labeled nodes per class for training and validation, while the rest form the test set. We ran 20 trials for each configuration and reported the average and standard deviation of the top 10 accuracies.

	r	Dataset		
		Chameleon	**Crocodile**	**Squirrel**
Classical	-	0.844 ± 0.000	0.853 ± 0.000	0.809 ± 0.000
FIT-GNN	0.1	$\mathbf{0.460 \pm 0.002}$	$\mathbf{0.362 \pm 0.002}$	$\mathbf{0.648 \pm 0.002}$
	0.3	0.499 ± 0.006	0.364 ± 0.001	0.672 ± 0.003

Table 3: This table shows the normalized MAE loss (lower is better) for node regression tasks on different datasets with **Gs-train-to-Gs-infer** experiment setup and **Cluster Nodes** method.

Graph Classification

Graph Classification has been conducted on the *PROTEINS* and *AIDS* dataset [Morris *et al.*, 2020]; we randomly split the dataset into a $2:1:1$ ratio of train, validation, and test.

Graph Regression

We perform graph regression on the QM9 [Wu *et al.*, 2017], and a subset of ZINC [Gómez-Bombarelli *et al.*, 2016] datasets, using the same splits as in graph classification. [Gilmer *et al.*, 2017] grouped the 19 properties of each molecule in *QM9* dataset into four broad categories. We predict one property from each of the four broad categories. The four specific properties are the following: the dipole moment (μ), the gap between ϵ_{HOMO} and ϵ_{LUMO}, zero-point vibration energy (ZPVE), and at 298.15K, the atomization energy (U^{ATOM}) of each molecule.

5.2 Result and Analysis

Table 3 shows the node regression error and standard deviation on coarsening ratios 0.1 and 0.3. All the experiments for node regression are shown with **Cluster node** method. With a higher coarsening ratio, more subgraphs are created. As the coarsening ratio increases, the size of each subgraph also reduces, and important neighboring node information gets lost. Hence, the performance decreases with an increase in the coarsening ratio.

Table 4 shows the results for node classification accuracy and standard deviation on different coarsening ratios on the *Cora* dataset with **Cluster Nodes** approach. With coarsening ratios 0.1 and 0.3, the observed performance is better than other coarsening ratios, and the inference time taken is also less. This motivates us to experiment on other datasets with lower coarsening ratios. Table 5 shows the results for node classification on different datasets using coarsening ratios 0.1 and 0.3. From the results, it can be observed that the accuracy is comparable to the baseline. Since experimenting on the full *OGBN-Products* dataset is not feasible due to memory constraints, we create a proxy graph by taking the maximum-sized communities with the number of nodes that sum up to 165000. For the subset of the *OGBN-Products* graph, the performance is better with the baseline even with 0.56% training nodes. For larger datasets like *Pubmed* and *Physics Coauthor*, a lower coarsening ratio doesn't perform better than the baseline.

	Fixed		Few	
r	**[Huang *et al.*, 2021]**	**FIT-GNN**	**[Huang *et al.*, 2021]**	**FIT-GNN**
0.1	0.772 ± 0.006	$\mathbf{0.829 \pm 0.005}$	0.676 ± 0.051	$\mathbf{0.695 \pm 0.016}$
0.3	$\mathbf{0.817 \pm 0.016}$	0.815 ± 0.005	0.694 ± 0.045	$\mathbf{0.719 \pm 0.011}$
0.5	$\mathbf{0.827 \pm 0.001}$	0.807 ± 0.005	0.688 ± 0.045	$\mathbf{0.729 \pm 0.007}$
0.7	$\mathbf{0.824 \pm 0.002}$	0.800 ± 0.008	0.679 ± 0.046	$\mathbf{0.706 \pm 0.012}$

Table 4: Results on *Cora* Dataset. The metrics shown here include the average and standard deviation of the top 10 accuracies over 20 runs. We show **Gc-train-to-Gs-train** experimental setup result for fixed and **Gc-train-to-Gs-infer** experimental result for few. The highest accuracy for each coarsening ratio is shown in bold.

		$\mathbf{r = 0.1}$		$\mathbf{r = 0.3}$	
Dataset	**Split**	**[Huang *et al.*, 2021]**	**FIT-GNN**	**[Huang *et al.*, 2021]**	**FIT-GNN**
Citeseer	Few	0.583 ± 0.063	$\mathbf{0.614 \pm 0.021}$	0.581 ± 0.052	$\mathbf{0.624 \pm 0.022}$
	Fixed	$\mathbf{0.711 \pm 0.004}$	0.677 ± 0.023	$\mathbf{0.714 \pm 0.003}$	0.703 ± 0.007
Pubmed	Few	$\mathbf{0.685 \pm 0.052}$	0.627 ± 0.116	$\mathbf{0.687 \pm 0.042}$	0.676 ± 0.038
	Fixed	$\mathbf{0.783 \pm 0.005}$	0.704 ± 0.022	$\mathbf{0.784 \pm 0.004}$	0.677 ± 0.013
DBLP	Few	0.679 ± 0.056	$\mathbf{0.683 \pm 0.072}$	0.648 ± 0.052	$\mathbf{0.670 \pm 0.034}$
	Random	0.760 ± 0.021	$\mathbf{0.789 \pm 0.005}$	0.745 ± 0.019	$\mathbf{0.771 \pm 0.007}$
Physics Coauthor	Few	$\mathbf{0.878 \pm 0.036}$	0.783 ± 0.028	$\mathbf{0.908 \pm 0.023}$	0.877 ± 0.019
	Random	$\mathbf{0.915 \pm 0.014}$	0.830 ± 0.048	$\mathbf{0.934 \pm 0.006}$	0.910 ± 0.011
OGBN-Products	Random	0.296 ± 0.034	$\mathbf{0.346 \pm 0.014}$	0.316 ± 0.016	$\mathbf{0.406 \pm 0.009}$

Table 5: Results for node classification tasks with accuracy as the metric (higher is better). We use the **Cluster Nodes** method to append additional nodes to subgraphs and **Gc-train-to-Gs-train** as experimental setup.

		FIT-GNN	
Dataset	**Classical**	$\mathbf{r = 0.3}$	$\mathbf{r = 0.5}$
ZINC (Subset)	0.688	**0.578**	0.657
QM9 (μ)	0.869	**0.841**	0.948
QM9 ($\Delta\epsilon$)	1.012	**0.875**	0.969
QM9 (ZPVE)	1.091	**0.818**	0.840
QM9 (U^{ATOM})	1.078	0.754	**0.630**

Table 6: Graph regression tasks results on *ZINC (Subset)* and *QM9*. The metrics shown are in terms of normalized MAE (lower is better) with **Gs-train-to-Gs-infer** experimental setup. The **Cluster Nodes** method is used to append additional nodes to subgraphs.

		Gc-train-to -Gs-infer		Gs-train-to -Gs-infer	
	Classical	$\mathbf{r = 0.3}$	$\mathbf{r = 0.5}$	$\mathbf{r = 0.3}$	$\mathbf{r = 0.5}$
AIDS	0.792	0.810	0.836	0.812	**0.843**
PROTEINS	0.623	0.636	0.681	0.812	**0.821**

Table 7: Results for the graph classification task on *AIDS* and *PROTEINS* are presented. The metrics reflect accuracy; higher values indicate better performance. The table also compares two experimental setups.

Table 6 shows the results for the graph regression task on ZINC (Subset) and QM9 dataset. All the results show that the FIT-GNN model's performance is comparable to the baselines. It is also observed that a lower coarsening ratio yields better loss, which implies that the model performs better on molecular graphs when the subgraph size is high. It implies that finer subgraphs lose out global information about the molecule, which is necessary for the prediction.

Table 7 displays graph classification results using two experimental setups across two datasets. Training on coarsened graphs outperforms the baseline but discards important information. Conversely, training on subgraphs significantly improves predictions compared to the baseline.

Table 8 and Table 9 show the comparison of performance with different coarsening algorithms used by [Huang *et al.*, 2021]. We find that **variation_neighborhoods** is more consistent among all other coarsening algorithms, therefore we use it to report other results also.

5.3 Inference Time and Memory Comparison

As mentioned in Section 4.3, our inference time is lesser than the standard GNN if certain conditions are met. Therefore, we empirically show reduction of time and space by our FIT-GNN model during inference. For recording the inference time, we use the Python **time** package to calculate the difference in time before and after the inference step.

In Table 10, we show the average time to predict for 1000 nodes in the baseline and FIT-GNN models for node regression and classification tasks. The baseline model processes the entire graph, increasing inference time, especially for large graphs. In contrast, the FIT-GNN model only requires the relevant subgraph, resulting in faster predictions. For larger datasets like the *OGBN-Products*, we see up to $\mathbf{100\times}$ **reduction in inference time**. Table 17 in the Ap-

Coarsening Method	Cora		Chameleon	
	$\mathbf{r = 0.1}$	$\mathbf{r = 0.3}$	$\mathbf{r = 0.1}$	$\mathbf{r = 0.3}$
variation_neighborhoods	0.821 ± 0.003	0.803 ± 0.005	$\mathbf{0.465 \pm 0.005}$	$\mathbf{0.481 \pm 0.006}$
algebraic_JC	$\mathbf{0.827 \pm 0.006}$	$\mathbf{0.804 \pm 0.003}$	0.544 ± 0.005	0.504 ± 0.003
kron	0.758 ± 0.004	0.800 ± 0.002	0.571 ± 0.004	0.580 ± 0.013
heavy_edge	0.736 ± 0.012	0.773 ± 0.006	0.572 ± 0.002	0.531 ± 0.002
variation_edges	0.484 ± 0.006	0.471 ± 0.012	0.513 ± 0.003	0.542 ± 0.005
variation_cliques	0.751 ± 0.012	0.801 ± 0.009	0.674 ± 0.009	0.679 ± 0.008

Table 8: Results comparing various coarsening methods on *Cora* and *Chameleon* datasets and the metric for each dataset are accuracy (higher the better) and normalized MAE (lower the better) respectively.

Coarsening Method	PROTEINS		ZINC (subset)	
	$\mathbf{r = 0.3}$	$\mathbf{r = 0.5}$	$\mathbf{r = 0.3}$	$\mathbf{r = 0.5}$
variation_neighborhoods	0.652	**0.739**	**0.629**	0.859
algebraic_JC	**0.783**	0.478	0.823	0.810
kron	0.522	0.652	0.663	1.201
heavy_edge	0.565	0.609	2.114	1.227
variation_edges	0.652	0.565	1.548	1.883
variation_cliques	0.696	0.652	0.689	**0.742**

Table 9: Results comparing various coarsening methods on *PROTEINS* and *ZINC (subset)* datasets. The metric for the *PROTEINS* dataset is accuracy (higher the better), and for the *ZINC (subset)* is normalized MAE (lower the better)

Dataset	Classical	FIT-GNN	
		$\mathbf{r = 0.1}$	$\mathbf{r = 0.3}$
Chameleon	0.0027	0.0016	**0.0014**
Squirrel	0.0081	0.0017	**0.0014**
Crocodile	0.0070	0.0015	**0.0015**
Cora	0.0026	**0.0019**	0.0020
Citeseer	0.0031	**0.0018**	0.0019
Pubmed	0.0042	0.0019	**0.0018**
DBLP	0.0063	0.0020	**0.0018**
Physics Coauthor	0.0252	0.0020	**0.0017**
OGBN-Products	0.1762	0.0017	**0.0016**

Table 10: Inference time (sec) comparison for a single node using standard GNN as a baseline and FIT-GNN with two different coarsening ratios. Lower is better. Here, we used **Cluster Nodes**.

pendix shows the detailed comparison of memory consumptions for different datasets for both **Extra Nodes** and **Cluster Nodes** method along with the baseline memory, which highlights that the FIT-GNN model uses up to $\mathbf{70}\times$ **less memory** than the baseline.

Table 11 compares the inference time of graph classification and graph regression task. We randomly select 1000 graphs from the test split and infer them for baseline and FIT-GNN. We also append additional nodes to each subgraph using **Cluster Nodes** method. The table shows how our method is comparable and sometimes faster than the baseline. Also, we observe that the inference time increases with a higher coarsening ratio. Because, with a higher coarsening ratio, the number of nodes in G' increases, resulting in more edges. In table 16 in the Appendix, we show the estimated $\phi_{\max}$ value

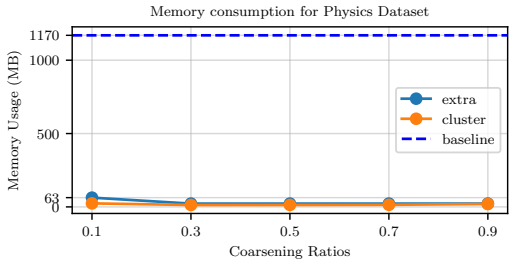

Figure 3: Comparison of inference time memory consumption (in MB) for *Coauthor Physics* Dataset between baseline and FIT-GNN

Dastaset	Classical	FIT-GNN	
		$\mathbf{r = 0.3}$	$\mathbf{r = 0.5}$
ZINC (subset)	**0.00184**	**0.00184**	0.00190
QM9	**0.00173**	0.00180	0.00191
AIDS	0.00163	**0.00155**	0.00163
PROTEINS	0.00165	**0.00160**	0.00163

Table 11: Inference time (sec) comparison for the graph-level task. We use the **Cluster Nodes** method to add additional nodes to each subgraph and **Gc-train-to-Gc-infer** experiment setup

from the feature dimension of the nodes and the coarsening ratio. Combinations with negative numbers indicate the FIT-GNN-based approach won't reduce the inference time.

Overall, the inference time and memory for all the tasks mentioned are drastically less than the classical approach while maintaining the performance mentioned in the previous section.

6 Conclusion

In this paper, we have focused on inference time and memory and presented a new way to utilize existing graph coarsening algorithms for GNNs. We have provided theoretical insight corresponding to the number of nodes in the graph for which the FIT-GNN model reduces the time and space complexity. Empirically, we have shown that our method is comparable to the uncoarsened baseline while being orders of magnitude faster in terms of the inference time and consuming a fraction of the memory. A few possible future directions involve studying directed and weighted graphs, focusing on the theoretical connections between Extra Nodes and Cluster Nodes.

References

[Chen and Zhu, 2018] Jianfei Chen and Jun Zhu. Stochastic training of graph convolutional networks, 2018.

[Chen *et al.*, 2018] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018.

[Chiang *et al.*, 2019] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 257–266, New York, NY, USA, 2019. Association for Computing Machinery.

[Cong *et al.*, 2020] Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 1393–1403, New York, NY, USA, 2020. Association for Computing Machinery.

[Fahrbach *et al.*, 2020] Matthew Fahrbach, Gramoz Goranci, Richard Peng, Sushant Sachdeva, and Chi Wang. Faster graph embeddings via coarsening. *CoRR*, abs/2007.02817, 2020.

[Giles *et al.*, 1998] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. Citeseer: an automatic citation indexing system. In *Proceedings of the Third ACM Conference on Digital Libraries*, DL '98, page 89–98, New York, NY, USA, 1998. Association for Computing Machinery.

[Gilmer *et al.*, 2017] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017.

[Gómez-Bombarelli *et al.*, 2016] Rafael Gómez-Bombarelli, David Duvenaud, José Miguel Hernández-Lobato, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *CoRR*, abs/1610.02415, 2016.

[Hu *et al.*, 2020] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: datasets for machine learning on graphs. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[Huang *et al.*, 2021] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. Scaling up graph neural networks via graph coarsening. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 675–684, New York, NY, USA, 2021. Association for Computing Machinery.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[Kumar *et al.*, 2023] Manoj Kumar, Anurag Sharma, Shashwat Saxena, and Sandeep Kumar. Featured graph coarsening with similarity guarantees. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 17953–17975. PMLR, 23–29 Jul 2023.

[Liu *et al.*, 2024] Juncheng Liu, Bryan Hooi, Kenji Kawaguchi, Yiwei Wang, Chaosheng Dong, and Xiaokui Xiao. Scalable and effective implicit graph neural networks on large graphs. In *The Twelfth International Conference on Learning Representations*, 2024.

[Loukas, 2018] Andreas Loukas. Graph reduction with spectral and cut guarantees, 2018.

[McCallum *et al.*, 2000] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, Jul 2000.

[Morris *et al.*, 2020] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.

[Rozemberczki *et al.*, 2019] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding, 2019.

[Sen *et al.*, 2008] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, Sep. 2008.

[Shchur *et al.*, 2018] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868, 2018.

[Tang *et al.*, 2008] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, page 990–998, New York, NY, USA, 2008. Association for Computing Machinery.

[Traag *et al.*, 2019] V. A. Traag, L. Waltman, and N. J. van Eck. From louvain to leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1):5233, 2019.

[Wu *et al.*, 2017] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay S. Pande. Moleculenet: A benchmark for molecular machine learning. *CoRR*, abs/1703.00564, 2017.

[Xue *et al.*, 2022] Zihui Xue, Yuedong Yang, Mengtian Yang, and Radu Marculescu. SUGAR: efficient subgraph-level training via resource-aware graph partitioning. *CoRR*, abs/2202.00075, 2022.

[Yang *et al.*, 2016] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 40–48. JMLR.org, 2016.

[Zeng *et al.*, 2020] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.

[Zou *et al.*, 2019] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. *Layer-dependent importance sampling for training deep and large graph convolutional networks*. Curran Associates Inc., Red Hook, NY, USA, 2019.

A Dataset Description

Dataset	Number of Graphs	Average Nodes	Average Edges	Features	Classes
PROTEINS	1113	19	72	3	2
AIDS	2000	7	16	38	2

Table 12: Summary of datasets used for Graph classification Dataset

Dataset	Number of Graphs	Average Nodes	Average Edges	Features	Number of Targets
QM9	130831	8	18	11	19
ZINC (subset)	10000	11	25	1	1

Table 13: Summary of datasets used for Graph regression Dataset

Dataset	Nodes	Edges	Features	Classes
Cora	2708	5278	1433	7
Citeseer	3327	4552	3703	6
Pubmed	19717	44324	500	3
DBLP	17716	52867	1639	4
Physics Coauthor	34493	247962	8415	5
OGBN-Products	165000	4340428	100	47

Table 14: Summary of datasets used for Node classification Dataset

Dataset	Nodes	Edges	Features	Number of Targets
Chameleon	2277	31396	128	1
Squirrel	5201	198423	128	1
Crocodile	11631	170845	128	1

Table 15: Summary of datasets used for Node regression Dataset

Dataset	$\mathbf{r = 0.1}$	$\mathbf{r = 0.3}$	$\mathbf{r = 0.5}$	$\mathbf{r = 0.7}$
PROTEINS	6.999	0.333	-1.000	-1.571
AIDS	322.000	82.000	34.000	13.429
QM9	79.000	19.000	7.000	1.857
ZINC (subset)	-11.000	-4.333	-3.000	-2.429

Table 16: Calculated $\phi_{\max}$ values for different datasets and coarsening ratios as per Lemma 1.

B More on Extra Nodes and Cluster Nodes

B.1 Extra Nodes

There are 3 sets of nodes in G_i.

- Nodes with 1-hop and 2-hop neighbours in G_i. Let us call this set S_1
- Nodes with 1-hop neighbours in G_i but $\exists v$ in 2-hop neighbourhood that is not in G_i. Let us call this set S_2
- Node where $\exists v$ in 1-hop and 2-hop neighbourhood that is not in G_i. Let us call this set S_3.

Let $\mathcal{I}_i^2$ be the number of nodes whose information doesn't get passed on after 2 layer of GNN for G_i.

$$\mathcal{I}_i^2 = \left| \bigcup_{v \in S_3} \mathcal{N}_2(v) - V(G_i) \right|$$

When we use **Extra Nodes**, the information loss can be written as follows:

$$\mathcal{I}_i^2 = \left| \bigcup_{v \in S_3} \mathcal{N}_2(v) - \mathcal{E}_{G_i} \right|$$

The above entity will depend on the density of the subgraphs formed and the number of connections each subgraph shares with each other. An algorithm with an objective to reduce this entity for all subgraphs will lose the least amount of information when **Extra Node** method is used.

B.2 Cluster Nodes

Given a partition matrix P, the features of the coarsened node are $X' = P^T X$. Given a normalized partition matrix, the features of a node $v_i' \in G'$ is the degree-weighted average of the features of nodes in C_i. This is one of the functions f used to create the features of the cluster node from C_i.

Previously, according to Lemma 1, there is no information loss when using the 1 layer of GNN and **Extra Nodes** method. It was also easy to quantify in terms of the number of nodes. However, it is different for **Cluster Nodes**. Let us discuss the issues first.

• Only a weighted version of node information is shared with the subgraph. Suppose $v_c, v_d \in G_i$ is connected to $v_a, v_b \in G_j$. Then the information contributed by these nodes is $\frac{d(v_a)x_a + d(v_b)x_b}{\sum_p d(v_p)}$. Here $d(v_p)$ represents the degree of node v_p, and x_p represents the feature of node v_p.

• Other node information will also be shared which is $\frac{\sum_{p \neq v_a, v_b} d(v_p)x_p}{\sum_p d(v_p)}$. This will capture further dependencies.

The performance of **Cluster Node** will depend on some distance or similarity metric between x_c, x_d and $f(C_j)$.

C Experiment Details (Parameters and Device Configuration)

For node classification and node regression tasks, we use Adam Optimizer with a learning rate of 0.01 and L_2 regularization with 0.0005 weight. We use Adam Optimizer with a learning rate of 0.0001 and L_2 regularization with 0.0005 weight for graph level tasks. For both coarsened graph and set of subgraph based model, we set epochs to 300 and the number of layers of GCN to 2 for training. We set the hidden dimensions for each layer of GCN to 512.

The device configurations are Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz, 256GB RAM, NVIDIA A100 40GB GPU. We use Pytorch Geometric to to train our models.

D Results

This section shows the memory consumption for different datasets with different methods. Also, we show how the FIT-GNN model performs with different methods of appending additional nodes to subgraphs at different coarsening ratios.

Dataset	Split	FIT-GNN r = 0.1	FIT-GNN r = 0.3	FIT-GNN r = 0.5	FIT-GNN r = 0.7	Baseline
Cora	**cluster**	1.310	0.648	0.694	0.868	15.743
	extra	2.072	1.134	1.017	1.017	
Citeseer	**cluster**	31.559	2.148	1.253	1.686	49.488
	extra	31.559	3.114	1.788	1.788	
Pubmed	**cluster**	0.613	0.569	0.571	0.573	41.227
	extra	0.798	0.590	0.590	0.590	
DBLP	**cluster**	6.067	2.940	1.847	1.966	118.121
	extra	13.907	4.663	2.649	2.274	
Physics Coauthor	**cluster**	24.025	13.088	13.388	14.353	1169.521
	extra	63.146	22.434	22.434	22.434	
Chameleon	**cluster**	0.212	0.246	0.291	0.438	2.198
	extra	0.653	0.653	0.653	0.592	
Crocodile	**cluster**	1.183	1.188	1.735	1.866	11.562
	extra	4.368	4.368	4.368	3.134	
Squirrel	**cluster**	1.507	1.607	1.903	3.009	9.075
	extra	7.275	7.275	7.275	6.953	
OGBN-Products	**cluster**	6.047	5.613	8.636	-	208.029

Table 17: Summary of memory consumption of datasets used for node-level tasks. All units are in MegaBytes (MB)

r	Extra Nodes	Cluster Nodes
0.1	0.798 ± 0.008	$\mathbf{0.824 \pm 0.003}$
0.3	0.798 ± 0.008	$\mathbf{0.807 \pm 0.004}$
0.5	0.807 ± 0.004	$\mathbf{0.814 \pm 0.004}$
0.7	0.797 ± 0.007	$\mathbf{0.808 \pm 0.006}$

Table 18: Average of top 10 accuracies (higher the better) of node classification task on *Cora* dataset with **Gs-train-to-Gs-infer** experimental setup. The table also demonstrates the superior performance of the Cluster Nodes method over Extra Nodes.

r	Extra Nodes	Cluster Nodes
0.1	0.504 ± 0.004	$\mathbf{0.460 \pm 0.003}$
0.3	0.579 ± 0.006	$\mathbf{0.499 \pm 0.006}$
0.5	0.585 ± 0.004	$\mathbf{0.505 \pm 0.004}$
0.7	$\mathbf{0.504 \pm 0.006}$	0.560 ± 0.004

Table 19: Average of best 10 losses (lesser the better) of node regression task on *Chameleon* dataset with **Gs-train-to-Gs-infer** experimental setup. The table also demonstrates the better performance of the Cluster Nodes method over Extra Nodes.

Dataset	Coarsening Ratio	Top 10 Loss
Chameleon	Baseline	0.844 ± 0.000
	0.1	$\mathbf{0.460 \pm 0.002}$
	0.3	0.499 ± 0.006
	0.5	0.587 ± 0.000
	0.7	0.591 ± 0.080
Crocodile	Baseline	0.853 ± 0.000
	0.1	$\mathbf{0.362 \pm 0.002}$
	0.3	0.364 ± 0.001
	0.5	0.375 ± 0.001
	0.7	0.384 ± 0.002
Squirrel	Baseline	0.809 ± 0.000
	0.1	$\mathbf{0.648 \pm 0.002}$
	0.3	0.672 ± 0.003
	0.5	0.729 ± 0.004
	0.7	0.694 ± 0.004

Table 20: Node Regression Results with **Gs-train-to-Gs-infer** experimental setup.