



Article

# A Scalable Deep Network for Graph Clustering via Personalized PageRank

Yulin Zhao , Xunkai Li, Yinlin Zhu, Jin Li, Shuo Wang and Bin Jiang \* 

School of Mechanical, Electrical and Information Engineering, Shandong University, Weihai 264209, China; zhaoyulin@mail.sdu.edu.cn (Y.Z.); 201800800586@mail.sdu.edu.cn (X.L.); zhuyinlin@mail.sdu.edu.cn (Y.Z.); huanguang@mail.sdu.edu.cn (J.L.); 201900800169@mail.sdu.edu.cn (S.W.)

\* Correspondence: jiangbin@sdu.edu.cn

**Abstract:** Recently, many models based on the combination of graph convolutional networks and deep learning have attracted extensive attention for their superior performance in graph clustering tasks. However, the existing models have the following limitations: (1) Existing models are limited by the calculation method of graph convolution, and their computational cost will increase exponentially as the graph scale grows. (2) Stacking too many convolutional layers causes the over-smoothing issue and neglects the local graph structure. (3) Expanding the range of the neighborhood and the model depth together is difficult due to the orthogonal relationship between them. Inspired by personalized pagerank and auto-encoder, we conduct the node-wise graph clustering task in the undirected simple graph as the research direction and propose a Scalable Deep Network (SDN) for graph clustering via personalized pagerank. Specifically, we utilize the combination of multi-layer perceptrons and linear propagation layer based on personalized pagerank as the backbone network (i.e., the Quasi-GNN module) and employ a DNN module for auto-encoder to learn different dimensions embeddings. After that, SDN combines the two embeddings correspondingly; then, it utilizes a dual self-supervised module to constrain the training of the embedding and clustering process. Our proposed Quasi-GNN module reduces the computational costs of traditional GNN models in a decoupled approach and solves the orthogonal relationship between the model depth and the neighborhood range. Meanwhile, it also alleviates the degraded clustering effect caused by the over-smoothing issue. We conducted experiments on five widely used graph datasets. The experimental results demonstrate that our model achieves state-of-the-art performance.

**Keywords:** graph embedding; deep clustering; auto-encoder; scalable GNN; deep learning



**Citation:** Zhao, Y.; Li, X.; Zhu, Y.; Li, J.; Wang, S.; Jiang, B. A Scalable Deep Network for Graph Clustering via Personalized PageRank. *Appl. Sci.* **2022**, *12*, 5502. <https://doi.org/10.3390/app12115502>

Academic Editor: Pasi Fränti

Received: 14 April 2022

Accepted: 27 May 2022

Published: 29 May 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Graph-structured data often contains abundant node features and topological information. Benefiting from its powerful expressive ability, graph-structured data are often used to model drug discovery [1], social networks [2], and recommender systems [3].

Moreover, the graph clustering task has attracted extensive attention as an important part of unsupervised learning on graphs. There are many directions in graph clustering tasks, such as node-wise graph clustering, and graph-wise graph clustering. In recent years, Graph Neural Networks (GNNs) have become a popular field in deep learning, which improves the performance of graph clustering tasks effectively. Graph Convolutional Networks (GCNs) [4] are one of the representative methods that can utilize node features and topology information to obtain low-dimensional embeddings. On this basis, many graph clustering models combined with deep learning techniques have been proposed to achieve state-of-the-art effects. Kipf et al. [5] learn embedding through GCN layers, then the decoder reconstructs the features as similar as possible to the original features. Ahn et al. [6] optimize the aforementioned model to solve the norm-zero tendency of isolated nodes. In addition, Pan et al. [7] combine graph convolution with the adversarial

training method. However, the above models fail to pay attention to the importance of nodes. Therefore, Veličković et al. [8] introduce the attention mechanism into the graph convolution, which can aggregate the information based on nodes' importance. Wang et al. [9] adopt the receptive field with an attention mechanism to encode the features and jointly optimize the clustering and embedding module. Although the models with the graph auto-encoder as the backbone network can generate the embeddings effectively; it ignores the information in the data structure. Combining different orders of the embeddings and the structural information, Bo et al. [10] integrate structural information into a deep clustering method for the first time to improve the clustering effect. In addition, Li et al. [11] adopts the combination of deep clustering and the graph auto-encoder to design a triple self-supervised module to supervise the embedding and clustering module.

However, the above models have the following limitations: (1) Existing models are limited by the calculation method of graph convolution, and their computational costs will increase exponentially as the scale of the graph grows. (2) Stacking too many convolutional layers will introduce the over-smoothing noises and ignore the local graph structure (3) The model depth is orthogonal to the neighborhood range, which makes it difficult to expand both.

Therefore, researchers expect to find more scalable models. Klicpera et al. [12] propose a simple model using the relation between graph convolution and pagerank, which enables an efficient neighborhood expansion. Wu et al. [13] entirely reduce the computational costs by decoupling the feature propagation from the training process. Following the idea of SGC, Frasca et al. [14] consider the features of different receptive layers and splice them without ignoring information, while Zhu et al. [15] average them to generate combined features with the same dimension. Meanwhile, Zhang et al. [16] simplify the GNN from the perspective of spectral graph theory and it can select different high-order information orders according to different graphs. Despite this, the above models ignore the difference in node importance in the aggregation process. Chen et al. [17] adopt a constant decay factor to solve this issue, while Zhang et al. [18] use the receptive field weighted aggregation with an attention mechanism to aggregate neighborhood information. However, the above methods are suitable for supervised or semi-supervised learning scenarios, lacking a task-oriented model framework for unsupervised clustering tasks.

In response to the above problems, we propose a network that can effectively utilize various types of information in a graph with high scalability. We adopt a dual self-supervision module to guide the training of the Quasi-GNN module and the DNN module. With this dual-supervised module, the entire model can be trained in an end-to-end manner for graph clustering. In addition, it should be mentioned that the algorithm of our proposed method requires a vector form of the data as input in addition to the graph.

In summary, our contributions are described as follows:

- A highly scalable deep network to process graph-structured data is proposed. This network can combine the topological information and the node features effectively to obtain potential embeddings for clustering tasks.
- A linear propagation based on personalized pagerank is proposed, which improves the performance of the clustering task and alleviates the over-smoothing issue.
- We conduct extensive experiments on five real-world datasets and achieve superior performance with fewer iterations. The experimental results show that our model outperforms the current state-of-the-art methods.

## 2. Related Work

Graph clustering divides the unlabeled nodes into different clusters with a certain metric. After that, we can mine the relationships between different nodes in a graph. The early graph clustering models perform poorly on real-world datasets due to their shallow architecture and learning capabilities, such as matrix factorization [19] and DeepWalk [20]. In addition, Sieranoja et al [21]. propose two complementary algorithms for graph clustering called K-algorithm and M-algorithm. The combination of these two algorithms can obtain

several local optimizations on the graph and they can be used with different cost functions. However, the two algorithms fail to integrate the graph topology information, which limits their final performance.

Recently, many more effective models applied to unsupervised learning are proposed, such as auto-encoder [22] and generative adversarial networks (GAN) [23]. On this basis, many graph clustering models combined with deep learning techniques have been proposed and they have achieved good performance. The Graph Auto-encoder (GAE) [5] combines the auto-encoder with graph convolution. It first utilizes the two GCN layers to capture the information between graph topology and node features and then reconstructs an adjacency matrix to be as similar to the original matrix. The ARGAN [7] adopts the adversarial training scheme to normalize the embedding process to obtain more robust embeddings. However, none of the above-mentioned models are clustering task-oriented joint optimization training methods. The DAEGC [9] combines the two components to jointly optimize the embedding module and the clustering module, which improves the quality of the embeddings and the clustering effect. Meanwhile, the SDCN [10] integrates the structural information into deep clustering by using a transfer operator to combine the auto-encoder with the GCN module. It can conduct end-to-end clustering training with the dual self-supervision module. Although these methods have superior performance, they still use the GCN module based on the message passing mechanism as the backbone network, limiting the scalability of these models.

However, the above models also have several drawbacks: (1) There is no solution to the orthogonal relationship between the model depth and the neighborhood range (2) Too many smoothing iterations or GCN layers stacking lead to over-smoothing issues.

To solve the limitations of the traditional GNN models, scalable GNN models are proposed. Early scalable GNNs simplified the model by sampling the graph, the GraphSAGE [24] samples the neighbors around the target node with the same probability, while the FastGCN [25] samples the nodes according to the importance of each node. Due to their node-wise sampling or layer-wise sampling method, they fail to learn large-scale sparse graphs effectively. On this basis, the GraphSAINT [26] proposes a subgraph-wise sampling method with high scalability, which decouples sampling from GNNs and further reduces the computational costs.

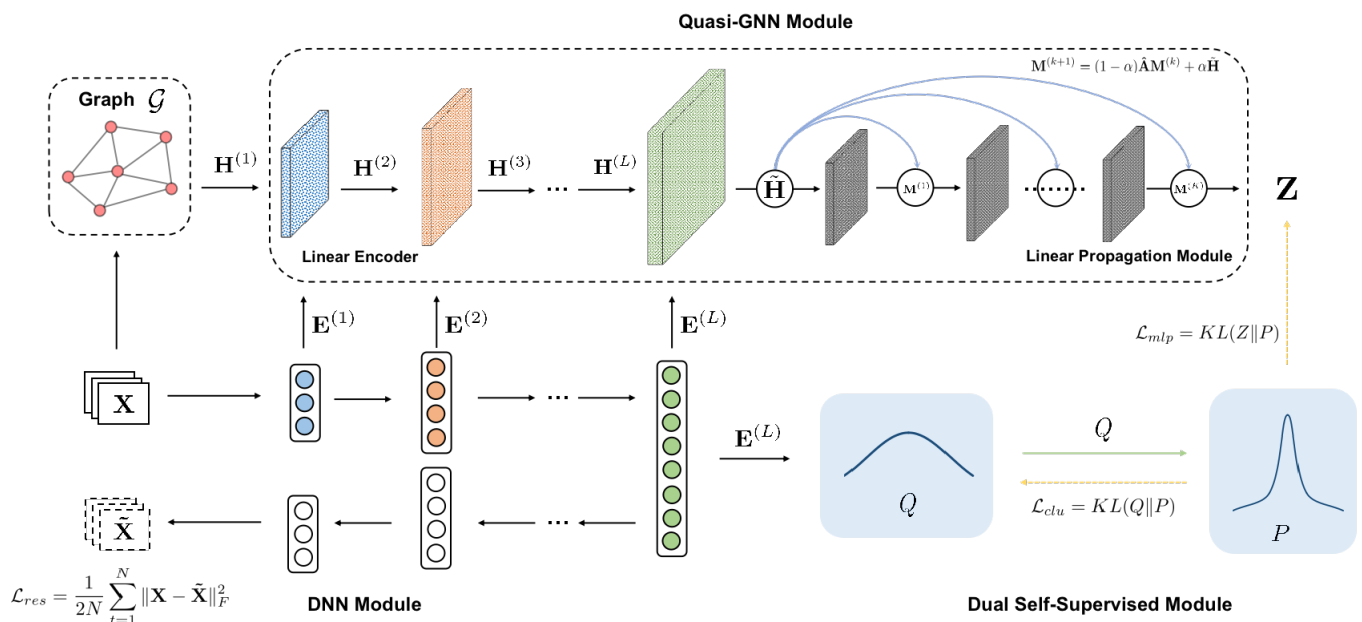
The other direction of scalable GNNs in recent years is to simplify the model structure. The SGC [13] transforms the nonlinear GCN into a simple linear model by repeatedly eliminating the nonlinear function between the GCN layers and folding the final function into a linear function. The PPNP [12] modifies the propagation scheme by adopting the relationship between GCN and pagerank. The AGC [16] advocates the use of high-order graph convolution to capture the global features of the graph and it can adaptively select the appropriate order according to different graphs, while the AGE [27] optimizes the model of GAE by decoupling the GCNs and modifies the GNN models from the perspective of graph signal processing. The S2GC [15] adopts an improved Markov diffusion kernel to derive a simpler variant of GCN that captures the global and local context of each node. Nevertheless, the SIGN [14] points out that the features among multiple layers should be considered together instead of a certain layer. They splice the features with different degrees of smoothness and utilize them for downstream tasks. Meanwhile, the GBP [17] adopts a constant weighted average decay factor to consider the difference in importance between the receptive fields of different nodes. On this basis, the GAMLP [18] integrates multi-scale node features effectively with three different attention mechanisms to improve the scalability and computational efficiency. However, the above models with high scalability lack a jointly training framework for graph clustering tasks.

In contrast, our proposed scalable deep network can not only effectively integrate the graph structure and node features, but also decouple the encoding process from the propagation process. We improve the scalability of the existing models and alleviate the over-smoothing issue. Moreover, SDN solves the issue of the orthogonal relationship between the model depth and the range of the neighborhood by the Quasi-GNN module.

Meanwhile, we utilize a dual self-supervised module to train the clustering task end-to-end, which enables high-confidence clustering results while obtaining high-quality embeddings.

### 3. The Proposed Method

In this section, we first introduce the definition of graph and clustering tasks. Then we introduce our proposed Scalable Deep Network (SDN). The overall framework of SDN is shown in Figure 1. Specifically, SDN consists of three modules, a DNN module for auto-encoder, a Quasi-GNN module, and a dual self-supervised module. We first utilize the DNN module and linear encoder to generate the intermediate embedding, then use the linear propagation module to obtain the final embeddings. Meanwhile, we utilize the dual self-supervised module to supervise the training of these two modules. We introduce the specific details of our model as follows.



**Figure 1.** The overall framework of SDN is as above.  $\mathbf{X}, \tilde{\mathbf{X}}$  are input data and reconstructed data, respectively.  $\mathbf{E}^{(l)}$  and  $\mathbf{H}^{(l)}$  are the results of the  $l$ -th layer of the linear encoder in the DNN and Quasi-GNN modules, respectively. Layers with different colors represent  $\mathbf{E}^{(l)}$  of different embeddings learned by the DNN module. The green solid line indicates that the target distribution  $P$  is calculated by the distribution  $Q$ , the yellow dotted line represents the dual self-supervision mechanism, and the target distribution  $P$  supervises the training of the DNN module and the Quasi-GNN module at the same time. The solid blue line in the linear propagation layer of the Quasi-GNN module represents the propagation mode.

#### 3.1. Problem Formalization

Graph-structured data can be defined as  $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathbf{X}\}$ , where  $\mathcal{V} = \{v_1, v_2, v_3, \dots, v_m\}$  is a vertex set with  $m$  vertices,  $\mathcal{E}$  is an edge set with  $n$  edges,  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_m\}^T$  is a feature matrix (input data). The topology of graph  $\mathcal{G}$  is described by an adjacency matrix (with self-loops)  $\tilde{\mathbf{A}}$ , where  $\tilde{\mathbf{A}} = \{a_{ij}\}$ . If there is an edge between  $v_i$  and  $v_j$ ,  $a_{ij} = 1$ , otherwise  $a_{ij} = 0$ . For non-graph data, we obtain their adjacency matrix  $\tilde{\mathbf{A}}$  by constructing a KNN graph with the Dot-product. We first calculate the similarity between different nodes by  $\mathbf{S}_{ij} = \mathbf{x}_j^T \mathbf{x}_i$ , and select  $K$  nodes with the highest similarity for each sample as their neighbors. Degree matrix  $\tilde{\mathbf{D}} = \text{diag}(d_1, d_2, d_3, \dots, d_n) \in \mathbb{R}^{n \times n}$ , where  $d_i = \sum_{v_j \in \mathcal{V}} a_{ij}$  represents the degree of any node  $v_i$ .

Graph clustering is to divide the nodes into  $t$  disjoint clusters  $\mathcal{C} = \{c_l \mid l = 1, 2, 3, \dots, t\}$  according to a selected criterion, and there is  $c_1 \cap c_2 = \emptyset$ . When the node  $v_i$  is divided into a certain cluster, it can be expressed as  $v_i \in c_l$ .

### 3.2. DNN Module for Auto-Encoder

It is not sufficient to obtain embeddings only based on node features and topology information, so we utilize auto-encoders to obtain high-dimensional representations of node features and integrate them into the embedding learning process of multi-layer perceptrons. For accommodating different data types, we adopt the most basic auto-encoder to obtain high-dimensional embeddings of nodes. First, the initial feature matrix (vector data)  $\mathbf{X}$  is fed into the fully connected neural network of the DNN module to obtain the high-dimensional embedding  $\mathbf{E}$ . The specific process and formula are defined as follows.

$$\mathbf{E}^{(l)} = \phi(\mathbf{W}_e^{(l)} \mathbf{E}^{(l-1)} + \mathbf{b}_e^{(l)}), \quad (1)$$

where  $\mathbf{E}^{(l)}$  represents the encoding result of the  $l$ -th layer, and for the 0-th layer of the network, we set  $\mathbf{E}^{(0)} = \mathbf{X}$ .  $\mathbf{W}_e^{(l-1)}$  represents the encoding weight matrix of the  $l$  layer,  $\phi$  represents the nonlinear function, such as  $ReLU(\cdot)$ . After encoding at layers  $l$ , we decode the embedding using a decoder that is fully symmetric to the encoder.

$$\mathbf{D}^{(l)} = \phi(\mathbf{W}_d^{(l)} \mathbf{D}^{(l-1)} + \mathbf{b}_d^{(l)}), \quad (2)$$

where  $\mathbf{D}^{(l)}$  represents the results of the  $l$ -th layer, for the 0-th layer of the decoding network, there is  $\mathbf{D}^{(0)} = \mathbf{E}^{(l)}$ .  $\mathbf{W}_d^{(l)}$  represents the decoding weight matrix of the  $l$  layer. After that, we set  $\tilde{\mathbf{X}} = \mathbf{D}^{(l)}$  and make the following results as the objective function.

$$\mathcal{L}_{res} = \frac{1}{2N} \sum_{i=1}^N \|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|_2^2 = \frac{1}{2N} \sum_{i=1}^N \|\mathbf{X} - \tilde{\mathbf{X}}\|_F^2. \quad (3)$$

### 3.3. Quasi-GNN Module

Although the auto-encoder can learn the embeddings from the data themselves, such as  $\mathbf{E}^{(1)}$ ,  $\mathbf{E}^{(2)}$ , and  $\mathbf{E}^{(3)}$ , it ignores the relationship between nodes. Therefore, the traditional deep clustering method needs to utilize the GCN module to capture the relationship between nodes as the supplements. The GCN module can solve this issue, but it is difficult to expand the model depth and the range of the neighborhood together, which limits the learning ability and architecture of the models. Therefore, we propose a Quasi-GNN module, which decouples the encoding process from the propagation process and it can not only capture the relationship between nodes, but also expand the range of the neighborhood and the depth of the model together, reducing the computational cost and improving the scalability.

#### 3.3.1. Linear Encoder

We utilize the multilayer perceptron (MLP) as our encoder to get the embeddings. The result of each layer can be defined as

$$\mathbf{H}^{(l)} = f(\mathbf{H}^{(l-1)}, \mathbf{W}_m^{(l-1)}) = \mathbf{H}^{(l-1)} \mathbf{W}_m^{(l-1)}, \quad (4)$$

where  $\mathbf{H}^{(l)}$  is the embeddings of the  $l$ -th layer. Specially,  $\mathbf{H}^{(0)} = \mathbf{X}$ ,  $\mathbf{W}_m$  is the weight matrix of MLP. To obtain a more complete and powerful embedding, we combine the high-dimensional representations  $\mathbf{E}^{(l)}$  learned from the DNN module with  $\mathbf{H}^{(l)}$ . The formula is as follows.

$$\tilde{\mathbf{H}}^{(l)} = (1 - \sigma) \mathbf{H}^{(l)} + \sigma \mathbf{E}^{(l)}, \quad (5)$$

$\mathbf{E}^{(l)}$  is the calculation result of the  $l$ -th layer DNN module.  $\sigma$  is the balance coefficient and we set it to 0.5. After that, we need to do the propagation operation on it to aggregate

information in the neighborhood. Then we utilize the  $\tilde{\mathbf{H}}^{(l)}$  as the input of the  $l$ -th layer in MLP to generate the embeddings

$$\mathbf{H}^{(l)} = f(\tilde{\mathbf{H}}^{(l-1)}, \mathbf{W}_m^{(l-1)}) = \tilde{\mathbf{H}}^{(l-1)} \mathbf{W}_m^{(l-1)}. \tag{6}$$

### 3.3.2. Linear Propagation Module

We first briefly review the message passing algorithm of traditional GCNs. A traditional two-layer GCN model can be defined as

$$\mathbf{Z}_{GCN} = \text{Softmax}(\hat{\mathbf{A}} \text{ReLU}(\hat{\mathbf{A}} \mathbf{X} \mathbf{W}_0) \mathbf{W}_1), \tag{7}$$

where  $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{r-1} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-r}$ ,  $\hat{\mathbf{A}}$  is the normalized adjacency matrix, by setting  $r = 1$  or  $0.5$ , we can obtain different regularization methods, such as  $\tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1}$ ,  $\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}$ , and  $\tilde{\mathbf{D}}^{-1} \tilde{\mathbf{A}}$ . The predicted labels is  $\mathbf{Z}_{GCN}$ . In a traditional two-layer GCN model, the calculation of each layer depends on the calculation result of the previous layer. Limited by this calculation method, the computational costs of the traditional GNN models increase exponentially. It is difficult to expand the model depth and the neighborhood range together for their orthogonal relationship. According to Xu et al. [28], the influence score of sample  $x$  on  $y$  in GNN can be defined as

$$I(x, y) = \sum_i \sum_j \frac{\partial \mathbf{Z}_{yi}}{\partial \mathbf{X}_{xj}}. \tag{8}$$

In the  $k$ -layers GNN,  $I(x, y) \propto P_{rw'}(x \rightarrow y, k)$ , where  $P_{rw'}(x \rightarrow y, k)$  is the random walk distribution after fine-tuning. When  $k \rightarrow \infty$ , if the graph is irreducible and aperiodic, the value will approach a stable distribution independent of  $x$  (i.e., the same amount of influence scaling), which indicates that the influence of the  $x$  on the  $y$  at this time will eventually be independent of the local graph structure. Assuming that this stable distribution is  $\pi_{lim}$ , we can calculate the distribution by the following formula

$$\pi_{lim} = \hat{\mathbf{A}} \pi_{lim}. \tag{9}$$

Obviously, the result is related to the structure of the whole graph and has no relation to the starting point of the random walk, which means that we finally consider the information of the whole graph and ignore the nodes themselves. In addition, the original pagerank also adopts this calculation method to obtain the full graph structure.

$$\pi_{pr} = \mathbf{A}_{rw} \pi_{pr} \quad \mathbf{A}_{rw} = \mathbf{A} \mathbf{D}^{-1}. \tag{10}$$

Based on this, we can adopt a variant of pagerank (i.e., personalized pagerank) to reconsider the root node. Assuming that  $\mathbf{i}_x$  is the indicator vector of node  $x$ , its vector representation after multiple propagations can be defined as

$$\pi_{ppr}(\mathbf{i}_x) = (1 - \alpha) \hat{\mathbf{A}} \pi_{ppr}(\mathbf{i}_x) + \alpha \mathbf{i}_x, \tag{11}$$

where  $\alpha$  is the transmission probability,  $\alpha \in [0, 1]$ ,  $\hat{\mathbf{A}}$  is the normalized adjacency matrix. In this way, we can obtain an approximate post-propagation matrix with respect to the entire graph data

$$\mathbf{M}^{(0)} = \mathbf{T} = g(\mathbf{X}), \tag{12}$$

$$\mathbf{M}^{(k+1)} = (1 - \alpha) \hat{\mathbf{A}} \mathbf{M}^{(k)} + \alpha \mathbf{T}, \tag{13}$$

where  $\mathbf{M}^{(k)}$  is the result of the  $k$ -th propagation,  $\mathbf{T}$  is the embedding obtained by the linear encoder. Therefore, we can deduce the final embeddings  $\mathbf{Z}$  by combining the intermediate embeddings  $\mathbf{H}$  in the Section 3.3.1.

$$\mathbf{M}^{(0)} = \tilde{\mathbf{H}} = \mathbf{H}^{(L)}, \tag{14}$$



$$\mathbf{M}^{(k+1)} = (1 - \alpha)\hat{\mathbf{A}}\mathbf{M}^{(k)} + \alpha\tilde{\mathbf{H}}. \tag{15}$$

The last layer of the linear propagation module is the multi-classification function of the softmax function

$$\mathbf{Z} = \text{Softmax}(\mathbf{M}^{(K)}) = \text{Softmax}((1 - \alpha)\hat{\mathbf{A}}\mathbf{M}^{(K)} + \alpha\tilde{\mathbf{H}}). \tag{16}$$

As a result,  $z_{ij} \in \mathbf{Z}$  indicates the probability that a node  $v_i$  belongs to the cluster  $j$ . Moreover, we can consider  $\mathbf{Z}$  as a kind of aggregation class distribution.

### 3.4. Dual Self-Supervised Module

Through the above two modules, we mechanically combine the DNN module and the Quasi-GNN module, they essentially are all used for unsupervised or supervised learning in different scenarios and we cannot apply them to our depth clustering task directly. Therefore, we need to unify the Quasi-GNN module and the DNN module with the same optimization objective. We set the goal of these modules to approximate the target distribution  $P$ , which makes the results tend to be consistent during the training process, and because of the strong connection between the two modules, we call it a dual self-Supervised module. This module does not require the participation of labels during the training process.

First, for the DNN module, we utilize Student's t-distribution as the kernel to measure the similarity between the node embeddings  $\mathbf{e}_i$  and the cluster center vector  $\mu_j$ :

$$q_{ij} = \frac{(1 + \|\mathbf{e}_i - \mu_j\|^2/v)^{-\frac{v+1}{2}}}{\sum_{j'} (1 + \|\mathbf{e}_i - \mu_{j'}\|^2/v)^{-\frac{v+1}{2}}}, \tag{17}$$

where  $\mathbf{e}_i$  is the  $i$ -th row of the embedding  $\mathbf{E}^{(l)}$ ,  $\mu_j$  is initialized by the K-means learned by the pre-train auto-encoder,  $v$  is the degree of freedom of Student's t-distribution.  $q_{ij}$  can be seen as the probability of assigning sample  $i$  to cluster  $j$ . From this, we can obtain the cluster distribution  $\mathbf{Q}$  about the nodes. To enable nodes to be assigned to different clusters with higher confidence, we calculate the target distribution  $\mathbf{P}$ .

$$p_{ij} = \frac{q_{ij}^2 / f_{ij}}{\sum_{j'} q_{ij}^2 / f_{ij'}}, \tag{18}$$

where  $f_{ij} = \sum_i q_{ij}$  is the soft clustering frequency. The target distribution  $\mathbf{P}$  normalizes the sum of squares of each distribution in the cluster distribution  $\mathbf{Q}$ . By using two distributions to constrain different embeddings, the embedding obtained by the DNN module and the Quasi-GNN module can be considered simultaneously to optimize the embedding and clustering quality jointly. On this basis, we can obtain the corresponding objective function

$$\mathcal{L}_{clu} = KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{19}$$

This objective function constrains the DNN module and we can obtain the superior embeddings for clustering by reducing the KL divergence loss of the two distributions  $\mathbf{Q}$  and  $\mathbf{P}$ . In addition, we need to utilize the  $\mathbf{P}$  distribution to constrain the Quasi-GNN module

$$\mathcal{L}_{mlp} = KL(P\|Z) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{z_{ij}}. \tag{20}$$

To sum up, the final loss can be defined as

$$\mathcal{L} = \mathcal{L}_{res} + \beta \mathcal{L}_{clu} + \gamma \mathcal{L}_{mlp}, \tag{21}$$

where  $\beta$  and  $\gamma$  are constraint coefficients,  $\beta \in [0, 1]$  and  $\gamma \in [0, 1]$ . Algorithm 1 shows the training process of our proposed model.

---

**Algorithm 1** Training process of SDN.

---

**Require:** Initial features:  $\mathbf{X}$ , Graph:  $\mathcal{G}$ , Numbers of clusters:  $K$ , Adjacency matrix:  $\mathbf{A}$ , Iteration number:  $MaxIter$ , Layer number of linear encoder:  $L_E$ , Layer number of linear propagation module:  $L_P$ ;

**Ensure:** Clustering results  $\mathbf{R}$ ;

- 1: Initialize  $\mathbf{W}_e^{(l)}, \mathbf{W}_d^{(l)}, \mathbf{b}_e^{(l)}, \mathbf{b}_d^{(l)}$  with pre-train auto-encoder;
  - 2: Initialize  $\mu$  with K-means on the representations learned by pre-train auto-encoder;
  - 3: Initialize  $\mathbf{W}_m^{(l)}$  randomly;
  - 4: **for**  $ite = 1$  to  $MaxIter$  **do**
  - 5:   Generate DNN embeddings  $\mathbf{E}^{(0)}, \mathbf{E}^{(1)}, \mathbf{E}^{(2)}, \dots, \mathbf{E}^{(L)}$ ;
  - 6:   Use  $\mathbf{E}^{(L)}$  to calculate the distribution  $Q$  by Equation (17);
  - 7:   Calculate target distribution  $P$  by Equation (18);
  - 8:   **for**  $l = 1$  to  $L_E$  **do**
  - 9:     Set the balance coefficient  $\sigma = 0.5$  to calculate  $\tilde{\mathbf{H}}^{(l)}$  by Equation (5);
  - 10:     Calculate the embeddings of the next layer of MLP by Equation (6);
  - 11:   **end for**
  - 12:   Set  $\tilde{\mathbf{H}} = \mathbf{H}^{(L)}$
  - 13:   **for**  $l = 1$  to  $L_P$  **do**
  - 14:     Calculate the embeddings by Equation (15);
  - 15:   **end for**
  - 16:   Set the transmission probability  $\alpha = 0.3$  to calculate the distribution  $Z$  by Equation (16);
  - 17:   Feed  $\mathbf{H}^{(L)}$  into the decoder to obtain the refactored feature  $\tilde{\mathbf{X}}$ ;
  - 18:   Calculate  $\mathcal{L}_{res}, \mathcal{L}_{clu}, \mathcal{L}_{mlp}$ , respectively;
  - 19:   Calculate the whole loss function  $\mathcal{L}$  by Equation (21);
  - 20:   Back propagation and update parameters in SDN;
  - 21: **end for**
  - 22: Calculate the clustering results based on distribution  $Z$ ;
  - 23: **return**  $\mathbf{R}$ ;
- 

## 4. Experiments

### 4.1. Datasets

To evaluate the performance of our model, we conduct extensive experiments on five public benchmark datasets, the specific details of them are shown in Table 1.

- **USPS** [29]: The USPS is a digit dataset automatically scanned from envelopes by the U.S. Postal Service containing a total of 9298  $16 \times 16$ -pixel grayscale samples; the images are centered, normalized, and show a broad range of font styles.
- **HHAR** [30]: The Heterogeneous Human Activity Recognition (HHAR) dataset contains 10,299 sensor records from different smart terminals. All samples are divided into 6 types of human activities, including riding, lying, sitting, standing, walking, and climbing stairs and down the stairs.
- **Reuters** [31]: Reuters is a simple, widely used dataset for text classification. It includes 46 different subjects: some subjects have more samples, but each subject in the training set has at least 10 samples.
- **ACM**: The ACM dataset is a paper network from ACM digital library. It contains papers published in KDD, SIGMOD, SIGCOMM, MobiCOMM, and VLDB, which can be divided into three categories (databases, wireless communication, data mining).
- **CiteSeer**: The CiteSeer is a citation network. Papers in this dataset are divided into Agents, AI (Artificial Intelligence), DB (Database), IR (Information Retrieval), ML



(Machine Learning), and HCI, containing a total of 3312 papers. It records information between cited papers or citations.

- **DBLP:** The DBLP dataset is an author network. If two authors are collaborators, then there is an edge connection between them. We label their research fields according to their papers published in international journals and conferences.
- **Flickr:** The Flickr is an image network which is constructed by forming links between shared Flickr public images. Edges are formed between pictures from the same location, pictures submitted to the same gallery, group, or collection, pictures that share a common tag, pictures taken by friends, etc.

**Table 1.** The statistics of the datasets.

Dataset	Type	Samples	Classes	Dimension	Description
USPS	Image	9298	10	256	KNN network
HHAR	Record	10,299	6	561	KNN network
Reuters	Text	10,000	4	2000	KNN network
ACM	Graph	3025	3	1870	Citation network
CiteSeer	Graph	3327	6	3703	Citation network
DBLP	Graph	4058	4	334	Author network
Flickr	Graph	89,250	7	500	Image network

#### 4.2. Methods

We compare SDN with various existing representative unsupervised models for clustering tasks. Moreover, these models can be divided into three categories according to different input data, models that only use feature matrix (vector data): K-means, AE, and Random Swap; models that only use adjacency matrix (graph data): K-algorithm and M-algorithm; models that use both the two data: DEC, IDEC, GAE, VGAE, DAEGC, ARG, SDCN, AGCN, SDN<sub>P</sub>, SDN<sub>E</sub>, and SDN. The following are specific descriptions of these models.

- **K-means** [32]: It is a traditional clustering method applied directly to the feature matrix (vector data). In this paper, we utilize the K-means supported by the sklearn package. For details, please refer to <https://github.com/scikit-learn/scikit-learn>, accessed on 12 April 2022.
- **AE** [22]: This auto-encoder consists of an encoder and a decoder. It uses the encoder to encode the initial data, then utilizes the decoder to reconstruct the embeddings. In addition, it calculates the reconstruction loss as the objective function. Finally, we employ K-means to perform clustering on the obtained high-dimensional embeddings.
- **Random Swap** [33]: Random swap algorithm aims at solving clustering by a sequence of prototype swaps and by fine-tuning their exact location by k-means. We utilize the Random Swap file of the python version in the UEF Machine Learning repository. For details, please refer to <https://github.com/uef-machine-learning/RandomSwap>, accessed on 12 April 2022.
- **K-algorithm** [21]: K-algorithm applies similar iterative local optimization but without the need to calculate the means. It inherits the properties of k-means clustering in terms of both good local optimization capability and the tendency to get stuck at a local optimum.
- **M-algorithm** [21]: M-algorithm gradually improves on the results of the K-algorithm to find new and potentially better local optima. It repeatedly merges and splits random clusters and tunes the results with the K-algorithm.
- **DEC** [34]: DEC is a deep clustering method that defines a centroid-based probability distribution and minimizes the KL divergence as an auxiliary objective distribution to improve both cluster assignment and feature representation. It implements the joint optimization of deep embedding and clustering.

- **IDEC [35]**: Taking into account the preserved data structure, IDEC manipulates the feature space to disperse the data points. Moreover, it can jointly perform the embedding and the clustering process.
- **GAE [5]**: This method is an effective combination of auto-encoder graph convolution. First, graph convolution is used to encode the data; then, the decoder is used to reconstruct its adjacency matrix. The loss function measures the difference between the reconstructed matrix and the original matrix.
- **VGAE [5]**: This model first obtains the embeddings through GCNs, then learns the distribution satisfied by them. Finally, it calculates the posterior probability to obtain the latent variable to reconstruct the adjacency matrix.
- **DAEGC [9]**: It adopts the attention network to learn node embeddings and employs a clustering loss to supervise the self-training clustering process.
- **AGRA [7]**: Using the adversarial regularization to normalize the process of encoding, ARGAs combines an adversarial training scheme with a graph auto-encoder to obtain the superior embeddings.
- **SDCN [10]**: To obtain the more robust embeddings, SDCN fuses the calculation results of the GCN module and the DNN module. Moreover, it utilizes a dual self-supervised module to constrain the two modules to train the model end-to-end.
- **AGCN [36]**: Considering the nodes' importance, AGCN employs the attention mechanism to merge the embeddings learned by the same layer of auto-encoder and GCNs.
- **SDN<sub>P</sub>**: It is a variant of SDN, which only employs the Quasi-GNN module.
- **SDN<sub>E</sub>**: It only utilizes the DNN module for encoding as a variant of SDN.
- **SDN**: The proposed method.

#### 4.3. Evaluation Metrics and Experimental Setup

##### 4.3.1. Evaluation Metrics

We adopt four widely used evaluation metrics: Accuracy (**ACC**), Normalized Mutual Information (**NMI**), Average Rand Index (**ARI**), and macro-F1 score (**F1**) [37]. For each metric, a larger value implies a better clustering result.

The specific calculation methods of the four indicators are as follows

##### Accuracy (ACC)

**ACC** is used to compare the obtained labels with the true labels, which can be calculated by the formula below

$$ACC = \frac{\sum_{i=1}^m \delta(s_i, \text{map}(r_i))}{n}, \quad (22)$$

where  $r_i, s_i$  represent the obtained label and true label corresponding to the data  $x_i$ , respectively,  $n$  is the total number of data,  $\delta$  indicates that the indicator function is as follows

$$\delta(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}. \quad (23)$$

The map in this formula represents the re-distribution of the best class label to ensure the correctness of the statistics.

##### Normalized Mutual Information (NMI)

**NMI** is often used in clustering to measure the similarity of two clustering results. Assuming that  $P_A(a), P_B(b)$  represent the probability distribution of  $A$  and  $B$ , and  $P_{AB}(a, b)$  represents the joint distribution probability of  $A$  and  $B$ , then we have

$$H(A) = - \sum_a P_A(a) \log P_A(a), \quad (24)$$

$$H(B) = - \sum_b P_B(b) \log P_B(b), \quad (25)$$

$$H(A, B) = - \sum_{a,b} P_{AB}(a, b) \log P_{AB}(a, b), \quad (26)$$

where  $H(A)$  is called the information entropy of  $A$  vector. According to the relationship between joint entropy and individual entropy, **NMI** is defined as

$$NMI = \frac{H(A) + H(B)}{H(A, B)}. \quad (27)$$

### Adjusted Rand index (ARI)

**ARI** reflects the degree of overlap between the two divisions. Suppose clustering is a series of decision-making processes, that is, making decisions on all  $N(N - 1)$  node pairs on the set. When only two nodes are similar, we group them into the same cluster. We utilize  $a$  to group two similar nodes into one cluster and  $b$  to group dissimilar nodes into different clusters. The Rand coefficient (RI) can be defined as

$$RI = \frac{a + b}{C_n^2}. \quad (28)$$

However, RI fails to guarantee that the RI value of randomly divided clustering results is close to 0. Therefore, the Adjusted Rand index (RI) is proposed.

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}, \quad (29)$$

where  $E[RI]$  represents the expectation of RI,  $ARI \in [-1, 1]$ .

### Macro-F1 Score (F1)

The **F1 score** measures the accuracy of a binary classification (or multi-task binary classification) model. It takes into account both the accuracy and recall of the classification model. F1 score can be regarded as a weighted average of model precision and recall, and  $F1 \in [0, 1]$ .

According to Table 2, *precision* refers to the proportion of samples with a predicted value of 1 and a true value of 1 in all samples with a predicted value of 1. In addition, *recall* refers to the proportion of samples with a predicted value of 1 and a true value of 1 among all samples with a true value of 1. Therefore, *precision* and *recall* can be defined as

$$precision = \frac{TP}{TP + FP}, \quad (30)$$

$$recall = \frac{TP}{TP + FN}, \quad (31)$$

On this basis, **F1 score** is defined as the harmonic mean of precision and recall

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} \quad (32)$$

For macro-F1, it is the average of the **F1 score** of each cluster in the set.

$$macroF_1 = \frac{1}{N} \sum_{i=1}^N F_1. \quad (33)$$

**Table 2.** Confusion matrix. For binary classification issues, the rows of the matrix represent the true values and the columns of the matrix represent the predicted values. **TP (True Positive)** means the number of positive samples predicted as positive samples, **FN (False Negative)** means the number of positive samples predicted as negative samples, **FP (False Positive)** means the number of negative samples predicted as positive samples, **TN (True Negative)** means the number of negative samples predicted as negative samples.

	Positive	Negative
Positive	TP	FN
Negative	FP	TN

#### 4.3.2. Experimental Setup

To ensure the consistency of the experiments, we utilize a unified pre-train auto-encoder to train the benchmark models involving the DNN module, such as **AE+K-means**, **DEC**, **IDEC**, **SDCN**, and **AGCN**. The structure of the pre-train auto-encoder is a 4-layer encoder and a 4-layer decoder with the dimension of 500–2000–500–10, and the two components are completely symmetrical to ensure the consistency of the constructed features. Meanwhile, we adopt the learning rate of  $10^{-3}$  and 30 epochs to train the auto-encoder and restore the optimal training results. In the subsequent training, we first employ the pre-train auto-encoder to encode the data, then we perform the K-means and initialize our clustering layer with the obtained clustering results. During the training, different learning rates and epochs are used for different datasets. Table 3 shows the detailed settings for training the Quasi-GNN module in different datasets. For the  $\beta$  and  $\gamma$  in the loss function, we set them as  $\beta = 10^{-1}$  and  $\gamma = 10^{-2}$ , respectively, in the experiment. Also, we set  $\alpha$  in the linear propagation layer to 0.3 and the degrees of freedom of the Student's t-distribution to 1.

For the application of Random Swap on each dataset, we set the number of iterations to 10 and perform K-means twice for each iteration, the rest of the settings are the default settings; please refer to <https://github.com/uef-machine-learning/RandomSwap>, accessed on 12 April 2022. Moreover, by applying the K-algorithm and M-algorithm, we need to reconstruct the data according to the corresponding input data format. Therefore, we obtain the corresponding neighbor nodes according to their adjacency matrix and measure their similarity according to the nodes' features like the weight of the edge. On this basis, when using the K-algorithm and the M-algorithm, we calculated the conductance as the cost function; in particular, for the M-algorithm, we set the number of iterations to 100. Other parameters are default; please refer to <https://github.com/uef-machine-learning/gclu>, accessed on 12 April 2022.

On the other hand, K-means, Random Swap, and AE perform graph clustering directly on the feature matrix (vector data), K-algorithm and M-algorithm perform graph clustering on the data after reconstructing the input format, and other methods based on graph neural networks utilize a combination of feature matrix (vector data) and adjacency matrix (graph data).

**Table 3.** Parameter settings used for each training set when training the Quasi-GNN module. K represents constructing a K-nearest neighbor graph for non-graph data, and if the value is none, it means that the original data is graph data.

Dataset	Learning Rate	Epochs	K
USPS	$10^{-3}$	200	3
HHAR	$10^{-3}$	550	5
Reuters	$10^{-4}$	120	5
ACM	$10^{-3}$	120	None
CiteSeer	$10^{-3}$	120	None
DBLP	$10^{-3}$	120	None

#### 4.4. Scalability Analysis

To further illustrate the scalability, we analyze our proposed model in time and space complexity. Moreover, we present time and memory consumption in different baselines.

##### 4.4.1. Complexity Analysis

In this paper, we assume that the dimension of the input data is  $d$  and the dimensions of each layer of the pre-train auto-encoder are  $d_1, d_2, d_3, \dots, d_L$ . Assuming the number of input data is  $N$ , the time complexity of the pre-train auto-encoder is  $\mathcal{O}(Nd^2d_1^2d_2^2 \dots d_L^2)$ . For the linear encoder in the Quasi-GNN module, the dimension used in this part must be the same as the pre-train auto-encoder, so the time complexity of the linear encoder is  $\mathcal{O}(Nd^2d_1^2d_2^2 \dots d_L^2)$ . In addition, the linear propagation module in the Quasi-GNN module requires an adjacency matrix to participate in the operation instead of parameters, so the time complexity of this part is related to the output dimension of the linear encoder and the number of nodes. Therefore, the time complexity is  $\mathcal{O}(NLpd_L^2|\mathcal{V}|^2)$ . Moreover, we suppose that there are  $K$  classes in the clustering task, and the time complexity of Equation (17) is  $\mathcal{O}(NK + N\log N)$  according to the analysis of Xie et al. [34]. In summary, the total time complexity of our proposed model is  $\mathcal{O}(Nd^2d_1^2d_2^2 \dots d_L^2 + NLpd_L^2|\mathcal{V}|^2 + NK + N\log N)$ .

Next, we analyze the space complexity of our proposed model. For neural networks, the space complexity is represented by the number of neural network layers and the number of parameters. The parameters needed in our model appear in the DNN module and the linear encoder in the Quasi-GNN module. For the encoder of the pre-train auto-encoder and the linear encoder in the Quasi-GNN module, to combine the embeddings of these two components, the dimensions of their weight matrix should correspond to each other. In addition, the decoder and the encoder are completely symmetrical. Therefore, the weight matrix size of these three components should be the same. The space complexity of  $\mathbf{W}_e, \mathbf{W}_d$ , and  $\mathbf{W}_m$  is  $\mathcal{O}(dd_1 + d_1 + d_1d_2 + d_2 + \dots + d_{L-1}d_L + d_L)$ . To sum up, the space complexity of SDN should be  $\mathcal{O}(dd_1 + d_1 + d_1d_2 + d_2 + \dots + d_{L-1}d_L + d_L)$ .

##### 4.4.2. Time and Memory Consumption Comparison

On the one hand, to fully demonstrate the superiority of SDN in terms of memory consumption, we conduct experiments on Flickr and compare the SDN with baselines that have the state-of-the-art (SOTA) performance, such as AGCN and SDCN. The statistics of the Flickr are shown in Table 1. The results of the comparative experiments, the total number of parameters, and memory consumption of AGCN, SDCN, and SDN are shown in Table 4.

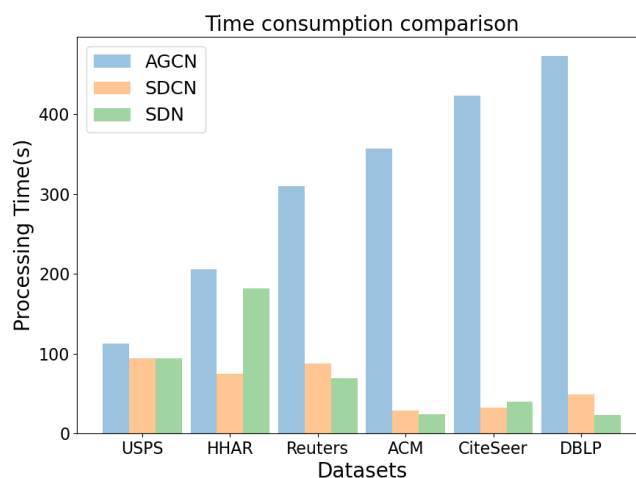
**Table 4.** Accuracy of graph clustering on Flickr. “OOM” means “out of memory”.

	Number of Parameters	Memory Consumption	Flickr
AGCN	4,614,831	Exceed 11G	OOM
SDCN	4,566,650	Exceed 11G	OOM
SDN	4,569,667	9.959G	43.60

On the other hand, to show the superiority of SDN in terms of time consumption, we record the time consumed by AGCN, SDCN, and SDN when processing the same dataset. The specific results are shown in Figure 2.

First, according to the results in Figure 2, the time consumption of AGCN and SDCN is mostly higher than that of SDN. In summary, the two-part experimental results show that existing SOTA methods have larger memory consumption and longer processing time than SDN. Second, for the experimental results in Table 4, AGCN and SDCN cannot be applied to large-scale datasets, such as Flickr, mainly because they adopt GNN and its variant methods as the result of the backbone network, while the computational costs of GNN recursively increase with the deepening of the network layer, which makes it difficult

for this type of model to handle large-scale graph data, and cannot effectively expand the neighborhood range to obtain better node embedding and clustering results.



**Figure 2.** Time consumption comparison chart of AGCN, SDCN, and SDN on six datasets. The experiments are conducted on a machine with Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz, and a single NVIDIA GeForce RTX 2080 Ti with 11GB memory. The operating system of the machine is Ubuntu 18.04.5 LTS. As for software versions, we use Python 3.9.12, Pytorch 1.11.0 and CUDA 11.4.

In contrast, SDN utilizes linear encoders as the backbone network and linear propagation layers for feature propagation. Our proposed model not only effectively reduces the computational costs when processing the large-scale graph-structured data but also solves the orthogonal issue of the neighborhood range and the model depth, which shows the high scalability of SDN.

## 5. Result Analysis

We compare SDN with representative benchmark models and conduct extensive experiments on five datasets, including **HHAR**, **Reuters**, **ACM**, **CiteSeer**, and **DBLP**. The benchmark models completely adopt the original parameter settings. Moreover, the specific experimental results for different metrics are shown in Tables 5–8, where the **bold** values represent the best performance, the underlined values indicate the second-best performance. Our model surpasses recent benchmark models and achieves SOTA results. Compared with SDCN, our module has the following advantages:

- We decouple the GCN module by employing the Quasi-GNN module to capture the information of graph topology and node features, and this module can be combined with methods such as smoothing or label propagation, which makes our model have high scalability.
- We solve the issue of the orthogonal relationship between the model depth and the range of neighborhood, which enables the two to scale together and reduces the computational costs.
- We simplify the model's structure and add the structural information to the Quasi-GNN module to alleviate the over-smoothing issue.



**Table 5.** Accuracy (ACC) results on six datasets.

	USPS	HHAR	Reuters	ACM	CiteSeer	DBLP	References
K-means	66.82	59.98	54.04	67.31	39.32	38.65	[32]
AE	71.04	68.69	74.90	81.83	57.08	51.43	[22]
Random swap	63.47	59.21	58.92	59.70	38.62	38.75	[33]
K-algorithm	56.27	45.06	44.24	37.82	32.40	32.44	[21]
M-algorithm	52.98	42.31	44.25	38.08	21.34	29.60	[21]
DEC	73.31	69.39	73.58	84.33	55.89	58.16	[34]
IDEC	76.22	71.05	75.43	85.12	60.49	60.31	[35]
GAE	63.10	62.33	54.40	84.52	61.35	61.21	[5]
VGAE	56.19	71.30	60.85	84.13	60.97	58.59	[5]
DAEGC	73.55	76.51	65.50	86.94	64.54	62.05	[9]
ARGA	66.80	63.30	56.20	86.10	56.90	61.60	[7]
SDCN	77.89	84.26	77.15	90.45	65.96	68.05	[10]
AGCN	<b>80.98</b>	<b>88.11</b>	79.30	90.59	68.79	<b>73.26</b>	[36]
<b>SDN<sub>P</sub></b>	77.66	78.61	<u>81.07</u>	<u>91.07</u>	<u>69.52</u>	60.56	Proposed
<b>SDN<sub>E</sub></b>	70.96	82.11	79.47	87.44	60.14	65.98	Proposed
<b>SDN</b>	<u>78.04</u>	<b>89.50</b>	<b>81.15</b>	<b>91.34</b>	<b>70.78</b>	<b>74.93</b>	Proposed

**Table 6.** Normalized Mutual Information (NMI) results on six datasets.

	USPS	HHAR	Reuters	ACM	CiteSeer	DBLP
K-means	62.63	58.86	41.54	32.44	16.94	11.45
AE	67.53	71.42	49.69	49.30	27.64	25.40
Random swap	60.51	58.86	29.06	16.40	17.26	11.24
K-algorithm	50.70	31.57	11.88	0.88	7.69	7.69
M-algorithm	57.54	50.60	4.03	1.27	0.71	0.38
DEC	70.58	72.91	47.50	54.54	28.34	29.51
IDEC	75.56	74.19	50.28	56.61	27.17	31.17
GAE	60.69	55.06	25.92	55.38	34.63	30.80
VGAE	51.08	62.95	25.51	53.20	32.69	26.92
DAEGC	71.12	69.10	30.55	56.18	36.41	32.49
ARGA	61.60	57.10	28.70	55.70	34.50	26.80
SDCN	<u>79.51</u>	79.90	50.82	68.31	38.71	39.50
AGCN	<b>79.64</b>	82.44	57.83	68.38	41.54	<u>39.68</u>
<b>SDN<sub>P</sub></b>	78.79	<u>82.31</u>	<b>59.52</b>	<u>69.70</u>	<u>41.95</u>	27.86
<b>SDN<sub>E</sub></b>	67.51	<u>78.96</u>	56.35	59.78	32.84	31.07
<b>SDN</b>	<b>79.64</b>	<b>83.02</b>	<u>59.49</u>	<b>70.35</b>	<b>44.28</b>	<b>41.84</b>

**Table 7.** Average Rand Index (ARI) results on six datasets.

	USPS	HHAR	Reuters	ACM	CiteSeer	DBLP
K-means	54.55	46.09	27.95	30.60	13.43	6.97
AE	58.83	60.36	49.55	54.64	29.31	12.21
Random swap	50.80	45.21	23.74	17.73	15.11	4.23
K-algorithm	35.71	24.46	11.17	0.76	7.97	7.97
M-algorithm	30.76	29.98	1.22	1.27	0.02	0.05
DEC	63.70	61.25	48.44	60.64	28.12	23.92
IDEC	67.86	62.83	51.26	62.16	25.70	25.37
GAE	50.30	42.63	19.61	59.46	33.55	22.02
VGAE	40.96	51.47	26.18	57.72	33.13	17.92
DAEGC	63.33	60.38	31.12	59.35	37.78	21.03
ARGA	51.10	44.70	24.50	62.90	33.40	22.70
SDCN	71.84	72.84	55.36	73.91	40.17	39.15
AGCN	<b>73.61</b>	<u>77.07</u>	60.55	74.20	43.79	<u>42.49</u>
<b>SDN<sub>P</sub></b>	71.26	72.78	<u>62.05</u>	<u>75.39</u>	<u>44.07</u>	27.71
<b>SDN<sub>E</sub></b>	58.74	71.33	58.07	66.39	33.42	30.45
<b>SDN</b>	<u>71.94</u>	<b>79.01</b>	<b>62.26</b>	<b>76.03</b>	<b>45.90</b>	<b>45.64</b>

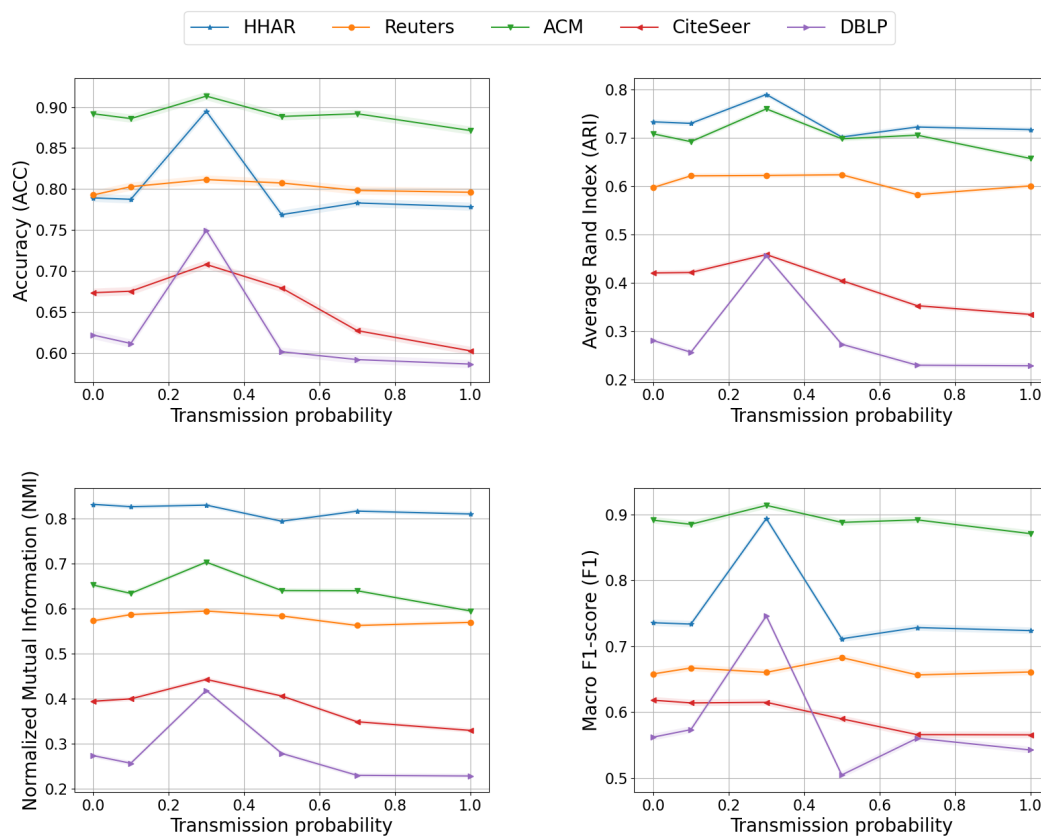
**Table 8.** Macro-F1 score (F1) results on six datasets.

	USPS	HHAR	Reuters	ACM	CiteSeer	DBLP
K-means	64.78	58.33	41.28	67.57	36.08	31.92
AE	69.74	66.36	60.96	82.01	53.80	52.53
Random swap	60.78	58.22	46.34	58.68	29.98	34.88
K-algorithm	57.71	44.69	37.76	36.93	29.51	29.51
M-algorithm	51.88	28.24	21.66	30.26	6.55	11.69
DEC	71.82	67.29	64.25	84.51	52.62	59.38
IDEC	74.63	68.63	63.21	85.11	61.62	61.33
GAE	61.84	62.64	43.53	84.65	57.36	61.41
VGAE	53.63	71.55	57.14	84.17	57.70	58.69
DAEGC	72.45	76.89	61.82	87.07	62.20	61.75
ARGA	66.10	61.10	51.10	86.10	54.80	61.80
SDCN	76.98	82.58	65.48	90.42	<b>63.62</b>	67.71
AGCN	<b>77.61</b>	<u>88.00</u>	<b>66.16</b>	90.58	<u>62.37</u>	<u>72.80</u>
<b>SDN<sub>P</sub></b>	76.60	73.21	65.38	<u>91.08</u>	62.16	51.28
<b>SDN<sub>E</sub></b>	69.62	80.18	63.30	87.38	56.50	64.97
<b>SDN</b>	<u>77.02</u>	<b>89.39</b>	<u>66.03</u>	<b>91.39</b>	61.48	<b>74.60</b>

## 6. Ablation Study

To further verify the effectiveness of our proposed model, we adopt two variant methods to verify the performance of each module. SDN<sub>P</sub> only employs the Quasi-GNN module and SDN<sub>E</sub> only utilizes the DNN module for encoding. Finally, we perform K-means on the embeddings obtained from them. Compared with other baselines, the experimental results show that removing either of the above two modules will lead to a decrease in accuracy and other metrics, which indicates that the two components of our proposed model are inseparable.

Moreover, it is worth noticing that some results obtained by SDN<sub>P</sub> are better than baselines. For instance, the accuracy in **Reuters**, **ACM**, and **CiteSeer** of SDN<sub>P</sub> is about **2%**, **0.5%**, and **1%** higher than that of AGCN. In addition, other metrics of SDN<sub>P</sub> also surpass AGCN with varying degrees, and this result indicates that the decoupled method we proposed still has superior performance while having high scalability.



**Figure 3.** Analysis of transmission probability  $\alpha$ . Through these four pictures, we can notice that when  $\alpha = 0.3$ , our model have the best performance for different metrics.

## 7. Analysis of Transmission Probability $\alpha$

The linear propagation layer adopts the propagation strategy defined by Equation (11). Owing to no parameters, it simply performs linear operations on the original matrix, which greatly improves scalability and reduces computational costs. We set  $\alpha = 0, 0.1, 0.3, 0.5, 0.7$ , and 1, respectively, on each dataset, and measure their final clustering effect. In this way, we obtain the most suitable transmission probability  $\alpha$  setting, and the experimental results are shown in Figure 3. The experimental results generally show a trend of increasing first and then decreasing with the increase in  $\alpha$ . The transition probability essentially indicates the probability that the target node learns from itself or its neighbor nodes. Based on the experimental results, we can infer that learning from only one of them is not sufficient. Therefore, it is necessary to find a suitable  $\alpha$  to integrate the information of the target node and its neighbor nodes to obtain the deep embeddings.

## 8. Conclusions

In this paper, we propose a scalable deep network with a Quasi-GNN module and a DNN module. First, we utilize the Quasi-GNN module to capture the information of graph topology and node features in different dimensions and employ the DNN module for auto-encoder to supplement the structural information. In addition, the combination of these two components can be combined with other post-processing methods to enable nodes further to be assigned to clusters with higher confidence, so it has high scalability. Moreover, our proposed model solves the issue of the orthogonal relationship between the model depth and the neighborhood range. It reduces the computational costs of the traditional GCN models and alleviates the over-smoothing issue caused by the stacking

of multiple GCN layers. Experiments on benchmark datasets show that our model has superior performance and achieves the SOTA effect.

For future work, we plan to optimize the Quasi-GNN module using the attention mechanism to consider the difference in importance between different nodes. On the other hand, we can add variants of GAE/VGAE to obtain more robust embeddings or propose different self-supervised modules to supervise the training of deep embeddings and clustering effectively.

**Author Contributions:** Conceptualization, Y.Z. (Yulin Zhao); methodology, Y.Z. (Yulin Zhao); software, Y.Z. (Yulin Zhao); validation, Y.Z. (Yulin Zhao); formal analysis, Y.Z. (Yulin Zhao); investigation, Y.Z. (Yulin Zhao) and X.L.; resources, Y.Z. (Yulin Zhao); data curation, Y.Z. (Yulin Zhao); writing—original draft preparation, Y.Z. (Yulin Zhao) and X.L.; writing—review and editing, Y.Z. (Yulin Zhao), X.L., Y.Z. (Yinlin Zhu), J.L. and S.W.; visualization, Y.Z. (Yulin Zhao); supervision, B.J.; project administration, B.J.; funding acquisition, B.J. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by Shandong Provincial Natural Science Foundation, China (No. ZR2020MA064). The APC was funded by Shandong Provincial Natural Science Foundation, China (No. ZR2020MA064).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Acknowledgments:** This paper was supported by the Shandong Provincial Natural Science Foundation (ZR2020MA064).

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Li, J.; Cai, D.; He, X. Learning graph-level representation for drug discovery. *arXiv* **2017**, arXiv:1709.03741.
2. Fan, W. Graph pattern matching revised for social network analysis. In Proceedings of the 15th International Conference on Database Theory, Berlin, Germany, 26–28 March 2012; pp. 8–21.
3. Guo, Z.; Wang, H. A deep graph neural network-based mechanism for social recommendations. *IEEE Trans. Ind. Inform.* **2020**, *17*, 2776–2783. [[CrossRef](#)]
4. Kipf, T.N.; Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv* **2016**, arXiv:1609.02907.
5. Kipf, T.N.; Welling, M. Variational graph auto-encoders. *arXiv* **2016**, arXiv:1611.07308.
6. Ahn, S.J.; Kim, M. Variational Graph Normalized AutoEncoders. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, Gold Coast, Australia, 1–5 November 2021; pp. 2827–2831.
7. Pan, S.; Hu, R.; Long, G.; Jiang, J.; Yao, L.; Zhang, C. Adversarially regularized graph autoencoder for graph embedding. *arXiv* **2018**, arXiv:1802.04407.
8. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y. Graph attention networks. *arXiv* **2017**, arXiv:1710.10903.
9. Wang, C.; Pan, S.; Hu, R.; Long, G.; Jiang, J.; Zhang, C. Attributed graph clustering: A deep attentional embedding approach. *arXiv* **2019**, arXiv:1906.06532.
10. Bo, D.; Wang, X.; Shi, C.; Zhu, M.; Lu, E.; Cui, P. Structural deep clustering network. In Proceedings of The Web Conference 2020, Taipei, Taiwan, 20–24 April 2020; pp. 1400–1410.
11. Li, X.; Hu, Y.; Sun, Y.; Hu, J.; Zhang, J.; Qu, M. A deep graph structured clustering network. *IEEE Access* **2020**, *8*, 161727–161738. [[CrossRef](#)]
12. Klicpera, J.; Bojchevski, A.; Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv* **2018**, arXiv:1810.05997.
13. Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; Weinberger, K. Simplifying graph convolutional networks. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 10–15 June 2019; pp. 6861–6871.
14. Frasca, F.; Rossi, E.; Eynard, D.; Chamberlain, B.; Bronstein, M.; Monti, F. Sign: Scalable inception graph neural networks. *arXiv* **2020**, arXiv:2004.11198.
15. Zhu, H.; Koniusz, P. Simple spectral graph convolution. In Proceedings of the International Conference on Learning Representations, Addis Ababa, Ethiopia, 26–30 April 2020.
16. Zhang, X.; Liu, H.; Li, Q.; Wu, X.M. Attributed graph clustering via adaptive graph convolution. *arXiv* **2019**, arXiv:1906.01210.
17. Chen, M.; Wei, Z.; Ding, B.; Li, Y.; Yuan, Y.; Du, X.; Wen, J.R. Scalable graph neural networks via bidirectional propagation. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 14556–14566.

18. Zhang, W.; Yin, Z.; Sheng, Z.; Ouyang, W.; Li, X.; Tao, Y.; Yang, Z.; Cui, B. Graph attention multi-layer perceptron. *arXiv* **2021**, arXiv:2108.10097.
19. Cao, S.; Lu, W.; Xu, Q. Grarep: Learning graph representations with global structural information. In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, Melbourne, Australia, 19–23 October 2015; pp. 891–900.
20. Perozzi, B.; Al-Rfou, R.; Skiena, S. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 24–27 August 2014; pp. 701–710.
21. Sieranoja, S.; Fränti, P. Adapting k-means for graph clustering. *Knowl. Inf. Syst.* **2022**, *64*, 115–142. [[CrossRef](#)]
22. Hinton, G.E.; Salakhutdinov, R.R. Reducing the dimensionality of data with neural networks. *Science* **2006**, *313*, 504–507. [[CrossRef](#)] [[PubMed](#)]
23. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Nets. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, Canada, 8–13 December 2014; Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2014; Volume 27.
24. Hamilton, W.; Ying, Z.; Leskovec, J. Inductive representation learning on large graphs. *arXiv* **2017**, arXiv:1706.02216.
25. Chen, J.; Ma, T.; Xiao, C. Fastgcn: Fast learning with graph convolutional networks via importance sampling. *arXiv* **2018**, arXiv:1801.10247.
26. Zeng, H.; Zhou, H.; Srivastava, A.; Kannan, R.; Prasanna, V. Graphsaint: Graph sampling based inductive learning method. *arXiv* **2019**, arXiv:1907.04931.
27. Cui, G.; Zhou, J.; Yang, C.; Liu, Z. Adaptive graph encoder for attributed graph embedding. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Online, 6–10 July 2020; pp. 976–985.
28. Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.i.; Jegelka, S. Representation learning on graphs with jumping knowledge networks. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 5453–5462.
29. Hull, J.J. A database for handwritten text recognition research. *IEEE Trans. Pattern Anal. Mach. Intell.* **1994**, *16*, 550–554. [[CrossRef](#)]
30. Stisen, A.; Blunck, H.; Bhattacharya, S.; Prentow, T.S.; Kjærgaard, M.B.; Dey, A.; Sonne, T.; Jensen, M.M. Smart devices are different: Assessing and mitigating mobile sensing heterogeneities for activity recognition. In Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems, Seoul, Korea, 1–4 November 2015; pp. 127–140.
31. Lewis, D.D.; Yang, Y.; Russell-Rose, T.; Li, F. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.* **2004**, *5*, 361–397.
32. Hartigan, J.A.; Wong, M.A. Algorithm AS 136: A k-means clustering algorithm. *J. R. Stat. Society. Ser. C* **1979**, *28*, 100–108. [[CrossRef](#)]
33. Fränti, P.; Kivijärvi, J. Randomised local search algorithm for the clustering problem. *Pattern Anal. Appl.* **2000**, *3*, 358–369. [[CrossRef](#)]
34. Xie, J.; Girshick, R.; Farhadi, A. Unsupervised deep embedding for clustering analysis. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 478–487.
35. Guo, X.; Gao, L.; Liu, X.; Yin, J. Improved deep embedded clustering with local structure preservation. In Proceedings of the International Joint Conference on Artificial Intelligence, Melbourne, Australia, 19–25 August 2017; pp. 1753–1759.
36. Peng, Z.; Liu, H.; Jia, Y.; Hou, J. Attention-driven Graph Clustering Network. In Proceedings of the 29th ACM International Conference on Multimedia, Online, 20–24 October 2021; pp. 935–943.
37. Gan, G.; Ma, C.; Wu, J. *Data Clustering: Theory, Algorithms, and Applications*; SIAM: Philadelphia, PA, USA, 2020.