

Towards Unsupervised Deep Graph Structure Learning

Yixin Liu¹, Yu Zheng², Daokun Zhang^{1,3}, Hongxu Chen⁴, Hao Peng⁵, Shirui Pan^{1*}

¹Monash University ²La Trobe University ³Monash Suzhou Research Institute

⁴University of Technology Sydney ⁵Beihang University

{yixin.liu, daokun.zhang, shirui.pan}@monash.edu;

yu.zheng@latrobe.edu.au; hongxu.chen@uts.edu.au; penghao@buaa.edu.cn

ABSTRACT

In recent years, graph neural networks (GNNs) have emerged as a successful tool in a variety of graph-related applications. However, the performance of GNNs can be deteriorated when noisy connections occur in the original graph structures; besides, the dependence on explicit structures prevents GNNs from being applied to general unstructured scenarios. To address these issues, recently emerged deep graph structure learning (GSL) methods propose to jointly optimize the graph structure along with GNN under the supervision of a node classification task. Nonetheless, these methods focus on a supervised learning scenario, which leads to several problems, i.e., the reliance on labels, the bias of edge distribution, and the limitation on application tasks. In this paper, we propose a more practical GSL paradigm, *unsupervised graph structure learning*, where the learned graph topology is optimized by data itself without any external guidance (i.e., labels). To solve the unsupervised GSL problem, we propose a novel **Str**Ucture **B**ootstrapping **L**earn**I**ng **f**ra**M**E**w**ork (**SUBLIME** for abbreviation) with the aid of self-supervised contrastive learning. Specifically, we generate a learning target from the original data as an “anchor graph”, and use a contrastive loss to maximize the agreement between the anchor graph and the learned graph. To provide persistent guidance, we design a novel bootstrapping mechanism that upgrades the anchor graph with learned structures during model learning. We also design a series of graph learners and post-processing schemes to model the structures to learn. Extensive experiments on eight benchmark datasets demonstrate the significant effectiveness of our proposed SUBLIME and high quality of the optimized graphs.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

graph neural networks, graph structure learning, unsupervised learning, contrastive learning

*Corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).
WWW '22, April 25–29, 2022, Lyon, France.

© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM... \$15.00
<https://doi.org/10.1145/XXXXXX.XXXXXX>

ACM Reference Format:

Yixin Liu¹, Yu Zheng², Daokun Zhang^{1,3}, Hongxu Chen⁴, Hao Peng⁵, Shirui Pan^{1*}. 2022. Towards Unsupervised Deep Graph Structure Learning. In *Proceedings of the ACM Web Conference 2022 (WWW '22)*, April 25–29, 2022, Lyon, France. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/XXXXXX.XXXXXX>

1 INTRODUCTION

Recent years have witnessed the prosperous development of graph-based applications in numerous domains, such as chemistry, bioinformatics and cybersecurity. As a powerful deep learning tool to model graph-structured data, graph neural networks (GNNs) have drawn increasing attention and achieved state-of-the-art performance in various graph analytical tasks, including node classification [22, 40], link prediction [21, 32], and node clustering [42, 55]. GNNs usually follow a message-passing scheme, where node representations are learned by aggregating information from the neighbors on an observed topology (i.e., the original graph structure).

Most GNNs rely on a fundamental assumption that the original structure is credible enough to be viewed as ground-truth information for model training. Such assumption, unfortunately, is usually violated in real-world scenarios, since graph structures are usually extracted from complex interaction systems which inevitably contain uncertain, redundant, wrong and missing connections [45]. Such noisy information in original topology can seriously damage the performance of GNNs. Besides, the reliance on explicit structures hinders GNNs’ broad applicability. If GNNs are capable of uncovering the implicit relations between samples, e.g., two images containing the same object, they can be applied to more general domains like vision and language.

To tackle the aforementioned problems, deep graph structure learning (GSL) is a promising solution that constructs and improves the graph topology with GNNs [7, 12, 20, 58]. Concretely, these methods parameterize the adjacency matrix with a probabilistic model [12, 45], full parameterization [20] or metric learning model [7, 11, 53], and jointly optimize the parameters of the adjacency matrix and GNNs by solving a downstream task (i.e., node classification) [58]. However, existing methods learn graph structures in a supervised scenario, which brings the following issues: (1) *The reliance on label information*. In supervised GSL methods, human-annotated labels play an important role in providing supervision signal for structure improvement. Such reliance on labels limits the application of supervised GSL on more general cases where annotation is unavailable. (2) *The bias of learned edge distribution*. Node classification usually follows a semi-supervised setting, where only a small fraction of nodes (e.g., 140/2708 in Cora dataset) are under the supervision of labels. As a result, the connections among these nodes and their neighbors would receive more guidance in

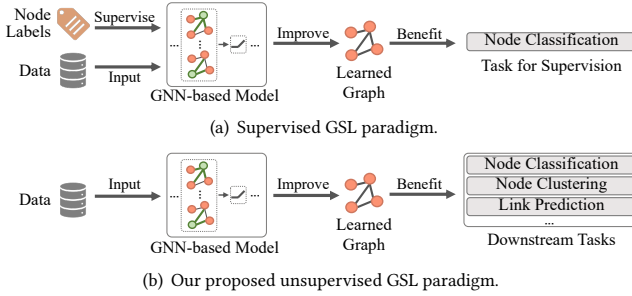


Figure 1: Concept maps of (a) the existing supervised GSL paradigm and (b) our proposed unsupervised GSL paradigm.

structure learning, while the relations between nodes far away from them are rarely discovered by GSL [11]. Such imbalance leads to the bias of edge distribution, affecting the quality of the learned structures. (3) *The limitation on downstream tasks.* In existing methods, the structure is specifically learned for node classification, so it may contain more task-specific information rather than general knowledge. Consequently, the refined topology may not benefit other downstream tasks like link prediction or node clustering, indicating the poor generalization ability of the learned structures.

To address these issues, in this paper, we investigate a novel unsupervised learning paradigm for GSL, namely *unsupervised graph structure learning*. As compared in Fig. 1, in our learning paradigm, structures are learned by data itself without any external guidance (i.e., labels), and the acquired universal, edge-unbiased topology can be freely applied to various downstream tasks. In this case, one natural question can be raised: how to provide sufficient supervision signal for unsupervised GSL? To answer this, we propose a novel **Str**ucture **B**ootstrapping **C**ontrastive **L**earning **f**ra**M**ework (**SUBLIME** for abbreviation) to learn graph structures with the aid of self-supervised contrastive learning [25]. Concretely, our method constructs an “anchor graph” from the original data to guide structure optimization, with a contrastive loss to maximize the mutual information (MI) between anchor graph and the learned structure. Through maximizing their consistency, informative hidden connections can be discovered, which well respects the node proximity conveyed by the original features and structures. Meanwhile, as we optimize the contrastive loss on the representations of every node, all potential edge candidates will receive the essential supervision, which promotes a balanced edge distribution in the inferred topology. Furthermore, we design a bootstrapping mechanism to update anchor graph with the learned edges, which provides a self-enhanced supervision signal for GSL. Besides, we carefully design multiple graph learners and post-processing schemes to model graph topology for diverse data. In summary, our core contributions are three-fold:

- **Problem.** We propose a novel unsupervised learning paradigm for graph structure learning, which is more practical and challenging than the existing supervised counterpart. To the best of our knowledge, this is the *first* attempt to learn graph structures with GNNs in an unsupervised setting.
- **Algorithm.** We propose a novel unsupervised GSL method **SUBLIME**, which guides structure optimization by maximizing the agreement between the learned structure and a crafted self-enhanced learning target with contrastive learning.

- **Evaluations.** We perform extensive experiments to corroborate the effectiveness and analyze the properties of **SUBLIME** via thorough comparisons with state-of-the-art methods on eight benchmark datasets.

2 RELATED WORK

2.1 Graph Neural Networks

Graph neural networks (GNNs) are a type of deep neural networks aiming to learn low-dimensional representations for graph-structure data [22, 48]. Modern GNNs can be categorized into two types: spectral and spatial methods. The spectral methods perform convolution operation to graph domain using spectral graph filter [3] and its simplified variants, e.g., Chebyshev polynomials filter [9] and the first-order approximation of Chebyshev filter [22]. The spatial methods perform convolution operation by propagating and aggregating local information along edges in a graph [15, 40, 50]. In spatial GNNs, different aggregation functions are designed to learn node representations, including mean/max pooling [15], LSTM [15], self-attention [40], and summation [50]. Readers may refer to the elaborate survey [48] for a thorough review.

2.2 Deep Graph Structure Learning

Graph structure learning (GSL) problem has been investigated by conventional machine learning techniques in graph signal processing [10], spectral clustering [2], and network science [26]. However, these methods are not capable of handling graph data with high-dimensional features, so they are not further discussed in our paper.

Very recently, there thrives a branch of research that investigates GSL for GNNs with the aim to boost their performance on downstream tasks, which is named deep graph structure learning [58]. These methods follow a general pipeline: the graph adjacency matrix is modeled with learnable parameters, and then jointly optimized along with GNN under the supervision of a downstream node classification task. In these methods, various techniques are leveraged to parameterize the adjacency matrix. Considering the discrete nature of graph structures, one type of methods adopts probabilistic models, such as Bernoulli probability model [12] and stochastic block model [45]. Another type of methods models structures with node-wise similarity computed by metric learning functions like cosine similarity [7] and dot production [11, 53]. Besides, directly treating each element in adjacency matrix as a learnable parameter is also an effective solution [11, 20]. Nevertheless, the existing deep GSL approaches follow a supervised scenario where node labels are always required to refine the graph structures. In this paper, differently, we advocate a more practical unsupervised learning paradigm where no extra information is needed for GSL.

2.3 Contrastive Learning on Graphs

After achieving significant performance in visual [6, 14] and linguistic [8, 13] domains, contrastive learning has shown competitive performance and become increasingly popular in graph representation learning [32, 39, 59]. Graph contrastive learning obeys the principle of mutual information (MI) maximization, which pulls the representations of samples with shared semantic information closer while pushing the representations of irrelevant samples away [25]. In graph data, the MI maximization can be carried out to samples in the same scale (i.e., node-level [19, 41, 59] and graph-level [52]) or

different scales (i.e., node v.s. graph [39, 57] and node v.s. subgraph [32]). Graph contrastive learning also benefits diverse applications, such as chemical prediction [47], anomaly detection [18, 24], federated learning [36, 56], and recommendation [54]. However, it still remains unclear how to effectively improve GSL using contrastive learning.

3 PROBLEM DEFINITION

Before we make the problem statement of unsupervised GSL, we first introduce the definition of graphs. An attributed graph can be represented by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}) = (\mathbf{A}, \mathbf{X})$, where \mathcal{V} is the set of $n = |\mathcal{V}|$ nodes, \mathcal{E} is the set of $m = |\mathcal{E}|$ edges, $\mathbf{X} \in \mathbb{R}^{n \times d}$ is the node feature matrix (where the i -th row \mathbf{x}_i is the feature vector of node v_i), and $\mathbf{A} \in [0, 1]^{n \times n}$ is the weighted adjacency matrix (where a_{ij} is the weight of the edge connecting v_i and v_j). Frequently used notations are summarized in Appendix A.

In this paper, we consider two unsupervised GSL tasks, i.e., structure inference and structure refinement. The former is applicable to general datasets where graph structures are not predefined or are unavailable. The latter, differently, aims to modify the given noisy topology and produce a more informative graph. Node labels are unavailable for structure optimization in both tasks.

Definition 3.1 (Structure inference). Given a feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, the target of structure inference is to automatically learn a graph topology $\mathbf{S} \in [0, 1]^{n \times n}$, which reflects the underlying correlations among data samples. In particular, $S_{ij} \in [0, 1]$ indicates whether there is an edge between two samples (nodes) \mathbf{x}_i and \mathbf{x}_j .

Definition 3.2 (Structure refinement). Given a graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ with a noisy graph structure \mathbf{A} , the target of structure refinement is to refine \mathbf{A} to be the optimized adjacency matrix $\mathbf{S} \in [0, 1]^{n \times n}$ to better capture the underlying dependency between nodes.

With the graph topology \mathbf{S} which is either learned automatically from data or refined from an existing graph structure, the hypothesis is that the model performance on downstream tasks can be essentially improved with $\mathcal{G}_l = (\mathbf{S}, \mathbf{X})$ as the input.

4 METHODOLOGY

This section elaborates our proposed SUBLIME, a novel unsupervised GSL framework. As shown in Fig. 2, SUBLIME on the highest level consists of two components: the *graph structure learning module* that models and regularizes the learned graph topology and the *structure bootstrapping contrastive learning module* that provides a self-optimized supervision signal for GSL. In the graph structure learning module, a sketched adjacency matrix is first parameterized by a graph learner, and then refined by a post-processor to be the learned adjacency matrix. Afterwards, in the structure bootstrapping contrastive learning module, we first establish two different views to contrast: learner view that discovers graph structure and anchor view that provides guidance for structure learning. Then, after data augmentation, the agreement between two views is maximized by a node-level contrastive learning. Specially, we design a structure bootstrapping mechanism to update anchor view with learned structures. The following subsections illustrate these crucial components respectively.

4.1 Graph Learner

As a key component of GSL, the graph learner generates a sketched adjacency matrix $\tilde{\mathbf{S}} \in \mathbb{R}^{n \times n}$ with a parameterized model. Most existing methods [7, 12, 20] adopt a single strategy to model graph structure, which cannot adapt to data with different unique properties. To find optimal structures for various data, we consider four types of graph learners, including a full graph parameterization (FGP) learner and three metric learning-based learners (i.e., Attentive, MLP, and GNN learner). In general, we formulate a graph learner as $p_\omega(\cdot)$, where ω is the learnable parameters.

FGP learner directly models each element of the adjacency matrix by an independent parameter [11, 12, 20] without any extra input. Formally, FGP learner is defined as:

$$\tilde{\mathbf{S}} = p_\omega^{FGP} = \sigma(\Omega), \quad (1)$$

where $\omega = \Omega \in \mathbb{R}^{n \times n}$ is a parameter matrix and $\sigma(\cdot)$ is a non-linear function that makes training more stable. The assumption behind FGP learner is that each edge exists independently in the graph.

Different from the FGP learner, metric learning-based learners [7, 58] first acquire node embeddings $\mathbf{E} \in \mathbb{R}^{n \times d}$ from the input data, and then model $\tilde{\mathbf{S}}$ with pair-wise similarity of the node embeddings:

$$\tilde{\mathbf{S}} = p_\omega^{ML}(\mathbf{X}, \mathbf{A}) = \phi(h_\omega(\mathbf{X}, \mathbf{A})) = \phi(\mathbf{E}), \quad (2)$$

where $h_\omega(\cdot)$ is a neural network-based embedding function (a.k.a. embedding network) with parameter ω , and $\phi(\cdot)$ is a non-parametric metric function (e.g., cosine similarity or Minkowski distance) that calculates pair-wise similarity. For different $h_\omega(\cdot)$, we provide three specific instances of metric learning-based learners: Attentive, MLP, and GNN learners.

Attentive Learner employs a GAT-like [40] attentive network as its embedding network, where each layer compute the Hadamard production of input feature vector and the parameter vector:

$$\mathbf{E}^{(l)} = h_w^{(l)}(\mathbf{E}^{(l-1)}) = \sigma([\mathbf{e}_1^{(l-1)} \odot \omega^{(l)}, \dots, \mathbf{e}_n^{(l-1)} \odot \omega^{(l)}]^\top), \quad (3)$$

in which $\mathbf{E}^{(l)}$ is the output matrix of the l -th layer of embedding network, $\mathbf{e}_i^{(l-1)} \in \mathbb{R}^d$ is the transpose of the i -th row vector of $\mathbf{E}^{(l-1)}$, $\omega^{(l)} \in \mathbb{R}^d$ is the parameter vector of the l -th layer, \odot is the Hadamard operation, $(\cdot)^\top$ is the transposition operation, and $\sigma(\cdot)$ is a non-linear operation. The input of the first layer $\mathbf{E}^{(0)}$ is the feature matrix \mathbf{X} , and the output of the final layer $\mathbf{E}^{(L)}$ (L is the layer number of embedding network) is the embedding matrix \mathbf{E} . Attentive learner assumes that each feature has different contribution to the existence of edge, but there is no significant correlation between features.

MLP Learner uses a Multi-Layer Perception (MLP) as its embedding network, where a single layer can be written by:

$$\mathbf{E}^{(l)} = h_w^{(l)}(\mathbf{E}^{(l-1)}) = \sigma(\mathbf{E}^{(l-1)} \Omega^{(l)}), \quad (4)$$

where $\Omega^{(l)} \in \mathbb{R}^{d \times d}$ is the parameter matrix of the l -th layer, and the other notations are similar to Eq. (3). Compared to attentive learner, MLP learner further considers the correlation and combination of features, generating more informative embeddings for downstream similarity metric learning.

GNN Learner integrates features \mathbf{X} and original structure \mathbf{A} into node embeddings \mathbf{E} via GNN-based embedding network. Due to the reliance on original topology, GNN learner is only used for

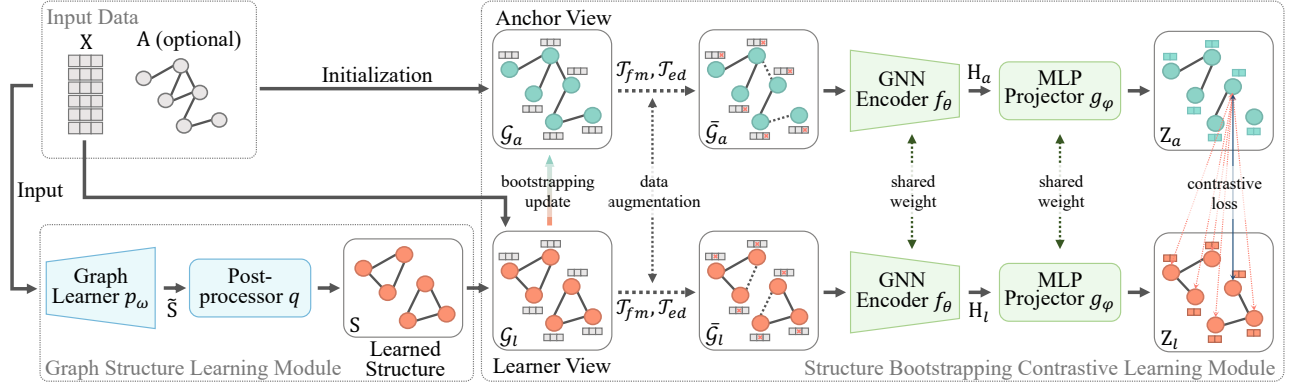


Figure 2: The overall pipeline of SUBLIME. In the graph structure learning module, the graph learner p_ω generates the sketched adjacency matrix \tilde{S} , and then the post processor q converts \tilde{S} into the learned structure S . After that, the structure bootstrapping contrastive learning module optimizes S by maximizing the agreement between the learner view and anchor view.

the structure refinement task. For simplicity, we take GCN layers [22] to form embedded network:

$$E^{(l)} = h_w^{(l)}(E^{(l-1)}, A) = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} E^{(l-1)} \Omega^{(l)}), \quad (5)$$

where $\tilde{A} = A + I$ is the adjacency matrix with self-loop, \tilde{D} is the degree matrix of \tilde{A} , and the other notations are similar to Eq. (4). GNN Learner assumes that the connection between two nodes is related to not only features but also the original structure.

In SUBLIME, we choose the most suitable learner to model \tilde{S} according to the characteristics of different datasets. In Appendix B, we analyze the properties of different graph learners and discuss how we allocate learners for each dataset.

4.2 Post-processor

The post-processor $q(\cdot)$ aims to refine the sketched adjacency matrix \tilde{S} into a sparse, non-negative, symmetric and normalized adjacency matrix S . To this end, four post-processing steps are applied sequentially, i.e., sparsification $q_{sp}(\cdot)$, activation $q_{act}(\cdot)$, symmetrization $q_{sym}(\cdot)$, and normalization $q_{norm}(\cdot)$.

Sparsification. The sketched adjacency matrix \tilde{S} is often dense, representing a fully connected graph structure. However, such adjacency matrix usually makes little sense for most applications and results in expensive computation cost [45]. Hence, we conduct a k-nearest neighbors (kNN)-based sparsification on \tilde{S} . Concretely, for each node, we keep the edges with top-k connection values and set the rest to 0. The sparsification $q_{sp}(\cdot)$ is expressed as:

$$\tilde{S}_{ij}^{(sp)} = q_{sp}(\tilde{S}_{ij}) = \begin{cases} \tilde{S}_{ij}, & \tilde{S}_{ij} \in \text{top-k}(\tilde{S}_i), \\ 0, & \tilde{S}_{ij} \notin \text{top-k}(\tilde{S}_i), \end{cases} \quad (6)$$

where $\text{top-k}(\tilde{S}_i)$ is the set of top-k values of row vector \tilde{S}_i . To keep the gradient flow, we do not apply sparsification for the FGP learner. For large-scale graphs, we perform the kNN sparsification with its locality-sensitive approximation [11] where the nearest neighbors are selected from a batch of nodes instead of all nodes, which reduces the requirement of memory.

Symmetrization and Activation. In real-world graphs, the connections are often bi-directional, which requires a symmetric adjacency matrix. In addition, the edge weights should be non-negative

according to the definition of adjacency matrix. To meet these conditions, the symmetrization and activation are performed as:

$$\tilde{S}^{(sym)} = q_{sym}(q_{act}(\tilde{S}^{(sp)})) = \frac{\sigma_q(\tilde{S}^{(sp)}) + \sigma_q(\tilde{S}^{(sp)})^T}{2}, \quad (7)$$

where $\sigma_q(\cdot)$ is a non-linear activation. For metric learning-based learners, we define $\sigma_q(\cdot)$ as ReLU function. For FGP learner, we apply the ELU function to prevent gradient from disappearing.

Normalization. To guarantee the edge weights are within the range $[0, 1]$, we finally conduct a normalization on \tilde{S} . In particular, we apply a symmetrical normalization:

$$S = q_{norm}(\tilde{S}^{(sym)}) = (\tilde{D}^{(sym)})^{-\frac{1}{2}} \tilde{S}^{(sym)} (\tilde{D}^{(sym)})^{-\frac{1}{2}}, \quad (8)$$

where $\tilde{D}^{(sym)}$ is the degree matrix of $\tilde{S}^{(sym)}$.

4.3 Multi-view Graph Contrastive Learning

Since we have obtained a well-parameterized adjacency matrix S , a natural question that arises here is: *how to provide an effective supervision signal guiding the graph structure learning without label information?* Our answer is to acquire the supervision signal from data itself via multi-view graph contrastive learning. To be concrete, we construct two graph views based on the learned structure and the original data respectively. Then, data augmentation is applied to both views. Finally, we maximize the MI between two augmented views with node-level contrastive learning.

4.3.1 Graph View Establishment. Different from general graph contrastive learning methods [19, 59] that obtain both views from the original data, SUBLIME defines the learned graph as one view, and constructs the other view with input data. The former, named *learner view*, explores potential structures in every step. The latter, named *anchor view*, provides a stable learning target for GSL.

Learner view is directly built by integrating the learned adjacency matrix S and the feature matrix X together, which is denoted as $\mathcal{G}_l = (S, X)$. In each training iteration, S and the parameters used to model it are directly updated by gradient descent to discover optimal graph structures. In SUBLIME, we initialize learner views as the kNN graph built on features, since it is an effective way to provide a starting point for GSL, as suggested in [11, 12]. Specifically, for FGP learner, we initialize the parameters corresponding to kNN edges as 1 while the rest as 0. For attentive learner, we let

each element in $\omega^{(l)} \in \omega$ to be 1. Then, feature-level similarities are computed according to the metric function, and the kNN graph is obtained by the sparsification post-processing. For MLP and GNN learners, similarly, we set the embedding dimension to be d and initialize $\Omega^{(l)} \in \omega$ as identity matrices.

Anchor view plays a “teacher” role that provides correct and stable guidance for GSL. For the structure refinement task where the original structure \mathbf{A} is available, we define anchor view as $\mathcal{G}_a = (\mathbf{A}_a, \mathbf{X}) = (\mathbf{A}, \mathbf{X})$; for the structure inference task where \mathbf{A} is inaccessible, we take an identity matrix \mathbf{I} as the anchor structure: $\mathcal{G}_a = (\mathbf{A}_a, \mathbf{X}) = (\mathbf{I}, \mathbf{X})$. To provide a stable learning target, anchor view is not updated by gradient descent but a novel bootstrapping mechanism which will be introduced in Section 4.4.

4.3.2 Data Augmentation. In contrastive learning, data augmentation is a key to benefiting the model through exploring richer underlying semantic information by making the learning tasks more challenging to solve [6, 24, 59]. In SUBLIME, we exploit two simple but effective augmentation schemes, i.e., *feature masking* and *edge dropping*, to corrupt the graphs views at both structure and feature levels.

Feature masking. To disturb the node features, we randomly select a fraction of feature dimensions and mask them with zeros. Formally, for a given feature matrix \mathbf{X} , a masking vector $\mathbf{m}^{(x)} \in \{0, 1\}^d$ is first sampled, where each element is drawn from a Bernoulli distribution with probability $p^{(x)}$ independently. Then, we mask the feature vector of each node with $\mathbf{m}^{(x)}$:

$$\bar{\mathbf{X}} = \mathcal{T}_{fm}(\mathbf{X}) = [\mathbf{x}_1 \odot \mathbf{m}^{(x)}, \dots, \mathbf{x}_n \odot \mathbf{m}^{(x)}]^\top, \quad (9)$$

where $\bar{\mathbf{X}}$ is the augmented feature matrix, $\mathcal{T}_{fm}(\cdot)$ is the feature masking transformation, and \mathbf{x}_i is the transpose of the i -th row vector of \mathbf{X} .

Edge dropping. Apart from masking features, we corrupt the graph structure by randomly dropping a portion of edges. Specifically, for a given adjacency matrix \mathbf{A} , we first sample a masking matrix $\mathbf{M}^{(a)} \in \{0, 1\}^{n \times n}$, where each element $\mathbf{M}_{ij}^{(a)}$ is drawn from a Bernoulli distribution with probability $p^{(a)}$ independently. After that, the adjacency matrix is masked with $\mathbf{M}^{(a)}$:

$$\bar{\mathbf{A}} = \mathcal{T}_{ed}(\mathbf{A}) = \mathbf{A} \odot \mathbf{M}^{(a)}, \quad (10)$$

where $\bar{\mathbf{A}}$ is the augmented adjacency matrix, and $\mathcal{T}_{ed}(\cdot)$ is the edge dropping transformation.

In SUBLIME, we jointly leverage these two augmentation schemes to generate augmented graphs on both learner and anchor views:

$$\bar{\mathcal{G}}_l = (\mathcal{T}_{ed}(\mathbf{S}), \mathcal{T}_{fm}(\mathbf{X})), \quad \bar{\mathcal{G}}_a = (\mathcal{T}_{ed}(\mathbf{A}_a), \mathcal{T}_{fm}(\mathbf{X})), \quad (11)$$

where $\bar{\mathcal{G}}_l$ and $\bar{\mathcal{G}}_a$ are the augmented learner view and anchor view, respectively. To obtain different contexts in the two views, the feature masking for two views employs different probabilities $p_l^{(x)} \neq p_a^{(x)}$. For edge dropping, since the adjacency matrices of two views are already significantly different, we use the same dropping probability $p_l^{(a)} = p_a^{(a)} = p^{(a)}$. Note that other advanced augmentation schemes can also be applied to SUBLIME, which is left for our future research.

4.3.3 Node-level Contrastive Learning. After obtaining two augmented graph views, we perform a node-level contrastive learning to maximize the MI between them. In SUBLIME, we adopt a simple

contrastive learning framework originated from SimCLR [6] which consists of the following components:

GNN-based encoder. A GNN-based encoder $f_\theta(\cdot)$ extracts node-level representations for augmented graphs $\bar{\mathcal{G}}_l$ and $\bar{\mathcal{G}}_a$:

$$\mathbf{H}_l = f_\theta(\bar{\mathcal{G}}_l), \quad \mathbf{H}_a = f_\theta(\bar{\mathcal{G}}_a), \quad (12)$$

where θ is the parameter of encoder $f_\theta(\cdot)$, and $\mathbf{H}_l, \mathbf{H}_a \in \mathbb{R}^{n \times d_1}$ (d_1 is the representation dimension) are the node representation matrices for learner/anchor views, respectively. In SUBLIME, we utilize GCN [22] as our encoder and set its layer number L_1 to 2.

MLP-based projector. Following the encoder, a projector $g_\varphi(\cdot)$ with L_2 MLP layers maps the representations to another latent space where the contrastive loss is calculated:

$$\mathbf{Z}_l = g_\varphi(\mathbf{H}_l), \quad \mathbf{Z}_a = g_\varphi(\mathbf{H}_a), \quad (13)$$

where φ is the parameter of projector $g_\varphi(\cdot)$, and $\mathbf{Z}_l, \mathbf{Z}_a \in \mathbb{R}^{n \times d_2}$ (d_2 is the projection dimension) are the projected node representation matrices for learner/anchor views, respectively.

Node-level contrastive loss function. A contrastive loss \mathcal{L} is leveraged to enforce maximizing the agreement between the projections $z_{l,i}$ and $z_{a,i}$ of the same node v_i on two views. In our framework, a symmetric normalized temperature-scaled cross-entropy loss (NT-Xent) [29, 35] is applied:

$$\mathcal{L} = \frac{1}{2n} \sum_{i=1}^n \left[\ell(z_{l,i}, z_{a,i}) + \ell(z_{a,i}, z_{l,i}) \right], \quad (14)$$

$$\ell(z_{l,i}, z_{a,i}) = \log \frac{e^{\text{sim}(z_{l,i}, z_{a,i})/t}}{\sum_{k=1}^n e^{\text{sim}(z_{l,i}, z_{a,k})/t}},$$

where $\text{sim}(\cdot, \cdot)$ is the cosine similarity function, and t is the temperature parameter. $\ell(z_{a,i}, z_{l,i})$ is computed following $\ell(z_{l,i}, z_{a,i})$.

4.4 Structure Bootstrapping Mechanism

With a fixed anchor adjacency matrix \mathbf{A}_a defined by \mathbf{A} or \mathbf{I} , SUBLIME can learn graph structure \mathbf{S} by maximizing the MI between two views. However, using a constant anchor graph may lead to several issues: (1) *Inheritance of error information.* Since \mathbf{A}_a is directly borrowed from the input data, it would carry some natural noise (e.g., missing or redundant edges) of the original graph. If the noise is not eliminated in the learning process, the learned structures will finally inherit it. (2) *Lack of persistent guidance.* A fixed anchor graph contains limited information to guide GSL. Once the graph learner captures this information, it will be hard for the model to gain effective supervision in the following training steps. (3) *Overfitting the anchor structure.* Driven by the learning objective that maximizes the agreement between two views, the learned structure tends to over-fit the fixed anchor structure, resulting in a similar testing performance to the original data.

Inspired by previous bootstrapping-based algorithms [5, 14, 37], we design a structure bootstrapping mechanism to provide a self-enhanced anchor view as the learning target. The core idea of our solution is to update the anchor structure \mathbf{A}_a with a slow-moving augmentation of the learned structure \mathbf{S} instead of keeping \mathbf{A}_a unchanged. In particular, given a decay rate $\tau \in [0, 1]$, the anchor structure \mathbf{A}_a is updated every c iterations as following:

$$\mathbf{A}_a \leftarrow \tau \mathbf{A}_a + (1 - \tau) \mathbf{S}. \quad (15)$$

Table 1: Node classification accuracy (percentage with standard deviation) in structure inference scenario. Available data for graph structure learning during the training phase is shown in the first column, where X, Y, A_{knn} correspond to node features, labels and the adjacency matrix of kNN graph, respectively. The highest and second highest results are highlighted with **boldface and underline, respectively. The symbol “OOM” means out of memory.**

| Available Data for GSL | Method | Dataset | | | | | | | | |
|------------------------|--------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|--|
| | | Cora | Citeseer | Pubmed | ogbn-arxiv | Wine | Cancer | Digits | 20news | |
| - | LR | 60.8±0.0 | 62.2±0.0 | 72.4±0.0 | 52.5±0.0 | 92.1±1.3 | 93.3±0.5 | 85.5±1.5 | 42.7±1.7 | |
| - | Linear SVM | 58.9±0.0 | 58.3±0.0 | 72.7±0.1 | 51.8±0.0 | 93.9±1.6 | 90.6±4.5 | 87.1±1.8 | 40.3±1.4 | |
| - | MLP | 56.1±1.6 | 56.7±1.7 | 71.4±0.0 | 54.7±0.1 | 89.7±1.9 | 92.9±1.2 | 36.3±0.3 | 38.6±1.4 | |
| - | GCN _{knn} [22] | 66.5±0.4 | 68.3±1.3 | 70.4±0.4 | 54.1±0.3 | 93.2±3.1 | 83.8±1.4 | 91.3±0.5 | 41.3±0.6 | |
| - | GAT _{knn} [40] | 66.2±0.5 | 70.0±0.6 | 69.6±0.5 | OOM | 91.5±2.4 | 95.1±0.8 | 91.4±0.1 | 45.0±1.2 | |
| - | SAGE _{knn} [15] | 66.1±0.7 | 68.0±1.6 | 68.7±0.2 | 55.2±0.4 | 87.4±0.8 | 93.7±0.3 | 91.6±0.7 | 45.4±0.4 | |
| X, Y | LDS [12] | 71.5±0.8 | 71.5±1.1 | OOM | OOM | 97.3±0.4 | 94.4±1.9 | 92.5±0.7 | 46.4±1.6 | |
| X, Y, A _{knn} | GRCN [53] | 69.6±0.2 | 70.4±0.3 | 70.6±0.1 | OOM | 96.6±0.4 | 95.4±0.6 | 92.8±0.2 | 41.8±0.2 | |
| X, Y, A _{knn} | Pro-GNN [20] | 69.2±1.4 | 69.8±1.7 | OOM | OOM | 95.1±1.5 | 96.5±0.1 | 93.9±1.9 | 45.7±1.4 | |
| X, Y, A _{knn} | GEN [45] | 69.1±0.7 | 70.7±1.1 | 70.7±0.9 | OOM | 96.9±1.0 | <u>96.8±0.4</u> | 94.1±0.4 | 47.1±0.3 | |
| X, Y | IDGL [7] | 70.9±0.6 | 68.2±0.6 | 70.1±1.3 | 55.0±0.2 | <u>98.1±1.1</u> | 95.1±1.0 | 93.2±0.9 | 48.5±0.6 | |
| X, Y | SLAPS [11] | 73.4±0.3 | <u>72.6±0.6</u> | 74.4±0.6 | 56.6±0.1 | 96.6±0.4 | 96.6±0.2 | 94.4±0.7 | 50.4±0.7 | |
| A _{knn} | GDC [23] | 68.1±1.2 | 68.8±0.8 | 68.4±0.4 | OOM | 96.1±1.0 | 95.9±0.4 | 92.6±0.5 | 46.4±0.9 | |
| X | SLAPS-2s [11] | 72.1±0.4 | 69.4±1.4 | 71.1±0.5 | 54.2±0.2 | 96.2±2.1 | 95.9±1.2 | 93.6±0.8 | 47.7±0.7 | |
| X | SUBLIME | <u>73.0±0.6</u> | 73.1±0.3 | <u>73.8±0.6</u> | <u>55.5±0.1</u> | 98.2±1.6 | 97.2±0.2 | <u>94.3±0.4</u> | <u>49.2±0.6</u> | |

Benefiting from the structure bootstrapping mechanism, SUBLIME has nice properties that can address the aforementioned problems. With the process of updating, the weights of some noise edges gradually decrease in A_a , which relieves their negative impact on structure learning. Meanwhile, since the learning target A_a is changing during the training phase, it can always incorporate more effective information to guide the learning of topology, and the over-fitting problem is naturally resolved. More importantly, our structure bootstrapping mechanism leverages the learned knowledge to improve the learning target in turn, pushing the model to discover increasingly optimal graph structure constantly. Besides, the slow-moving average (with $\tau > 0.99$) updating ensures the stability of training.

4.5 Overall Framework

In this subsection, we first illustrate the training process of SUBLIME, and then introduce the tricks to help apply it to large-scale graphs. **Model training.** In our training process, we first initialize the parameters and anchor adjacency matrix A_a . Then, in each iteration, we perform forward propagation to compute the contrastive loss \mathcal{L} , and update all the parameters jointly via back propagation. After back propagation, we update A_a by bootstrapping structure mechanism every c iterations. Finally, we acquire the learned topology represented by S . As analyzed in Appendix C, the time complexity of SUBLIME is $O(n^2d + md_1L_1 + nd_1^2L_1 + nd_2^2L_2 + nk)$. The algorithmic description is provided in Appendix D.

Scalability extension. To extend the scalability of SUBLIME, the key is to avoid $O(n^2)$ space complexity and time complexity. To this end, we adopt the following measures: (1) To avoid explosive number of parameters, we use metric learning-based learners instead of FGP learner. (2) For sparsification post-processing, we consider a locality-sensitive approximation for kNN graph [11]. (3) For graph contrastive learning, we compute the contrastive loss \mathcal{L} for a mini-batch of samples instead of all nodes. (4) To reduce the space complexity of the bootstrapped structure, we perform the update in Eq. (15) with a larger iteration interval c ($c \geq 10$).

5 EXPERIMENTS

In this section, we conduct empirical experiments to demonstrate the effectiveness of the proposed framework SUBLIME. We aim to answer five research questions as follows: **RQ1:** How effective is SUBLIME for learning graph structure under unsupervised settings? **RQ2:** How does the structure bootstrapping mechanism influence the performance of SUBLIME? **RQ3:** How do key hyper-parameters impact the performance of SUBLIME? **RQ4:** How robust is SUBLIME to adversarial graph structures? and **RQ5:** What kind of graph structure is learned by SUBLIME?

5.1 Experimental Setups

Downstream tasks for evaluation. We use node classification and node clustering tasks to evaluate the quality of learned topology. For node classification, We conduct experiments on both structure inference/refinement scenarios, and use classification accuracy as our metric. For node clustering, the experiments are conducted on structure refinement scenario, and four metrics are employed, including clustering accuracy (C-ACC), Normalized Mutual Information (NMI), F1-score (F1) and Adjusted Rand Index (ARI).

Datasets. We evaluate SUBLIME on eight real-world benchmark datasets, including four graph-structured datasets (i.e., Cora, Citeseer [34], Pubmed [27] and ogbn-arxiv [17]) and four non-graph datasets (i.e., Wine, Cancer, Digits and 20news [1]). Details of datasets are summarized in Appendix E.

Baselines. For node classification, we mainly compare SUBLIME with two categories of methods, including three structure-fixed GNN methods (i.e., GCN [22], GAT [40] and GraphSAGE (SAGE for short) [15]), and six supervised GSL methods (i.e., LDS [12], GRCN [53], Pro-GNN [20], GEN [45], IDGL [7] and SLAPS [11]). We also consider GDC [23], a diffusion-based graph structure improvement method, and SLAPS-2s, a variant of SLAPS [11] which only uses denoising autoencoder to learn topology, as two baselines of unsupervised GSL. In structure inference scenario, we further add three conventional feature-based classifiers (Logistic Regression, Linear SVM and MLP) for comparison. For node clustering task, we consider baseline methods belonging to the following three

Table 2: Node classification accuracy (percentage with standard deviation) in structure refinement scenario.

| Available Data for GSL | Method | Dataset | | | |
|------------------------|---------|-----------------|-----------------|-----------------|-----------------|
| | | Cora | Citeseer | Pubmed | ogbn-arxiv |
| - | GCN | 81.5 | 70.3 | 79.0 | 71.7±0.3 |
| - | GAT | 83.0±0.7 | 72.5±0.7 | 79.0±0.3 | OOM |
| - | SAGE | 77.4±1.0 | 67.0±1.0 | 76.6±0.8 | 71.5±0.3 |
| X, Y, A | LDS | 83.9±0.6 | 74.8±0.3 | OOM | OOM |
| X, Y, A | GRCN | 84.0±0.2 | 73.0±0.3 | 78.9±0.2 | OOM |
| X, Y, A | Pro-GNN | 82.1±0.4 | 71.3±0.4 | OOM | OOM |
| X, Y, A | GEN | 82.3±0.4 | 73.5±1.5 | 80.9±0.8 | OOM |
| X, Y, A | IDGL | 84.0±0.5 | 73.1±0.7 | 83.0±0.2 | 72.0±0.3 |
| A | GDC | 83.6±0.2 | 73.4±0.3 | 78.7±0.4 | OOM |
| X, A | SUBLIME | 84.2±0.5 | 73.5±0.6 | 81.0±0.6 | 71.8±0.3 |

Table 3: Node clustering performance (4 metrics in percentage) in structure refinement scenario.

| Method | Cora | | | | Citeseer | | | |
|---------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| | C-ACC | NMI | F1 | ARI | C-ACC | NMI | F1 | ARI |
| K-means | 50.0 | 31.7 | 37.6 | 23.9 | 54.4 | 31.2 | 41.3 | 28.5 |
| SC | 39.8 | 29.7 | 33.2 | 17.4 | 30.8 | 9.0 | 25.7 | 8.2 |
| GE | 30.1 | 5.9 | 23.0 | 4.6 | 29.3 | 5.7 | 21.3 | 4.3 |
| DW | 52.9 | 38.4 | 43.5 | 29.1 | 39.0 | 13.1 | 30.5 | 13.7 |
| DNGR | 41.9 | 31.8 | 34.0 | 14.2 | 32.6 | 18.0 | 30.0 | 4.3 |
| M-NMF | 42.3 | 25.6 | 32.0 | 16.1 | 33.6 | 9.9 | 25.5 | 7.0 |
| RMSC | 46.6 | 32.0 | 34.7 | 20.3 | 51.6 | 30.8 | 40.4 | 26.6 |
| TADW | 53.6 | 36.6 | 40.1 | 24.0 | 52.9 | 32.0 | 43.6 | 28.6 |
| VGAE | 59.2 | 40.8 | 45.6 | 34.7 | 39.2 | 16.3 | 27.8 | 10.1 |
| ARGA | 64.0 | 44.9 | 61.9 | 35.2 | 57.3 | 35.0 | 54.6 | 34.1 |
| MGAE | 68.1 | 48.9 | 53.1 | 56.5 | 66.9 | 41.6 | 52.6 | 42.5 |
| AGC | 68.9 | 53.7 | 65.6 | 44.8 | 67.0 | 41.1 | 62.5 | 41.5 |
| DAEGC | 70.4 | 52.8 | 68.2 | 49.6 | 67.2 | 39.7 | 63.6 | 41.0 |
| SUBLIME | 71.3 | 54.2 | 63.5 | 50.3 | 68.5 | 44.1 | 63.2 | 43.9 |

categories: 1) feature-based clustering methods (i.e., K-means [16] and Spectral Clustering (SC for short) [28]); 2) structure-based clustering methods (i.e., GraphEncoder (GE for short) [38], DeepWalk (DW for short) [33], DNGR [4] and M-NMF [46]); and 3) attributed graph clustering methods (i.e., RMSC [49], TADW [51], VGAE [21], ARGA [30], MGAE [43], AGC [55] and DAEGC [42]).

For other experimental details, including infrastructures and hyper-parameter, interested readers can refer to Appendix F. Our code is available at <https://github.com/GRAND-Lab/SUBLIME>.

5.2 Performance Comparison (RQ1)

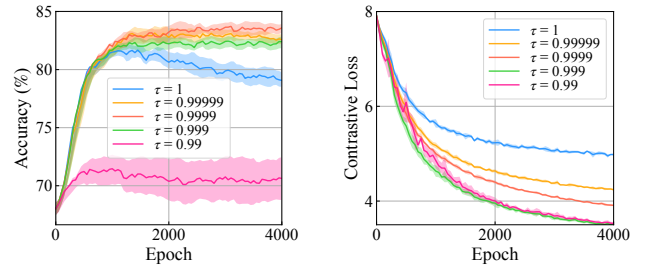
Node classification in structure inference scenario. Table 1 reports the classification accuracy of our method and other baselines in structure inference scenario. For structure-fixed GNNs (i.e., GCN, GAT and GraphSAGE) and GSL methods designed for structure refinement scenarios (i.e., GRCN, Pro-GNN, GEN and GDC), we use kNN graphs as their input graphs, where k is tuned in the same search space to our method.

As can be observed, without the guidance of labels, our proposed SUBLIME outperforms all baselines on 3 out of 8 benchmarks and achieves the runner-up results on the rest datasets. This competitive performance benefits from the novel idea of guiding GSL with a self-enhanced learning target by graph contrastive learning. Besides, the result on ogbn-arxiv exhibits the scalability of SUBLIME.

We make other observations as follows. Firstly, the performance of structure-fixed GNNs (taking kNN graphs as input) is superior

Table 4: Test accuracy corresponding to different bootstrapping decay rate τ in structure refinement scenario.

| Dataset | Bootstrapping decay rate τ | | | | |
|----------|---------------------------------|---------|-------------|-------|------|
| | 1 | 0.99999 | 0.9999 | 0.999 | 0.99 |
| Cora | 82.1 | 83.2 | 84.2 | 82.4 | 70.9 |
| Citeseer | 71.9 | 72.6 | 73.5 | 73.4 | 72.6 |
| Pubmed | 80.1 | 80.3 | 81.0 | 80.8 | 80.5 |



(a) Test accuracy w.r.t. epoch.

(b) Contrastive loss value w.r.t. epoch.

Figure 3: Curves of training process on Cora dataset.

to conventional feature-based classifiers on most datasets, which shows the benefit of considering the underlying relationship among samples. Secondly, GSL methods achieve better performance than structure-fixed methods, indicating the significance of structure optimization. Thirdly, compared to supervised GSL methods, the unsupervised methods also achieve competitive results without the supervision of labels, which shows their effectiveness.

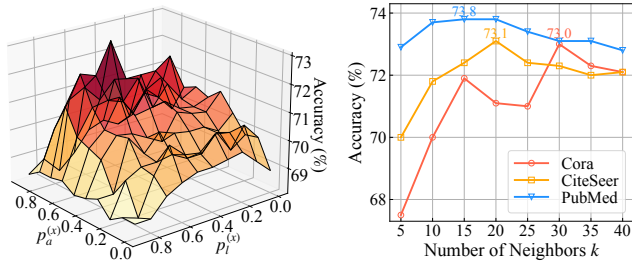
Node classification in structure refinement scenario. Table 2 summarizes the classification performance of each method in structure refinement scenario. We find that SUBLIME still shows very promising results against not only the self-supervised but also supervised methods, indicating that SUBLIME can leverage self-supervision signal to improve the original graphs effectively.

Node clustering in structure refinement scenario. In Table 3, we report the results of node clustering. Compared to baselines, our performance improvement illustrates that optimizing graph structures is indeed helpful to the clustering task. Meanwhile, the implementation of SUBLIME for node clustering task suggests that our learned topology can be applied to not only node classification task but also a wide range of downstream tasks.

5.3 Ablation Study (RQ2)

In our structure bootstrapping mechanism, the bootstrapping decay rate τ control the trade-off between updating anchor graph too sharply (with smaller τ) and too slowly (with larger τ). When $\tau = 1$, anchor graph is never updated and remains as a constant structure. To verify the effectiveness of the proposed mechanism, we adjust the value of τ and the results are shown in Table 4. We also plot the curves of accuracy and loss value w.r.t. training epoch with different τ , which are shown in Fig. 3.

As shown in Table 4, without structure bootstrapping mechanism ($\tau = 1$), the classification accuracy decreases by 1.5% on average, indicating the mechanism helps improve the quality of learned graphs. From Fig. 3(a), we further find an obvious drop after around 1500 iterations when $\tau = 1$, demonstrating the lack of effective guidance hurts the performance. When τ is within [0.999, 0.99999], the accuracy can converge to a high value (as shown in Fig. 3(a)),



(a) Accuracy w.r.t. feature masking rates. (b) Accuracy w.r.t. number of neighbors.

Figure 4: Sensitivity of hyper-parameters $p^{(x)}$ and k .

meaning that SUBLIME can learn a stable and informative structure with the bootstrapping mechanism. However, the performance declines with τ becoming smaller, especially on Cora dataset. We conjecture that with sharp updating, the anchor graph tends to be polluted by the learned graph obtained in the early training stage, which fails to capture accurate connections. Another problem caused by a too small τ is the unstable training, which can be seen in Fig. 3(a) and 3(b).

5.4 Sensitivity Analysis (RQ3)

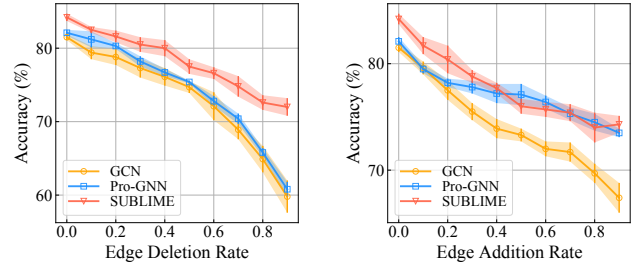
Using the structure inference case, we investigate the sensitivity of critical hyper-parameters in SUBLIME, including the probabilities $p^{(x)}$, $p^{(a)}$ for data augmentation and the number of neighbors k in kNN for sparsification and learner initialization. The discussion for $p^{(x)}$ and k are provided below while the analysis for $p^{(a)}$ is given in Appendix G.

Feature masking probability $p^{(x)}$. Fig. 4(a) shows the performance under different combinations of masking probabilities of two views on Cora dataset. We observe that the value of $p_a^{(x)}$ between 0.6 and 0.8 produces higher accuracy. Compared to $p_a^{(x)}$, SUBLIME is less sensitive to the choice of $p_l^{(x)}$, suggesting a good performance when $p_l^{(x)} \in [0, 0.7]$. When $p^{(x)}$ is larger than 0.8, the features will be heavily undermined, resulting worse results.

Number of neighbors k . To investigate its sensitivity, we search the number of neighbors k in the range of $\{5, 10, \dots, 40\}$ for three datasets. As is demonstrated in Fig. 4(b), the best selection for each dataset is different, i.e., $k = 30$ for Cora, $k = 20$ for Citeseer, and $k = 15$ for Pubmed. A common phenomenon is that a too large or too small k results in poor performance. We conjecture that an extremely small k may limit the number of beneficial neighbors, while an overlarge k causes some noisy connections.

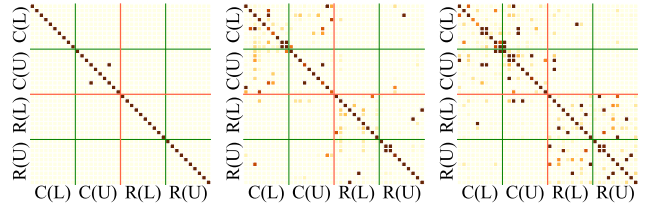
5.5 Robustness Analysis (RQ4)

To evaluate the robustness of SUBLIME against adversarial graphs, we randomly remove edges from or add edges to the original graph structure of Cora dataset and validate the performance on the corrupted graphs. We change the ratios of modified edges from 0 to 0.9 to simulate different attack intensities. We compared our method to GCN [22] and Pro-GNN [20], a supervised graph structure method for graph adversarial defense. As we can see in Fig. 5, SUBLIME consistently achieves better or comparable results in both settings. When the edge deletion rates become larger, our method shows more significant performance gains, indicating that SUBLIME has stronger robustness against serious structural attacks.



(a) Accuracy w.r.t. edge deletion rate.

(b) Accuracy w.r.t. edge addition rate.

Figure 5: Test accuracy in the scenarios where graph structures are perturbed by edge deletion or addition.

(a) Original graph.

(b) Graph learned by Pro-GNN.

(c) Graph learned by SUBLIME.

Figure 6: Heatmaps of the subgraph adjacency matrices of (a) the original graph with self-loop, the graph learned by (b) Pro-GNN and (c) SUBLIME on Cora dataset. A block in darker color indicates a larger edge weight between two nodes.

5.6 Visualization (RQ5)

To investigate what kind of graph structure is learned by SUBLIME, we select a subgraph from Cora dataset with nodes in two categories and visualize the edge weights in original graph, graphs learned by Pro-GNN and SUBLIME, respectively. The selected categories are Case base (C) and Rule learning (R), each of which has 10 labeled nodes (L) and 10 unlabeled nodes (U). Note that the labels of the labeled nodes are used to refine the graph structures in Pro-GNN, but are not used to optimize topology in SUBLIME. As we can see in Fig. 6, numerous intra-class edges are learned by SUBLIME, while the learned inter-class edges are far fewer than intra-class edges. In contrast, the original graph only provides scarce intra-class edges. We conclude that SUBLIME can learn connections between two nodes sharing similar semantic information, which improves the quality of graph topology. Moreover, in Pro-GNN, there are more connections built across labeled nodes than unlabeled nodes, indicating an edge distribution bias in the graph learned by such a supervised method. Conversely, SUBLIME equally constructs edges across all nodes belonging to the same class as each node can receive the essential supervision from the contrastive objective.

6 CONCLUSION

In this paper, we make the first investigation on the problem of unsupervised graph structure learning. To tackle this problem, we design a novel method, SUBLIME, which is capable of leveraging data itself to generate optimal graph structures. To learn graph structures, our method uses contrastive learning to maximize the agreement between the learned topology and a self-enhanced learning target. Extensive experiments demonstrate the superiority of SUBLIME and rationality of the learned structures.

ACKNOWLEDGMENTS

The corresponding author is Shirui Pan. This work was supported by an ARC Future Fellowship (No. FT210100097).

REFERENCES

- [1] Arthur Asuncion and David Newman. 2007. UCI machine learning repository.
- [2] Aleksandar Bojchevski, Yves Matkovic, and Stephan Günnemann. 2017. Robust spectral clustering for noisy data: Modeling sparse corruptions improves latent embeddings. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 737–746.
- [3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*.
- [4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.
- [5] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. 2018. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision*. 132–149.
- [6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*. PMLR, 1597–1607.
- [7] Yu Chen, Lingfei Wu, and Mohammed Zaki. 2020. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *Advances in Neural Information Processing Systems* 33 (2020).
- [8] Zewen Chi, Li Dong, Furu Wei, Nan Yang, Saksham Singhal, Wenhui Wang, Xia Song, Xian-Ling Mao, Heyan Huang, and Ming Zhou. 2021. InfoXLM: An Information-Theoretic Framework for Cross-Lingual Language Model Pre-Training. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.
- [9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, Vol. 29. 3844–3852.
- [10] Hilmi E Egilmez, Eduardo Pavez, and Antonio Ortega. 2017. Graph learning from data under Laplacian and structural constraints. *IEEE Journal of Selected Topics in Signal Processing* 11, 6 (2017), 825–841.
- [11] Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. 2021. SLAPS: Self-Supervision Improves Structure Learning for Graph Neural Networks. In *Advances in Neural Information Processing Systems*.
- [12] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. 2019. Learning discrete structures for graph neural networks. In *International Conference on Machine Learning*. PMLR, 1972–1982.
- [13] John Giorgi, Osvald Nitski, Bo Wang, and Gary Bader. 2021. DeCLUTR: Deep Contrastive Learning for Unsupervised Textual Representations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 879–895.
- [14] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, koray kavukcuoglu, Remi Munos, and Michal Valko. 2020. Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning. In *Advances in Neural Information Processing Systems*, Vol. 33. 21271–21284.
- [15] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1025–1035.
- [16] John A Hartigan and Manchek A Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)* 28, 1 (1979), 100–108.
- [17] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *Advances in Neural Information Processing Systems*, Vol. 33. 22118–22133.
- [18] Ming Jin, Yixin Liu, Yu Zheng, Lianhua Chi, Yuan-Fang Li, and Shirui Pan. 2021. ANEMONE: Graph Anomaly Detection with Multi-Scale Contrastive Learning. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3122–3126.
- [19] Ming Jin, Yizhen Zheng, Yuan-Fang Li, Chen Gong, Chuan Zhou, and Shirui Pan. 2021. Multi-Scale Contrastive Siamese Networks for Self-Supervised Graph Representation Learning. In *International Joint Conference on Artificial Intelligence*.
- [20] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 66–74.
- [21] Thomas N Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. In *Neural Information Processing Systems Workshop*. 1–3.
- [22] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.
- [23] Johannes Klicpera, Stefan Weissenberger, and Stephan Günnemann. 2019. Diffusion improves graph learning. *Advances in Neural Information Processing Systems* 32 (2019), 13354–13366.
- [24] Yixin Liu, Zhao Li, Shirui Pan, Chen Gong, Chuan Zhou, and George Karypis. 2021. Anomaly Detection on Attributed Networks via Contrastive Self-Supervised Learning. *IEEE Transactions on Neural Networks and Learning Systems* (2021).
- [25] Yixin Liu, Shirui Pan, Ming Jin, Chuan Zhou, Feng Xia, and Philip S Yu. 2021. Graph self-supervised learning: A survey. *arXiv preprint arXiv:2103.00111* (2021).
- [26] Travis Martin, Brian Ball, and Mark EJ Newman. 2016. Structural inference for uncertain networks. *Physical Review E* 93, 1 (2016), 012306.
- [27] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. 2012. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, Vol. 8. 1.
- [28] Andrew Y Ng, Michael I Jordan, and Yair Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*. 849–856.
- [29] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).
- [30] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. 2018. Adversarially regularized graph autoencoder for graph embedding. In *International Joint Conference on Artificial Intelligence*. 2609–2615.
- [31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* 32 (2019), 8026–8037.
- [32] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. 2020. Graph representation learning via graphical mutual information maximization. In *Proceedings of The Web Conference 2020*. 259–270.
- [33] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.
- [34] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.
- [35] Kihyuk Sohn. 2016. Improved deep metric learning with multi-class n-pair loss objective. In *Advances in Neural Information Processing Systems*. 1857–1865.
- [36] Yue Tan, Guodong Long, Lu Liu, Tianyi Zhou, Qinghua Lu, Jing Jiang, and Chengqi Zhang. 2022. Fedproto: Federated prototype learning over heterogeneous devices. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
- [37] Antti Tarvainen and Harri Valpola. 2017. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems*. 1195–1204.
- [38] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. 2014. Learning deep representations for graph clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28.
- [39] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax. In *International Conference on Learning Representations*.
- [40] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.
- [41] Sheng Wan, Yibing Zhan, Liu Liu, Baosheng Yu, Shirui Pan, and Chen Gong. 2021. Contrastive Graph Poisson Networks: Semi-Supervised Learning with Extremely Limited Labels. *Advances in Neural Information Processing Systems* 34 (2021).
- [42] Chun Wang, Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Attributed Graph Clustering: A Deep Attentional Embedding Approach. In *International Joint Conference on Artificial Intelligence*. 3670–3676.
- [43] Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. 2017. Mgae: Marginalized graph autoencoder for graph clustering. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 889–898.
- [44] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315* (2019).
- [45] Ruijia Wang, Shuai Mou, Xiao Wang, Wanpeng Xiao, Qi Ju, Chuan Shi, and Xing Xie. 2021. Graph Structure Estimation Neural Networks. In *Proceedings of the Web Conference 2021*. 342–353.
- [46] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community preserving network embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

- [47] Yingheng Wang, Yaosen Min, Xin Chen, and Ji Wu. 2021. Multi-view Graph Contrastive Representation Learning for Drug-Drug Interaction Prediction. In *Proceedings of the Web Conference 2021*. 2921–2933.
- [48] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24.
- [49] Rongkai Xia, Yan Pan, Lei Du, and Jian Yin. 2014. Robust multi-view spectral clustering via low-rank and sparse decomposition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28.
- [50] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.
- [51] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. 2015. Network representation learning with rich text information. In *International Joint Conference on Artificial Intelligence*.
- [52] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems* 33 (2020), 5812–5823.
- [53] Donghan Yu, Ruohong Zhang, Zhengbao Jiang, Yuexin Wu, and Yiming Yang. 2020. Graph-revised convolutional network. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 378–393.
- [54] Junliang Yu, Hongzhi Yin, Jundong Li, Qinyong Wang, Nguyen Quoc Viet Hung, and Xiangliang Zhang. 2021. Self-Supervised Multi-Channel Hypergraph Convolutional Network for Social Recommendation. In *Proceedings of the Web Conference 2021*. 413–424.
- [55] Xiaotong Zhang, Han Liu, Qimai Li, and Xiao-Ming Wu. 2019. Attributed Graph Clustering via Adaptive Graph Convolution. In *International Joint Conference on Artificial Intelligence*. 4327–4333.
- [56] Yin Zhang, Fanglin An, and Jun Ye. 2022. A Vertical Federation Framework Based on Representation Learning. In *2021 International Conference on Big Data Analytics for Cyber-Physical System in Smart City*. Springer, 627–633.
- [57] Yizhen Zheng, Ming Jin, Shirui Pan, Yuan-Fang Li, Hao Peng, Ming Li, and Zhao Li. 2021. Towards Graph Self-Supervised Learning with Contrastive Adjusted Zooming. *arXiv preprint arXiv:2111.10698* (2021).
- [58] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Qiang Liu, Shu Wu, and Liang Wang. 2021. Deep Graph Structure Learning for Robust Representations: A Survey. *arXiv preprint arXiv:2103.03036* (2021).
- [59] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*. 2069–2080.

A NOTATIONS

In this paper, we denote scalars with letters (e.g., k), column vectors with boldface lowercase letters (e.g., \mathbf{x}), matrices with boldface uppercase letters (e.g., \mathbf{X}), and sets with calligraphic fonts (e.g., \mathcal{V}). The frequently used notations are listed in Table 5.

Table 5: Frequently used notations.

| Notation | Description |
|--|--|
| $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ | The (original) graph. |
| n, m, d | The number of nodes/edges/features. |
| $\mathbf{A} \in [0, 1]^{n \times n}$ | The (original) adjacency matrix. |
| $\mathbf{X} \in \mathbb{R}^{n \times d}$ | The feature matrix. |
| $\mathcal{G}_l = (\mathbf{S}, \mathbf{X})$ | The learned graph / Learner graph view. |
| $\mathbf{S} \in [0, 1]^{n \times n}$ | The learned adjacency matrix. |
| $\tilde{\mathbf{S}} \in \mathbb{R}^{n \times n}$ | The sketched adjacency matrix. |
| $\mathbf{E} \in \mathbb{R}^{n \times d}$ | The embedding matrix. |
| $\mathcal{G}_a = (\mathbf{A}_a, \mathbf{X})$ | Anchor graph view. |
| $\mathbf{A}_a \in [0, 1]^{n \times n}$ | The anchor adjacency matrix. |
| $\tilde{\mathcal{G}}_l, \tilde{\mathcal{G}}_a$ | The augmented learner/anchor view. |
| d_1, d_2 | The dimension of node representation/projection. |
| $\mathbf{H}_l, \mathbf{H}_a \in \mathbb{R}^{n \times d_1}$ | The representation matrix of learner/anchor view. |
| $\mathbf{Z}_l, \mathbf{Z}_a \in \mathbb{R}^{n \times d_2}$ | The projected representation matrix of learner/anchor view. |
| \mathcal{L} | The contrastive loss function. |
| $p_\omega(\cdot)$ | The graph learner with parameter ω . |
| $q(\cdot)$ | The post-processor. |
| $\mathcal{T}_{fm}(\cdot), \mathcal{T}_{ed}(\cdot)$ | The feature masking/edge dropping augmentation. |
| $f_\theta(\cdot)$ | The GNN-based encoder with parameter θ . |
| $g_\varphi(\cdot)$ | The MLP-based projector with parameter φ . |
| k | The number of neighbors in kNN. |
| $p^{(x)}, p^{(a)}$ | The masking/dropping probability for $\mathcal{T}_{fm}(\cdot)/\mathcal{T}_{ed}(\cdot)$. |
| τ, c | The decay rate/interval for bootstrapping updating. |
| \odot | The Hadamard operation. |
| \cdot^T | The transposition operation. |

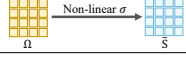

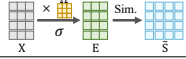
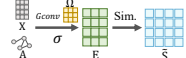
B ANALYSIS OF GRAPH LEARNERS

In Table 6, We summarize the properties of the proposed graph learners, including their memory, parameter and time complexity. For metric learning-based graph learners, we consider the complexities with locality-sensitive kNN sparsification post-processing [11] where the neighbors are selected from a batch of nodes (batch size = b_1). We provide our analysis as follows:

- Since FGP learner can model each edge independently and directly, it enjoys several advantages such as the flexibility to model connections and low time complexity. However, its $O(n^2)$ space complexity makes it hard to be applied to the modeling of large-scale graphs.
- Among all metric learning-based learners, attentive learner has the lowest parameter and time complexity w.r.t. dimension d . It is suitable for the situation with high feature dimension and low correlation between features.
- Compared to attentive learner, MLP and GNN learner require larger space and time complexity to consider the correlation between features and original topology.
- With the effective kNN sparsification, the memory and time complexity are reduced from $O(n^2)$ to $O(n)$, which improves the scalability of the metric learning-based learners.

Considering these properties, we allocate the suitable learner for each dataset. Specifically, for small datasets whose node numbers are less than 3000 (e.g., Cora), we use FGP learners to model them due to the flexibility and acceptable complexity. For larger datasets with high-dimensional raw features (e.g., Citeseer), we

Table 6: Properties of graph learners.

| Learner | Sketch | Memory | Params | Time |
|-----------|--|---------------|-----------|--------------------------|
| FGP |  | $O(n^2)$ | $O(n^2)$ | $O(1)$ |
| Attentive |  | $O(ndL + nk)$ | $O(dL)$ | $O(ndL + ndb_1)$ |
| MLP |  | $O(ndL + nk)$ | $O(d^2L)$ | $O(nd^2L + ndb_1)$ |
| GNN |  | $O(ndL + nk)$ | $O(d^2L)$ | $O(mdL + nd^2L + ndb_1)$ |

Algorithm 1: The training algorithm of SUBLIME

Input: Feature matrix \mathbf{X} ; Adjacency matrix \mathbf{A} (optional); Number of nearest neighbors k ; Bootstrapping decay rate and interval τ, c ; Feature masking probability $p_l^{(x)}, p_a^{(x)}$; Edge dropping probability $p^{(a)}$; Temperature t ; Number of epochs E .

Output: Learned Adjacency Matrix \mathbf{S}

```

1 Initialize parameters  $\omega, \theta, \varphi$ ;
2 if  $\mathbf{A}$  is provided then
3   | Initialize the anchor adjacency matrix by:  $\mathbf{A}_a \leftarrow \mathbf{A}$ ;
4 else
5   | Initialize the anchor adjacency matrix by:  $\mathbf{A}_a \leftarrow \mathbf{I}$ ;
6 end
7 for  $e = 1, 2, \dots, E$  do
8   | Calculate  $\tilde{\mathbf{S}}$  with graph learner  $p_\omega$  by Eq. (1) or (2);
9   | Calculate  $\mathbf{S}$  with post-processor  $q(\tilde{\mathbf{S}})$  by Eq. (6) - Eq. (8);
10  | Establish two graph views by  $\mathcal{G}_l = (\mathbf{S}, \mathbf{X}), \mathcal{G}_a = (\mathbf{A}_a, \mathbf{X})$ ;
11  | Obtain augmented graph views  $\tilde{\mathcal{G}}_l, \tilde{\mathcal{G}}_a$  by Eq. (9) - Eq.
    | (11) with probability  $p_l^{(x)}, p_a^{(x)}, p^{(a)}$ ;
12  | Calculate node representations  $\mathbf{H}_l, \mathbf{H}_a$  with encoder  $f_\theta$ 
    | by Eq. (12);
13  | Calculate projections  $\mathbf{Z}_l, \mathbf{Z}_a$  with encoder  $g_\phi$  by Eq. (13);
14  | Calculate the contrastive loss  $\mathcal{L}$  by Eq. (14);
15  | Update parameters  $\omega, \theta, \varphi$  by applying gradient descent;
16  if  $e \bmod c = 0$  then
17    | Bootstrapping update  $\mathbf{A}_a$  with decay  $\tau$  by Eq. (15);
18  end
19 end

```

choose attentive learner considering its low parameter/time complexity w.r.t. dimension d . For large-scale datasets with relevantly low feature dimensions (e.g., 20news), we adopt MLP learners to capture the correlation between features. In graph refinement scenarios where original graphs are available, GNN learners can be further considered to leverage the extra topology information.

C COMPLEXITY ANALYSIS

We analyze the time complexity of each component of SUBLIME. For graph learner, the complexity has been described in Table 6. The time complexity of post-processor is mainly contributed by sparsification, which is $O(ndb_1)$ for effective kNN and $O(n^2d)$ for

Table 7: Statistics of datasets.

| Dataset | Nodes | Edges | Classes | Features | Label Rate |
|------------|---------|-----------|---------|----------|------------|
| Cora | 2,708 | 5,429 | 7 | 1,433 | 0.052 |
| Citeseer | 3,327 | 4,732 | 6 | 3,703 | 0.036 |
| Pubmed | 19,717 | 44,338 | 3 | 500 | 0.003 |
| ogbn-arxiv | 169,343 | 1,166,243 | 40 | 128 | 0.537 |
| Wine | 178 | N/A | 3 | 13 | 0.056 |
| Cancer | 569 | N/A | 2 | 30 | 0.018 |
| Digits | 1,797 | N/A | 10 | 64 | 0.028 |
| 20news | 9,607 | N/A | 10 | 236 | 0.010 |

conventional kNN. In the contrastive learning module, the complexities of feature masking and edge dropping are $O(d)$ and $O(m)$, respectively. For the encoder and projector, the total complexity is $O(md_1L_1 + nd_1^2L_1 + nd_2^2L_2)$. For contrastive loss computation, the complexity is $O(n^2)$ for its full-graph version, $O(nb_2)$ for the mini-batch version, where b_2 is the batch size of contrastive learning.

D ALGORITHM

The training algorithm of SUBLIME is summarized in Algorithm 1.

E DATASETS

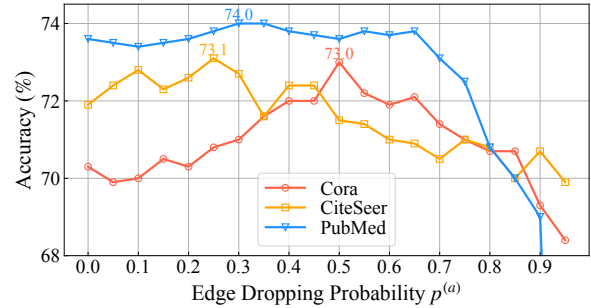
In Table 7, we summarize the statistics of benchmark datasets. The dataset splitting follows the previous works [7, 12]. Details of these datasets are introduced as follows.

- **Cora** [34] is a citation network where each node is a machine learning paper belonging to 7 research topics and each edge is a citation between papers.
- **Citeseer** [34] is a citation network containing 6 types of machine learning papers: Agents, AI, DB, IR, ML, and HCI. Nodes denote papers and edges denote citation relationships.
- **Pubmed** [27] is a citation network from the PubMed database, where nodes are papers about three diabetes types and edges are citations among them.
- **ogbn-arxiv** [17] is a citation network with Computer Science arXiv papers. The features are the embeddings of words in its title and abstract. The labels are 40 subject areas.
- **Wine** [1] is a non-graph dataset containing the results of a chemical analysis of 178 wines derived from three different cultivars. Features are the quantities of 13 constituents found.
- **Cancer** [1] is a binary classification dataset of diagnosis of breast tissues (malignant/benign). The features are computed from a digitized image of a breast mass.
- **Digits** [1] is a non-graph dataset containing handwritten digits in 10 classes. Each sample is a 8×8 image of a digit.
- **20news** [1] is a non-graph dataset comprising newsgroups posts on 20 topics. Following [12], we select 10 topics with 9,607 samples in our experiments.

F IMPLEMENTATION DETAILS

F.1 Computing Infrastructures

We implement SUBLIME using PyTorch 1.7.1 [31] and DGL 0.7.1 [44]. All experiments are conducted on a Linux server with an Intel Xeon 4214R CPU and four Quadro RTX 6000 GPUs.

**Figure 7: Sensitivity analysis for $p^{(a)}$.**

F.2 Evaluation Details

Through node classification tasks, we evaluate the quality of the learned structures by re-training a classifier with the learned structure as its constant input. Specifically, we use the learned adjacency matrices to train GCN-based classification models, and record the testing result with the highest validation accuracy. The averaged accuracy over five rounds of running is used to assess the classification performance. For ogbn-arxiv dataset, we utilize a three-layer GCN with 256 hidden units as the evaluation model. For the rest datasets, a two-layer GCN with 32 hidden units is employed.

For node clustering tasks, we evaluate the performance of our method by measuring the quality of the learned representations. Concretely, following the baseline methods [42, 55], we train our framework for a fixed number of epochs and apply K-means algorithm for 10 runs to group the learned representations. The representations are generated by the contrastive learning encoder f_θ taking learned graph $\mathcal{G}_l = (S, X)$ as its input without augmentation.

F.3 Hyper-parameter Specifications

We perform grid search to select hyper-parameters on the following searching space: the dimension of representation and projection is searched in $\{16, 32, 64, 128, 256, 512\}$; k on kNN is tuned amongst $\{5, 10, 15, 20, 25, 30, 35, 40\}$; feature masking probability $p^{(x)}$ is tuned from 0.1 to 0.9; edge dropping probability $p^{(a)}$ is searched in $\{0, 0.25, 0.5, 0.75\}$; the bootstrapping decay rate is chosen from $\{0.99, 0.999, 0.9999, 0.99999, 1\}$; and the learning rate of Adam optimizer is selected from $\{0.01, 0.001, 0.0001\}$. The temperature for contrastive loss is fixed to 0.2. The layer numbers of encoder (L_1), projector (L_2), and embedding network (L) are set to 2.

For our baselines, we reproduce the experiments using their official open-source codes or borrow the reported results in their papers. We carefully tune their hyper-parameters to achieve optimal performance. To compare fairly, we use the random seeds $\{0, 1, 2, 3, 4\}$ for all classification methods, and fix the seed to 0 for clustering methods.

G PARAMETER SENSITIVITY OF $p^{(a)}$

We vary the dropping rate $p^{(a)}$ from 0 to 0.95 on Cora, Citeseer and Pubmed datasets, and the results are shown in Fig. 7. As we can see, when $p^{(a)}$ is between 0.2 and 0.65, SUBLIME achieves better performance. When the edge dropping rate is overlarge, the structures on both views will be deteriorated, causing a sharp drop of performance.