Multi-Grained Semantics-Aware Graph Neural Networks

Zhiqiang Zhong, Cheng-Te Li, and Jun Pang

Abstract—Graph Neural Networks (GNNs) are powerful techniques in representation learning for graphs and have been increasingly deployed in a multitude of different applications that involve node- and graph-wise tasks. Most existing studies solve either the node-wise task or the graph-wise task independently while they are inherently correlated. This work proposes a unified model, AdamGNN, to interactively learn node and graph representations in a mutual-optimisation manner. Compared with existing GNN models and graph pooling methods, AdamGNN enhances the node representation with the learned multi-grained semantics and avoids losing node features and graph structure information during pooling. Specifically, a differentiable pooling operator is proposed to adaptively generate a multi-grained structure that involves meso- and macro-level semantic information in the graph. We also devise the unpooling operator and the *flyback* aggregator in AdamGNN to better leverage the multi-grained semantics to enhance node representations. The updated node representations can further adjust the graph representation in the next iteration. Experiments on 14 real-world graph datasets show that AdamGNN can significantly outperform 17 competing models on both node- and graph-wise tasks. The ablation studies confirm the effectiveness of AdamGNN's components, and the last empirical analysis further reveals the ingenious ability of AdamGNN in capturing long-range interactions.

Index Terms—Graph neural networks, multi-grained semantic, hierarchical structure, representation learning

1 Introduction

In many real-world applications, such as social networks, recommendation systems, and biological networks, data can be naturally organised as graphs [10]. Nevertheless, working with this powerful node and graph representations remains a challenge, since it requires integrating the rich inherent features and complex structural information. To address this challenge, Graph Neural Networks (GNNs), which generalise deep neural networks to graph-structured data, have drawn remarkable attention from academia and industry, and achieve state-of-the-art performances in a multitude of applications [32], [43]. The current literature on GNNs can be used for tasks with two categories. One is to learn node representations to perform tasks such as link prediction [41] and node classification [15], [34]. The other is to learn graph representations for tasks, such as graph classification [9], [37], [39].

On node-wise tasks, existing GNN models on learning node representations rely on a similar methodology that utilises a GNN layer to aggregate the sampled neighbouring nodes' features in a number of iterations, via non-linear transformation and aggregation functions. Its effectiveness has been widely proved, however, a major limitation of these GNN models is that they are inherently *flat* as they only propagate information across the observed edges in the original graph. Thus, they lack the capacity to encode features in the high-order neighbourhood in the graphs [1], [38]. For example, in an academic collaboration network, *flat* GNN models could capture the micro semantic (e.g., co-authorships) between authors, but neglect their macro semantics (e.g., belonging to different research institutes).

On the other hand, graph classification is to predict the label associated with an entire graph by using the given graph structure and initial node features. Nevertheless, existing GNNs for graph classification are unable to learn graph representations in a *multi-grained* manner, which is crucial to encode better meso- and macro-level graph semantics hidden in the graphs for practical applications such as drug design [27] and program analysis [18]. To remedy this limitation, novel pooling approaches have been proposed, where sets of nodes are recursively aggregated to form super nodes in the pooled graph. DiffPool [37] is a differentiable pooling operator but its assignment matrix is too *dense* [4] to apply on large graphs. g-U-Net [9], SagPool [16], GXN [20] and ASAP [26] are four recent methods that adopt the Top-k selection strategy to address the sparsity concerns of DiffPool. They score nodes based on a learnable projection vector and select a fraction of high scoring nodes as super nodes. However, the pre-defined pooling ratio limits these models' adaptivity on graphs with different sizes, and the Top-k selection may easily lose important node features or graph

- *Zhiqiang Zhong is with the Faculty of Science, Technology and Medicine, University of Luxembourg, 4365 Esch-sur-Alzette, Luxembourg. E-mail: zhiqiang.zhong@uni.lu.*
- *Cheng-Te Li is with the Institute of Data Science and the Department of Statistics, National Cheng Kung University, Tainan 701, Taiwan. E-mail: chengte@mail.ncku.edu.tw.*
- *Jun Pang is with the Faculty of Science, Technology and Medicine, Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, 4365 Esch-sur-Alzette, Luxembourg. E-mail: jun.pang@uni.lu.*

Manuscript received 6 September 2021; revised 25 May 2022; accepted 15 July 2022. Date of publication 29 July 2022; date of current version 5 June 2023. This work was supported in part by the Luxembourg National Research Fund under Grant PRIDE15/10621687/SPsquared, and in part by Ministry of Science and Technology (MOST) of Taiwan under Grants 110-2221-E-006-136-MY3, 110-2221-E-006-001, and 110-2634-F-002-051. (Corresponding authors: Cheng-Te Li and Jun Pang.) Recommended for acceptance by V. Moreira. Digital Object Identifier no. 10.1109/TKDE.2022.3195004

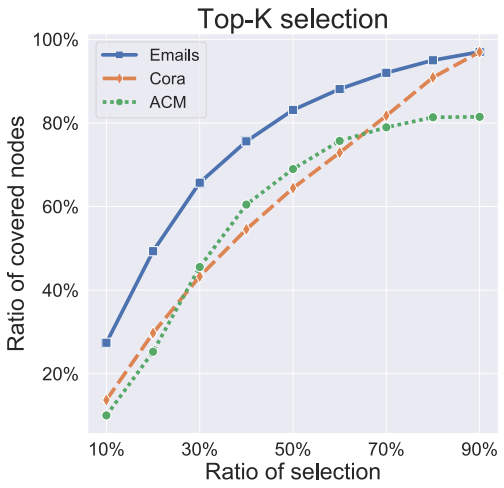

Fig. 1. Ratio of covered nodes with various selection ratio.

structure by simply ignoring low scoring nodes. As shown in Fig. 1 (Section 2), different numbers of k will significantly affect the number of covered nodes of super nodes in the pooled graph, which means the important nodes' features could get lost during the trivial pooling strategy. The hyper-parameter k is also crucial for the final performance [9], thus reduces their convenience in applications.

In the end, we argue that node- and graph-wise tasks are inherently correlated with one another. That said, node representations form graph representation, and graph representation can provide node representations with meso/macro-level semantic information in the graph. Joint modelling with node- and graph-wise tasks will allow GNNs to overcome the limitation of *flat* propagation mode in capturing multi-grained semantics, and the enriched node representation could further ameliorate the graph representation. However, to the best of our knowledge, none of the existing work simultaneously exploit node- and graph-wise tasks, along with capturing multi-grained semantics hidden in the graph, to learn representations of nodes and the graph.

In this work, we propose a novel framework, *Adaptive Multi-grained Graph Neural Networks* (AdamGNN), which integrates graph convolution, adaptive pooling and unpooling operations into one framework to generate both node and graph level representations interactively. Unlike the above-mentioned GNN models, we treat node and graph representation learning tasks in a unified framework so that they can collectively optimise each other during training. In modelling multi-grained semantics, the adaptive pooling and unpooling operators preserve the important node features and hierarchical structural features. More concretely, as shown in Fig. 2a, we employ *(i)* an adaptive graph pooling (AGP) operators to generate a multi-grained structure based on the derived primary node representations by a GNN layer, *(ii)* graph unpooling (GUP) operators to further distribute the explored meso- and macro-level semantics to the corresponding nodes of the original graph, and *(iii)* a *flyback* mechanism to integrate all received multi-grained semantic messages as the evolved node representations. Besides, the attention-enhanced *flyback* aggregator provides a reasonable explanation of the importance of messages from different grains. Experimental results reveal the effectiveness of AdamGNN, and the ablation and empirical studies confirm the effectiveness and flexibility of different components in AdamGNN. At last, through case studies, AdamGNN is shown to highlight variant-range node interactions in different graph datasets.

Our contributions can be summarised as follows. (1) We propose a novel framework, AdamGNN[1], that adaptively integrates multi-grained semantics into node representations and achieves mutual optimisation between node-wise and graph-wise tasks in one unified process. (2) An adaptive and efficient pooling operator is devised in AdamGNN to generate the multi-grained structure without introducing any hyper-parameters. (3) An attention-based flyback aggregation can provide model explainability on how different grains benefit the prediction tasks. (4) Extensive experiments on 14 real-world datasets demonstrate the promising performance of AdamGNN, along with providing insightful explanations with case studies.

2 Related Work

Graph Neural Networks. The existing GNN models can be generally categorised into spectral and spatial approaches. The spectral approach utilises the Fourier transformation to define convolution operation in the graph domain [3]. However, its incurred heavy computation cost hinders it from being applied to large-scale graphs. Later on, a series of spatial models drawn remarkable attention due to their effectiveness and efficiency in node-wise tasks [6], [7], [8], [10], [15], [22], [29], [31], [34], such as link prediction, node classification and node clustering. They mainly rely on the *flat* message-passing mechanism that defines convolution by iteratively aggregating messages from the neighbouring nodes. Recent studies have proved that the spatial approach is a special form of Laplacian smoothing and is limited to summarising each node's local information [5], [21]. Besides, they are either unable to capture global information or incapable of aggregating messages in a multi-grained manner to support graph classification tasks.

Graph Pooling. Pooling operation overcomes GNN's weakness in generating graph-level representation by recursively merge sets of nodes to form super nodes in the pooled graph. DiffPool [37] is a differentiable pooling operator, which learns a soft assign matrix that maps each node to a set of clusters, treated as super nodes. Since this assignment is rather *dense* that incurs high computation cost, it is not scalable for large graphs [4]. Following this direction, a Top-k selection based pooling layer (g-U-Net) is proposed to select important nodes from the original graph to build a pooled graph [9]. SagPool [16] and ASAP [26] further use attention and self-attention for cluster assignment, GXN [20] uses mutual information estimation for super node selection. They address the problem of sparsity in DiffPool, however, such a manual-defined hyper-parameter k is quite sensitive to the final performance [9], thus limits the adaptivity of these models on graphs of different sizes. In addition, as shown in Fig. 1, different k values will significantly affect the number of covered nodes in the graph, which means the important node features could get lost during the trivial pooling operation. Note that

1. Code and data are available at: https://github.com/zhiqiangzhongddu/AdamGNN

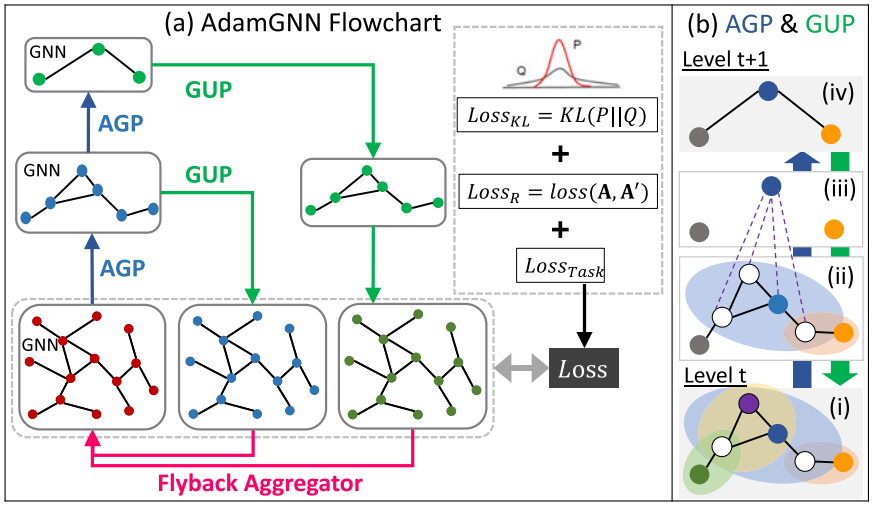

Fig. 2. (a) An illustration of AdamGNN with 3 levels. AGP: adaptive graph pooling, GUP: graph unpooling. (b) An example of performing adaptive graph pooling on a graph: (i) ego-network formation, (ii-iii) super node generation, (iv) maintaining super graph connectivity.

nodes covered by a super node refer to nodes involved in the super node's aggregation tree.

Recently, EigenPool [23] proposes a pooling operator based on graph Fourier which does not rely on the Top-k selection strategy, StructPool [39] designs strategies to involve both node and graph structures, and includes conditional random fields technique to ameliorate the cluster assignment. However, StructPool treats the graph assignment as a *dense* clustering problem, which gives rise to a high computation complexity as in DiffPool.

Discussion. Table 1 summarises the key advantages of the proposed AdamGNN and compares it with a number of state-of-the-art methods. Among the existing GNN models, g-U-Net, GXN and AdamGNN support both node- and graph-level tasks. However, *(i)* the Top-k selection strategy of g-U-Net and GXN introduces a new hyper-parameter and may lose important node features or graph structure; *(ii)* g-U-Net and GXN generate super graph only with k selected super nodes, which ruins multi-grained semantics of the original graph; *(iii)* g-U-Net does not support mini-batch because it needs to compute scores of all nodes in one big batch to select super nodes; *(iv)* GXN requires positive and negative node sets sampling which introduces additional operation and the random sampling strategy brings unmanageable uncertainty on model performance [36]. On the contrary, AdamGNN is a unified framework that adaptively integrates multi-grained semantics into node representations, and achieves a mutual optimisation between node- and graph-wise tasks. Besides, AdamGNN supports efficient mini-batch pooling and unpooling, and also provides model explanation via the multi-grained semantics. Therefore, we believe AdamGNN's framework is more general, effective and scalable than G-U-Net.

TABLE 1
Model Comparison from Various Aspects: Node-Wise Task (NT), Graph-Wise Task (GT), Pooling And/Or Unpooling (P/U), Adaptive Pooling (AP), Efficient Pooling (EP), Multi-Grained Explanation (ME)

	NT	GT	P/U	AP	EP	ME
GCN [15]	✓					
GraphSAGE [10]	✓					
GAT [31]	✓					
GIN [34]	✓	✓				
PNA [7]	✓	✓				
GCNII [6]	✓					
GRAND [8]	✓					
DiffPool [37]		✓	P			
g-U-Net [9]	✓	✓	P, U		✓	
SagPool [16]		✓	P		✓	
EigenPool [23]		✓	P	✓	✓	
GXN [20]	✓	✓	P, U		✓	
StructPool [39]		✓	P			
ASAP [26]		✓	P		✓	
AdamGNN	✓	✓	P, U	✓	✓	✓

3 Proposed Approach

3.1 Preliminaries

A graph with n nodes can be formally represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathcal{V} = \{v_1, \ldots, v_n\}$ is the node set, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges, and $\mathbf{X} \in \mathbb{R}^{n \times \pi}$ represents nodes' features (π is the dimensionality of node features). Besides, $\mathcal{V}$ and $\mathcal{E}$ can be summarised in adjacency matrix $\mathbf{A} \in \{0,1\}^{n \times n}$.

For node-wise tasks, the goal is to learn a mapping function $f_n : \mathcal{G} \rightarrow \mathbf{Z}$, where $\mathbf{Z} \in \mathbb{R}^d$, and each row $\mathbf{z}_i \in \mathbf{Z}$ corresponds to node v_i's representation. For graph-wise tasks, similarly it aims to learn a mapping $f_g : \mathcal{D} \rightarrow \mathbf{Z}$, where $\mathcal{D} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots\}$ is a set of graphs, each row $\mathbf{z}_i \in \mathbf{Z}$ corresponds to the graph $\mathcal{G}_i$'s representation. The mapping function's effectiveness f_n and f_g is evaluated by applying $\mathbf{Z}$ to different downstream tasks.

Primary Node Representation. We use Graph Convolution Network (GCN) [15] as an example primary GNN encoder to obtain the node representation:

$$\mathbf{H}^{(\ell+1)} = \mathrm{ReLU}(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{\frac{1}{2}}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)}), \quad (1)$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, $\hat{\mathbf{D}} = \sum_j \hat{\mathbf{A}}_{ij}$ and $\mathbf{W}^{(\ell)} \in \mathbb{R}^{n \times d}$ is a trainable weight matrix for layer ℓ. $\mathbf{H}^{(\ell)}$ is the generated node representation of layer ℓ which is defined as the primary node representations $\mathbf{H} = \mathbf{H}^{(\ell)}$. Node representations are generated

based on each target node's local neighbours, which are aggregated via learning based on the adjacency matrix $\mathbf{A}$. GCN cannot capture meso/macro-level knowledge, even with stacking multiple layers. Hence we term such generated node representations as primary node representations.

3.2 Adaptive Graph Pooling for Multi-Grained Structure Generation

The proposed model, AdamGNN, adaptively generates a multi-grained structure to realise the collective optimisation between the node and graph level tasks within one unified framework. The key idea is that apply an adaptive graph pooling operator to present the multi-grained semantics of $\mathcal{G}$ explicitly and improve the node representation generation with the derived meso/macro information. While AdamGNN is usually performed under multiple levels of granularity (T different grains), in this section, we present how level t's super graph is adaptively constructed based on graph of level $t-1$, i.e., $\mathcal{G}_{t-1} = (\mathcal{V}_{t-1}, \mathcal{E}_{t-1}, \mathbf{X}_{t-1})$.

Ego-Network Formation. We initially consider the graph pooling as an ego-network selection problem, i.e., each ego node can determine whether to aggregate its local neighbours to form a super node, resolving the *dense* issue of DIFFPOOL by avoiding to use a dense assignment matrix. As shown in Fig. 2b-(i), each ego-network c_λ contains the ego and its local neighbours $\mathcal{N}_i^\lambda$ within λ-hops., i.e., $\mathcal{N}_i^\lambda = \{v_j \mid d(v_i, v_j) \leq \lambda\}$, where $d(v_i, v_j)$ means the shortest-path length between v_i and v_j. Thus an ego-network can be formally presented as: $c_\lambda(v_i) = \{v_j \mid \forall v_j \in \mathcal{N}_i^\lambda\}$, and a set of ego-networks $\mathcal{C}_\lambda = \{c_\lambda(v_1), \ldots, c_\lambda(v_n)\}$ can be generated from $\mathcal{G}$. We will investigate the impact of the ego-network size in the ablation studies of Section 4.2

Super Node Determination. Given $\mathcal{G}$ with n nodes has n ego-networks, forming a super graph with all ego-networks will blur the useful multi-grained semantics and lead to a high computation cost. We select a fraction of ego-networks from $\mathcal{C}_\lambda$ to organise the multi-grained semantics of $\mathcal{G}$. We make the selection based on a closeness score ϕ_i that evaluates the representativeness of the ego v_i to its local neighbours $v_j \in c_\lambda(v_i)$. We first create a function to calculate the closeness score ϕ_{ij} between v_i and v_j:

$$\begin{aligned}\phi_{ij} &= f_\phi^{Metric}(v_i, v_j) \times f_\phi^{NN}(v_i, v_j) \\ &= \text{Sigmoid}(\mathbf{H}^{\mathrm{T}}[j] \cdot \mathbf{H}[i]) \\ &\quad \times \text{Softmax}(\overrightarrow{\mathbf{a}}^{\mathrm{T}} \sigma(\mathbf{WH}[j] \parallel \mathbf{WH}[i])),\end{aligned} \tag{2}$$

where $\parallel$ is the concatenation operator, $\overrightarrow{\mathbf{a}} \in \mathbb{R}^{2\pi}$ is the weight vector and σ is an activation function (LeakReLU) to avoid the vanishing gradient problem [12] during the model training process. Consequently, nodes with similar features and structure information to the ego will contribute to higher closeness scores. $f_\phi^{Metric}(v_i, v_j) = \text{Sigmoid}(\mathbf{H}^{\mathrm{T}}[j] \cdot \mathbf{H}[i])$ is a metric function on dot product similarity, Sigmoid ensures the proper range of output values. However, we observed in our practical experiments that, due to the local-smoothing speciality of the GNN encoder [21], ego nodes often have high $f_\phi^{Metric}(v_i, v_j)$ to many local neighbours, leading to egos' closeness scores with less differences. Thus, we further add another component $f_\phi^{NN}(v_i, v_j) = \frac{\exp(\overrightarrow{\mathbf{a}}^{\mathrm{T}} \sigma(\mathbf{WH}[j] \| \mathbf{WH}[i]))}{\sum_{v_r \in \mathcal{N}_j^\lambda} \exp(\overrightarrow{\mathbf{a}}^{\mathrm{T}} \sigma(\mathbf{WH}[j] \| \mathbf{WH}[r]))}$ to supercharge ϕ_{ij} to magnify the closeness difference between nodes. Compared with f_ϕ^{Metric}, the neural function f_ϕ^{NN} is enhanced with trainable parameters $\mathbf{W}$ to be able to capture the complex relationship pattern between ego v_i' and node v_j's representations. The output of f_ϕ^{NN} lies in $(0, 1)$ as a valid probability for ego-network selection. In the end, we produce the representativeness of the ego v_i as:

$$\phi_i = \frac{1}{|\mathcal{N}_i^\lambda|} \sum_{v_j \in \mathcal{N}_i^\lambda} \phi_{ij}, \tag{3}$$

where $|\mathcal{N}_i^\lambda|$ indicates the number of nodes in $\mathcal{N}_i^\lambda$.

After obtaining ego-networks' closeness scores, we propose an adaptive approach to select a fraction of ego-networks to form super nodes without pre-defined hyper-parameters (cf. the Top-k selection strategy [9]). Our key intuition is that a high diameter ego-network could be composed of multiple low diameter ego-networks. Therefore, we intend first to find these low diameter ego-networks, then recursively aggregate them to form a super node that contains these ego-networks. Specifically, we form ego-networks by selecting a fraction of egos $\hat{N}_p$ as: $\hat{N}_p = \{v_i \mid \phi_i > \phi_j, \forall v_j \in \mathcal{N}_i^1\}$, where $\mathcal{N}_i^1$ means the neighbour nodes of node v_i within the first hop. Note that each node may belong to various ego-networks since they may play different roles in different groups. Therefore, we allow overlapping between different selected ego-networks and utilise $\mathcal{N}_i^1$ instead of $\mathcal{N}_i^\lambda$. If we adopt $\mathcal{N}_i^\lambda$ here, the selected ego node v_i cannot be involved in other ego-networks anymore. Following this, we can select a fraction of ego-networks to form super nodes at granularity level t.

Proposition 1. *Let $\mathcal{G}$ be a connected graph with n nodes, and a total number of n ego-networks can be formed from the graph $\mathcal{G}$, i.e., $\mathcal{C}_\lambda = \{c_\lambda(v_1), c_\lambda(v_2), \ldots, c_\lambda(v_n)\}$. Each ego-network $c_\lambda(v_i)$ will be assigned with a closeness score ϕ_i. Then, there exist at least one ego-network $c_\lambda(v_i)$ that satisfies $\phi_i > \phi_j$, $\forall v_j \in \mathcal{N}_i^1$.*

Proof of Proposition 1. For $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ with n nodes. n ego-networks can be generated by following the procedures, $c_\lambda(v_i) = \{j \mid \forall j \in \mathcal{N}_i^\lambda\}$, and each ego-network will be given a closeness score ϕ_i as Equation 3. We assume that these cluster closeness scores are not all the same, thus there exists at least one maximum ϕ_{max}. Hence, the clusters with closeness score ϕ_{max} satisfy the requirements of ego-network selection requirement that:

$$\phi_{max} > \phi_j, \quad \forall v_j \in \mathcal{N}_{max}^1, \tag{4}$$

where $\mathcal{N}_{max}^1 = \{v_j \mid if\ d(v_i, v_j) = 1\}$. So, for any connected $\mathcal{G}$ with n nodes, there exists at least one cluster that satisfies the requirements of our ego-network selection approach. □

Proposition 1 ensures that our super node determination method can find at least one ego-network to generate a super graph for any graph. It guarantees the generality of our strategy.

Meanwhile, we would also retain nodes that do not belong to any selected ego-networks, denoted as $\hat{N}_r$, to maintain the graph structure: $\hat{N}_r = \{v_j \mid v_j \notin c_\lambda(v_i), \forall v_i \in \hat{N}_p\}$. In this way, a super node formation matrix $\mathbf{S}_t \in$

$\mathbb{R}^{n\times(|\hat{N}_p|+|\hat{N}_r|)}$ can be formed, where $(|\hat{N}_p|+|\hat{N}_r|)$ is number of nodes of the generated super graph, rows of $\mathbf{S}_t$ corresponds to the n nodes of $\mathcal{G}_{t-1}$, and columns of $\mathbf{S}_t$ corresponds to the selected ego-networks ($\hat{N}_p$) plus the remaining nodes ($\hat{N}_r$). We have $\mathbf{S}_t[i,j]=\phi_{ij}$ if node v_j belongs to the selected ego-network $c_\lambda(v_i)$ and $\mathbf{S}_t[i,j]=1$ if node v_j is a remaining node corresponds to node v_i in the super graph; otherwise $\mathbf{S}_t[i,j]=0$. The weighted super node formation matrix $\mathbf{S}_t$ can better maintain the relation between different super nodes in the pooled graph.

Maintaining Super Graph Connectivity. After selecting the ego-networks and retaining nodes in level $t-1$, as shown in Fig. 2b-(iii-iv), we construct the new adjacent matrix $\mathbf{A}_t$ for the super graph using $\hat{\mathbf{A}}_{t-1}$ and $\mathbf{S}_t$ as follows: $\mathbf{A}_t=\mathbf{S}_t^{\mathrm{T}}\hat{\mathbf{A}}_{t-1}\mathbf{S}_t$. This formula makes any two super nodes connected if they share any common nodes or any two nodes are already neighbours in $\mathcal{G}_{t-1}$. In addition, $\mathbf{A}_t$ will retain the edge weights passed by $\mathbf{S}_{t-1}$ that involves the relation weights between super nodes. Eventually, we obtain a generated super graph $\mathcal{G}_t$ at granularity level t.

Super Node Feature Initialisation. All nodes in the super graph $\mathcal{G}_t$ need initial feature vectors to support the graph convolution operation. Recall that we have the closeness score as calculated in Eq. 2, between node v_j to the ego v_i. However, this is not equivalent to the contribution of node v_j's feature to the super node feature, since we need to compare the relationship strength between ego v_i and v_j with the relation between other $v_r\in c_\lambda(v_i)$. Therefore, we further propose a super node feature initialisation method through a self-attention mechanism [31]. Specifically, it can be described as:

$$\mathbf{X}_t[i]=\mathbf{H}_{t-1}[i]+\sum_{v_j\in c_\lambda(v_i)\setminus v_i}\alpha_{ij}\mathbf{H}_{t-1}[j], \tag{5}$$

where $\mathbf{H}_{t-1}$ is the generated node representation by the $(t-1)$-th primary GNN layer, i.e., at level $t-1$ with a similar method as Equation 1, α_{ij} describes the importance of node v_j to the initial feature of $c_\lambda(v_i)$ at level t. And α_{ij} can be learned as follows: $\alpha_{ij}=\frac{\exp(\overrightarrow{\mathbf{a}_1}^{\mathrm{T}}\,\sigma(\mathbf{W}(\phi_{ij}\mathbf{H}_{t-1}[j])\|\mathbf{H}_{t-1}[i]))}{\sum_{v_r\in c_\lambda(v_i)}\exp(\overrightarrow{\mathbf{a}_1}^{\mathrm{T}}\,\sigma(\mathbf{W}(\phi_{ir}\mathbf{H}_{t-1}[r])\|\mathbf{H}_{t-1}[i]))}\Big)$ where $\overrightarrow{\mathbf{a}_1}\in\mathbb{R}^{2\pi}$ is the weight vector. For the remaining nodes $\hat{N}_r$ that do not belong to any super nodes, we keep their representations of $\mathbf{H}_{t-1}$ as initial node features.

3.3 Graph Unpooling

Different from existing graph pooling models [4], [16], [26], [37], [39] which only coarsen graphs to generate graph representations, we aim to mutually utilise node-wise and graph-wise tasks to better encode multi-grained semantics into both node and graph representations under a unified framework. We design a mechanism to allow the learned multi-grained semantics to enrich the node representations of the original graph $\mathcal{G}$ as shown in Fig. 2a. Vice versa, the updated node representation can further ameliorate the graph representation in the next training iteration. A reasonable unpooling operation that passes macro-level information to original nodes has not been well studied in the literature. For instance, Gao et al. [9] directly relocate the super node back into the original graph and utilise other GNN layers to spread its message to other nodes. However, these additional aggregation operations cannot allow each node to receive meaningful information since some nodes may be distant from super nodes, in such case these operations can exacerbate local-smoothing [21].

We implement the unpooling process by devising a *top-down* message-passing mechanism, which endows GNN models with meso/macro level knowledge. Specifically, since $\mathbf{S}_t$ records how nodes of $\mathcal{G}_{t-1}$ form the super nodes of $\mathcal{G}_t$, so we utilise $\mathbf{S}_t$ to restore the generated ego-network representation at level t to that at level $t-1$ until we arrive at the original graph $\mathcal{G}$, i.e., $t\rightarrow 0$, as follows:

$$\hat{\mathbf{H}}_t=(\mathbf{H}_t^{\mathrm{T}}\mathbf{S}_t^{\mathrm{T}}\mathbf{S}_{t-1}^{\mathrm{T}}\ldots\mathbf{S}_1^{\mathrm{T}})^{\mathrm{T}}, \tag{6}$$

where $\hat{\mathbf{H}}_t\in\mathbb{R}^{n\times d}$. At the end of each iteration, nodes in the original graph $\mathcal{G}$ will receive high-level semantic messages from the different levels, i.e., $\{\hat{\mathbf{H}}_1,\ldots,\hat{\mathbf{H}}_T\}$. As illustrated in Fig. 2b-(iv-i), the graph unpooling process can be treated as an inverse process of adaptive graph pooling process.

3.4 Flyback Aggregation

Since the super graphs at different granularity levels present multi-grained semantics and how each node utilises the received semantic information with its *flat* representation is a challenging question. And nodes of the same graph may need different granularity levels' information. Therefore, we propose a novel attention mechanism to integrate the derived representations at different levels, given by:

$$\mathbf{Z}=\mathbf{H}+\sum_{\forall t}\boldsymbol{\beta}_t\hat{\mathbf{H}}_t, \tag{7}$$

where the attention score $\boldsymbol{\beta}_t$ estimates the importance of the message from level t, given by: $\boldsymbol{\beta}_t[i]=\frac{\exp(\overrightarrow{\mathbf{a}_2}^{\mathrm{T}}\,\sigma(\mathbf{W}\hat{\mathbf{H}}_t[i])\|\mathbf{H}[i]))}{\sum_{j\in T}\exp(\overrightarrow{\mathbf{a}_2}^{\mathrm{T}}\,\sigma(\mathbf{W}\hat{\mathbf{H}}_j[i])\|\mathbf{H}[i]))}$ where $\overrightarrow{\mathbf{a}_2}\in\mathbb{R}^{2\pi}$ is the weight vector. We term this process as the *flyback* aggregation, which considers the attention scores of different levels and allows each node to decide whether/how to utilise semantic information from different granularity levels. We will verify the effectiveness of *flyback* aggregation in the ablation study of Section 4.2 and discuss the explainability in Section 4.3.

3.5 Training Strategy

Till now, there are still two challenges when training the model. The first is how to highlight the difference among nodes' representations from different ego-networks. Nodes belonging to neighbouring ego-networks receive closely related messages from super nodes, since their super nodes are connected in the super graph, and local smoothing makes their representation vectors similar. Representations of proximal nodes could be further closer to each other in the representation latent space. To address this problem and enhance the discrimination capability between ego-networks, we exploit a self-optimisation strategy [33], which makes nodes in different ego-networks distinguishable. Specifically, we use the Student's t-distribution (Q) as a kernel to measure the similarity between representation vectors of v_j and ego v_i: $q_{ij}=\frac{(1+\|\mathbf{Z}[j]-\mathbf{Z}[i]\|^2/\mu)^{-\frac{\mu+1}{2}}}{\sum_{i'}(1+\|\mathbf{Z}[j]-\mathbf{Z}[i']\|^2/\mu)^{-\frac{\mu+1}{2}}}$, where $v_j\in$

$c_\lambda(v_i)$, $v_{i'}$ are other ego nodes, μ are the degrees of freedom of Student's t-distribution. Following this, q_{ij} can be integrated the probability of assigning node v_j to ego v_i. In this paper, we set $\mu = 1$ the same as [33]. After, we propose to learn better node representations by matching Q to the auxiliary target distribution (P), and we choose the proposition of [33] which first raises q_i to the second power and then normalises by frequency per ego-network: $p_{ij} = \frac{q_{ij}^2/g_i}{\sum_{i'}(q_{ij}^2/g_{i'})}$, where $g_i = \sum_j q_{ij}$. Therefore, apart from the task-related loss function $\mathcal{L}_{task}$, we further define a *KL* divergence loss as:

$$\mathcal{L}_{KL} = KL(P \parallel Q) = \sum_{v_j}\sum_{v_i} p_{ij}\log\frac{p_{ij}}{q_{ij}}, \tag{8}$$

The second challenge is to avoid the over-smoothing problem that nodes of a graph tend to have indistinguishable representations. GNN is proved as a special form of Laplacian smoothing [5], [21] that naturally assimilates the nearby nodes' representations. AdamGNN will further exacerbate this problem because it distributes semantic information from one super node representation to all nodes of the ego network. Therefore, we introduce the reconstruction loss, which can alleviate the over-smoothing issue and drive the node representations to retain the structure information of $\mathcal{G}$ by differentiating non-connected nodes' representations. Specifically, the reconstruction loss is defined as:

$$\mathcal{L}_R = -\frac{1}{n}\sum(\mathbf{A}_{ij}\cdot\log(\mathbf{A}'_{ij}) + (1-\mathbf{A}_{ij})\cdot\log(1-\mathbf{A}'_{ij})), \tag{9}$$

where $\mathbf{A}' = \mathrm{Sigmoid}(\mathbf{Z}^{\mathrm{T}}\mathbf{Z})$. Therefore, the overall loss function consists of the training task $\mathcal{L}_{Task}$, the self-optimising task $\mathcal{L}_{KL}$, and the reconstruction task $\mathcal{L}_R$, given by:

$$\mathcal{L} = \mathcal{L}_{Task} + \gamma\mathcal{L}_{KL} + \delta\mathcal{L}_R, \tag{10}$$

where $\mathcal{L}_{Task}$ is a flexible task-specific loss function, and γ and δ are two hyper-parameters that we will discuss in Section 4.1. Note that for link prediction task we have $\mathcal{L} = \mathcal{L}_R + \gamma\mathcal{L}_{KL}$, since $\mathcal{L}_{Task}$ equals to $\mathcal{L}_R$. Moreover, we will demonstrate the effectiveness of each component of loss function $\mathcal{L}$ in the ablation study of Section 4.2.

3.6 Algorithm

We have presented the idea of AdamGNN and the design details of each component in Section 3. Given a graph $\mathcal{G}$, we first apply a primary GNN encoder to generate the primary node embedding (line 1). Then we construct a multi-grained structure with t-th level (line 3-13) with the proposed adaptive graph pooling operator. Meanwhile, we also propose a method to define the initial features of pooled super nodes (line 14-19). The graph connectivity of the pooled graph is maintained by line 20. We apply GNN encoder on the pooled graph to summarise the relationships between super nodes (line 21) to learn macro grained semantics of t-th granularity level. The learned multi-grained semantics will be further distributed to the original graph following an unpooling operator (line 22). Last, the *flyback* aggregator generates the meso/macro level knowledge from different levels as the node representations of $\mathcal{G}$ (line 24), and additional READOUT operators [7] produce the node representations as to the graph representation (line 25).

Algorithm 1. *Adaptive Multi-Grained GNNs*

Input: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$.
Output: node representations $\mathbf{Z}$, graph representations $\mathbf{Z}_g$

1: $\mathbf{H} = \mathrm{ReLU}(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{\frac{1}{2}}\mathbf{X}\mathbf{W})$;
2: **for** $t \leftarrow \{1, 2, \ldots, T\}$ **do**
3: **for** $v_i \leftarrow \{v_1, v_2, \ldots, v_n\}$ **do**
4: **for** $v_j \in \mathcal{N}_i^\lambda$ **do**
5: $\phi_{ij} = f_\phi^{Metric}(v_i, v_j) \times f_\phi^{NN}(v_i, v_j)$;
6: **end**
7: $\phi_i = \frac{1}{|\mathcal{N}_i^\lambda|}\sum_{v_j\in\mathcal{N}_i^\lambda}\phi_{ij}$;
8: **end**
9: **for** $v_i \leftarrow \{v_1, v_1, \ldots, v_n\}$ **do**
10: $\hat{N}_p = \{v_i \,|\, \phi_i > \phi_j, \textbf{for}all v_j \in \mathcal{N}_i^1\}$;
11: **end**
12: $\hat{N}_r = \{v_j \,|\, v_j \notin c_\lambda(v_i), \textbf{for}all v_i \in \hat{N}_c\}$;
13: Generate the super-node formation matrix: $\mathbf{S}_t$;
14: **for** $v_i \in \hat{N}_r$ **do**
15: $\mathbf{X}_t[i] = \mathbf{H}_{t-1}[i]$;
16: **end**
17: **for** $v_i \in \hat{N}_p$ **do**
18: $\mathbf{X}_t[i] = \mathbf{H}_{t-1}[i] + \sum_{v_j\in c_\lambda(v_i)\setminus v_i}\alpha_{ij}\mathbf{H}_{t-1}[j]$;
19: **end**
20: $\mathbf{A}_t = \mathbf{S}_t^{\mathrm{T}}\hat{\mathbf{A}}_{t-1}\mathbf{S}_t$;
21: $\mathbf{H}_t = \mathrm{ReLU}(\hat{\mathbf{D}}_t^{-\frac{1}{2}}\hat{\mathbf{A}}_t\hat{\mathbf{D}}_t^{\frac{1}{2}}\mathbf{X}_t\mathbf{W}_t)$;
22: $\hat{\mathbf{H}}_t = (\mathbf{H}_t^T\mathbf{S}_t^{\mathrm{T}}\mathbf{S}_{t-1}^{\mathrm{T}}\ldots\mathbf{S}_1^{\mathrm{T}})^{\mathrm{T}}$;
23: **end**
24: $\mathbf{Z} = \mathbf{H} + \sum_t^T \boldsymbol{\beta}_t\hat{\mathbf{H}}_t$;
25: $\mathbf{Z}_g = \mathrm{READOUT}(\{\mathbf{Z}, \hat{\mathbf{H}}_1, \ldots, \hat{\mathbf{H}}_T\}) \in \mathbb{R}^d$;

Model Scalability. According to the design for AdamGNN framework, we can find that the primary node representation learning module of each level and the adaptive graph pooling and unpooling operators are categorised as a local network algorithm [28], which only involves local exploration of the network structure. Therefore, our design enables AdamGNN to scale to representation learning on large-scale networks and to be friendly to distributed computing settings [25]. We present instances that utilise a multi-GPU computing framework to accelerate the training process of AdamGNN in Section 4.3.

4 Experiments

4.1 Experimental Setup

We evaluate our proposed model, AdamGNN, on 14 benchmark datasets, and compare with 16 competing methods over both node- and graph-wise tasks, including node classification, link prediction and graph classification.

Datasets. To validate the effectiveness of our model on real-world applications, we adopt datasets come from different domains with different topics and relations. We use 8 datasets for node-wise tasks (data statistics are summarised in Table 2). Ogbn-arxiv [13], ACM [2], Cora [15], Citeseer [15] and Pubmed [15] are paper citation graph datasets. DBLP [2] is an author graph dataset from the DBLP dataset. Emails [17] is an email communication graph dataset. Wiki [35] is a webpage graph dataset.

For the graph classification task, we adopt 6 bioinformatics datasets [39] (data statistics are summarised in Table 3). D&D and PROTEINS are dataset containing proteins as

TABLE 2
Data Statistics for Node-Wise Tasks and the Split for the Semi-Supervised Node Classification Task

Dataset	#Nodes	#Edges	#Features	#Classes	#Train-#Val-#Test
ACM	3,025	13,128	1,870	3	60-500-1,000
Citeseer	3,327	4,552	3,703	6	120-500-1,000
Cora	2,708	5,278	1,433	7	140-500-1,000
DBLP	4,057	3,528	334	4	80-500-1,000
Emails	799	10,182	N.A.	18	180-309-310
Pubmed	19,717	88,648	500	3	60-500-1,000
Wiki	2,405	1,2178	4973	17	481-962-962
Ogbn-arxiv	169,343	1,166,243	128	40	N.A.

N.A. means a dataset does not contain node attributes or does not support semi-supervised settings.

TABLE 3
Data Statistics for Graph Classification

Dataset	#Graphs	#Nodes (avg)	#Edges (avg)	#Features	#Classes
NCI1	4,110	29.87	32.3	37	2
NCI109	4,127	29.68	32.13	38	2
D&D	1,178	284.32	715.66	89	2
MUTAG	188	17.93	19.79	7	2
Mutagenicity	4,337	30.32	30.77	14	2
PROTEINS	1,113	39.06	72.82	32	2

graphs. NCI1 and NCI109 involve anticancer activity graphs. The MUTAG and Mutagenicity consist of chemical compounds divided into two classes according to their mutagenic effect on a bacterium. Note that all datasets can be downloaded with our published code automatically.

Competing Methods. For node-wise tasks, we adopt 10 competing methods that include 7 GNN models with flat message-passing mechanism, and 2 state-of-the-art methods that contain a hierarchical structure: MLP [11], GCN [15], GraphSAGE [10], GAT [31], GIN [34], PNA [7], GCNII [6], GRAND [8], GXN [20] and g-U-Net [9]. For the graph classification task, except for GIN, PNA, GXN and g-U-Net which support graph-wise tasks, we adopt extra 7 competing methods that involve the state-of-the-art models: 3WL-GNN [24], SortPool [42], DiffPool [37], SagPool [16], EigenPool [23], StructPool [39] and ASAP [26]. Note that we already carefully discussed these competing methods in Section 2; therefore, we do not repeat the method description to save page space. The competing model implementations can be found in our published project on Github.

Evaluation Settings. For the *node-wise* tasks, we follow the *supervised node classification* (Sup-NC) settings of PNA [7], i.e., using two sets of 10% labelled nodes as validation and test sets, with the remaining 80% labelled nodes used as the training set. Meanwhile, we follow the *semi-supervised node classification* (Semi-NC) settings of GCN [15], and the data split is shown in Table 2. That said, for Cora, Citeseer and Pubmed, we use the fixed splits and for other datasets, we randomly assign 20 labelled nodes for each class for training, and 500 and 1000 nodes for validation and testing, respectively. Note that since the Email does not have a sufficient number of nodes for classic Semi-NC setting, we choose 10 labelled nodes for each class for training, and the rest data is evenly separated as validation and test sets. Wiki is imbalanced, where some classes only have very few labelled nodes, e.g., class 12 has 9 labelled nodes and class 4 has 10 labelled nodes, which cannot support Semi-NC settings. Therefore, we follow Sup-NC settings to split Wiki for the Semi-NC experimental parts but use only 20% labelled nodes for training and the remaining nodes for validation and testing, respectively. Ogbn-arxiv follows the fixed split of OGB leaderboard [13]. For the *Link Prediction* (LP) task, we follow the settings of [38], i.e., using two sets of 10% existing edges as validation and test sets, with the remaining 80% edges used as the training set. Note that, an equal number of non-existent links are randomly sampled and used for every set. We present the average performance of 10 times experiments with random seeds. The AUC score evaluates link prediction, and node classification tasks are evaluated by accuracy. We conduct the experiments with random parameter initialisation with 10 random seeds and report the average performance.

For the *graph-wise* task, i.e., *graph classification* (GC) task, we perform all experiments following the pooling pipeline of SagPool [16]. 80% of the graphs are randomly selected as training, and the rest two 10% graphs are used for validation and testing, respectively. We conduct the experiments using 10-fold cross-validation and report the average classification accuracy on 10 random seeds.

Model Configuration. For all methods, we set the embedding dimension $d = 64$ and utilise the same learning rate $= 0.01$, Adam optimiser, number of training epochs $= 1000$ with early stop (100). In terms of the neural network layers, we report the one with better performance of GCNII with better performance among $\{8, 16, 32, 64, 128\}$; for other models, we report the one with better performance between $2-4$; For all models with hierarchical structure (including AdamGNN), we use GCN as the GNN encoder for fair comparision. In terms of the number of levels that is required by hierarchical models, we present the one with better performance, between $2-5$. On other hyper-parameter settings of competing methods, we employ the default values of each competing method as shown in the paper's official implementation. Particularly, for AdamGNN, by tuning the hyper-parameters based on the validation set, we have $\gamma = 0.1$ and $\delta = 0.01$ for Eq. 10 for the experiments to let loss values lie in a reasonable range, i.e., $(0, 10)$. We employ Pytorch and PyTorch Geometric to implement all models. Experiments were conducted with GPU (NVIDIA Tesla V100) machines.

4.2 Experimental Evaluation and Ablation Study

Performance on Node-Wise Tasks. We compare AdamGNN with 7 GNN models and one pooling-based model, i.e., g-U-Net, since other pooling approaches do not provide an unpooling operator and thus cannot support node-wise tasks. Results on node classification (with supervised and semi-supervised settings) are summarised in Table 4. They show that AdamGNN can outperform most competing methods with up to 10.47% and 5.39% improvements on semi-supervised and supervised settings, respectively. AdamGNN brings the most significant improvement in Wiki data with semi-supervised settings, and the competing method that only adopts node features, i.e., MLP, achieve terrible accuracy, 17.46%. We argue that because the node features and node labels are weakly correlated in this

TABLE 4
Results in Accuracy (%) for Supervised and Semi-Supervised Node Classification on Eight Datasets

Models	ACM		Citeseer		Cora		Emails		DBLP		Pubmed		Wiki		Ogbn-arxiv
	Sup	Semi	Sup	Semi	Sup	Semi	Sup	Semi	Sup	Semi	Sup	Semi	Sup	Semi	Sup
MLP [11]	87.08	76.14	70.87	63.27	76.12	57.82	N.A.	N.A.	79.17	66.55	83.41	71.83	20.42	17.46	55.50‡
GCN [15]	92.25	86.14	76.13	71.13	88.90	81.50	85.03	77.32	82.68	72.22	86.04	79.02	57.36	46.19	71.74‡
GraphSAGE [10]	92.48	87.18	76.75	71.19	**88.92**	81.17	85.80	78.19	83.20	72.20	86.01	79.14	57.24	49.21	71.49‡
GAT [31]	91.69	87.52	**76.96**	72.43	88.33	83.00	84.67	77.35	84.04	73.24	86.21	79.01	58.07	50.27	72.06
GIN [34]	90.66	85.98	76.39	70.27	87.74	82.19	87.18	79.23	82.54	73.42	87.10	79.11	66.29	49.88	71.76
PNA [7]	**93.97**	**89.81**	72.67	67.81	88.78	79.54	81.25	73.23	**86.21**	72.40	87.63	76.61	21.67	19.58	72.37
GCNII [6]	93.05	87.80	76.37	**73.37**	88.07	**85.43**	82.51	72.90	84.48	72.82	84.48	80.01	60.24	**56.18**	**72.74**‡
GRAND [8]	93.05	86.90	76.88	**75.25**	87.82	**85.41**	52.53	71.94	85.47	70.46	86.63	**82.06**	59.58	27.94	70.97
g-U-Net [9]	93.42	89.34	75.59	72.97	87.68	84.40	**89.16**	**80.48**	85.27	**73.86**	**87.67**	79.06	**71.33**	54.18	71.78
GXN [20]	92.95	87.70	75.93	71.07	86.90	83.10	88.68	77.91	84.66	71.27	86.02	78.01	68.90	51.89	70.02
AdamGNN	**94.37**	**90.72**	**78.92**	73.03	**90.92**	84.66	**91.88**	**83.23**	**88.36**	**74.60**	**89.81**	**80.48**	**73.37**	**62.06**	**72.65**

‡ *Indicates the results from OGB leaderboard [13]. The bold numbers represent the top-2 results.*

TABLE 5
Results in AUC for Link Prediction on Seven Datasets

Models	ACM	Citeseer	Cora	Emails	DBLP	Pubmed	Wiki
GCN [15]	0.975	0.887	0.918	0.930	0.904	0.941	0.523
GraphSAGE [10]	0.972	0.884	0.908	0.923	0.889	0.905	0.577
GAT [31]	0.968	0.910	0.912	0.930	0.889	0.947	0.594
GIN [34]	0.787	0.808	0.878	0.859	0.820	0.927	0.501
PNA [7]	0.978	0.974	0.731	0.932	0.908	0.896	0.538
GCNII [6]	0.968	0.969	0.871	0.926	0.890	0.933	0.709
g-U-Net [9]	0.890	0.918	0.932	0.936	0.934	0.962	0.734
GXN [20]	0.862	0.894	0.909	0.915	0.911	0.934	0.516
AdamGNN	**0.988**	**0.975**	**0.948**	**0.957**	**0.965**	**0.969**	**0.920**

TABLE 6
Results in Accuracy (%) for Graph Classification on Six Datasets

Models	NCI1	NCI109	D&D	MUTAG	Mutagenicity	PROTEINS
GIN [34]	76.17±1.12	77.31±1.42	78.05±1.89	75.11±2.64	77.24±2.26	75.37±1.62
3WL-GNN [24]	79.38±1.73	78.34±1.90	78.32±2.44	78.34±3.39	81.52±2.23	77.92±2.09
PNA [7]	78.96±1.01	79.06±1.15	75.47±2.52	**81.91**±2.59	81.72±1.46	77.72±2.25
SortPool [42]	72.25±1.33	73.21±2.21	73.31±2.43	71.47±2.31	74.65±3.35	70.49±2.37
DiffPool [37]	76.47±0.96	76.17±1.11	76.16±1.69	73.61±3.94	76.30±0.37	71.90±2.75
g-U-Net [9]	77.56±1.92	77.02±2.30	73.98±2.63	76.60±5.03	78.64±3.11	72.94±3.68
SagPool [16]	75.76±1.29	73.67±2.32	76.21±1.56	75.27±1.92	77.09±1.17	75.27±0.57
EigenPool [23]	77.54±1.82	77.20±1.81	78.25±1.78	76.21±2.74	78.60±1.24	75.19±1.95
GXN [20]	74.18±2.20	71.39±1.90	74.92±2.04	74.10±4.36	75.53±1.83	72.26±2.24
StructPool [39]	77.61±1.08	78.39±1.23	80.10±1.77	77.13±3.93	80.94±1.67	78.84±1.70
ASAP [26]	78.21±1.75	78.16±1.62	79.50±1.80	80.17±1.77	81.52±1.24	**78.92±1.45**
AdamGNN	**79.77±1.29**	**79.36±1.03**	**81.51±1.56**	80.11±2.58	**82.04±1.73**	77.04±0.78

dataset, multi-grained semantics provided by AdamGNN help to ameliorate the performance.

Link prediction results in Table 5 show that AdamGNN can significantly outperform the 7 competing methods by up to 25.3% improvement in terms of AUC. It indicates the versatility of AdamGNN on different node-wise tasks and exhibits the usefulness of modelling multi-grained semantics into node representations. Similar to node classification task, AdamGNN again brings the most significant AUC improvement on the Wiki dataset, i.e., achieving 29.76% improvement compared with the *flat* GNN models.

Performance on Graph-Wise Task. Experimental results are summarised in Table 6. It is apparent that our AdamGNN achieves the best performance on 4 of the 6 datasets, and consistently outperforms most of competing pooling-based techniques by 1.76% improvement. For the datasets MUTAG and PROTEINS, our results are still competitive since PNA and ASAP only slightly outperform AdamGNN. Nevertheless, AdamGNN is still better than other baselines. This is because our model involves adaptive pooling and unpooling operators to update node- and graph-wise information interactively, and further enhance the representations of nodes and graphs during the training process.

Ablation Study of Different Loss Functions. The loss function of our AdamGNN consists of three parts, i.e., $\mathcal{L}_{Task}$, $\mathcal{L}_R$ and $\mathcal{L}_{KL}$. We examine how each part contributes to the

TABLE 7
Comparison of AdamGNN with Different Loss Functions on Three Tasks

	DBLP (LP)	Citeseer (NC)	Mutagenicity (GC)
AdamGNN w/ $\mathcal{L}_{Task}$	0.956	76.63	79.04
AdamGNN w/ $\mathcal{L}_{Task}$+$\mathcal{L}_{KL}$	-	77.17	78.94
AdamGNN w/ $\mathcal{L}_{Task}$+$\mathcal{L}_{R}$	-	77.64	80.65
AdamGNN (Full model)	**0.965**	**78.92**	**82.04**

NC task follows the supervised settings.

TABLE 8
Comparison of AdamGNN With and Without *flyback* Aggregation in Terms of Graph Classification Accuracy on NCI1, NCI109 and Mutagenicity Datasets

AdamGNN	NCI1	NCI109	Mutagenicity
No *flyback* aggregation	75.54	77.49	79.89
Full model	**79.77**	**79.36**	**82.04**

TABLE 9
Comparison of AdamGNN with Different Number of Granularity Levels in Terms of Different Tasks

# Levels (T)	DBLP LP	Wiki LP	ACM NC	Citeseer NC	Emails NC	Mutagenicity GC
1	0.951	0.912	92.60	77.68	86.83	78.16
2	0.958	0.913	93.38	74.67	**91.88**	**82.04**
3	0.959	0.917	**94.37**	76.15	90.61	81.58
4	**0.965**	**0.920**	90.84	**78.92**	-	81.01

NC task follows the supervised settings.

performance. Table 7 provides the results. For the link prediction task, we have $\mathcal{L} = \mathcal{L}_R + \gamma\mathcal{L}_{KL}$, since $\mathcal{L}_{Task}$ equals to $\mathcal{L}_R$. Thus, two comparison experiments are missing in link prediction. From the results, we can see that $\mathcal{L}_R$ can significantly improve the performance over three tasks. This is because it can eliminate the over-smoothing problem caused by the received messages from different granularity levels. Meanwhile, $\mathcal{L}_{KL}$ can slightly improve the results of node-wise tasks as well.

Ablation Study of the Flyback Aggregation. Experimental results of node-wise tasks confirm that capturing multi-grained semantics in AdamGNN can help to learn more meaningful node representations. Here, we further study whether *flyback* aggregator can improve graph representations. Specifically, we aim to see how the *flyback* aggregator contributes to graph classification performance by removing and keeping it. The results are summarised in Table 8. It is clear that the node representations enhanced by the *flyback* aggregation can indeed improve the graph representation in the classification task.

Ablation Study of Number of Granularity Levels. As it has been proved that the existing GNN models will have worse performance when the neural network goes deeper [21], here we examine how AdamGNN can be benefited from more granularity levels. By varying the number of granularity levels, we report the performance of AdamGNN on link prediction, supervised node classification and graph classification, as summarised in Table 9. We can observe that

TABLE 10
Comparison of AdamGNN with Different Primary GNN Encoder, Follow the Semi-Supervised Node Classification Settings

Models	Cora	Citeseer	Pubmed
GCN	81.50	71.13	79.02
AdamGNN w/ GCN	**84.66**	**73.03**	**80.48**
PNA	79.54	67.81	76.61
AdamGNN w/ PNA	**81.21**	**70.90**	**78.73**
GCNII	85.43	73.37	80.01
AdamGNN w/ GCNII	**85.81**	**74.13**	**81.37**

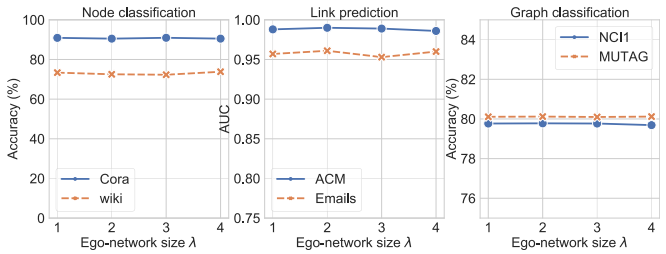

Fig. 3. Ablation study of Ego-network size λ in terms of different tasks. NC task follow the supervised settings.

increasing the number of granularity levels can improve both link prediction and node classifications. As for graph classification, 2 levels would be a proper choice.

Ablation Study of Different Primary GNN Encoders. We adopted GCN as the default primary GNN encoder in model presentation (Section 3) and previous experiments. Here, we present more experimental results by endowing AdamGNN with advanced GNN encoders in Table 10. The table demonstrates that advanced GNN encoders can still benefit from the multi-grained semantics of AdamGNN. For instance, GCNII can stack lots of layers to capture long-range information; however, it still follows a *flat* message-passing mechanism hence naturally ignoring the multi-grained semantics. AdamGNN further ameliorates this problem for better performance.

Ablation Study of Ego-Network Size (λ). The size of an ego-network as defined in Section 3 is captured by λ. We present an ablation study to investigate the influence of λ on AdamGNN's performance, results are summarised in Fig. 3. The figure indicates that λ has no significant influence on the model performance. We simply adopt $\lambda = 1$ throughout the paper.

4.3 More Model Analysis

Running Time Comparison. We present the average epoch training time of different node and graph classification models in Tables 11 and 12, respectively. In terms of node classification task, AdamGNN requires more training time due to the computation cost of α_{ij} and $\boldsymbol{\beta}_t$, similar to any attention-mechanism enhanced models [30]. However, AdamGNN is designed as a local network algorithm, maintaining good scalability; hence it can be easily accelerated by mini-batch and multi-GPUs computing frameworks [14]. It will significantly mitigate the computational issues. On the other hand, AdamGNN follows the sparse design similar to SAGPOOL, ASAP, striking a balance between performance improvement and maintaining proper time efficiency. DIFFPOOL and STRUCTPOOL employ a *dense* mechanism that is not easily

TABLE 11
Average One Epoch Running Time (In Seconds) for Supervised Node Classification Task

Models	ACM	Citeseer	Cora
GCN	0.008	0.008	0.009
GAT	0.010	0.011	0.011
GCNII	0.045	0.040	0.041
g-U-Net	0.076	0.064	0.069
AdamGNN ($\times$1)	0.087	0.072	0.074
AdamGNN ($\times$2)	0.059	0.050	0.052
AdamGNN ($\times$3)	0.050	0.047	0.048

($\times$2) means accelerated by 2 GPUs [14].

scalable to larger graphs [4], and g-U-Net uses convolution operations, which bring additional computation cost, to distribute the received information to the graph.

Messages From Different Levels. We aim to figure out the importance of received messages from different granularity levels since messages from different levels contain the meso/macro-level knowledge encoded by super nodes. Here we consider the node classification on the ACM and DBLP as an example. Specifically, ACM's paper nodes are labelled with 3 topics: database (DB), data mining (DM) and wireless communication (WC); DBLP's author nodes have 4 research areas: AI, DB, DM and computer vision. The node classification task on these two datasets is predicting the paper/scholar's research area. The attention scores of nodes that highlight the importance of different levels' messages are plotted in Fig. 4. We can find different distributions of attention weights over different granularity levels for various areas' classifications. The relatively general topics, i.e., AI and WC, receive messages from different levels with relatively indistinguishable weights, i.e., higher attention scores of nodes are distributed across levels. The DM topic in two datasets has different attention patterns: it receives messages from level 1 with the greatest attention in ACM but receives greatest attention messages from level 3 in DBLP. This is because DM is not closely related to the other two topics of ACM dataset, Scholars of DM-related papers are less possible collaborate with researchers from DB or WC. DM papers are close to each other in the network. Thus DM papers only need to receive level 1 granularity semantic information summarised from neighbouring nodes. In contrast, DBLP's other 3 topics are close to DM. DM-related scholars may cite any other scholars' papers, and information related to DM is scattered over author nodes in DBLP network. Therefore, DM researchers tend to be characterised by level 3 semantics from a wide range.

TABLE 12
Average One Epoch Running Time (In Seconds) for Graph Classification Task

Models	NCI1	NCI109	PROTEINS
DiffPool	6.23	3.22	3.65
SagPool	1.95	1.55	0.45
g-U-Net	4.58	4.45	1.46
EigenPool	3.88	3.54	1.38
ASAP	2.04	1.83	1.09
StructPool	6.31	6.04	1.34
AdamGNN	3.62	3.24	1.03

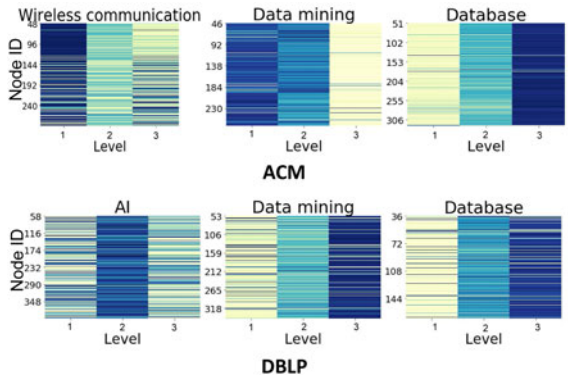

Fig. 4. Visualisation of attention weight for messages at different granularity levels. Dark colours indicate higher weights.

Visualisation of Different Granularity Levels. To better understand the process of learning multi-grained semantics, we report the relative numbers of nodes (i.e., node ratio concerning the original graph) at different granularity levels generated by AdamGNN in 7 datasets, as shown in Fig. 5. In particular, we set the max number of granularity levels as 5, and level 0 indicates the original graph. We train AdamGNN with semi-supervised node classification and report the relative number of nodes and selected ego-nodes at each level. In the right, we can find out that the number of ego-nodes can stay stable after $1-2$ times of our adaptive pooling which indicates that AdamGNN can effectively find a compact structure that contains multi-grained semantics. In the left, we can see that the number of nodes of each level stabilises after $3-4$ times of our adaptive pooling which illustrates AdamGNN will maintain the graph size at a proper level to avoid dense super graph.

Visualisation of Adjacency Matrices at Different Granularity Levels. One of the fundamental limitations of existing GNNs is the inability of capturing long-range node interactions in the graph [19]. We find that AdamGNN can provide a possible solution to overcome this limitation. AdamGNN allows nodes to receive messages from far-away nodes with the support of the adaptive multi-grained structure. That said, the learned multi-grained structure can be regarded as a kind of *short-cuts* to let far-away nodes be aware of each other. We visualise these short-cut connections at different levels on the Wiki dataset in Fig. 6. Figure-level 0 plots the original adjacency matrix, Figure-level 1 exhibits the learned short-cuts by the first pooled super graph (in green). Similarly, figure-levels 2 and 3 present the derived short-cuts by further stacking the

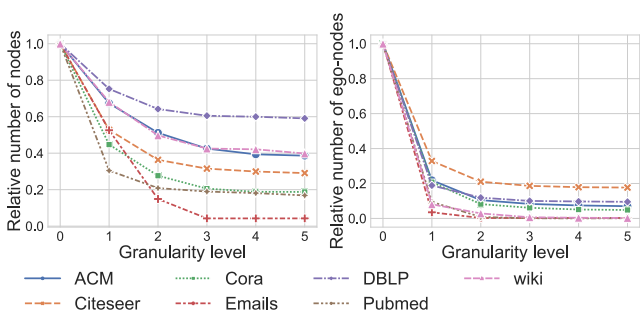

Fig. 5. Visualisation of different granularity levels.

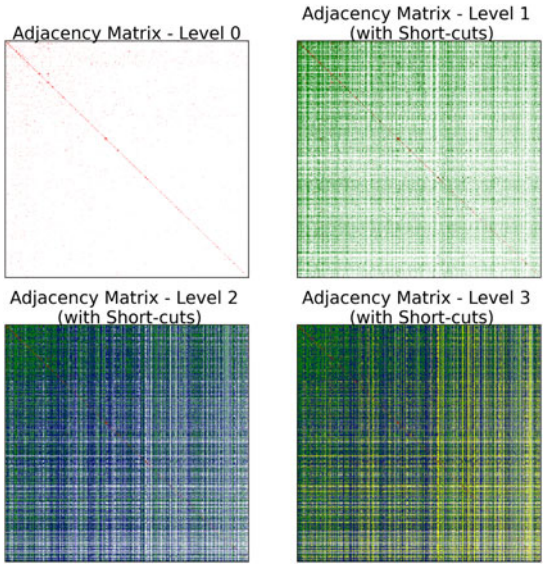

Fig. 6. Visualisation of the adjacency matrices stacking the 1-st (green), 2-nd (blue), and 3-rd (yellow) granularity levels on the Wiki dataset.

TABLE 13
Performance Comparison for 1-Shot-NC Task on *Karate*-Club Dataset

Model	1-Layer	2-Layers
GCN	65.26	69.74
AdamGNN (1-level)	88.65	97.91

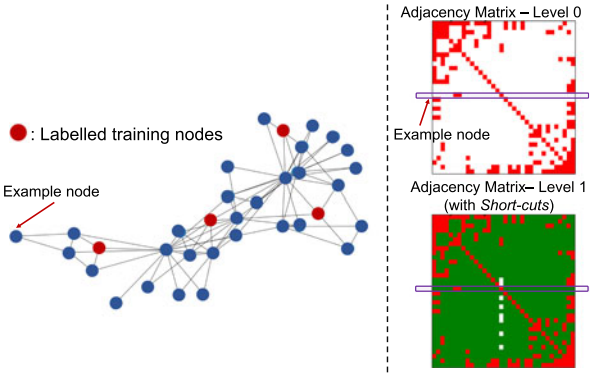

Fig. 7. Visualisation of network structure and adjacency matrix at different granularity levels of *Karate*-club dataset

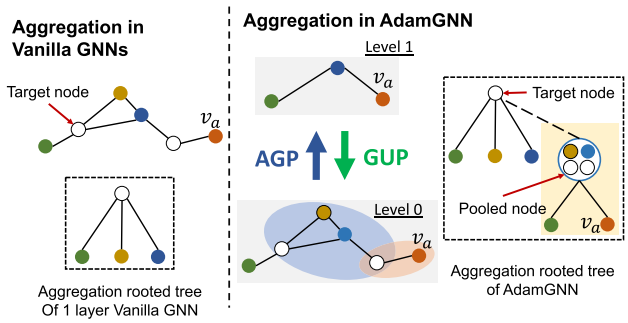

Fig. 8. A toy example of the aggregation schemas of vanilla GNNs (left) and AdamGNN (right).

adjacency matrices of the second (in blue) and third (in yellow) pooled graphs, respectively. We can clearly see that the original graph of the Wiki dataset is very sparse, and AdamGNN adds short-cuts between nodes with the help of the learned multi-grained structure. In this way, AdamGNN allows nodes to capture global information with few adaptively pooled graphs.

Exploration of Short-Cuts of AdamGNN. To explore the *short-cuts* derived by AdamGNN, we perform another empirical analysis on one additional network, i.e., the *Karate* [40] club network. We choose 1-shot NC as target task, where we randomly select one sample from each class as training set, an equal number nodes as validation set and the rest nodes for test. Network structure, experimental results is summarised in Table 13 and two adjacency matrices are shown in Fig. 7. Short-cuts derived by the first pooled super graph are depicted in green in the level 1 adjacency matrix. We find that AdamGNN outperforms GCN on 1-shot NC task with up to 40.5% performance improvements. The two figures in the right part of Fig. 7 clearly demonstrate that the *short-cuts* derived by AdamGNN make the example node aware of far-away nodes.

Comparing Aggregation Mechanisms. To demonstrate the internal aggregation mechanism of AdamGNN and figure out the reason that it leads to performance improvements as shown in Section 4.2, we give a toy example of the aggregation schemas in vanilla GNNs and AdamGNN. As shown in Fig. 8, 1-layer vanilla GNN can only capture limited information as presented in the left rooted tree. Nevertheless, thanks to the adaptive hierarchical structure learned by AdamGNN, target nodes can receive multi-grained semantics as well as endowed with the ability to capture information from nodes from a long-range. For instance, node v_a's message cannot be obtained by target node with a few-layers vanilla GNN, but AdamGNN allows the target node to receive v_a's information with 1 granularity level. The level 1's graph allows the super node to receive a message from node v_a and pass it to the target nodes by *flyback* aggregator.

5 Conclusion

To summarise, we proposed AdamGNN, a method that integrates multi-grained semantics into node representations and realises collective optimisation between node- and graph-wise tasks in one unified process. We have designed an adaptive and efficient pooling operator with a novel ego-network selection approach to encode the multi-grained structural semantics, and a training strategy to overcome the over-smoothing problem. Extensive experiments conducted on 14 real-world datasets showed the promising effectiveness of AdamGNN on node- and graph-wise downstream tasks. One future direction is to appropriately apply the adaptive multi-grained structure on heterogeneous networks for node and graph level tasks.

References

[1] P. Barceló, E. V. Kostylev, M. Monet, J. Pérez, J. L. Reutter, and J. P. Silva, "The logical expressiveness of graph neural networks," in *Proc. Int. Conf. Learn. Representations*, 2020.

[2] D. Bo, X. Wang, C. Shi, M. Zhu, E. Lu, and P. Cui, "Structural deep clustering network," in *Proc. Int. Conf. World Wide Web*, 2020, pp. 1400–1410.

[3] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. Int. Conf. Learn. Representations*, 2014.
[4] C. Cangea, P. Velickovic, N. Jovanovic, T. Kipf, and P. Liò, "Towards sparse hierarchical graph classifiers," 2018, *arXiv:1811.01287*.
[5] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 3438–3445.
[6] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 1725–1735.
[7] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Velickovic, "Principal neighbourhood aggregation for graph nets," in *Proc. Annu. Conf. Neural Informat. Process. Syst.*, 2020, pp. 13260–13271.
[8] W. Feng et al., "Graph random neural networks for semi-supervised learning on graphs," in *Proc. Annu. Conf. Neural Informat. Process. Syst.*, 2020, pp. 22 092–22 103.
[9] H. Gao and S. Ji, "Graph U-Nets," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2083–2092.
[10] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Annu. Conf. Neural Informat. Process. Syst.*, 2017, pp. 1025–1035.
[11] S. Haykin, "Neural networks: A comprehensive foundation," *Knowl. Eng. Rev.*, vol. 13, pp. 409–412, 1999.
[12] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *Int. J. Uncertain. Fuzziness Knowl. Based Syst.*, vol. 6, no. 2, pp. 107–116, 1998.
[13] W. Hu et al., "Open graph benchmark: Datasets for machine learning on graphs," in *Proc. Annu. Conf. Neural Informat. Process. Syst.*, 2020, pp. 22118–22133.
[14] T. Kaler et al., "Accelerating training and inference of graph neural networks with fast sampling and pipelining," 2021, *arXiv:2110.08450*.
[15] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017.
[16] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3734–3743.
[17] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," Jun. 2014. [Online]. Available: http://snap.stanford.edu/data
[18] B. Li, X. Fan, J. Pang, and J. Zhao, "A model to slice Java programs hierarchically," *J. Comput. Sci. Technol.*, vol. 19, no. 6, pp. 848–858, 2004.
[19] G. Li et al., "DeepGCNs: Can GCNs go as deep as CNNs?," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 9267–9276.
[20] M. Li, S. Chen, Y. Zhang, and I. W. Tsang, "Graph cross networks with vertex infomax pooling," in *Proc. Annu. Conf. Neural Informat. Process. Syst.*, 2020, pp. 14 093–14 105.
[21] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 3538–3545.
[22] H. Liu, Y. Yang, and X. Wang, "Overcoming catastrophic forgetting in graph neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 8653–8661.
[23] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in *Proc. ACM Conf. Knowl. Discov. Data Mining*, 2019, pp. 723–731.
[24] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," in *Proc. Annu. Conf. Neural Informat. Process. Syst.*, 2019, pp. 2153–2164.
[25] J. Qiu et al., "GCC: Graph contrastive coding for graph neural network pre-training," in *Proc. ACM Conf. Knowl. Discov. Data Mining*, 2020, pp. 1150–1160.
[26] E. Ranjan, S. Sanyal, and P. P. Talukdar, "ASAP: Adaptive structure aware pooling for learning hierarchical graph representations," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 5470–5477.
[27] C. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, and J. Tang, "Graphaf: A flow-based autoregressive model for molecular graph generation," in *Proc. Int. Conf. Learn. Representations*, 2020.
[28] S. Teng, "Scalable algorithms for data and network analysis," *Found. Trends Theor. Comput. Sci.*, vol. 12, no. 1/2, pp. 1–274, 2016.
[29] V. Thost and J. Chen, "Directed acyclic graph neural networks," in *Proc. Int. Conf. Learn. Representations*, 2021.
[30] A. Vaswani et al., "Attention is all you need," in *Proc. Annu. Conf. Neural Informat. Process. Syst.*, 2017, pp. 5998–6008.
[31] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representations*, 2018.
[32] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.
[33] J. Xie, R. B. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 478–487.
[34] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *Proc. Int. Conf. Mach. Learn.*, 2019.
[35] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information," in *Proc. Int. Joint Conf. Artif. Intell.*, 2015, pp. 2111–2117.
[36] Z. Yang, M. Ding, C. Zhou, H. Yang, J. Zhou, and J. Tang, "Understanding negative sampling in graph representation learning," in *Proc. ACM Conf. Knowl. Discov. Data Mining*, 2020, pp. 1666–1676.
[37] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proc. Annu. Conf. Neural Informat. Process. Syst.*, 2018, pp. 4805–4815.
[38] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7134–7143.
[39] H. Yuan and S. Ji, "Structpool: Structured graph pooling via conditional random fields," in *Proc. Int. Conf. Learn. Representations*, 2020.
[40] W. W. Zachary, "An information flow model for conflict and fission in small groups," *J. Anthropological Res.*, vol. 33, pp. 452–473, 1977.
[41] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proc. Annu. Conf. Neural Informat. Process. Syst.*, 2018, pp. 5171–5181.
[42] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 4438–4445.
[43] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 1, pp. 249–270, Jan. 2020.

Zhiqiang Zhong received the PhD degree from the University of Luxembourg, in 2022. He is currently a postdoctoral researcher with the Department of Computer Science, Aarhus University, Aarhus, Denmark. His principal research interest is in graph machine learning, biomedicine design and social network analysis.

Cheng-Te Li received the PhD degree from the Graduate Institute of Networking and Multimedia, National Taiwan University, 2013. He is an associate professor with the Institute of Data Science, National Cheng Kung University (NCKU), Tainan, Taiwan. Before joining NCKU, He was an assistant research fellow (2014-2016) with CITI, Academia Sinica. His research interests include machine learning, deep learning, data mining, social media analysis, recommender systems, and natural language processing. He has a number of papers published with top conferences, including KDD, WWW, ICDM, CIKM, SIGIR, IJCAI, ACL, EMNLP, and NAACL.

Jun Pang received the PhD degree from Vrije Universiteit Amsterdam in 2004. He is currently a senior researcher with the Faculty of Science, Technology and Medicine & Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg. His research interests include formal methods, social media mining, computational systems biology, security and privacy.