Kernel Readout for Graph Neural Networks

Jiajun Yu[1,3,5], **Zhihao Wu**[2,3], **Jinyu Cai**[4], **Adele Lu Jia**[1*] and **Jicong Fan**[2,3*]
[1]College of Information and Electrical Engineering, China Agricultural University, Beijing, China
[2]School of Data Science, The Chinese University of Hong Kong, Shenzhen (CUHK-Shenzhen), China
[3]Shenzhen Research Institute of Big Data, Shenzhen, China
[4]Institute of Data Science, National University of Singapore, Singapore
[5]College of Computer Science and Technology, Zhejiang University, Hangzhou, China
2017307070311@cau.edu.cn, zhihaowu1999@gmail.com, jinyucai1995@gmail.com, ljia@cau.edu.cn, fanjicong@cuhk.edu.cn

Abstract

Graph neural networks (GNNs) for graph classification or representation learning require a pooling operation to convert the nodes' embeddings of each graph to a vector as the graph-level representation and the operation has a significant impact on model accuracy. The paper presents a novel graph pooling method called Kernel Readout (KerRead). KerRead maps the node embeddings from the sample space with limited nodes to an augmented sample space with infinite nodes, and then calculates the inner product between some learnable adaptive centers and the augmented node embeddings, which forms a final graph-level feature vector. We apply the proposed strategy to six supervised and two unsupervised graph neural networks such as GCN, GIN, GUNet, InfoGraph, and GraphCL, and the experiments on eight benchmark datasets show that the proposed readout outperforms classical pooling methods such as Sum and seven state-of-the-art pooling methods such as SRead and Janossy GRU. Code and Appendix are both available at https://github.com/jiajunCAU/KerRead.

1 Introduction

In recent years, graph neural networks (GNNs) have flourished in many fields such as machine learning [Yu and Jia, 2023; Wu *et al.*, 2023b; Zhang *et al.*, 2024], computer vision [Han *et al.*, 2022a; Chen *et al.*, 2023b; Cai *et al.*, 2024b], and bioinformatics [Gilmer *et al.*, 2017; Gasteiger *et al.*, 2021]. Originating from the theoretical investigation of both spatial and spectral domains, GNN has become a general paradigm powerful for modeling non-Euclidean graph data. Typically, GNNs generate representations that preserve structural information by recursive neighborhood aggregation. Numerous studies [Welling and Kipf, 2017; Hamilton *et al.*, 2017; Zheng *et al.*, 2023; Jin *et al.*, 2021] have been devoted to designing various GNN architectures for node-level tasks like node classification and clustering, and the most prevalent GNNs have been validated to generate expressive node representations.

Beyond that, some critical real-world scenarios, e.g. protein interface prediction [Fout *et al.*, 2017; Liu *et al.*, 2020] and compounds functionality analysis [Kojima *et al.*, 2020], require predicting for entire graphs, leading to graph-level tasks, where each graph, rather than a node, is a sample. A key challenge is the graph isomorphism problem, that is, how to recognize whether two graphs are isomorphic or not. Xu *et al.* [2018] first explored the connection between GNNs and the graph isomorphic problem, proving that GNNs are at least as powerful as the Weisfeiler-Lehman graph isomorphism test. Motivated by these, many researchers have proposed GNNs focusing on graph-level tasks [Gilmer *et al.*, 2017]. Generally, these methods apply readout functions, also known as graph pooling functions, on node representations, and then obtain graph-level representations, i.e., a vector representing the entire graph.

Readout is a crucial technique bridging powerful GNNs and graph-level learning. The most widely adopted graph-level GNN backbones [Xu *et al.*, 2018] only used some simple readout functions like Sum, Max, and Mean. These functions typically collect first-order statistics only, which does not accurately characterize the distribution of graphs. Meanwhile, they are fixed and non-data-driven operations, so they may not fit different graphs well. Some recent work has put forward learnable readout functions to solve these issues. They are mainly categorized as attention-based [Li *et al.*, 2016; Itoh *et al.*, 2022; Fan *et al.*, 2020] and sequence-based [Vinyals *et al.*, 2016; Buterez *et al.*, 2022] approaches. These methods have made some progress, but the gains are not steady, and most critically, no unified framework to guide the design of readout. [Sun and Fan, 2024] proposed a deep MMD graph kernel (MMD-GK) that integrates graph kernel learning with graph neural networks and has promising performance in graph classification and clustering. Although Deep MMD-GK does not require any readout operation, it has quadratic time and space complexities.

In this article, we propose a novel kernel-based readout function named KerRead, which utilizes the kernel technology in readout. The detailed process of KerRead is depicted in Figure 1. To be specific, we treat each dimension as a point with coordinates determined by the feature values of each

*Corresponding authors.

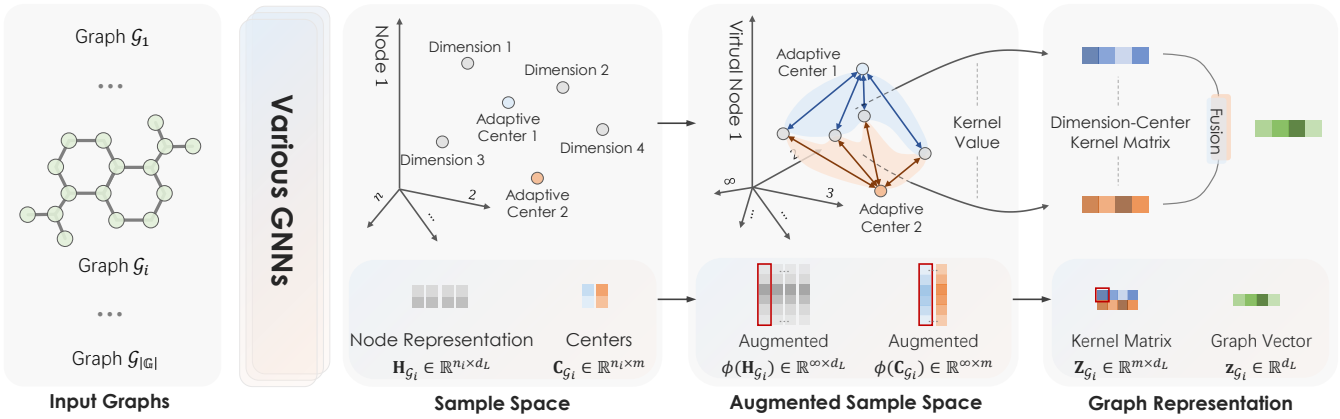

Figure 1: The overall framework of KerRead. The kernel readout function first designs the adaptive center, and then uses a kernel function to map the node embedding and the adaptive center to the augmented sample space, and then calculates the kernel value based on each dimension of augmented node embedding and the augmented adaptive center to obtain the final graph vector. This can be formulated as:

$$\mathcal{G}_i = (\mathbf{X}_{\mathcal{G}_i} \in \mathbb{R}^{n_i \times d}, \mathbf{A}_{\mathcal{G}_i} \in \mathbb{R}^{n_i \times n_i}) \xrightarrow{\text{GNN}} \mathbf{H}_{\mathcal{G}_i} \in \mathbb{R}^{n_i \times d_L} \xrightarrow{\phi} \phi(\mathbf{H}_{\mathcal{G}_i}) \in \mathbb{R}^{\infty \times d_L} \xrightarrow{\phi(\mathbf{C}_{\mathcal{G}_i}) \in \mathbb{R}^{\infty \times m}} \mathbf{Z}_{\mathcal{G}_i} \in \mathbb{R}^{m \times d_L} \xrightarrow{\mathbf{w}} \mathbf{z}_{\mathcal{G}_i} \in \mathbb{R}^{d_L}.$$

node in this dimension, then we define several adaptive centers and calculate kernels between these centers and dimension points, leading to several advantages over other methods: 1) KerRead implicitly augments the number of nodes in a graph to be large or even infinite through the Gaussian kernel; 2) Graph representations are generated by computing the kernel matrices so they are naturally in the same scale; 3) The centers are trainable and the number of centers can be flexibly defined. 4) KerRead adheres to the crucial property of permutation invariance, a vital characteristic for robust graph learning. In general, KerRead employs kernel technology to better reflect the latent distributions of diverse node representations, which is a more powerful and scalable method.

Our contributions are as follows.

- We propose a new paradigm to analyze and design readout functions from the perspective of kernel functions.
- We put forward KerRead which solves several drawbacks of existing methods.
- We conduct comprehensive experiments to evaluate the proposed KerRead, including graph classification and clustering with various GNN backbones.

2 Related Work

2.1 Graph Neural Networks

Graph neural networks (GNNs) aim to learn low-dimensional and dense node representations from high-dimensional sparse graphs while preserving the original structural information. Initially, GNNs were operated in the spectral domain [Bruna *et al.*, 2014; Defferrard *et al.*, 2016; Welling and Kipf, 2017; Wang and Zhang, 2022; Wu *et al.*, 2024] and transformed the feature signals of the graph into the frequency domain through Fourier transformation and subsequently designing various filters in the frequency domain, where the goal was to make the signal of target node more similar with its neighbors. Meanwhile, other graph neural network methods operated in the spatial domain [Hamilton *et al.*, 2017; Veličković *et al.*, 2018; Xu *et al.*, 2018; Cai *et al.*, 2024a; Yu and Jia, 2024], which can acquire the node representation by message passing mechanism. For instance, GraphSAGE [Hamilton *et al.*, 2017] aggregated the sampled neighborhoods of the target nodes to generate the target node representations. Graph Attention Network (GAT) [Veličković *et al.*, 2018] introduced the multi-attentional module mechanism into GraphSAGE. Meanwhile, many widely developed domains, such as machine learning [Chen *et al.*, 2023a; Fan *et al.*, 2022; Chen *et al.*, 2022; Fan, 2022] and data mining [Cai *et al.*, 2022; Zheng *et al.*, 2022; Fan, 2021; Cai and Fan, 2022], have extensively employed numerous GNN variants [Chen *et al.*, 2023c; Chen *et al.*, 2023d; Wu *et al.*, 2023a; Zheng *et al.*, 2021].

2.2 Graph-level Prediction

Graph-level tasks treat each entire graph as a sample, posing many new challenges for GNNs. For this purpose, many GNN methods were focusing on the graph-level tasks [Xu *et al.*, 2018; Gilmer *et al.*, 2017; Gao and Ji, 2019; Buterez *et al.*, 2022; Ju *et al.*, 2022; Sun and Fan, 2024; Han *et al.*, 2022b]. For instance, [Gilmer *et al.*, 2017] proposed a general framework for quantum chemistry representation called Message Passing Neural Networks. GUNet [Gao and Ji, 2019] added the graph pooling and unpooling layers during graph convolution. [Luo *et al.*, 2023] explored the semi-supervised classification problem from a subgraph perspective. [Cai *et al.*, 2024a] advanced graph clustering with a novel pseudo-label mechanism. [Liu *et al.*, 2022] considered the long-tailed problem in the graph classification. [Sun *et al.*, 2023] utilized the Lovász principle to build an unsupervised graph representation learning framework.

2.3 Readout Function

The readout function, a vital component in GNNs, pools the learned graph embedding matrix into a graph embedding vector for the subsequent graph classification task. The aforementioned graph-level methods use simple and naive

readout functions, such as sum, mean, or max, to generate the final graph representation vector. [Li *et al.*, 2016; Itoh *et al.*, 2022; Fan *et al.*, 2020] proposed attention-based readout functions to capture the importance of each node. [Lee *et al.*, 2021] proposed learnable modules for readout functions to further improve their expressive power. To further enhance the expressiveness of readout functions, certain methods incorporate techniques designed for sequence data types into the readout process [Navarin *et al.*, 2019; Xu *et al.*, 2018; Buterez *et al.*, 2022]. There are also some hierarchical readout methods, that is, reading out smaller and smaller graphs step by step as the graph is embedded until it is aggregated into a node [Zhang *et al.*, 2018; Ying *et al.*, 2018; Du *et al.*, 2021], however, these hierarchical methods also need a readout function to gradually reduce the size of the graph, so in this article, we only focus the readout function which map the node embedding into graph vector directly.

3 Proposed Method

3.1 Notation and Problem Formulation

A graph is presented as $\mathcal{G} = (\mathbf{A}, \mathbf{X})$, where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is the adjacency matrix, where $\mathbf{A}_{ij} = 1$ denotes that v_i is connected to v_j and $\mathbf{A}_{ij} = 0$ otherwise. Let $\mathbb{G} = \{\mathcal{G}_1, ..., \mathcal{G}_N\}$ be the graph set. We aim to learn the graph representation vector $\mathbf{z}_{\mathcal{G}_i}$ for each graph $\mathcal{G}_i$ to facilitate downstream tasks.

Encoding Graph GNNs are usually used to encode each graph and consist of some graph convolutional layers. Each layer is based on the message-passing mechanism, i.e., aggregating the neighborhood information for each target node, which is formed by two steps

$$\bar{\mathbf{H}}^{(l+1)}_{\mathcal{G}_i} = \mathrm{AGGREGATE}\big(\mathbf{A}_{\mathcal{G}_i}, \mathbf{H}^{(l)}_{\mathcal{G}_i}\big), \tag{1}$$

$$\mathbf{H}^{(l+1)}_{\mathcal{G}_i} = \mathrm{COMBINE}\big(\mathbf{H}^{(l)}_{\mathcal{G}_i}, \bar{\mathbf{H}}^{(l+1)}_{\mathcal{G}_i}\big), \tag{2}$$

where we have $\mathbf{H}^{(0)}_{\mathcal{G}_i} = \mathbf{X}_{\mathcal{G}_i}$. For convenience in expression, we will use $\mathbf{H}_{\mathcal{G}_i} \in \mathbb{R}^{n_i\times d_L}$ to denote the node embedding matrix of graph $\mathcal{G}_i$ at final layers in the following part.

Readout Function After convolution by GNNs and getting node emebdding matrix $\mathbf{H}_{\mathcal{G}_i}$, a readout function is utilized to map $\mathbf{H}_{\mathcal{G}_i} \in \mathbb{R}^{n_i\times d_L}$ into a graph vector $\mathbf{z}_{\mathcal{G}_i} \in \mathbb{R}^{d_L}$ for the following graph classification task:

$$\mathbf{z}_{\mathcal{G}_i} = \mathrm{READOUT}\big(\{\mathbf{H}_{\mathcal{G}_i}[j,:] \mid v_j \in \mathcal{G}_i\}\big), \tag{3}$$

where $\mathbf{H}_{\mathcal{G}_i}[j,:]$ refers to the j-th row of $\mathcal{G}_i$ node embedding matrix, i.e., node v_j in graph $\mathcal{G}_i$.

3.2 Detailed Method

In this part, we will introduce our proposed KerRead, which incorporates the principles of kernels into the realm of readout mechanisms. KerRead is a learnable readout function that matches GNNs' expressiveness well by solving the following challenges of existing methods:

- **Challenge 1:** How to 'readout' distinguishable graph representations in a data-driven manner?
- **Challenge 2:** How to fairly learn graph representations for all graphs in a unified size sample space?
- **Challenge 3:** How to capture latent distributions from graphs with scarce and varying numbers of nodes?

The following involves devising an adaptive center, and we leverage expressive kernel functions to calculate graph representations, e.g., Gaussian kernel. Moreover, we put forth the concept of multi-head centers to further enhance the model's capabilities. The overall framework is shown in Figure 1.

Adaptive Center In graph-level tasks, although graphs vary, nodes of different graphs are described by features with the same meaning and amount. Therefore, it makes more sense to think of readout in a dimension-wise perspective than in the sample-wise one, i.e., we describe graphs by considering how to express distributions in each dimension instead of directly considering fusing representations of all nodes.

More specifically, $\mathbf{H}_{\mathcal{G}_i}[:, j]$ in the node representations can be viewed as a description or distribution for the distribution of graph $\mathcal{G}_i$ in the corresponding dimension, and these distributions of different graphs are naturally comparable. From this view, mapping the node representation matrix into a graph representation vector can be regarded as representing the distributions of each dimension by one value. Assume there is a central dimension called *center* in this paper, we can represent the distribution by calculating certain distances between each dimension and the center, and our key idea is employing the effective kernel method to achieve this. It is obvious that the design of the center is critical in our method. On the one hand, one single and fixed center can not adapt to different graphs; On the other hand, centers of different graphs can not be totally individual otherwise the graph representations may not be compared. For this purpose, we design an *adaptive center* which can adapt different graphs and share information among graphs. First we leverage a shared model $f_\Theta : \mathbb{R}^{d_L} \to \mathbb{R}$ to generate the adaptive center as

$$\mathbf{c}_{\mathcal{G}_i} = s \cdot f_\Theta(\mathbf{H}_{\mathcal{G}_i}), \tag{4}$$

where $s \in \mathbb{R}$ is a learnable scale factor, and f_Θ can be a certain neural network like Multi-Layer Perceptrons (MLP) performed on each row of $\mathbf{H}_{\mathcal{G}_i}$ and Θ is the set of learnable parameters. Note that f_Θ can also be a simple pooling function, in which Θ is the empty set. For example, the adaptive center $\mathbf{c}_{\mathcal{G}_i} \in \mathbb{R}^{n_i}$ can be

$$\mathbf{c}_{\mathcal{G}_i} = [s \cdot f_\Theta(\mathbf{H}_{\mathcal{G}_i}[1,:]), \cdots, s \cdot f_\Theta(\mathbf{H}_{\mathcal{G}_i}[n_i,:])]^\top. \tag{5}$$

In this way, the learning of centers is data-driven and optimized in conjunction with GNNs. More importantly, the designed model can address graphs with different sizes as it adaptively generates a center for each graph. Meanwhile, all graphs share f_Θ and the scale factor s, so that the training process incorporates information from various graphs. These provide bases to solve **Challenge 1** and **2**. After obtaining $\mathbf{c}_{\mathcal{G}_i}$ with flexible size n_i, i.e., the number of nodes in $\mathcal{G}_i$, we can calculate the kernel value between each dimension of node embedding and the adaptive center to obtain the final graph vector in the following section.

Kernel Calculation Compared to node-level tasks, graphs commonly contain very scarce nodes in graph-level tasks (e.g., molecules), and the number of nodes in different graphs

may be imbalanced. Thus, it is difficult to capture latent distributions and derive discriminative graph representations (**Challenges 3**). To tackle this, we further employ a transformation function ϕ to map each column of node embedding and the adaptive center into augmented sample space with more virtual nodes:

$$\phi(\mathbf{H}_{\mathcal{G}_i}) = \left[\phi\big(\mathbf{H}_{\mathcal{G}_i}[:,1]\big), \cdots, \phi\big(\mathbf{H}_{\mathcal{G}_i}[:,d_L]\big)\right], \quad (6)$$

where $\mathbf{H}_{\mathcal{G}_i}[:,j]$ denotes the j-th column of $\mathbf{H}_{\mathcal{G}_i}$, and $\phi : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i'}$ maps a vector refers to n_i nodes to a vector refers to n_i' nodes, where $n_i' \gg n_i$. In general, explicitly designing a ϕ is cost-prohibitive, particularly when n_i' is significantly large or even infinite. Instead, we leverage kernel functions $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ capable of implicitly inducing ϕ:

$$k\big(\mathbf{c}_{\mathcal{G}_i}, \mathbf{H}_{\mathcal{G}_i}[:,j]\big) = \phi(\mathbf{c}_{\mathcal{G}_i})^\top \phi(\mathbf{H}_{\mathcal{G}_i}[:,j]). \quad (7)$$

Thus the final graph representation vector is:

$$\mathbf{z}_{\mathcal{G}_i} = \left[k\big(\mathbf{c}_{\mathcal{G}_i}, \mathbf{H}_{\mathcal{G}_i}[:,1]\big), \cdots, k\big(\mathbf{c}_{\mathcal{G}_i}, \mathbf{H}_{\mathcal{G}_i}[:,d_L]\big)\right]. \quad (8)$$

There are numerous available kernel functions, each inducing a unique feature map ϕ. Perhaps the best-known kernel function is the Gaussian kernel, i.e.,

$$k(\mathbf{c}_{\mathcal{G}_i}, \mathbf{H}_{\mathcal{G}_i}[:,j]) = \exp\left(-\frac{\|\mathbf{c}_{\mathcal{G}_i} - \mathbf{H}_{\mathcal{G}_i}[:,j]\|^2}{2\sigma^2}\right), \quad (9)$$

where $\mathbf{c}_{\mathcal{G}_i}$ is the proposed learnable adaptive center corresponding to $\mathcal{G}_i$, and the parameter $\sigma \in \mathbb{R}$ is also learnable. The mapping ϕ induced by a Gaussian kernel is infinite-dimension. In our experiment, we evaluated various kernel functions, including the Laplace kernel, Sigmoid kernel, Polynomial kernel, and others. The performance results for these kernels can be found in Section 4.3.

Multi-head Centers In order to enhance the richness of information captured from node representation $\mathbf{H}_{\mathcal{G}_i}$, relying solely on a single center may be insufficient. To address this limitation, we further introduce the concept of multi-head center, which initializes m learnable scale factors denoted as $\mathbf{s} = [s_1, ..., s_m]$. The adaptive center matrix $\mathbf{C}_{\mathcal{G}_i} \in \mathbb{R}^{n_i \times m}$ is then defined as follows:

$$\mathbf{C}_{\mathcal{G}_i} = g_\Psi(\mathbf{H}_{\mathcal{G}_i}) = [s_1 \cdot f_\Theta(\mathbf{H}_{\mathcal{G}_i}), \cdots, s_m \cdot f_\Theta(\mathbf{H}_{\mathcal{G}_i})], \quad (10)$$

where $\Psi = \{\mathbf{s}, \Theta\}$ is the parameter set containing scale factors and parameters of f_Θ. Subsequently, the multi-head kernel matrix $\mathbf{Z}_{\mathcal{G}_i} \in \mathbb{R}^{m \times d_L}$ is expressed as:

$$\mathbf{Z}_{\mathcal{G}_i} = \begin{bmatrix} k(\mathbf{C}_{\mathcal{G}_i}[:,1], \mathbf{H}_{\mathcal{G}_i}[:,1]) & \cdots & k(\mathbf{C}_{\mathcal{G}_i}[:,1], \mathbf{H}_{\mathcal{G}_i}[:,d_L]) \\ \vdots & \ddots & \vdots \\ k(\mathbf{C}_{\mathcal{G}_i}[:,m], \mathbf{H}_{\mathcal{G}_i}[:,1]) & \cdots & k(\mathbf{C}_{\mathcal{G}_i}[:,m], \mathbf{H}_{\mathcal{G}_i}[:,d_L]) \end{bmatrix}, \quad (11)$$

where $\mathbf{Z}_{\mathcal{G}_i}$ is only related to the number of heads m and hidden dimension d_L. Finally, the fusion is performed on all graph representation vectors:

$$\mathbf{z}_{\mathcal{G}_i} = h_\Upsilon(\mathbf{Z}_{\mathcal{G}_i}), \quad (12)$$

where $h_\Upsilon : \mathbb{R}^m \to \mathbb{R}$ is a fusion model and Υ is the parameter set. As a simple example, h_Υ can be a weighted summation of $\{\mathbf{Z}_{\mathcal{G}_i}[j,:]\}_{j=1}^m$, allowing for adaptive learning of contributions from each center to the final graph vector.

Overall Framework The entire pipeline of graph representation learning on graph $\mathcal{G}_i$ with KerRead is formulated as

$$\begin{aligned}\mathbf{z}_{\mathcal{G}_i} &= h_\Upsilon\Bigg(\underbrace{\phi\Big(g_\Psi\big(f_{\mathcal{W}}^{\mathrm{GNN}}(\mathcal{G}_i)\big)\Big)^\top}_{m \times n_i'} \underbrace{\phi\Big(f_{\mathcal{W}}^{\mathrm{GNN}}(\mathcal{G}_i)\Big)}_{n_i' \times d_L}\Bigg) \\ &\triangleq \mathrm{M}_{\Upsilon,\Psi,\mathcal{W}}(\mathcal{G}_i),\end{aligned} \quad (13)$$

where $f_{\mathcal{W}}^{\mathrm{GNN}}$ is a certain GNN with parameter set $\mathcal{W}$ and we denote $\mathrm{M}_{\Upsilon,\Psi,\mathcal{W}}$ the over-all graph representation learning model with KerRead for convenience. Recall that ϕ is the implicit mapping induced by the kernel function k. When k is a Gaussian kernel, then $n_i' = \infty$.

Theorem 1. *Suppose the kernel k is feature permutation invariant, then $\mathrm{M}_{\Upsilon,\Psi,\mathcal{W}}$ is node permutation invariant—that is, for any graph $\mathcal{G}$ and its any permuted counterpart $\mathcal{G}'$, $\mathrm{M}_{\Upsilon,\Psi,\mathcal{W}}(\mathcal{G}') = \mathrm{M}_{\Upsilon,\Psi,\mathcal{W}}(\mathcal{G})$ always holds.*

The proof is deferred to Appendix A[1]. Note that the assumption in the theorem is very general. Many popular kernels such as Gaussian kernel, Laplacian kernel, and polynomial kernels are feature permutation invariant. Theorem 1 indicates the permutation invariance of our proposed framework, which has been proven to be very critical in graph-level learning [Buterez *et al.*, 2022].

The training strategy of KerRead is based on the selected GNN backbone. For the graph classification task, we employ a supervised GNN algorithm, using cross-entropy as the loss function for supervised learning. Conversely, in handling graph clustering tasks, we opt for an unsupervised GNN backbone and utilize such as InfoNCE loss [Oord *et al.*, 2018] for representation learning, and then input learned graph embeddings into clustering algorithms, such as K-means [Lloyd, 1982] or spectral clustering [Ng *et al.*, 2001] to obtain the clustering results. This ensures that our model can be flexibly and effectively applied to different tasks.

For the complexity analysis of KerRead and $\mathrm{M}_{\Upsilon,\Psi,\mathcal{W}}$, we denote the number of graphs as N and the initial node feature dimension as d_0, $n = \max(\{n_i\}_{i=1}^N)$ is the max number of nodes and $|\mathcal{E}| = \max(\{|\mathcal{E}_i|\}_{i=1}^N)$ is the max number of edges among graphs, where $\mathcal{E}_i$ is the edge set of $\mathcal{G}_i$. Considering an L-layer GNN, we define $d = \max(\{d_i\}_{i=1}^L)$ is the max hidden dimension. For the forward pass of $\mathrm{M}_{\Upsilon,\Psi,\mathcal{W}}$ in per iteration, the time complexity of KerRead is $\mathcal{O}(Nnmd)$, including adaptive centers generation and kernel calculation. The overall time complexity is $\mathcal{O}(N(nd+|\mathcal{E}|)(Ld+d_0))$ with $m \ll d_0$. The space complexity of KerRead is $\mathcal{O}(m(n+d))$, and that of the overall framework is $\mathcal{O}(N((n+d)(Ld+d_0)+L|\mathcal{E}|))$. Generally, the additional costs our approach imposes on the framework are negligible.

3.3 Connection to Existing Work

For brief formulation, we adopt one center vector $\mathbf{c}_{\mathcal{G}_i}$ for graph $\mathcal{G}_i$, meaning a feature dimension in graph $\mathcal{G}_i$, and compute the j-th element of graph representation $\mathbf{z}_{\mathcal{G}_i}$ as

$$\mathbf{z}_{\mathcal{G}_i}[j] = \phi(\mathbf{c}_{\mathcal{G}_i})^\top \phi(\mathbf{H}_{\mathcal{G}_i}[:,j]). \quad (14)$$

[1]The appendix can be found at https://github.com/jiajunCAU/KerRead/blob/master/Appendix.pdf

Backbone	Readout	MUTAG	DD	PROTEINS	NCI1	Mutagenicity	IMDB-B	IMDB-M	Avg
GCN [Welling and Kipf, 2017]	Sum	86.7 (5.5)	68.8 (4.7)	73.9 (3.5)	75.8 (2.3)	78.0 (1.8)	69.9 (4.3)	48.1 (4.1)	71.6
	Max	81.8 (9.7)	72.8 (4.3)	66.0 (5.7)	72.4 (3.9)	79.6 (1.7)	71.2 (4.1)	46.7 (4.3)	70.1
	Mean	87.2 (7.3)	71.1 (3.0)	68.1 (4.1)	73.6 (1.8)	78.5 (2.2)	71.1 (3.3)	**48.7 (3.8)**	71.2
	Set2set	84.0 (4.2)	**74.2 (4.9)**	72.5 (4.0)	**80.9 (1.9)**	**82.0 (0.9)**	69.3 (4.3)	48.5 (3.8)	**73.1**
	Attention	**87.2 (4.4)**	71.4 (4.0)	70.5 (3.5)	74.5 (2.6)	80.2 (2.3)	72.0 (4.6)	47.1 (5.5)	71.8
	Deep Sets	84.6 (4.9)	73.6 (3.3)	**75.5 (5.3)**	76.2 (1.9)	78.2 (1.1)	72.0 (2.9)	48.5 (2.8)	72.7
	SRead	86.1 (5.6)	68.8 (3.2)	74.1 (5.0)	75.8 (2.0)	78.4 (1.9)	70.4 (4.7)	47.6 (3.9)	71.6
	Set Transformer	84.6 (6.0)	69.7 (3.9)	71.7 (5.8)	80.7 (2.6)	82.0 (1.7)	**72.1 (3.0)**	48.3 (1.7)	72.7
	Janossy MLP	76.7 (11.4)	54.0 (3.4)	65.6 (4.0)	71.8 (2.1)	74.6 (1.7)	67.8 (3.3)	48.3 (3.1)	65.5
	Janossy GRU	84.6 (7.7)	58.6 (0.4)	59.5 (0.2)	80.2 (1.6)	73.5 (10.8)	69.3 (3.6)	46.1 (6.6)	67.4
	KerRead	**88.3 (6.1)**	**77.8 (3.0)**	**75.9 (2.8)**	**82.6 (2.0)**	**82.6 (2.0)**	**72.5 (2.4)**	**49.0 (2.9)**	**75.5**
GAT [Veličković *et al.*, 2018]	Sum	76.0 (8.8)	73.6 (3.9)	73.6 (3.1)	70.9 (3.0)	75.0 (2.1)	50.0 (0.0)	**34.7 (2.3)**	64.8
	Max	75.5 (8.4)	74.3 (4.1)	66.9 (5.0)	62.0 (3.5)	73.3 (2.9)	50.0 (0.0)	33.3 (0.0)	62.2
	Mean	74.9 (9.7)	70.3 (4.2)	70.0 (5.4)	69.1 (2.2)	75.8 (2.6)	50.0 (0.0)	33.7 (1.4)	63.4
	Set2set	76.0 (10.9)	**75.2 (5.4)**	72.6 (4.1)	73.5 (1.6)	78.7 (2.4)	50.0 (0.0)	33.3 (0.0)	**65.6**
	Attention	**78.7 (8.1)**	67.7 (4.2)	71.4 (5.4)	69.5 (2.2)	76.9 (1.8)	50.0 (0.0)	33.3 (0.0)	63.9
	Deep Sets	74.0 (12.0)	72.8 (4.6)	**74.2 (4.9)**	72.0 (1.7)	74.7 (1.3)	50.0 (0.0)	33.3 (0.0)	64.4
	SRead	74.4 (7.8)	70.0 (4.5)	73.8 (4.0)	70.2 (3.1)	74.1 (1.8)	**50.2 (0.6)**	34.4 (2.3)	63.9
	Set Transformer	74.0 (8.6)	66.8 (4.0)	73.4 (5.2)	72.9 (2.3)	**79.6 (1.8)**	50.0 (0.0)	33.3 (0.0)	64.3
	Janossy MLP	74.4 (7.4)	52.3 (3.9)	65.0 (4.7)	68.4 (1.8)	71.4 (1.7)	50.0 (0.0)	33.3 (0.0)	59.3
	Janossy GRU	75.5 (9.1)	69.9 (5.0)	59.5 (0.3)	**75.5 (3.1)**	65.3 (11.5)	49.8 (1.1)	32.9 (0.8)	61.2
	KerRead	**79.0 (5.4)**	**76.1 (3.2)**	**75.1 (3.2)**	**76.0 (2.3)**	**79.8 (1.8)**	**53.9 (3.1)**	**36.1 (2.5)**	**68.0**
GIN [Xu *et al.*, 2018]	Sum	87.3 (5.9)	72.5 (3.2)	70.3 (4.1)	79.0 (1.6)	81.0 (1.4)	71.0 (3.7)	48.1 (2.2)	72.7
	Max	82.5 (4.6)	72.4 (3.3)	72.5 (2.7)	**81.0 (2.1)**	82.0 (1.9)	71.2 (4.3)	48.2 (3.7)	72.8
	Mean	87.2 (5.0)	69.7 (3.3)	68.1 (4.1)	77.7 (1.9)	81.2 (1.6)	71.4 (4.3)	47.3 (2.3)	71.8
	Set2set	**87.3 (4.2)**	73.5 (5.5)	69.0 (4.4)	80.7 (1.2)	**82.5 (1.8)**	**71.8 (4.0)**	47.6 (3.3)	73.2
	Attention	85.1 (6.1)	70.0 (5.0)	68.4 (4.5)	78.4 (1.7)	81.9 (1.5)	70.9 (3.3)	**48.4 (4.6)**	71.9
	Deep Sets	86.8 (4.8)	**73.6 (3.3)**	**75.4 (4.8)**	79.1 (2.0)	81.1 (2.3)	71.7 (2.7)	47.9 (2.6)	**73.7**
	SRead	86.2 (6.9)	69.0 (3.7)	71.7 (5.0)	77.7 (1.7)	80.1 (2.1)	70.8 (3.7)	46.5 (3.8)	71.7
	Set Transformer	87.3 (4.8)	66.9 (4.3)	72.1 (3.9)	79.3 (1.7)	81.7 (1.5)	71.6 (2.0)	47.5 (3.3)	72.3
	Janossy MLP	79.4 (11.6)	53.1 (2.7)	60.7 (3.7)	70.8 (1.8)	77.3 (1.8)	69.5 (4.6)	47.9 (2.6)	65.5
	Janossy GRU	82.4 (6.6)	58.6 (0.4)	61.3 (4.1)	79.3 (2.8)	60.9 (10.0)	68.2 (5.8)	36.4 (7.8)	63.9
	KerRead	**88.8 (7.6)**	**76.5 (4.2)**	**72.7 (5.2)**	**83.0 (2.7)**	**82.8 (2.4)**	**72.5 (5.5)**	**49.0 (4.6)**	**75.0**

Table 1: Graph classification accuracy (mean and std%) with three GNN backbones (GCN, GAT, and GIN), where **orange shading** denotes the best performance and **blue shading** indicates the second-best performance. Note that GAT performs abnormally on the IMDB-B and IMDB-M datasets due to the absence of features which results in meaningless attention calculations.

Based on this formula, the process of readout can be described as performing a kernel on the dimensions of node representations and a center vector:

$$\mathbf{z}_{\mathcal{G}_i}[j] = k(\mathbf{c}_{\mathcal{G}_i}, \mathbf{H}_{\mathcal{G}_i}[:, j]). \tag{15}$$

For example, the widely adopted Sum can be formulated as

$$\mathrm{SUM}(\mathbf{H}_{\mathcal{G}_i})[j] = \mathbf{1}_{\mathcal{G}_i}^{\top}(\mathbf{H}_{\mathcal{G}_i}[:, j]) = k(\mathbf{1}_{\mathcal{G}_i}, \mathbf{H}_{\mathcal{G}_i}[:, j]) \tag{16}$$

where $\mathbf{1}_{\mathcal{G}_i}$ is an all-one vector and a linear kernel is employed, thus ϕ is an identity mapping. Similarly, we have

$$\begin{cases} \mathrm{MEAN}(\mathbf{H}_{\mathcal{G}_i})[j] = k\big(\mathbf{e}_{\mathcal{G}_i}, \mathbf{H}_{\mathcal{G}_i}[:, j]\big), \\ \mathrm{MAX}(\mathbf{H}_{\mathcal{G}_i})[j] = k\big(\mathcal{I}_{max}(\mathbf{H}_{\mathcal{G}_i}[:, j]), \mathbf{H}_{\mathcal{G}_i}[:, j]\big), \\ \mathrm{ATTENTION}(\mathbf{H}_{\mathcal{G}_i})[j] = k\big(f_{\Omega}^{att}(\mathbf{H}_{\mathcal{G}_i}), \mathbf{H}_{\mathcal{G}_i}[:, j]\big), \end{cases} \tag{17}$$

where k is a linear kernel, $\mathbf{e}_{\mathcal{G}_i} = [1/n_i, \cdots, 1/n_i]$, and $\mathcal{I}_{max}(\mathbf{x})_i = \begin{cases} 1, & \text{if } i = \arg\max(\mathbf{x}) \\ 0, & \text{otherwise} \end{cases}$. f_{Ω}^{att} is an attention model to generate the attention score as the node weight. In this way, we innovatively explain several common readout methods from the perspective of the kernel, and find these traditional readout functions both are simplified versions and special instances of KerRead. Furthermore, this perspective also explains the causes of the above challenges: Challenge 1—fixed center; Challenge 2—non-adaptive center for all graphs; Challenge 3—linear kernel.

4 Experiments

In this section, we first introduce the benchmark datasets, the backbone, the baseline readout function, and parameter settings, and then we present the comparison results of our methods with other readout functions in different GNN backbones on graph classification tasks and graph clustering tasks.

4.1 Experimental Setting

Datasets This study employs a total of 8 graph-level datasets, comprising 6 chemical molecule datasets (MUTAG, DD, PROTEINS, NCI1, Mutagenicity and OGBG-Molhiv) and 2 social network graph datasets (IMDB-B, IMDB-M), these datasets are collected in TU datasets[2] and open graph benchmark[3]. The description and statistics of the dataset are in Appendix C.

Backbone and Baseline We use 6 widely adopted supervised GNN methods, including three baseline GNNs (GCN [Welling and Kipf, 2017], GraphSAGE [Hamilton *et al.*, 2017], SGC [Wu *et al.*, 2019]), three more expressive GNNs (GAT [Veličković *et al.*, 2018], GIN [Xu *et al.*, 2018] and GUNet [Gao and Ji, 2019]), and two unsupervised GNNs (InfoGraph [Sun *et al.*, 2019] and GraphCL [You *et al.*, 2020]) are also used in this article. To ensure a fair comparison, we select 10 readout functions as our baseline for comparison. These encompass simple readout functions (Sum, Mean, Max), along with Attention [Li *et al.*, 2016], Set2set [Vinyals *et al.*, 2016], Deep Sets [Zaheer *et al.*, 2017], SRead [Lee *et al.*, 2021], Set Transformer, Janossy MLP, and Janossy GRU [Buterez *et al.*, 2022].

[2]https://chrsmrrs.github.io/datasets/

[3]https://ogb.stanford.edu/

Backbone	Readout	MUTAG	DD	PROTEINS	NCI1	Mutagenicity	IMDB-B	IMDB-M	Avg
GUNet [Gao and Ji, 2019]	Sum	74.1 (6.3)	70.6 (3.1)	71.3 (4.8)	76.9 (3.2)	77.9 (2.0)	63.7 (5.8)	41.4 (3.6)	68.0
	Max	73.9 (6.2)	70.5 (4.8)	71.2 (5.0)	75.6 (2.0)	**80.5 (1.8)**	63.6 (5.9)	40.9 (3.4)	68.0
	Mean	76.6 (9.2)	72.9 (4.4)	71.8 (4.4)	**77.7 (2.3)**	**80.2 (1.4)**	62.2 (5.4)	40.3 (3.5)	69.1
	Set2set	74.4 (10.9)	72.5 (3.9)	71.2 (4.5)	74.0 (3.4)	79.8 (2.0)	**64.9 (5.8)**	40.0 (3.4)	68.1
	Attention	74.4 (6.3)	70.6 (3.1)	71.3 (4.8)	76.9 (3.2)	77.9 (2.0)	63.7 (5.8)	42.4 (3.8)	68.2
	Deep Sets	75.5 (14.5)	71.6 (3.8)	**75.3 (3.1)**	77.2 (1.9)	78.6 (2.0)	62.0 (3.8)	38.5 (3.8)	68.4
	SRead	75.0 (11.1)	**75.8 (2.9)**	72.5 (3.1)	76.4 (2.4)	80.2 (2.3)	64.5 (4.7)	**42.7 (3.8)**	**69.6**
	Set Transformer	76.6 (11.4)	65.7 (3.2)	72.2 (5.3)	75.0 (2.1)	80.2 (1.6)	64.2 (2.1)	40.5 (3.8)	67.8
	Janossy MLP	72.9 (11.6)	56.0 (2.2)	68.4 (5.5)	75.1 (1.1)	77.4 (2.0)	62.4 (3.8)	38.5 (2.6)	64.4
	Janossy GRU	**77.0 (6.7)**	67.0 (7.9)	59.4 (0.5)	73.4 (2.5)	57.1 (3.8)	60.3 (7.9)	32.6 (8.4)	61.0
	KerRead	**77.1 (6.0)**	**76.1 (4.4)**	**74.9 (3.5)**	**78.4 (3.0)**	**81.9 (1.4)**	**65.2 (3.2)**	**41.7 (2.5)**	**70.8**
GraphSAGE [Hamilton *et al.*, 2017]	Sum	87.7 (5.5)	69.3 (3.8)	74.0 (3.7)	71.1 (1.8)	76.3 (1.4)	65.2 (4.8)	45.0 (2.9)	69.8
	Max	82.9 (7.6)	74.9 (3.4)	74.1 (4.4)	**73.7 (1.6)**	79.7 (2.1)	58.8 (3.8)	44.6 (2.5)	69.8
	Mean	87.8 (3.5)	70.8 (3.6)	65.1 (3.4)	67.3 (1.5)	77.3 (1.4)	65.6 (6.1)	43.5 (5.3)	68.2
	Set2set	85.6 (4.2)	75.6 (4.9)	73.1 (4.2)	73.2 (3.4)	81.2 (1.7)	**71.1 (4.2)**	46.3 (3.4)	**72.3**
	Attention	87.2 (5.5)	69.9 (2.0)	69.0 (3.4)	69.2 (2.6)	79.0 (1.9)	70.1 (4.0)	45.4 (2.6)	70.0
	Deep Sets	**88.8 (4.4)**	**78.7 (5.1)**	**74.9 (3.4)**	71.0 (2.5)	77.3 (2.0)	67.9 (3.4)	43.6 (3.5)	71.7
	SRead	86.2 (4.2)	68.8 (2.9)	72.9 (4.0)	71.1 (1.8)	76.9 (1.6)	69.9 (4.5)	44.5 (4.4)	70.0
	Set Transformer	80.3 (4.9)	67.7 (3.7)	71.0 (3.0)	77.2 (1.3)	**81.5 (1.5)**	70.4 (2.4)	**47.4 (2.5)**	70.8
	Janossy MLP	77.7 (8.7)	52.0 (4.5)	64.1 (4.7)	68.7 (1.9)	73.1 (1.6)	65.1 (4.6)	44.2 (3.5)	63.6
	Janossy GRU	80.3 (5.9)	71.1 (2.2)	60.6 (4.2)	78.2 (2.5)	75.3 (10.0)	66.1 (8.3)	32.1 (6.9)	66.2
	KerRead	**88.9 (6.3)**	**75.8 (2.9)**	**75.0 (2.9)**	**77.6 (2.7)**	**81.6 (1.4)**	**71.2 (4.7)**	**47.5 (3.5)**	**73.9**
SGC [Wu *et al.*, 2019]	Sum	72.8 (7.9)	67.0 (2.5)	74.0 (3.3)	64.9 (1.9)	68.1 (1.8)	56.7 (6.9)	37.7 (2.3)	63.0
	Max	72.8 (9.7)	72.2 (3.5)	71.4 (4.4)	69.7 (1.9)	77.2 (1.3)	58.0 (4.9)	40.3 (3.9)	65.9
	Mean	73.9 (6.2)	70.3 (3.4)	66.8 (2.7)	59.3 (1.8)	67.3 (1.8)	60.6 (6.2)	39.7 (2.1)	62.6
	Set2set	**80.2 (12.3)**	72.8 (5.0)	73.0 (4.8)	67.7 (3.2)	76.2 (1.9)	61.0 (5.2)	40.9 (1.9)	67.4
	Attention	74.9 (7.3)	69.7 (4.8)	66.8 (2.6)	61.3 (1.8)	69.4 (2.0)	59.3 (6.1)	39.4 (3.1)	63.0
	Deep Sets	70.7 (7.7)	**76.8 (4.2)**	**74.5 (4.3)**	64.9 (2.3)	68.3 (2.0)	61.4 (3.5)	**41.1 (3.5)**	65.4
	SRead	77.6 (8.5)	69.4 (5.0)	69.8 (3.1)	68.1 (2.4)	71.4 (2.1)	58.9 (4.8)	39.6 (3.4)	65.0
	Set Transformer	76.7 (9.0)	70.2 (3.6)	72.9 (4.5)	**77.6 (2.9)**	**77.4 (1.3)**	59.1 (5.3)	39.1 (3.1)	**67.6**
	Janossy MLP	78.7 (10.0)	57.4 (3.4)	63.4 (3.3)	61.7 (2.0)	66.3 (3.0)	53.5 (4.3)	37.7 (2.8)	59.8
	Janossy GRU	**81.9 (8.3)**	58.6 (0.4)	73.9 (5.0)	77.5 (1.1)	66.0 (11.7)	**63.1 (2.8)**	**43.9 (2.5)**	66.4
	KerRead	77.7 (9.5)	**76.9 (2.8)**	**75.5 (2.9)**	**74.9 (2.4)**	**77.5 (2.3)**	**61.5 (2.6)**	40.7 (3.3)	**69.3**

Table 2: Graph classification accuracy (mean and std%) with three GNN backbones (GUNet, GraphSAGE, and SGC), where **orange shading** denotes the best performance and **blue shading** indicates the second-best performance.

Dataset	Readout	GCN	GIN	GraphSAGE	GAT	GUNet	SGC	Avg
OGBG-Molhiv	Sum	60.9 (2.9)	71.3 (1.7)	58.3 (2.3)	62.3 (2.1)	59.0 (1.7)	59.7 (2.1)	61.9
	Max	58.4 (2.3)	**71.8 (2.0)**	66.9 (3.6)	67.7 (2.0)	66.7 (4.4)	66.1 (3.2)	66.3
	Mean	57.4 (1.8)	70.9 (2.4)	51.0 (2.2)	60.4 (2.3)	50.3 (1.0)	50.3 (0.7)	56.7
	Set2set	66.8 (1.8)	70.9 (1.3)	59.4 (1.8)	65.7 (2.0)	56.7 (2.1)	58.0 (1.1)	62.9
# Graphs: 41,127	Attention	62.0 (1.7)	70.4 (1.5)	51.1 (1.9)	58.5 (3.0)	51.0 (1.6)	51.8 (2.2)	57.5
# Features: 9	Deep Sets	63.3 (2.0)	52.9 (1.0)	53.1 (0.8)	62.2 (2.2)	53.4 (1.4)	52.7 (1.0)	56.3
# Classes: 2	SRead	63.0 (1.4)	70.9 (2.5)	57.2 (1.2)	63.2 (2.9)	59.1 (3.4)	51.8 (2.2)	60.9
Avg. # Nodes: 25.5	Set Transformer	**72.2 (1.6)**	71.2 (1.3)	61.3 (1.4)	**69.4 (1.5)**	61.8 (2.2)	**69.2 (2.1)**	67.5
Avg. # Edges: 27.5	Janossy MLP	52.4 (0.6)	50.0 (0.1)	50.0 (0.1)	51.5 (1.2)	50.0 (0.0)	50.0 (0.0)	50.7
	Janossy GRU	71.1 (1.4)	67.9 (2.6)	**68.1 (2.4)**	69.1 (1.2)	**67.1 (1.9)**	68.1 (2.2)	**68.6**
	KerRead	**72.5 (1.6)**	**72.2 (1.5)**	**67.1 (1.2)**	**70.4 (0.7)**	**67.9 (2.6)**	**68.9 (2.1)**	**69.8**

Table 3: Graph classification AUC (mean and std%) on OGBG-Molhiv dataset with six GNN backbones, where **orange shading** denotes the best performance and **blue shading** indicates the second-best performance.

Parameter Settings We employ the officially released source code and adopt the parameters suggested in the respective papers for all baseline readout functions and GNN backbones. Additional information on experimental settings can be found in Appendix C.

4.2 Performance Analysis

In this section, we analyze the performance of graph classification and graph clustering.

Graph Classifications We conducted a comprehensive analysis comparing the graph classification performance of KerRead with ten other readout functions across six GNN backbone models on eight datasets, as detailed in Table 1, Table 2, and Table 3. We obtain the following observations from this comparison:

(i) KerRead consistently outperforms other readout methods across various GNN backbones. KerRead surpasses the most powerful baseline Set2set readout by 2.3% and 2.4% on GCN and GAT backbones on average.

(ii) KerRead has achieved the best performance on datasets with different sizes. Compared to baseline readout functions, we observe an AUC increase ranging from 0.8% to 19.2% on large-scale dataset OGBG-Molhiv on average.

(iii) The average performance of different readouts in SGC was significantly lower than GUNet and GraphSAGE, which indicates that SGC does not possess a substantial advantage in graph-level tasks similar to its superiority in node-level tasks.

(iv) Sequence-based readout methods, such as Janossy GRU, Set2Set, and Set Transformer, demonstrated better average performance on SGC. We speculate that incorporating complex sequential information can enhance the expressive power of final graph embedding when the backbone cannot capture enough information.

In particular, KerRead outperforms Sum, Mean, Max, and

Backbone	Readout	MUTAG		DD		PROTEINS		NCI1		Mutagenicity		IMDB-B		IMDB-M	
		ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI
InfoGraph [Sun *et al.*, 2019]	Sum	71.3	14.1	59.7	2.6	60.4	2.6	56.6	1.3	56.4	1.7	62.3	4.4	40.8	2.2
	Max	**80.9**	25.6	60.8	4.7	61.7	2.8	60.4	3.3	57.2	1.6	61.1	4.7	38.9	2.2
	Mean	71.3	14.1	56.5	2.2	62.4	**3.4**	56.8	1.4	60.0	3.5	63.1	5.2	41.0	2.4
	Set2set	74.5	17.5	63.2	6.0	**62.5**	3.2	59.0	2.4	62.7	4.3	62.9	5.1	40.5	**3.2**
	Attention	80.3	24.4	55.2	1.0	59.9	2.4	57.3	1.6	63.3	5.1	62.9	5.0	40.3	1.8
	Deep Sets	76.1	**33.5**	60.9	3.2	62.3	3.2	57.8	1.9	60.5	3.7	62.4	4.6	41.3	2.2
	SRead	79.3	31.7	60.4	3.6	59.5	2.0	58.3	2.0	61.1	4.2	62.6	4.6	41.5	2.5
	Set Transformer	73.9	16.9	**70.3**	**13.0**	59.8	2.6	**60.9**	**3.6**	**65.2**	6.4	**63.6**	**5.4**	41.5	1.9
	Janossy MLP	62.8	11.5	55.4	1.3	58.8	0.4	59.4	3.0	62.4	**7.3**	62.2	4.3	41.9	2.1
	Janossy GRU	70.7	26.0	58.6	0.0	59.4	0.0	58.1	0.0	62.3	0.1	50.1	0.0	**42.3**	**3.5**
	KerRead	**83.5**	**36.4**	**74.8**	**20.5**	**63.4**	**6.3**	**61.8**	**3.8**	**65.7**	**7.0**	**63.5**	**5.3**	**42.8**	2.6
GraphCL [You *et al.*, 2020]	Sum	73.9	16.9	66.0	8.3	64.8	4.7	57.2	1.6	61.5	3.7	57.6	2.4	42.5	3.0
	Max	79.3	21.8	59.1	3.1	64.4	4.5	**58.7**	2.2	61.1	3.7	62.6	5.0	39.9	3.4
	Mean	73.9	16.9	66.3	8.7	64.4	4.5	55.2	0.8	61.6	3.4	61.6	4.0	42.9	3.3
	Set2set	70.7	17.5	60.7	2.5	63.4	3.2	58.3	**2.6**	63.6	**6.6**	**64.8**	6.4	43.8	**3.8**
	Attention	76.1	24.3	55.8	0.5	62.0	2.1	57.6	2.3	61.1	3.1	60.9	4.5	39.9	3.5
	Deep Sets	75.0	18.2	58.7	4.1	61.2	2.5	57.8	2.1	**64.5**	**6.2**	59.6	3.3	41.8	2.8
	SRead	76.6	24.9	**68.1**	**8.9**	**65.1**	**5.0**	57.1	2.1	62.2	5.8	63.9	**8.6**	43.9	**4.0**
	Set Transformer	79.8	32.2	53.7	0.3	58.3	1.7	55.7	1.0	61.4	4.0	59.6	4.2	41.9	2.5
	Janossy MLP	72.3	9.8	52.0	0.2	62.1	2.3	54.3	0.7	64.1	6.0	60.7	3.4	42.3	2.9
	Janossy GRU	**81.4**	**33.8**	55.4	0.6	65.1	4.7	56.2	1.2	56.6	1.3	57.7	2.3	**44.7**	3.0
	KerRead	**83.5**	**34.1**	**75.4**	**18.7**	**70.9**	**11.2**	**58.8**	**2.8**	**64.2**	6.1	**64.4**	**6.5**	**45.0**	3.6

Table 4: Graph clustering accuracy (ACC) and normalized mutual information (NMI) with two unsupervised GNN backbones (InfoGraph, GraphCL), where **orange shading** denotes the best performance and **blue shading** indicates the second-best performance.

	MUTAG	DD	PROTEINS	NCI1	Mutagenicity	IMDB-B	IMDB-M	Avg
Linear	86.3 (7.9)	71.0 (5.2)	72.6 (4.2)	79.0 (2.1)	81.3 (1.4)	71.0 (2.5)	47.2 (2.5)	72.6
Sigmoid	86.0 (5.2)	73.3 (4.2)	75.0 (5.5)	80.2 (2.1)	81.0 (1.7)	71.0 (3.0)	47.4 (2.5)	73.4
Polynomial	87.5 (6.8)	72.2 (2.6)	73.0 (5.6)	80.2 (2.2)	81.9 (1.2)	70.8 (2.6)	47.6 (2.4)	73.3
Laplacian	88.1 (5.5)	76.7 (5.4)	72.5 (3.9)	82.2 (1.8)	81.4 (1.4)	71.5 (2.4)	47.2 (1.7)	74.2
Gaussian	88.3 (6.1)	77.8 (3.0)	75.9 (2.8)	82.6 (2.0)	82.6 (2.0)	72.5 (2.4)	49.0 (2.9)	75.5

Table 5: Mean accuracy (10 folds) and standard deviation on the 7 graph classification datasets with 5 kernel functions, where GCN is employed as the backbone.

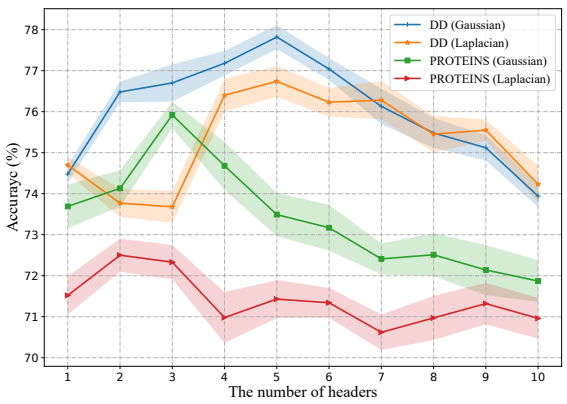

Figure 2: The performance of graph classification with different kernels and different numbers of heads.

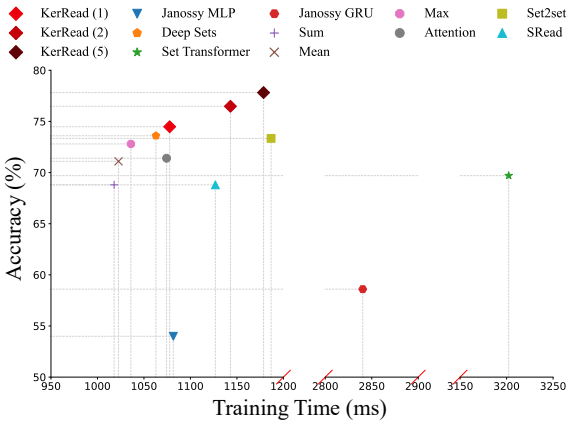

Figure 3: Performance over training time on DD dataset. KerRead (1), (2), and (5) refer to KerRead with 1, 2, and 5 heads respectively.

Attention in all comparisons, which aligns with earlier analysis since these readout functions are special cases of KerRead, and their expressivity is weaker than KerRead. Furthermore, KerRead's performance is more stable than some sequence-based methods, we consider KerRead satisfies permutation invariance, so it is not influenced by the order of input nodes, thus avoiding any impact on performance.

Graph Representation Learning & Clustering To provide a more comprehensive evaluation of KerRead, we conducted graph clustering experiments on seven benchmark datasets using two classic unsupervised graph backbone networks, as shown in Table 4. In general, KerRead continues to demonstrate highly competitive performance in graph clustering tasks. Compared to other readout functions, we observed an increase in ACC from 2.6% to 20.7% and an improvement in NMI from 2.9% to 24.9% on MUTAG when using InfoGraph. On GraphCL, ACC increased from 2.1% to 12.8%, and NMI increased from 0.3% to 24.3%. However, KerRead did not achieve optimal performance on the IMDB-B and IMDB-M datasets. We speculate that this may be at-

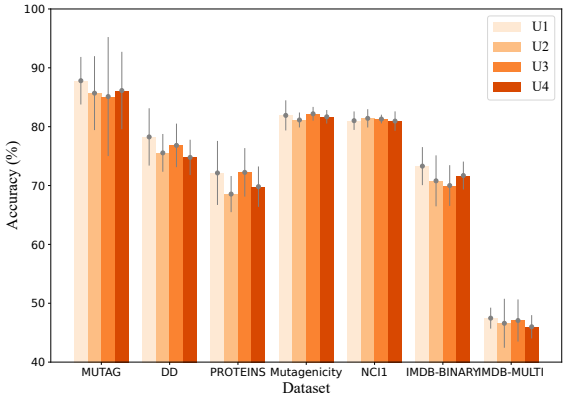

Figure 4: The graph classification performance of KerRead with different learning strategies. (U1: both c and σ are learnable, U2: only c is learnable while σ is fixed, U3: only σ is learnable while c is fixed, U4: both c and σ are fixed.)

tributed to insufficient initial features in these two datasets, leading to the inability to capture enough information when computing the kernel values between each dimension of node embedding and the adaptive center.

4.3 Sensitivity Analysis

In this section, we will focus on conducting a sensitivity analysis of the parameters of our proposed KerRead. This includes examining the impact of varying the number of heads, different kernel functions, distinct update strategies on the final results, and the efficiency analysis.

Kernel Function Selection As depicted in Figure 2, we analyze two common kernel functions, namely the Laplace kernel function and the Gaussian kernel, on the DD and PROTEINS datasets. Overall, the performance of the Gaussian kernel surpasses that of the Laplace kernel. Besides, we have further compared the graph classification performance of five commonly used kernel functions on seven datasets. The experimental results are shown in Table 5, revealing the following observations: (i) Overall, KerRead outperforms most baselines regardless of the kernel function used. (ii) The use of non-linear kernel functions yields better performance compared to linear kernel functions. Specifically, the Gaussian kernel outperforms the Linear kernel by an average accuracy increase of 2.9%.

The Number of Heads To explore the impact of the number of heads on the results, we present the graph classification results on the DD and PROTEINS datasets under different numbers of heads as shown in Figure 2. Overall, optimal performance is observed when the number of heads ranges from 2 to 6. However, as the number of heads increases, there may be a negative impact on performance. Specifically, the Laplace kernel achieves optimal performance on the DD dataset when the number of heads is 5, surpassing the performance at 1 and 10 heads by 2.04% and 2.51%, respectively. Similarly, the Gaussian kernel attains its best performance on the DD dataset with 5 heads, exhibiting improvements of 3.46% and 3.88% compared to 1 and 10 heads, respectively. These findings suggest that an appropriate increase in the number of heads can enhance performance, but an excessive number may lead to detrimental effects on the model.

Training Time We conducted an efficiency analysis comparing our model with a baseline readout. As illustrated in Figure 3, we depict the training time (measured in milliseconds per epoch) variation of KerRead under 1, 2, and 5 heads, as well as other methods. Generally, KerRead requires more training time as the number of heads increases. However, its training time remains closely aligned with the linear readout function and consistently stays below the complex readout functions (Set Transformer, Janossy GRU), however, KerRead achieves significant performance improvements compared to these linear readout functions. Furthermore, KerRead demonstrates optimal performance on the DD dataset, where each epoch takes approximately 4.1% longer than the linear readout function, which indicates that the gains in accuracy achieved by KerRead outweigh the marginal pressure introduced by the training time.

Learning Strategies We mentioned that the parameters c and σ in the model are learnable as shown in Eq (9) when employing Gaussian kernel. However, what impact would ensue if these parameters were fixed? As illustrated in Figure 4, we present the performance comparison of these four update strategies across seven datasets. Our observations indicate that, on average, the optimal performance is attained when both parameters are learnable. Across the seven datasets, U1 outperforms U4 by 1.67%, 3.48%, 2.33%, 0.24%, 0.07%, 1.6%, and 1.47%, respectively. This suggests that concurrently updating both parameters indeed facilitates the learning of a superior model.

4.4 Supplementary Experiment

Due to space constraints, we have included the other curial experiments in Appendix C, which include the t-SNE visualizations.

5 Conclusions

This work proposed a novel graph pooling strategy called kernel readout. The main idea is to transfer the node embedding from the sample space with limited nodes to the augmented sample space with infinite virtual nodes, and then generate the graph vector by kernel methods. The strategy has been applied to six supervised GNN backbones and two unsupervised GNN backbones and outperformed previous methods of graph pooling, on eight graph datasets.

Acknowledgements

This work was partially supported by the National Natural Science Foundation of China under Grant No.62376236, the General Program JCYJ20210324130208022 of Shenzhen Fundamental Research, the Guangdong Key Lab of Mathematical Foundations for Artificial Intelligence (PPA00003), the funding UDF01001770 of The Chinese University of Hong Kong, Shenzhen, and the National Natural Science Foundation for Young Scholars of China under Grants No.61502500 and No.61602500.

Contribution Statement

Jiajun Yu and Zhihao Wu contributed equally as co-first authors.

References

[Bruna *et al.*, 2014] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and deep locally connected networks on graphs. In *ICLR*, 2014.

[Buterez *et al.*, 2022] David Buterez, Jon Paul Janet, Steven J Kiddle, Dino Oglic, and Pietro Liò. Graph neural networks with adaptive readouts. In *NeurIPS*, pages 19746–19758, 2022.

[Cai and Fan, 2022] Jinyu Cai and Jicong Fan. Perturbation learning based anomaly detection. In *NeurIPS*, 2022.

[Cai *et al.*, 2022] Jinyu Cai, Jicong Fan, Wenzhong Guo, Shiping Wang, Yunhe Zhang, and Zhao Zhang. Efficient deep embedded subspace clustering. In *CVPR*, pages 21–30, 2022.

[Cai *et al.*, 2024a] Jinyu Cai, Yi Han, Wenzhong Guo, and Jicong Fan. Deep graph-level clustering using pseudo-label-guided mutual information maximization network. *NCAA*, pages 1–16, 2024.

[Cai *et al.*, 2024b] Jinyu Cai, Yunhe Zhang, Shiping Wang, Jicong Fan, and Wenzhong Guo. Wasserstein embedding learning for deep clustering: A generative approach. *IEEE T-MM*, 2024.

[Chen *et al.*, 2022] Man-Sheng Chen, Tuo Liu, Chang-Dong Wang, Dong Huang, and Jian-Huang Lai. Adaptively-weighted integral space for fast multiview clustering. In *MM*, pages 3774–3782, 2022.

[Chen *et al.*, 2023a] Man-Sheng Chen, Jia-Qi Lin, Chang-Dong Wang, Wu-Dong Xi, and Dong Huang. On regularizing multiple clusterings for ensemble clustering by graph tensor learning. In *MM*, pages 3069–3077, 2023.

[Chen *et al.*, 2023b] Zhaoliang Chen, Lele Fu, Jie Yao, Wenzhong Guo, Claudia Plant, and Shiping Wang. Learnable graph convolutional network and feature fusion for multi-view learning. *Information Fusion*, 95:109–119, 2023.

[Chen *et al.*, 2023c] Zhaoliang Chen, Zhihao Wu, Zhenghong Lin, Shiping Wang, Claudia Plant, and Wenzhong Guo. Agnn: Alternating graph-regularized neural networks to alleviate over-smoothing. *IEEE T-NNLS*, pages 1–13, 2023.

[Chen *et al.*, 2023d] Zhaoliang Chen, Zhihao Wu, Shiping Wang, and Wenzhong Guo. Dual low-rank graph autoencoder for semantic and topological networks. In *AAAI*, pages 4191–4198, 2023.

[Defferrard *et al.*, 2016] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, 2016.

[Du *et al.*, 2021] Jinlong Du, Senzhang Wang, Hao Miao, and Jiaqiang Zhang. Multi-channel pooling graph neural networks. In *IJCAI*, pages 1442–1448, 2021.

[Fan *et al.*, 2020] Xiaolong Fan, Maoguo Gong, Yu Xie, Fenlong Jiang, and Hao Li. Structured self-attention architecture for graph-level representation learning. *Pattern Recognition*, 100:107084, 2020.

[Fan *et al.*, 2022] Jicong Fan, Yiheng Tu, Zhao Zhang, Mingbo Zhao, and Haijun Zhang. A simple approach to automated spectral clustering. In *NeurIPS*, pages 9907–9921, 2022.

[Fan, 2021] Jicong Fan. Large-scale subspace clustering via k-factorization. In *SIGKDD*, pages 342–352, 2021.

[Fan, 2022] Jicong Fan. Multi-mode deep matrix and tensor factorization. In *ICLR*, 2022.

[Fout *et al.*, 2017] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *NeurIPS*, 2017.

[Gao and Ji, 2019] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *ICML*, pages 2083–2092, 2019.

[Gasteiger *et al.*, 2021] Johannes Gasteiger, Florian Becker, and Stephan Günnemann. Gemnet: Universal directional graph neural networks for molecules. In *NeurIPS*, pages 6790–6802, 2021.

[Gilmer *et al.*, 2017] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, pages 1263–1272, 2017.

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

[Han *et al.*, 2022a] Kai Han, Yunhe Wang, Jianyuan Guo, Yehui Tang, and Enhua Wu. Vision gnn: An image is worth graph of nodes. In *NeurIPS*, pages 8291–8303, 2022.

[Han *et al.*, 2022b] Xiaotian Han, Zhimeng Jiang, Ninghao Liu, and Xia Hu. G-mixup: Graph data augmentation for graph classification. In *ICML*, pages 8230–8248, 2022.

[Itoh *et al.*, 2022] Takeshi D Itoh, Takatomi Kubo, and Kazushi Ikeda. Multi-level attention pooling for graph neural networks: Unifying graph representations with multiple localities. *Neural Networks*, 145:356–373, 2022.

[Jin *et al.*, 2021] Ming Jin, Yizhen Zheng, Yuan-Fang Li, Chen Gong, Chuan Zhou, and Shirui Pan. Multi-scale contrastive siamese networks for self-supervised graph representation learning. In *International Joint Conference on Artificial Intelligence 2021*, pages 1477–1483. Association for the Advancement of Artificial Intelligence (AAAI), 2021.

[Ju *et al.*, 2022] Wei Ju, Xiao Luo, Meng Qu, Yifan Wang, Chong Chen, Minghua Deng, Xian-Sheng Hua, and Ming Zhang. Tgnn: A joint semi-supervised framework for graph-level classification. In *IJCAI*, 2022.

[Kojima *et al.*, 2020] Ryosuke Kojima, Shoichi Ishida, Masateru Ohta, Hiroaki Iwata, Teruki Honma, and Yasushi Okuno. kgcn: a graph-based deep learning framework for chemical structures. *Journal of Cheminformatics*, 12:1–10, 2020.

[Lee *et al.*, 2021] Dongha Lee, Su Kim, Seonghyeon Lee, Chanyoung Park, and Hwanjo Yu. Learnable structural semantic readout for graph classification. In *ICDM*, pages 1180–1185, 2021.

[Li *et al.*, 2016] Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. Gated graph sequence neural networks. In *ICLR*, 2016.

[Liu *et al.*, 2020] Yi Liu, Hao Yuan, Lei Cai, and Shuiwang Ji. Deep learning of high-order interactions for protein interface prediction. In *SIGKDD*, pages 679–687, 2020.

[Liu *et al.*, 2022] Zemin Liu, Qiheng Mao, Chenghao Liu, Yuan Fang, and Jianling Sun. On size-oriented long-tailed graph classification of graph neural networks. In *WWW*, pages 1506–1516, 2022.

[Lloyd, 1982] Stuart Lloyd. Least squares quantization in pcm. *IEEE T-IT*, 28(2):129–137, 1982.

[Luo *et al.*, 2023] Xiao Luo, Yusheng Zhao, Yifang Qin, Wei Ju, and Ming Zhang. Towards semi-supervised universal graph classification. *IEEE T-KDE*, 2023.

[Navarin *et al.*, 2019] Nicolò Navarin, Dinh Van Tran, and Alessandro Sperduti. Universal readout for graph convolutional neural networks. In *IJCNN*, pages 1–7, 2019.

[Ng *et al.*, 2001] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *NeurIPS*, 2001.

[Oord *et al.*, 2018] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[Sun and Fan, 2024] Yan Sun and Jicong Fan. Mmd graph kernel: Effective metric learning for graphs via maximum mean discrepancy. In *ICLR*, 2024.

[Sun *et al.*, 2019] Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *ICLR*, 2019.

[Sun *et al.*, 2023] Ziheng Sun, Chris Ding, and Jicong Fan. Lovász principle for unsupervised graph representation learning. In *NeurIPS*, 2023.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[Vinyals *et al.*, 2016] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. In *ICLR*, 2016.

[Wang and Zhang, 2022] Xiyuan Wang and Muhan Zhang. How powerful are spectral graph neural networks. In *ICML*, pages 23341–23362, 2022.

[Welling and Kipf, 2017] Max Welling and Thomas N Kipf. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[Wu *et al.*, 2019] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, pages 6861–6871, 2019.

[Wu *et al.*, 2023a] Zhihao Wu, Xincan Lin, Zhenghong Lin, Zhaoliang Chen, Yang Bai, and Shiping Wang. Interpretable graph convolutional network for multi-view semi-supervised learning. *IEEE T-MM*, 25:8593–8606, 2023.

[Wu *et al.*, 2023b] Zhihao Wu, Zhao Zhang, and Jicong Fan. Graph convolutional kernel machine versus graph convolutional networks. In *NeurIPS*, 2023.

[Wu *et al.*, 2024] Zhihao Wu, Zhaoliang Chen, Shide Du, Sujia Huang, and Shiping Wang. Graph convolutional network with elastic topology. *Pattern Recognition*, 151:110364, 2024.

[Xu *et al.*, 2018] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2018.

[Ying *et al.*, 2018] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

[You *et al.*, 2020] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *NeurIPS*, pages 5812–5823, 2020.

[Yu and Jia, 2023] Jiajun Yu and Adele Lu Jia. Mlgal: Multi-level label graph adaptive learning for node clustering in the attributed graph. *Knowledge-Based Systems*, 278:110876, 2023.

[Yu and Jia, 2024] Jiajun Yu and Adele Lu Jia. Agcl: Adaptive graph contrastive learning for graph representation learning. *Neurocomputing*, 566:127019, 2024.

[Zaheer *et al.*, 2017] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *NeurIPS*, 2017.

[Zhang *et al.*, 2018] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

[Zhang *et al.*, 2024] Yunhe Zhang, Yan Sun, Jinyu Cai, and Jicong Fan. Deep orthogonal hypersphere compression for anomaly detection. In *ICLR*, 2024.

[Zheng *et al.*, 2021] Yizhen Zheng, Vincent CS Lee, Zonghan Wu, and Shirui Pan. Heterogeneous graph attention network for small and medium-sized enterprises bankruptcy prediction. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 140–151. Springer, 2021.

[Zheng *et al.*, 2022] Yizhen Zheng, Shirui Pan, Vincent Lee, Yu Zheng, and Philip S Yu. Rethinking and scaling up graph contrastive learning: An extremely efficient approach with group discrimination. In *NeurIPS*, pages 10809–10820, 2022.

[Zheng *et al.*, 2023] Yizhen Zheng, He Zhang, Vincent Lee, Yu Zheng, Xiao Wang, and Shirui Pan. Finding the missing-half: Graph complementary learning for homophily-prone and heterophily-prone graphs. In *ICML*, pages 42492–42505, 2023.