LOBSTUR: A Local Bootstrap Framework for Tuning Unsupervised Representations in Graph Neural Networks

So Won Jeong
The University of Chicago
Chicago, Illinois, USA
sowonjeong@chicagobooth.edu

Claire Donnat
The University of Chicago
Chicago, Illinois, USA
cdonnat@uchicago.edu

ABSTRACT

Graph Neural Networks (GNNs) are increasingly used in conjunction with unsupervised learning techniques to learn powerful node representations, but their deployment is hindered by their high sensitivity to hyperparameter tuning and the absence of established methodologies for selecting the optimal models. To address these challenges, we propose LOBSTUR-GNN (**Lo**cal **B**oot**s**trap for **T**uning **U**nsupervised **R**epresentations in GNNs) i), a novel framework designed to adapt bootstrapping techniques for unsupervised graph representation learning. LOBSTUR-GNN tackles two main challenges: (a) adapting the bootstrap edge and feature resampling process to account for local graph dependencies in creating alternative versions of the same graph, and (b) establishing robust metrics for evaluating learned representations without ground-truth labels. Using locally bootstrapped resampling and leveraging Canonical Correlation Analysis (CCA) to assess embedding consistency, LOBSTUR provides a principled approach for hyperparameter tuning in unsupervised GNNs. We validate the effectiveness and efficiency of our proposed method through extensive experiments on established academic datasets, showing an 65.9% improvement in the classification accuracy compared to an uninformed selection of hyperparameters. Finally, we deploy our framework on a real-world application, thereby demonstrating its validity and practical utility in various settings.

CCS CONCEPTS

• **Computing methodologies** → **Cross-validation**; **Learning latent representations**; **Unsupervised learning**; **Neural networks**.

KEYWORDS

Graph Neural Network, Hyperparameter Tuning, Unsupervised Representations, Bootstrap, Cross-validation

ACM Reference Format:
So Won Jeong and Claire Donnat. 2025. LOBSTUR: A Local Bootstrap Framework for Tuning Unsupervised Representations in Graph Neural Networks. In *Proceedings of Machine Learning on Graphs in the Era of Artificial General Intelligence (MLoG-GenAI@KDD).* ACM, New York, NY, USA, 38 pages. https://doi.org/10.48550/arXiv.2505.14867

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 the author(s) 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.
MLoG-GenAI@KDD, August 03–07, 2025, Toronto, Canada
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.48550/arXiv.2505.14867

1 INTRODUCTION

With the expanding availability of network and spatial data in the sciences, Graph Neural Networks (GNNs) have emerged as a compelling approach to identify interaction patterns within complex systems. Examples include spatial transcriptomics [Dong and Zhang 2022; Zhu et al. 2018], where graph-based neural networks are used to learn representations of cell neighborhoods that can be correlated with cancer outcomes, and microbiome studies [Lamurias et al. 2022; Le et al. 2020] where they are used for genome assembly or to predict metabolite information from microbes. In many of these applications, scientists are increasingly interested in using GNNs in conjunction with unsupervised learning techniques for learning informative representations, due to the paucity of available labeled data, or as a way of automatically detecting structure or patterns [Dong and Zhang 2022; Lamurias et al. 2022; Le et al. 2020; Zhu et al. 2018]. Consequently, the last few years have seen the development of a number of unsupervised GNN methods for the sciences [Hu et al. 2021; Ishiai et al. 2024; Li et al. 2021; Partel and Wählby 2021; Zhang et al. 2020], usually adapting known methods for Euclidean data such as tSNE [van der Maaten and Hinton 2008], k-Means clustering, or UMAP [McInnes et al. 2018] to accommodate graph data and GNNs. These methods typically involve few hyperparameters, but their scope of application is typically confined to a specific data type or use case.

Within the methods community, on the other hand, recent advances in unsupervised node representation learning seem to have primarily been driven by contrastive learning [Stokes et al. 2020; You et al. 2021; Zhang et al. 2021]. This popular self-supervised learning framework has indeed demonstrated impressive performance for learning rich and versatile data representations across various domains. However, in the graph-setting, despite their impressive performance and promising results on academic benchmarks, these methods are not tuning-free, making them difficult to deploy in real-world applications. In fact, they rely heavily on selecting appropriate values for several of their hyperparameters, but incorrect hyperparameter values can lead to severely distorted data representations. We illustrate this effect in Figure 1 by showing how different choices of hyperparameters can decrease the accuracy of a linear classifier (trained on the learned node embeddings of a toy benchmark) from 73% to 30%, thus indicating a severe loss in embedding quality.

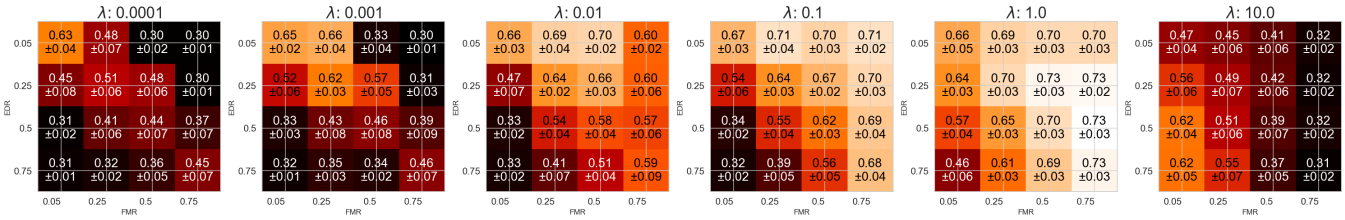


Figure 1: Cora. Evaluation of the CCA-SSG embeddings [Zhang et al. 2021], an unsupervised learning method, for each combination of the hyperparameters (loss parameter λ, edge drop rate (EDR), feature mask rate (FMR)). Each entry denotes the mean and standard deviation of the node classification accuracy of a linear classifier trained on the learned representations (averaged over 20 experiments).

Despite the empirical importance of hyperparameter tuning, there is currently no valid hyperparameter selection procedure for unsupervised GNN representation learning. In the methods community, new unsupervised learning approaches are commonly tested on established benchmark datasets, with hyperparameters selected based on performance in a downstream node classification task. However, this procedure essentially converts the problem into a supervised learning setting, making it unsuitable for genuinely unsupervised, real-world use cases.

Hyperparameter tuning in unsupervised settings is made difficult by two main challenges: (a) the absence of a clear ground truth or statistical framework for unsupervised learning, and (b) the lack of an established metric to evaluate the learned embeddings. To our knowledge, the only study that attempts to measure the quality of latent representations is that of Tsitsulin et al., which empirically evaluates various metrics. Yet, without a proper inference framework, pinpointing a suitable metric remains a significant challenge.

Contributions. In this paper, we propose the first bootstrapped-based method for selecting hyperparameters for unsupervised GNN representation learning. Hyperparameters can be broadly categorized into two types: (1) those that require tuning within a specific model (e.g. parameters in the loss function) and (2) those that involve tuning across a family of models (e.g. different GNN architectures or sizes). In this paper, we aim to tackle both, as both are essential to self-supervised learning. More specifically,

(1) We cast the learning of representations as an estimation problem: we posit that the learned representations correspond to a learned low-dimensional manifold, which must therefore be consistent under a noise model, as explicited in Section 2.
(2) To generate independent copies of the same graph, we propose a bootstrap procedure based on nonparametric modeling of the graph as a graphon [Su et al. 2020] (Section 3).
(3) To evaluate the quality of the embeddings learned on independent copies of the same graph in the absence of labels, we suggest using Canonical Correlation Analysis [Hotelling 1936] as a translation- and rotation-invariant tool to quantify the stability of the learned embedding spaces (Section 4).

2 PROBLEM FORMALIZATION

Establishing a framework for hyperparameter tuning in unsupervised learning requires us to address two fundamental questions: what are we aiming to estimate, and where does the randomness come from?

In the graph setting, the data is presented in two modalities: a feature matrix, and an adjacency matrix. Unsupervised learning can be framed as learning what information is shared across modalities, and what information is specific to each one in a condensed format. This approach is typically described in the data-integration literature using a latent variable space model [Bishop 1998; Hoff et al. 2002], which we adapt here for the graph domain.

Inference setting. We consider a graph G on n nodes with features $X \in \mathbb{R}^{n \times p}$, and denote by $A \in \{0,1\}^{n \times n}$ its corresponding (binary) adjacency matrix. We assume the graph is sampled from a graphon W (see for instance Gao et al.) — a non-parametric random graph model–, and that node features are a noisy transformation of the latent variable U_i:

$$\begin{aligned} &\forall i \in [n], U_i \sim \text{Unif}([0,1]), \\ &\forall j \in [n], A_{ij} \sim \text{Bernoulli}(W(U_i, U_j)), \\ &X_i \sim g(U_i) + \epsilon_i, \end{aligned} \tag{1}$$

where $W(U_i, U_j)$ denotes the graphon function evaluated at the latent positions U_i and U_j, and ϵ_i denotes some independent, mean-zero noise. This model allows us to reason on the randomness of the generation procedure without making assumptions on the specifics of the graph generation process. While graphons are known to generate dense graphs, their output can be sparsified by scaling W by a sparsity factor that tends to 0 as $n \to \infty$, e.g. $\rho_n = \frac{\log(n)}{n}$ [Davison and Austern 2023; Gaucher and Klopp 2021].

Learning Conditional or Marginal Representations. In the unsupervised context, we differentiate between two main scenarios based on whether the goal is to learn representations conditioned on the realized U_i or not.

(a) Learning ***marginal data representations*** *that do not depend on the realized U_is* : In some applications, embeddings are assumed to capture clusters or predict specific outcomes [Hu et al. 2021; Wu et al. 2022]. In these cases, the learned embeddings serve as a way to extract inherent structure from the data — such as clusters, an underlying manifold, or more generally, functions of the graphon W — that should be consistent across datasets of the same type. For example, in single-cell transcriptomics studies (e.g. Wu et al. [2022] and Hu et al. [2021]), GNNs are employed to learn embeddings that capture patterns in cell-neighborhood interactions. Here, the

assumption is that the clusters derived from one dataset should be reproducible on another dataset drawn from the same distribution. An oracle data generation procedure would therefore generate unseen data using the full model in (1), resampling latent variables U_i to generate new nodes and edges.

(b) Learning ***conditional data representations****, i.e. conditional on the realized* U_i*s*: However, in other applications, these assumptions do not hold: there might not necessarily be an obvious notion of "another similar graph". In citation networks, for example, where nodes represent users, the focus shifts to learning properties specific to individual nodes. In this case, data generation procedures must accommodate another type of randomness, this time conditioned on U_i: randomness arises solely in sampling the edges and features, as per the last two lines of (1).

In both settings, however, the quality of the learned embeddings might be evaluated based on their reproducibility, or the alignment between the latent structure stemming from representations learned on one dataset to those learned on another. Devising a criterion leveraging this notion would require two main components: (a) a data generation procedure, to create independent draws of the same datasets (Section 3), and (b) a metric to measure the alignment between representations (Section 4).

3 A LOCAL GRAPH BOOTSTRAPPING PROCEDURE

In the GNN literature, data splitting and resampling are usually done in one of two ways: by resampling the nodes or by resampling the edges. However, in the unsupervised setting, these two sampling procedures are not necessarily suitable for two reasons. First, this type of sampling can considerably disrupt the structure of the graph (by thinning nodes or edges, respectively), as reflected in Figure 2, Table 6 and Figure 8 in the Appendix — an indication that the stochastic process underpinning the graph creation is not correctly simulated. Second, these procedures require the specification of the node (respectively edge) drop rate. However, without any theoretical underpinning, it is difficult to set these correctly, as the distance between the original graph and the thinned ones increases monotonously with the drop rate.

Relationship with existing literature. Relatively few studies have been conducted on extending bootstrapping to network data [Chen and Lei 2018; Hoff 2007; Levin and Levina 2021; Li et al. 2020]. Most of these works assume a specific distribution for the graph, also typically assuming latent variables for each node (e.g. a random product graph [Levin and Levina 2021]). The edges are assumed to be drawn from a Bernoulli distribution with probability $f(U_i, U_j)$, where f is a parametric function. For instance, in the random dot product graph [Levin and Levina 2021]), edges are assumed to be sampled as $A_{ij} \sim \text{Bernoulli}(U_i^\top U_j)$. However, these works (a) rely on strong parametric assumptions about the graph's nature, which, if unverified, can lead to resampled graphs that differ substantially from the original; (b) usually do not address the challenge of simultaneously resampling both edges and node features; and (c) typically generate new graphs conditioned on the imputed latent variables, but are not easily extendable to the marginal case, which would require resampling the node representations U_i's themselves.

By contrast, we propose a nonparametric technique for resampling graphs based on the model detailed in (1), thus requiring minimal assumptions about the underlying graph structure.

3.1 The Oracle Case

We begin by assuming that, for each latent variable U_i, we have oracle knowledge of its k-nearest neighbors. We denote the resulting directed k-nearest neighbor graph as $\mathcal{G}_{knn}$. Under sufficiently smooth functions W and g (as defined in the next paragraphs), for a given node i, its neighbors in $\mathcal{G}_{knn}$ have similar distributions, and can thus be viewed as alternative realizations of the same underlying stochastic process (conditioned on U).

We leverage this observation to propose a bootstrap procedure conditioned on the realized U_i:

(1) *Feature resampling:* we resample the features of each node by drawing at random a feature vector from one of k-nearest neighbors. This ensures preserving the covariance between features by sampling full vectors.
(2) *Edge rewiring:* let $\mathcal{N}_m(i)$ denote the mth closest neighbor of node i according to the oracle graph $\mathcal{G}_{knn}$. For each pair of node (i, j), sample an edge with probability $\hat{p}_{ij} = \frac{1}{k}\sum_{m=1}^{k} A_{\mathcal{N}_m(i),j}$, effectively estimating the underlying probability $\mathbb{P}[A_{ij} = 1|U_i, U_j]$. An efficient procedure for resampling is presented in Algorithm 2.

To extend this procedure to generate *marginally-resampled* graphs, we propose simply sampling with replacement nodes (which effectively implies resampling the U_i), and applying the same procedure as above. The whole procedure is described in more detail in Algorithm 3. In either case (marginally or conditionally on U), this framework preserves local latent-space similarities while generating plausible bootstrap replicates of the graph.

Algorithm 1 Non-parametric Resampling of Node Features

1: **Input:** Graph G with node features $\{X_i\}_{i=1}^n$, $\mathcal{G}_{knn}$ k-nearest neighbor graph on U.
2: **for** each node $i \in [n]$ **do**
3: Identify the set of neighboring nodes $N(i) = \{j : j \sim i\}$ in the graph $\mathcal{G}_{knn}$,
4: Construct the candidate set for resampling: $C_i = \{X_i\} \cup \{X_j\}_{j \in N(i)}$.
5: Resample the feature vector for node i by selecting a vector uniformly from C_i: $X_i^{\text{new}} \sim \text{Unif}(C_i)$.
6: **end for**
7: **Output:** Resampled node features $\{X_i^{\text{new}}\}_{i=1}^n$.

The following theorem characterizes the consistency of the procedure in deriving nodes with similar features.

Theorem 3.1. *Assume that* $\mathcal{G}_{knn}$*, the directed* k*- nearest neighbor graph induced by the latent variable* $\{U_i\}_{i=1}^n$ *is known, with* k *such that* $\lim_{n\to\infty} \frac{k}{n} = 0$*. Suppose* g *is an* α*-Hölder-continuous function on the interval* $[0, 1]$*, so that there exists a constant* C *such that:* $|g(U_i) - g(U_j)| \leq C|U_i - U_j|^\alpha$ *for* $\alpha > 0$*.*

Let $\mathcal{X}$ *denote the domain of the features (so that for each node* i*, its features are denoted* $X_i \in \mathcal{X}$*). Then, the procedure described in*

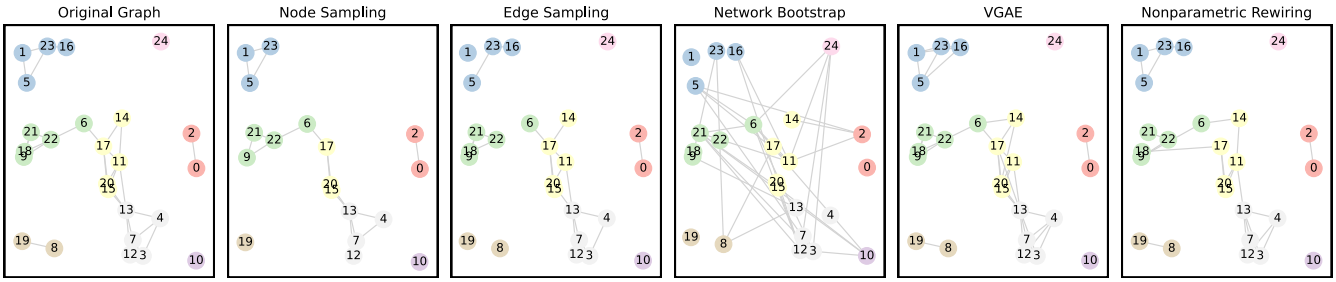


Figure 2: Illustration of different techniques for generating new copies of a simple graph (left-most image). The original graph has a distinctive community structure. Note that node sampling or edge sampling randomly removes either nodes or edges, disrupting the original graph structure.

Algorithm 1 is asymptotically consistent in that for any set $\mathcal{A} \in \mathcal{X}$:

$$\forall j \in \mathcal{N}_{knn}(i), \lim_{n\to\infty} \left|\mathbb{P}(X_j \in \mathcal{A}|U_j) - \mathbb{P}(X_i \in \mathcal{A}|U_i)\right| = 0,$$

where $\mathcal{N}_{knn}(i)$ denotes any of the k-nearest neighbors of i.

Proof. Under (1), $\forall i, \quad X_i = g(U_i) + \epsilon_i$, where ϵ_i is independent, identically distributed centered noise, and $g(U_i)$ is the expectation of X given the latent U_i. Since the ϵ are assumed to be i.i.d, we can write for any nodes i and j:

$$X_i \overset{d}{=} g(U_i) + \epsilon_j = g(U_i) + X_j - g(U_j).$$

The quantity $g(U_i) - g(U_j)$ represents the bias in using the expectation X_j to approximate the distribution of X_i, and since g is assumed to be Hölder-continuous: $\|g(U_i) - g(U_j)\| \leq c|U_i - U_j|^\alpha$.

Consider now j to be chosen to be one of the k-nearest neighbors of node i. $|U_i - U_j|^\alpha$ is a monotonously decreasing function of n, and with high probability (over the distribution of $U_1, \cdots, U_n$), we have $|U_i - U_j| \leq c_0 \frac{k}{n}$, for all $j \in \mathcal{N}(U_i)$, and a constant c_0. Therefore, $\|g(U_i) - g(U_j)\|$ tends to 0 (in probability) as n goes to ∞. Therefore, by Slutsky's lemma, as n goes to ∞, $\{X_j|U_j\}_{j\in\mathcal{N}_{knn}(i)} \xrightarrow{d} X_i|U_i$. □

The following theorem highlights the consistency of the edge rewiring procedure.

Theorem 3.2. *Suppose that the k_n-nearest neighbor graph $\mathcal{G}_{knn}$ induced by the latent variables $\{U_i\}_{i=1}^n$ is known, where k_n is such that $\lim_{n\to\infty} \frac{k_n}{n} = 0$, and $\lim_{n\to\infty} k_n = 0$. Suppose that W is an α-Hölder graphon function [Gao et al. 2015] (see definition A.1 in the Appendix) with $\alpha \in (0, 1]$.*

Then, the quantity $\hat{p}_{ij} = \frac{1}{k_n}\sum_{m=1}^{k_n} A_{\mathcal{N}_m(i),j}$ is a consistent estimator of p_{ij} in the sense that:

$$\lim_{n\to\infty} \hat{p}_{ij} = \mathbb{P}[A_{ij}|U_i, U_j].$$

Proof. The proof follows a similar argument to the previous theorem and is deferred to Appendix A.2. □

Remark 3.3. We note that the noise ϵ_i on the features does not need to be globally identically distributed for the previous construction to hold. Instead, since the procedure only relies on the k-nearest neighborhood of each node, it suffices to assume that these properties hold locally.

Algorithm 2 Non-parametric Resampling of Edges

1: **Input:** Graph $G = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes; flattened list of edge stems

$$L = \{u \mid u \in E[:, 0]\} \cup \{v \mid v \in E[:, 1]\},$$

where $E \in \mathbb{R}^{|\mathcal{E}|\times 2}$, and k-nearest neighbor graph $\mathcal{G}_{\text{knn}}$ on U.

2: Initialize an empty graph G' with n nodes.

3: **while** $\texttt{len}(L) > 0$ **do**

4: Sample a source node u uniformly at random from L and remove it: $u \leftarrow \text{pop}(L)$.

5: Sample a target node v from

$$v \sim L \cap \left(\bigcup_{m=1}^{k} \mathcal{N}_A\left(\mathcal{N}_m^{\text{knn}}(u)\right)\right),$$

where $\mathcal{N}_A(i)$ denotes the set of neighbors of node i in G, and $\mathcal{N}_m^{\text{knn}}(u)$ denotes the m-th nearest neighbor of node u in $\mathcal{G}_{\text{knn}}$.

6: Remove the selected node v from L.

7: Add an undirected edge between u and v in G'.

8: **end while**

9: **Output:** Resampled edge structure $\{A_{ij}^{\text{new}}\}_{i,j=1}^n$.

3.2 The Noisy Setting

The resampling procedure highlighted in the previous paragraph requires oracle knowledge of the k-NN graph on the latent U. In practice, the graph $\mathcal{G}_{knn}$ has to be estimated from the data. To this end, we suggest two strategies:

Solution 1: Constructing 2 kNN graphs based on features and topology, respectively. The theorems established earlier ensure the consistency of our resampling procedure, provided that the kNN graph is constructed independently of the data. Therefore, one approach is to build a kNN graph based on a notion of graph distance (e.g., shortest-path, shared -neighbors, or Jaccard similarity, depending on what metric is adapted for the graph), which can then be used for resampling the node features, while using a kNN graph based on feature similarity for resampling the edges. This approach is expected to perform well as long as the graph distance chosen is reflective of the underlying distance (e.g. shortest-path distance for homophilic graphs, or shared neighbors for more general classes of

graphs), and the features are sufficiently informative.

Solution 2: Using solely the kNN Graph from the adjacency matrix. In practice, while solution 1 may work well for feature-based resampling, the kNN induced by the features (and used to resample edges) might not be as reliable as the one induced by the edges (see Table 5 in the Appendix). This is because, in high-dimensional feature spaces, kNN suffers from the curse of dimensionality, making it difficult to ensure the consistency of the kNN graph. As an alternative, one can define the kNN graph solely based on the graph structure for all components of the algorithm. While this approach does not guarantee theoretical consistency in estimating the relevant quantities, it exhibits promising empirical performance, as shown in the experiments.

3.3 Validation of Bootstrap Samples

To evaluate the quality of bootstrapped samples, we propose bootstrapping different graphs (synthetic and real), and to compare key graph statistics—including node and edge counts, average degree, and degree distribution—, against those of the original graph.

Table 1 summarizes these comparisons for a synthetic graphon function and three well-established graph benchmarks. Additional results for more datasets and graphon settings, including the effect of the choice of k, are provided in Appendix F.2. While not exhaustive, these comparisons help assess whether the structural properties of the original graph are preserved in the bootstrapped samples. In particular, we note that our approach typically produces graphs with a closer average degree and edge count than other methods (see for instance Table 6 and Figure 8 in the Appendix). When the underlying graph is a graphon, our model is in fact very good at reproducing graphs with the similar statistics (see Table 7, 8, 9, 10). On real datasets, our method seems to produce reasonable copies of the same graph as well, as reflected by similar average degrees and number of connected components (Table 11 and 12). However, the graphon assumption upon which our method relies seems to hit a limit in the ability of the method to reproduce a graph with as many triangles (see results Cora in Table 1).

4 EVALUATION METRICS

If the generation of independent copies of the same graph poses a significant challenge, determining an appropriate evaluation metric in the absence of known labels poses another. While contrastive learning is based on the idea of turning unsupervised learning into a supervised problem by learning to recognize positive pairs, we note that we cannot use this objective as our hyperparameter tuning criterion. First, the loss function is designed to optimize the model's internal objective, which may not necessarily reflect meaningful patterns or structures in the data. For example, minimizing the loss in contrastive learning could lead to trivial solutions [Hua et al. 2021; Tsitsulin et al. 2023] that satisfy the objective but fail to capture important relationships in the graph, or the model may overfit to spurious correlations in the data, such as background features in images or noise in graphs [Chen et al. 2020]. Second, the scale of the loss function can vary with different hyperparameters (particularly for those who directly impact the loss function, as for the composite loss used in Zhang et al. [2021]), complicating direct comparisons even within the same model architecture. To ensure robust evaluation, it is essential to employ a separate, universal metric that directly evaluates the learned embeddings to assess model performance.

However, due to the nonconvexity of the method, we do not expect the learned embeddings to be close, even when fitted by the same algorithm on the same dataset. Scale and location can in fact vary greatly from one run to the next. To remedy these issues, we propose here using a procedure based on Canonical Correlation Analysis (CCA) [Hotelling 1936]. Canonical correlation analysis is a classical method for finding the correspondence between two datasets on the same samples by finding linear transformations of X and Y that maximizes their correlation. The CCA objective can be written as a prediction problem:

$$\begin{aligned} \hat{U}, \hat{V} \in \underset{U \in \mathbb{R}^{p_1 \times r},\, V \in \mathbb{R}^{p_2 \times r}}{\arg\min} \quad & \|XU - YV\|_F^2 \\ \text{subject to} \quad & U^T \Sigma_X U = I_r, \quad V^T \Sigma_Y V = I_r. \end{aligned} \tag{2}$$

where Σ_X and Σ_Y denote the covariance matrices of X and Y respectively.

Remark 4.1. We emphasize that the normalization $U^T \Sigma_X U = V^T \Sigma_Y V = I_r$ is here indispensable to ensure that the learned mappings between representations remain independent of the varying scales introduced by different GNN representations.

Remark 4.2. We argue that the linear nature of mappings learned by CCA is not overly restrictive. While deep variants of CCA could be employed, self-supervised embeddings are often used with simple linear models (e.g. the outputs are processed with a simple linear classifier to produce labels). Thus, restricting our objective to consider linear mappings seems reasonable.

4.1 CCA for Aligning Representations

As we seek to evaluate unsupervised representations, in this section, we assume that we have generated $3n_b$ independent versions of the dataset with the procedure described in Section 3. For each $i \in [2n_b]$, we learn an unsupervised representation of the nodes: $H_i = GNN_i(G_i, \theta)$, where θ indicates the tunable hyperparameters. We propose evaluating the quality of the learned representation by comparing the alignment of the embeddings learned by different models on replicas of the same dataset as per (2).

The solution to the CCA problem (2) has a closed-form expression. Let U_0, V_0 be the left and right singular vectors of the cross-covariance matrix:

$$\mathrm{corr}(H_i, H_j) = \hat{\Sigma}_{H_i}^{-1/2} \hat{\Sigma}_{H_i H_j} \hat{\Sigma}_{H_j}^{-1/2} = U_0 \Lambda_0 V_0^\top,$$

where $\hat{\Sigma}_{H_i}$ is the empirical covariance of embeddings from dataset i, and $\hat{\Sigma}_{H_i H_j}$ is the empirical cross-covariance of embeddings from datasets i and j. The solutions to (2) are

$$\hat{U}(i,j) = \hat{\Sigma}_{H_i}^{-1/2} U_0, \quad \hat{V}(i,j) = \hat{\Sigma}_{H_j}^{-1/2} V_0. \tag{3}$$

and we can compute the alignment between versions of the dataset as:

$$\text{alignment} = \|H_i \hat{U}(i,j) - H_j \hat{V}(i,j)\|_F,$$

where the alignment is evaluated and aggregated over the bootstrapped samples $i, j \in [n_b]$, $i \neq j$.

Statistic	Graphon (n = 500)		Cora		Citeseer		Twitch	
	True	Ours	True	Ours	True	Ours	True	Ours
$\lvert\mathcal{V}\rvert$	500	500±0	2708	2708±0	3327	3327±0	1912	1912±0
$\lvert\mathcal{E}\rvert$	769	757.9±2.5	5278	5171.78±7.34	4552	4127.78±10.93	31299	31082.05±22.83
Avg. Degree	3.08	3.03±0.01	3.90	3.82±0.01	2.74	2.48±0.01	32.74	32.51±0.02
Density	0.01	0.01±0	0.00	0.00±0	0.00	0.00±0	0.02	0.02±0
Clustering Coefficient	0.01	0.01±0	0.24	0.05±0	0.14	0.03±0	0.32	0.17±0
Connected Components	29.00	31±2	78	67.91±6.70	438	635.09±11.87	1.00	1.14±0.39
Giant Component Size	471.00	467±3	2485	2620.80±10.80	2120	2418.12±35.69	1912	1911.72±0.77
Assortativity	-0.04	-0.08±0.03	-0.07	-0.07±0	0.05	-0.08±0	-0.23	-0.29±0
PageRank Sum	249.5	249.5±0	1353.50	1353±0	1663	1663±0	955.50	955.5±0
Transitivity	0.01	0.01±0	0.09	0.03±0	0.13	0.04±0	0.13	0.08±0
Number of Triangles	7	5±2.3	1630	471±27	1167	304.6±19.59	173510	105534.51±1904.54

Table 1: Graph statistics for synthetic graphon data, citation networks (Cora, Citeseer), and a social network (Twitch) [Huang et al. 2023]. We generated 500 bootstrap samples and report the mean and standard deviation. The size of the neighborhood (k) used for sample generation is fixed at 20. Results for additional datasets and different graphon settings are included in Appendix F.2.

4.2 Validation of the Evaluation Metric

We evaluate the validity of our metric (2) on three biological datasets: the spleen dataset [Goltsev et al. 2018], the MIBI-TOF breast cancer dataset [Keren et al. 2018], and the colorectal cancer (CRC) dataset [Schürch et al. 2020]. Each dataset comprises multiple graphs, allowing us to assess the validity of our proposed metric (2) independently of the graph bootstrapping procedure. For instance, the spleen dataset includes samples from 3 mice, while the CRC dataset contains data from 31 patients. Table 2 presents both the evaluation of our metric along with the downstream task performance. In addition, visualizations provided in Appendix Figure 7 further support the utility of our metric in guiding the hyperparameter selection (e.g., λ), effectively recovering biologically meaningful cell microenvironments. Detailed descriptions of the datasets and downstream tasks are provided in Appendix F.1.1.

4.3 Proposed Hyperparameter Tuning Framework

We now describe the full procedure, which we call LOBSTUR (**Lo**cal **B**oot**s**trap for **T**uning **U**nsupervised **R**epresentations in GNNs). The previous two sections have described the two core components of our procedure. We now detail an additional step to safeguard our pipeline against degeneracies.

Adjustment for Dimensional Collapse. Our proposed alignment metric is grounded in a straightforward statistical method, Canonical Correlation Analysis (CCA). The strength of this method lies in its assessment of *correlations* between representations. However, because it accounts for different variances, this method may struggle to accurately reflect the quality of embeddings in the presence of dimensional collapse [Hua et al. 2021]. Dimensional collapse, a phenomenon common in self-supervised representation learning, occurs when the learned representations are confined to a low-dimensional manifold. For example, when training a model with an embedding dimension of $p = 2$, dimensional collapse may result in embeddings that lie along a single line (reduced to a one-dimensional representation) or form a blob. In such cases, although the embeddings lack informative structure, their alignment across different samples may still be high, leading to an over-inflated metric.

The *StableRank* metric [Tsitsulin et al. 2023] is defined as $\sum_i \sigma_i^2/\sigma_1^2$, where σ_i are the singular values of the embeddings $H \in \mathbb{R}^{n\times p}$ in descending order, and assesses the numerical rank of the embedding space. We will use this metric to filter out embeddings that are clearly suboptimal [Jing et al. 2022] before applying our CCA-based metric to tune hyperparameters. An alternative choice for the threshold metric could be *RankMe* proposed by Garrido et al. [2023].

Our full procedure is highlighted in Algorithm 3.

5 EXPERIMENTS

Benchmark Datasets. We demonstrate the validity of our entire framework on GNN benchmark datasets such as Cora, Citeseer, and Pubmed. We show that hyperparameter and model selection using our suggested framework results in robust, high downstream task performance on benchmark datasets, thereby indicating embeddings of good quality. More specifically, we consider the task of learning unsupervised GNN embeddings using four different methods (CCA-SSG, BGRL, DGI, and GRACE, see Appendix B), and choosing the correct set of hyperparameters in each method. *Note that we do not look at the classification accuracy ahead of time and use them for choosing the model and hyperparameters.* Instead, we only report them after choosing the model to validate the approach, reflecting a more practical scenario to apply unsupervised GNNs on real datasets. In Table 3, we report the downstream task performance (classification or regression) of the model chosen by our framework (Algorithm 3) and metrics proposed in Tsitsulin et al. [2023]. Our method shows a robust performance and achieves

	Spleen			TNBC			CRC		
λ	ACC	Mean	SD	AUC	Mean	SD	AUC	Mean	SD
0.000001	0.4114	56,955	23,032	0.7566	4,765	3,617	0.8039	26,385	3,812
0.00001	0.4135	58,398	30,425	0.7487	5,217	4,284	0.8039	26,699	3,746
0.0001	**0.4146**	40,017	12,422	0.7249	4,734	3,348	0.8170	25,972	5,934
0.001	0.4128	**21,741**	5,732	0.7513	3,781	1,782	0.7974	8,844	1,893
0.01	0.3691	42,336	1,970	0.7328	3,425	1,566	0.8431	6,425	1,319
0.1	0.3986	50,351	2,550	**0.8757**	**3,149**	1,111	**0.9412**	5,940	1,538
1	0.3914	55,264	2,788	0.8307	3,516	1,309	0.9346	**5,543**	889
10	0.3184	61,804	2,339	0.8704	3,689	1,443	0.8627	5,951	1,301

Table 2: For each dataset, the first column reports the downstream task performance, while the second and third columns present the mean and standard deviation of the evaluation metric defined in Equation 2. We adopt the architecture from Zhang et al. [2021] and fix all hyperparameters except for λ in the CCA-SSG loss (Equation 8). Using Algorithm 3, the minimum average distances are achieved at $\lambda_{\text{MS}} = 0.001$ for the mouse spleen dataset [Goltsev et al. 2018], $\lambda_{\text{TNBC}} = 0.1$ for Triple Negative Breast Cancer (TNBC) [Keren et al. 2018], and $\lambda_{\text{CRC}} = 1.0$ for colorectal cancer (CRC) [Schürch et al. 2020]. Notably, strong downstream performance coincides with improved embedding alignment, as indicated by lower average distances reported in the second column for each dataset.

Dataset	Default	Ours	α-ReQ	pseudo-κ	RankME	NESum	SelfCluster	Stable Rank	Coherence
Classification tasks									
Cora	0.36	0.65	0.66	0.54	0.63	0.63	**0.69**	0.59	0.47
PubMed	0.62	0.81	0.75	0.75	0.75	0.75	**0.82**	0.75	0.76
Citeseer	0.32	**0.51**	**0.51**	**0.51**	**0.51**	**0.51**	0.48	**0.51**	0.22
CS	0.47	0.79	**0.86**	0.72	**0.86**	**0.86**	**0.86**	**0.86**	0.76
Photo	0.29	0.73	0.79	0.79	0.79	0.79	0.57	**0.81**	0.69
Computers	0.37	0.57	0.45	0.57	0.45	0.39	0.39	0.57	**0.65**
Regression tasks									
Chicago	0.39	0.34	0.35	0.35	0.35	0.35	0.35	0.29	**0.40**
Anaheim	0.13	**0.23**	0.12	0.18	0.18	0.12	**0.23**	0.18	0.12
Twitch	0.47	**0.52**	0.15	0.15	0.15	0.15	0.46	0.15	0.48
Education	0.23	0.26	**0.33**	**0.33**	**0.33**	**0.33**	**0.33**	**0.33**	0.26
Avg clf	0.41	**0.68**	0.67	0.65	0.66	0.65	0.63	**0.68**	0.59
Avg reg	0.30	**0.34**	0.24	0.25	0.25	0.24	**0.34**	0.24	0.32

Table 3: Downstream task (classification or regression) performance of the best model and hyperparameters chosen by each criterion. The best value is bolded and the second best is underlined. We compare to the BGRL [Thakoor et al. 2021] with default hyperparameters (fmr $= 0.5$, edr $= 0.25$, $\lambda = 10^{-2}$) in the left-most column.

either the best or the second best performance compared to the existing metrics for 7 out of 10 datasets, and achieving the best overall accuracy. A similar table reporting the performance by different GNN architectures [Thakoor et al. 2021; Zhang et al. 2021; Zhu et al. 2020] is presented in Table 13, 14, 15 in the Appendix.

Choice of the Threshold. We turn to the problem of selecting a stable-rank threshold. We suggest using the reasonable lower bound for the latent (effective) dimension as sufficient. For Tables 3 and 16, we set the threshold to $t = 2$. This choice ensures that the embeddings retain a minimum effective dimensionality, preventing collapse to a single line. Consequently, our alignment metric accurately measures meaningful signal alignment rather than trivial, collapsed patterns. It is important to highlight the trade-off associated with this threshold: setting a higher threshold enhances robustness but may inadvertently exclude beneficial models, while a lower threshold allows greater model diversity but risks increased variability and potential collapse of representations.

Algorithm 3 Hyperparameter Tuning Procedure

Input: An input graph G and a set of hyperparameters Θ from which to choose an optimal value.
Create $3n_b$ bootstrap samples of the graph, denoted as $\{\hat{G}_i\}_{i=1}^{3n_b}$ (Algorithm 1 , 2).
for each value $\theta \in \Theta$ **do**
  for $i = 1, \ldots, 2n_b$ **do**
    Train an unsupervised GNN, $f_i(\cdot, \theta)$ on $\hat{G}_i$.
  end for
  for each pair of models $f_i(\cdot, \theta)$ and $f_{i+n_b}(\cdot, \theta)$ with $i \in \{1, \cdots, n_b\}$ **do**
    Compute the distance between embeddings from models f_i and f_{i+n_b} on the test graph $\hat{G}_{i+2n_b}$:

$$d_i(\theta) = \ell\big(f_i(\hat{G}_{i+2n_b}, \theta), f_{i+n_b}(\hat{G}_{i+2n_b}, \theta)\big),$$

    where $\ell(\cdot)$ is some metric, like the one we proposed in Section 4.
  end for
end for
Choose the optimal hyperparameters: $\hat{\theta} = \underset{\theta \in \Theta, StableRank \geq t}{\operatorname{argmin}} \bar{d}(\theta)$, where $\bar{d}(\theta)$ is the average distance across $i \in [n_b]$, and t is the *StableRank* threshold.

6 CONCLUSION

Hyperparameter tuning for unsupervised GNNs presents two primary challenges – the complexity of generating multiple samples out of one observed graph and the difficulty in evaluating model performance without labeled data. In this paper, we propose a novel method for performing bootstrapping specifically tailored for unsupervised GNNs, facilitating both hyperparameter tuning and model selection. Although our validation is primarily empirical, we believe that this study highlights a more systematic approach for tuning graph neural networks and machine learning models in general, encouraging further exploration in this direction. Notably, our approach is applicable to graphs of moderate size (a few thousand nodes), but may not scale directly to larger graphs. A potential solution is to partition the graph and bootstrap within blocks. We present preliminary results in the appendix (see Appendix C.4, D) suggesting the promise of this approach, though adapting the procedure for large-scale graphs remains an open question for future research.

ACKNOWLEDGMENTS

This work was completed in part with resources provided by the University of Chicago Research Computing Center, and was supported in part through the computational resources and staff contributions provided for the Mercury high performance computing cluster at The University of Chicago Booth School of Business which is supported by the Office of the Dean.

REFERENCES

Stephen Bates, Trevor Hastie, and Robert Tibshirani. 2024. Cross-Validation: What Does It Estimate and How Well Does It Do It? *J. Amer. Statist. Assoc.* 119, 546 (2024), 1434–1445. https://doi.org/10.1080/01621459.2023.2197686

Mikhail Belkin and Partha Niyogi. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in neural information processing systems* 14 (2001).

Christopher M Bishop. 1998. Latent variable models. In *Learning in graphical models*. Springer, 371–403.

Sergio Castillo-Páez, Rubén Fernández-Casal, and Pilar García-Soidán. 2019. A non-parametric bootstrap method for spatial data. *Computational Statistics & Data Analysis* 137 (2019), 1–15. https://doi.org/10.1016/j.csda.2019.01.017

Kehui Chen and Jing Lei. 2018. Network Cross-Validation for Determining the Number of Communities in Network Data. *J. Amer. Statist. Assoc.* 113, 521 (2018), 241–251. https://doi.org/10.1080/01621459.2016.1246365 arXiv:https://doi.org/10.1080/01621459.2016.1246365

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. arXiv:2002.05709 [cs.LG] https://arxiv.org/abs/2002.05709

Andrew Davison and Morgane Austern. 2023. Asymptotics of network embeddings learned via subsampling. *Journal of Machine Learning Research* 24, 138 (2023), 1–120.

Kangning Dong and Shihua Zhang. 2022. Deciphering spatial domains from spatially resolved transcriptomics with an adaptive graph attention auto-encoder. *Nature communications* 13, 1 (2022), 1739.

B. Efron. 1979. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics* 7, 1 (1979), 1 – 26. https://doi.org/10.1214/aos/1176344552

Bradley Efron. 2012. Bayesian inference and the parametric bootstrap. *The annals of applied statistics* 6, 4 (2012), 1971.

Wei Fu and Patrick O. Perry. 2017. Estimating the number of clusters using cross-validation. arXiv:1702.02658 [stat.ME]

Chao Gao, Yu Lu, and Harrison H Zhou. 2015. Rate-optimal graphon estimation. (2015).

Quentin Garrido, Randall Balestriero, Laurent Najman, and Yann Lecun. 2023. RankMe: Assessing the downstream performance of pretrained self-supervised representations by their rank. arXiv:2210.02885 [cs.LG] https://arxiv.org/abs/2210.02885

Solenne Gaucher and Olga Klopp. 2021. Maximum likelihood estimation of sparse networks with missing observations. *Journal of Statistical Planning and Inference* 215 (2021), 299–329.

Yury Goltsev, Nikolay Samusik, Julia Kennedy-Darling, Salil Bhate, Matthew Hale, Gustavo Vazquez, Sarah Black, and Garry P Nolan. 2018. Deep profiling of mouse splenic architecture with CODEX multiplexed imaging. *Cell* 174, 4 (2018), 968–981.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2001. *The Elements of Statistical Learning*. Springer New York Inc., New York, NY, USA.

Peter D. Hoff. 2007. Modeling homophily and stochastic equivalence in symmetric relational data. arXiv:0711.1146 [stat.ME]

Peter D Hoff, Adrian E Raftery, and Mark S Handcock. 2002. Latent space approaches to social network analysis. *Journal of the american Statistical association* 97, 460 (2002), 1090–1098.

Joel L Horowitz. 2019. Bootstrap methods in econometrics. *Annual Review of Economics* 11, 1 (2019), 193–224.

Harold Hotelling. 1936. Relations between two sets of variates. In *Biometrika*. Biometrika, 28(3/4), 321–337.

Jian Hu, Xiangjie Li, Kyle Coleman, Amelia Schroeder, Nan Ma, David J Irwin, Edward B Lee, Russell T Shinohara, and Mingyao Li. 2021. SpaGCN: Integrating gene expression, spatial location and histology to identify spatial domains and spatially variable genes by graph convolutional network. *Nature methods* 18, 11 (2021), 1342–1351.

Tianyu Hua, Wenxiao Wang, Zihui Xue, Sucheng Ren, Yue Wang, and Hang Zhao. 2021. On feature decorrelation in self-supervised learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 9598–9608.

Kexin Huang, Ying Jin, Emmanuel Candes, and Jure Leskovec. 2023. Uncertainty quantification over graph with conformalized graph neural networks. *NeurIPS* (2023).

L. Hubert and P. Arabie. 1985. Comparing partitions. *Journal of classification* 2, 1 (1985), 193–218. http://scholar.google.de/scholar.bib?q=info:IkrWWF2JxwoJ:scholar.google.com/&output=citation&hl=de&ct=citation&cd=0

Satoki Ishiai, Ikki Yasuda, Katsuhiro Endo, and Kenji Yasuoka. 2024. Graph-Neural-Network-Based Unsupervised Learning of the Temporal Similarity of Structural Features Observed in Molecular Dynamics Simulations. *Journal of Chemical Theory and Computation* (2024).

Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. 2022. Understanding Dimensional Collapse in Contrastive Self-supervised Learning. arXiv:2110.09348 [cs.CV] https://arxiv.org/abs/2110.09348

Leeat Keren, Marc Bosse, Diana Marquez, Roshan Angoshtari, Samir Jain, Sushama Varma, Soo-Ryum Yang, Allison Kurian, David Van Valen, Robert West, et al. 2018. A structured tumor-immune microenvironment in triple negative breast cancer revealed by multiplexed ion beam imaging. *Cell* 174, 6 (2018), 1373–1387.

Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

Andre Lamurias, Alessandro Tibo, Katja Hose, Mads Albertsen, and Thomas Dyhre Nielsen. 2022. Graph Neural Networks for Microbial Genome Recovery. *arXiv preprint arXiv:2204.12270* (2022).

Vuong Le, Thomas P Quinn, Truyen Tran, and Svetha Venkatesh. 2020. Deep in the bowel: highly interpretable neural encoder-decoder networks predict gut metabolites from gut microbiome. *BMC genomics* 21, 4 (2020), 1–15.

James Leiner and Aaditya Ramdas. 2024. Graph fission and cross-validation. arXiv:2401.15063 [stat.ME]

Keith Levin and Elizaveta Levina. 2021. Bootstrapping Networks with Latent Space Structure. arXiv:1907.10821 [math.ST]

Junyi Li, Wei Jiang, Henry Han, Jing Liu, Bo Liu, and Yadong Wang. 2021. ScGSLC: an unsupervised graph similarity learning framework for single-cell RNA-seq data clustering. *Computational Biology and Chemistry* 90 (2021), 107415.

Tianxi Li, Elizaveta Levina, and Ji Zhu. 2020. Network cross-validation by edge sampling. arXiv:1612.04717 [stat.ME]

Leland McInnes, John Healy, and James Melville. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018).

Anna Neufeld, Ameer Dharamshi, Lucy L. Gao, and Daniela Witten. 2023. Data thinning for convolution-closed distributions. arXiv:2301.07276 [stat.ME]

Gabriele Partel and Carolina Wählby. 2021. Spage2vec: Unsupervised representation of localized spatial gene expression signatures. *The FEBS Journal* 288, 6 (2021), 1859–1870.

Patrick O. Perry. 2009. Cross-Validation for Unsupervised Learning. arXiv:0909.3052 [stat.ME]

Dimitris N. Politis and Joseph P. Romano. 1994. The Stationary Bootstrap. *J. Amer. Statist. Assoc.* 89, 428 (1994), 1303–1313. http://www.jstor.org/stable/2290993

Olivier Roy and Martin Vetterli. 2007. The effective rank: A measure of effective dimensionality. In *2007 15th European signal processing conference*. IEEE, 606–610.

Donald B Rubin. 1981. The bayesian bootstrap. *The annals of statistics* (1981), 130–134.

Christian M Schürch, Salil S Bhate, Graham L Barlow, Darci J Phillips, Luca Noti, Inti Zlobec, Pauline Chu, Sarah Black, Janos Demeter, David R McIlwain, et al. 2020. Coordinated cellular neighborhoods orchestrate antitumoral immunity at the colorectal cancer invasive front. *Cell* 182, 5 (2020), 1341–1359.

Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, et al. 2020. A deep learning approach to antibiotic discovery. *Cell* 180, 4 (2020), 688–702.

Yi Su, Raymond KW Wong, and Thomas CM Lee. 2020. Network estimation via graphon with node features. *IEEE Transactions on Network Science and Engineering* 7, 3 (2020), 2078–2089.

Shantanu Thakoor et al. 2021. Large-Scale Representation Learning on Graphs via Bootstrapping. *arXiv preprint arXiv:2102.06514* (2021).

Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L. Dyer, Rémi Munos, Petar Veličković, and Michal Valko. 2023. Large-Scale Representation Learning on Graphs via Bootstrapping. arXiv:2102.06514 [cs.LG]

Robert Tibshirani and Guenther Walther. 2005. Cluster Validation by Prediction Strength. *Journal of Computational and Graphical Statistics* 14, 3 (2005), 511–528. https://doi.org/10.1198/106186005X59243 arXiv:https://doi.org/10.1198/106186005X59243

Anton Tsitsulin, Marina Munkhoeva, and Bryan Perozzi. 2023. Unsupervised embedding quality evaluation. In *Topological, Algebraic and Geometric Learning Workshops 2023*. PMLR, 169–188.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9, 86 (2008), 2579–2605. http://jmlr.org/papers/v9/vandermaaten08a.html

Zhenqin Wu, Alexandro E Trevino, Eric Wu, Kyle Swanson, Honesty J Kim, H Blaize D'Angio, Ryan Preska, Gregory W Charville, Piero D Dalerba, Ann Marie Egloff, et al. 2022. SPACE-GM: geometric deep learning of disease-associated microenvironments from multiplex spatial protein profiles. *bioRxiv* (2022), 2022–05.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2021. Graph Contrastive Learning with Augmentations. arXiv:2010.13902 [cs.LG] https://arxiv.org/abs/2010.13902

Hengrui Zhang, Qitian Wu, Junchi Yan, David Wipf, and Philip S. Yu. 2021. From Canonical Correlation Analysis to Self-supervised Graph Neural Networks. arXiv:2106.12484 [cs.LG]

Ruochi Zhang, Jianzhu Ma, and Jian Ma. 2020. DANGO: Predicting higher-order genetic interactions. *bioRxiv* (2020), 2020–11.

Qian Zhu, Sheel Shah, Ruben Dries, Long Cai, and Guo-Cheng Yuan. 2018. Identification of spatially associated subpopulations by combining scRNAseq and sequential fluorescence in situ hybridization data. *Nature biotechnology* 36, 12 (2018), 1183–1190.

Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2020. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131* (2020).

A ADDITIONAL DEFINITIONS AND PROOFS

A.1 Definitions

Throughout this manuscript, we assume the same conventions as in the general literature on graphon estimation (see, for instance, Gao et al. [2015]; Gaucher and Klopp [2021]).

In particular, for a function $f : [0,1] \times [0,1] \to [0,1]$, the derivative operator is defined by

$$\nabla_{jk} f(x,y) = \frac{\partial^{j+k}}{(\partial x)^j (\partial y)^k} f(x,y),$$

and we adopt the convention $\nabla_{00} f(x,y) = f(x,y)$.

Definition A.1 (Hölder class for Graphon functions (from Gao et al. [2015])). The Hölder norm is defined as

$$\begin{aligned} \|f\|_{\mathcal{H}_\alpha} &= \max_{j+k \le \lfloor \alpha \rfloor} \sup_{x,y \in \mathcal{D}} \left| \nabla_{jk} f(x,y) \right| \\ &\quad + \max_{j+k=\lfloor \alpha \rfloor} \sup_{(x,y) \neq (x',y') \in \mathcal{D}} \frac{\left| \nabla_{jk} f(x,y) - \nabla_{jk} f(x',y') \right|}{(|x-x'| + |y-y'|)^{\alpha - \lfloor \alpha \rfloor}}. \end{aligned}$$

The Hölder class is defined by

$$\mathcal{H}_\alpha(M) = \left\{ \|f\|_{\mathcal{H}_\alpha} \le M : f(x,y) = f(y,x) \text{ for } x \ge y \right\},$$

where $\alpha > 0$ is the smoothness parameter and $M > 0$ is the size of the class, which is assumed to be a constant.

Definition A.2 (Distance Measures). For nodes $i, j \in [n]$:

$$\begin{aligned} d_L(i,j) &= |U_i - U_j| \quad \text{(Latent distance)} \\ d_F(i,j) &= \|X_i - X_j\|_2 \quad \text{(Feature distance)} \\ d_G(i,j) &= \text{length of shortest path from } i \text{ to } j \quad \text{(Graph distance)} \end{aligned}$$

Note that in the actual implementation, other graph distances are available as an option, but for the analysis purpose, we assume $D_G(\cdot,\cdot)$ is a shortest-path distance.

Definition A.3 (k-NN Neighborhoods). For node i:

$$\begin{aligned} \mathcal{N}_k^U(i) &= \{j : U_j \text{ is among k-nearest neighbors of } U_i\} \\ \mathcal{N}_k^X(i) &= \{j : X_j \text{ is among k-nearest neighbors of } X_i\} \\ \mathcal{N}_k^G(i) &= \{j : \text{node } j \text{ is among k-nearest neighbors of node } i\} \end{aligned}$$

where $X_i = g(U_i) + \epsilon_i$, and $U_i \sim \text{Unif}[0,1]$. The neighborhood is determined by corresponding distance. For example, the neighborhood in the latent space is determined by latent distance.

A.2 Proof of Theorem 3.2

Proof. Letting $\mathcal{N}_k(i)$ denote the kth closest neighbor of node i according to the oracle graph $\mathcal{G}_{knn}$.

For any pair of nodes (i,j), as we are resampling, we are effectively replacing the underlying connection probability $\mathbb{P}[A_{ij} = 1 | U_i, U_j]$ by:

$$\hat{p}_{ij} = \frac{1}{K} \sum_{k=1}^{K} A_{\mathcal{N}_k(i),j}$$

We decompose the risk of this estimator as:

$$\mathbb{E}\left[\left(\mathbb{P}[A_{ij} = 1 | U_i, U_j] - \hat{p}_{ij} \right)^2 \right] = \text{Bias}^2 + \text{Variance}$$

where

$$\begin{aligned} \text{Bias} &= \mathbb{P}[A_{ij} = 1 | U_i, U_j] - \mathbb{E}[\hat{p}_{ij}] \\ &= \frac{1}{k} \sum_{m=1}^{k} \left(\mathbb{P}[A_{ij} = 1 | U_i, U_j] - \mathbb{P}[Y_{\mathcal{N}_m(i),j} = 1 | U_{\mathcal{N}_m(i)}, U_j] \right) \\ \text{Variance} &= \mathbb{E}\left[\left(\frac{1}{k} \sum_{m=1}^{k} (\mathbb{P}[Y_{\mathcal{N}_m(i),j} = 1 | U_{\mathcal{N}_m(i)}, U_j] - A_{\mathcal{N}_m(i),j}) \right)^2 \right] \end{aligned} \tag{4}$$

By assumption, since W is assumed to be α-Hölder, as emphasized in Gao et al. [2015], when $\alpha \in (0,1]$, a function $f \in \mathcal{H}_\alpha(M)$ satisfies the Lipschitz condition

$$\left|f(x,y) - f(x',y')\right| \leq M\left(\left|x-x'\right| + \left|y-y'\right|\right)^{\alpha}, \tag{5}$$

Therefore, we have:

$$\begin{aligned} |\text{Bias}| &= \left|\mathbb{P}[A_{ij}=1|U_i,U_j] - \frac{1}{k}\sum_{m=1}^{k}\mathbb{P}[Y_{\mathcal{N}_m(i),j}=1|U_{\mathcal{N}_m(i)},U_j]\right| \\ &\leq \frac{1}{k}\sum_{m=1}^{k} M|U_i - U_{\mathcal{N}_m(i)}|^{\alpha}. \end{aligned} \tag{6}$$

The quantity $|U_i - U_m|^{\alpha}$ (with m a k-nearest neighbor of i) is a monotonously decreasing function of n, and with high probability (over the distribution of $U_1, \cdots, U_n$), we have $|U_i - U_m|_2 \leq c_0\frac{k}{n}$, for all $m \in \mathcal{N}(U_i)$, and a constant c_0. Therefore, as n goes to infinity, $\lim_{n\to\infty}|\text{Bias}| = 0$.

Similarly, for the variance:

$$\begin{aligned} \text{Variance} &= \mathbb{E}\left[\left(\frac{1}{k}\sum_{m=1}^{k}(\mathbb{P}[Y_{\mathcal{N}_m(i),j}=1|U_{\mathcal{N}_m(i)},U_j] - A_{\mathcal{N}_m(i),j})\right)^2\right] \\ &= \frac{1}{k^2}\sum_{m=1}^{k}\mathbb{P}[Y_{\mathcal{N}_m(i),j}=1|U_{\mathcal{N}_m(i)},U_j](1-\mathbb{P}[Y_{\mathcal{N}_m(i),j}=1|U_{\mathcal{N}_m(i)},U_j]) \\ &\leq \frac{1}{k}. \end{aligned} \tag{7}$$

As $k \to \infty$, this converges to 0.

This shows that $\hat{p}_{ij}$ is a consistent estimator of p_{ij}.

□

B SUMMARY OF SELECTED UNSUPERVISED GNNS

CCA-SSG: CCA-SSG [Zhang et al. 2021] is inspired by statistical canonical correlation analysis(CCA) that constructs the loss on the feature-level rather than instance-level discrimination, which is typical in contrastive methods. They augment the original graph in a random fashion by dropping edges or masking the node features to make a pair of graphs for learning.

$$\mathcal{L} = \underbrace{\|\tilde{Z}_A - \tilde{Z}_B\|^2}_{\text{invariance term}} + \lambda \underbrace{\|\tilde{Z}_A^\top\tilde{Z}_A - I\|_F^2 + \|\tilde{Z}_B^\top\tilde{Z}_B - I\|_F^2}_{\text{decorrelation term}} \tag{8}$$

Although their model structure is relatively simple and does not require a parametrized mutual information estimator or additional projection network, they still have the issue of choosing hyperparameters(e.g. λ) which has a non-negligible impact on the model performance.

GRACE: Contrastive learning or self-supervised method has gotten increasing attention as they do not require label availability as supervised GNN does. Deep Graph Contrastive Representation Learning(GRACE) [Zhu et al. 2020] is one of the popular graph constrastive learning methods.

(1) For each iteration, GRACE generates two graph views, $\tilde{G}_1, \tilde{G}_2$, by either randomly removing edges or randomly masking node features.
(2) Let $U = f(\tilde{X}_1, \tilde{A}_1), V = f(\tilde{X}_2, \tilde{A}_2)$ be the embedded representation of two graph views, and their corresponding node features and adjacency matrices.
(3) Positive samples: For any node v_i, its corresponding representation in another view u_i is treated as natural positive pair.
(4) Negative samples: For given node v_i, any nodes in another view $u_{k\neq i}$ are treated as negative pair.
(5) Node-wise objective:

$$\ell(u_i,v_i) = \log\frac{e^{\theta(u_i,v_i)/\tau}}{\underbrace{e^{\theta(u_i,v_i)/\tau}}_{\text{the positive pair}} + \underbrace{\sum_{k=1}^{N}\mathbb{1}_{k\neq i}e^{\theta(u_i,v_k)/\tau}}_{\text{inter-view negative pairs}} + \underbrace{\sum_{k=1}^{N}\mathbb{1}_{k\neq i}e^{\theta(u_i,u_k)/\tau}}_{\text{intra-view negative pairs}}}$$

(6) Overall loss function: $\ell = \frac{1}{2N}\sum_{i=1}^{N}\left[\ell(u_i,v_i) + \ell(v_i,u_i)\right]$
(7) Optimization: apply stochastic gradient descent.

DGI: Deep Graph Infomax [Stokes et al. 2020] is another option for the unsupervised graph representation learning. DGI optimizes the mutual information between the local patch representation of the graph and the overall high-level summaries.

$$\mathcal{L} = \frac{1}{N+M}\Big(\sum_{i=1}^{N}\mathbb{E}_{(X,A)}\left[log\mathcal{D}(\vec{h}_i,\vec{s})\right] + \sum_{j=1}^{M}\mathbb{E}_{(X,A)}\left[log(1-\mathcal{D}(\vec{\tilde{h}}_i,\vec{\tilde{s}}]\right)$$

BGRL: Large-Scale Representation Learning on Graphs via Bootstrapping(BGRL) [Thakoor et al. 2023] similar to CCA-SSG, BGRL uses node and feature masking to augment the original graph. At the core of BGRL is a bootstrapping mechanism that updates the target representations gradually, borrowing ideas from consistency regularization and contrastive learning. Unlike contrastive learning methods that require negative samples, BGRL avoids the computational overhead associated with negative sampling by using a bootstrapping approach. This involves maintaining two networks: an online network that is updated using gradients and a target network that is slowly updated with the parameters of the online network. This setup encourages the embeddings to become more stable and consistent over iterations.

(1) Update the online encoder:

$$\ell(\theta,\phi) = -\frac{2}{N}\sum_{i=0}^{N-1}\frac{\tilde{Z}_{(1,i)}\tilde{H}^{\top}_{(2,i)}}{\|\tilde{Z}_{(1,i)}\|\|\tilde{H}^{\top}_{(2,i)}\|}$$

(2) Update the target encoder: $\theta \leftarrow \tau\phi + (1-\tau)\theta$

GCA: Graph Contrastive Learning with Augmentations (GCA) [You et al. 2021] introduces a contrastive learning framework designed specifically for graph data. GCA applies data augmentation techniques on both the node features and graph structure, creating different views of the same node. The central idea is to maximize the agreement between the representations of the same node in different augmented views, while ensuring that the representations of different nodes remain distinguishable.

The contrastive loss is designed to encourage the representations of different views, a and b of the same node i, with temperature scaling τ.

$$\mathcal{L}_{\text{GCA}} = \frac{1}{N}\sum_{i=1}^{N} -\log\frac{\exp(\text{sim}(\mathbf{z}_i^a,\mathbf{z}_i^b)/\tau)}{\sum_{j=1}^{N}\exp(\text{sim}(\mathbf{z}_i^a,\mathbf{z}_j^b)/\tau)}$$

where $\text{sim}(z_i, z_j) = z_i^T z_j/(\|z_i\|\cdot\|z_j\|)$ is a cosine similarity.

VGAE: Variational Graph Autoencoder (VGAE) [Kipf and Welling 2016] is a framework designed for learning graph embeddings through variational inference. It is a probabilistic approach that leverages both graph structure and node features to infer latent node representations. VGAE aims to model the underlying distribution of the graph data, capturing the uncertainty in the embeddings by using a variational autoencoder architecture. This setup allows VGAE to generate robust embeddings that generalize well to unseen nodes or links. The model consists of an encoder that approximates the posterior distribution over latent variables and a decoder that reconstructs the graph from these variables.

The loss function comprises two components: a reconstruction loss that encourages the model to accurately predict the adjacency matrix, and a regularization term in the form of the KL-divergence, which ensures the latent variables follow the prior distribution.

(1) Update the encoder by maximizing the evidence lower bound (ELBO):

$$\mathcal{L} = \mathbb{E}_{q(Z|X,A)}\left[\log p(A|Z)\right] - KL(q(Z|X,A)||p(Z))$$

(2) The prior over the latent variables Z is typically set to a standard Gaussian: $p(Z) = \mathcal{N}(0, I)$.

C ADDITIONAL LITERATURE REVIEW

C.1 Cross-Validation

In the supervised learning literature, cross-validation (CV) [Hastie et al. 2001; Tibshirani and Walther 2005] stands as a fundamental strategy for selecting hyperparameters and evaluating models. In the usual (Euclidean) setting, this technique involves partitioning the dataset into distinct subsets: a "training set" for model training and a "test set" for its evaluation. The partitioning is justified by the independence between observations, which implies that the subsamples still follow the same distribution as the original data. A commonly used method is K-fold cross-validation, where the dataset is divided into K subsets or folds. For simplicity, we assume there are n samples, and each fold has m data points so that $n = K \times m$. We denote a set of index for the k-th fold as I_k. The model is trained K times, each time using $K-1$ folds for training and the remaining fold for validation. Evaluation of the validation set is performed through an appropriate evaluation function $\ell(\cdot)$ measuring the discrepancy between the observations y_i and their predicted values $\hat{y}_i = \hat{f}(x_i, \theta)$. This loss is usually taken to be the mean squared error(MSE) in the regression case, ($\text{MSE}_k = \frac{1}{m}\sum_{i\in I_k}(y_i - \hat{y}_i)^2$), or to be the classification accuracy in the classification setting. By averaging this metric over all k folds, cross-validation provides a reliable estimate of the model's prediction error on unseen data.

While the implementation and practice of cross-validation is simple and straightforward, its interpretation has only recently been investigated in work by Bates et al. [2024]. The authors' key finding is that the cross-validation does not estimate the prediction error for the

model trained on a specific dataset but rather the “average" prediction error across all possible training datasets from the same distribution.

$$\widehat{Err}^{(CV)} = \frac{1}{n}\sum_{i=1}^{n} e_i = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{m}\sum_{i\in I_k}\ell(\hat{f}(x_i,\hat{\theta}^{(-k)}), y_i). \tag{9}$$

The intuition is the inner summation in Equation 9 estimates the prediction error of the model at hand, and the outer summation calculates the empirical average over all possible training sets of the same size. In the previous equation, $\hat{\theta}^{(-k)}$ denotes the parameters of the model fitted on all but the kth fold, and $\hat{f}(x_i,\hat{\theta}^{(-k)})$ indicates the estimator of y.

C.2 Cross-Validation for Unsupervised Learning

Despite the popularity and simplicity of the cross-validation procedure, its application in unsupervised learning has been relatively underexplored, largely due to the absence of clear evaluation metrics. Perry [2009] addressed this gap by examining cross-validation in unsupervised settings and proposing several solutions, with a focus on methods utilizing Singular Value Decomposition (SVD). Among the strategies reviewed, two are particularly relevant for this discussion. The first is a traditional hold-out method, where a portion of the data is set aside for validation, and the second involves treating random elements of the dataset as "missing values." For a detailed explanation of these methods, refer to Perry [2009], Chapter 5. However, it is important to note that these methods were originally designed for conventional, independent, tabular data for unsupervised tasks. In this study, we build on Perry’s framework, focusing on its connection to graph neural networks (GNNs) and extending its use to evaluate unsupervised learning methods in the context of GNNs in Section 4.

For the hold-out method, we randomly partition the data $Z \in \mathbb{R}^{n\times p}$ into $\begin{pmatrix} Z_1 \\ Z_2 \end{pmatrix}$, where $Z_1 \in \mathbb{R}^{n_1\times p}$ is a training set, $Z_2 \in \mathbb{R}^{n_2\times p}$ is a test set, and $n_1 + n_2 = n$. We want to approximate the test data by projecting it onto the principal spaces of the training data. To do so, one can calculate the k-dimensional reduced SVD of Z_1, where $\hat{Z}_1(k) = \sqrt{(n)}\hat{U}_1\hat{D}_1(k)\hat{V}_1$. Project the test set onto the principal space of Z_1.

$$\hat{Z}_2(k) = Z_2\hat{Z}_1(k)^\top(\hat{Z}_1(k)\hat{Z}_1(k)^\top)^\dagger\hat{Z}_1(k) = Z_2\hat{V}_1\hat{V}_1^\top.$$

$X^\dagger$ denotes the pseudo-inverse of X. The performance ban be evaluated using ℓ_2 loss, $\|Z_2 - \hat{Z}_2(k)\|_F^2$. Although this method cannot be used in practice because the loss is a decreasing function with k, the idea of using projection to compute the projection error for unsupervised tasks was insightful.

The second is called either missing value strategy or Wold hold-outs. Instead of simply splitting the data, one could randomly select the indices $I \in \mathcal{I}$, which denote the missing elements. Then, $Z_I = \begin{cases} Z_i & i \in I \\ * & \text{o.w} \end{cases}$; similarly, $Z_{\bar{I}} = \begin{cases} Z_i & i \notin I \\ * & \text{o.w} \end{cases}$. Apply k-rank missing value SVD algorithm to find the decomposition of $Z_{\bar{I}}(k) = U_kD_kV_k^\top$. There are many options[] including the one proposed by Perry [2009]. The performance can again be evaluated using $\|U_kD_kV_k^\top - Z_I\|_{F,I}^2$

The last method is basically to convert the unsupervised task into the supervised task, and called Gabriele hold-outs. Given the data, we could randomly permute the row and column so that we have the following decomposition $P^\top ZQ = \begin{pmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{pmatrix}$, where P and Q are the permutation matrices.

There is continuing work on applying this hold-out approach (especially Gabriele’s hold-out on clustering analysis [Fu and Perry 2017].

C.3 Cross Validation for Network Analysis

There have been relatively few studies [Chen and Lei 2018; Hoff 2007; Li et al. 2020] on the cross-validation of network data. In Li et al. [2020], the key assumption for the entire analysis is that the edge is the realization of independent Bernoulli random variables, and the probability of connection M, which is realized by the observed adjacency matrix A, is approximately of low rank. The edge cross-validation proposed in this study is different from traditional node-splitting methods in that the random dropping applies to the connected pair of nodes. The model by Chen and Lei [2018] is particularly designed for determining the number of communities within the network data, as well as choosing between the regular stochastic block model and the degree-corrected stochastic block model(DCSBM). The core idea is a block-wise node-pair splitting, which is then combined with an integrated step of community recovery using sub-blocks of the adjacency matrix.

Leiner and Ramdas [2024] introduces another cross-validation method for graphs but approaches the problem from a different angle. The study applies data thinning to data following convolution-closed distributions by Neufeld et al. [2023]. This procedure creates data folds that maintain the same distribution as the original data, are independent of each other, and sum to the original random variable. A canonical example of it is a normal variable. Given data $X \sim (\mu, \sigma^2)$, with unknown parameter of interest μ. Through data thinning algorithm, we could thin X into $X^{(1)} \sim N(\epsilon\mu, \epsilon^2\sigma^2)$ and $X^{(2)} \sim N((1-\epsilon)\mu, (1-\epsilon)^2\sigma^2)$, where these two thinned variables are independent to each other. Leiner and Ramdas [2024] is an extension of this concept to graph data, applying data thinning to node features while treating the adjacency matrix as fixed.

However, all these statistical methods heavily rely on the certain generation mechanism of underlying networks, such as the stochastic block model [Chen and Lei 2018] or low-rank structure of expected value of adjacency matrix [Li et al. 2020]. The assumptions of the

aforementioned approaches on which part of the graph is a random component are also different. Leiner and Ramdas [2024] treats the graph structural component (V, E) as non-random and the node feature as random; however, [Li et al. 2020] treats edge as then random realization based on statistical graph generation model, such as stochastic block model.

C.4 Bootstrap

The bootstrap [Efron 1979] has been widely used as a non-parametric method for estimating the distribution of a statistic through resampling with replacement. This method is useful because it does not rely on assumptions about the underlying distribution, making it applicable across various fields where such assumptions are challenging. The validity of the bootstrap is supported by its consistency [Horowitz 2019] under mild assumptions, where the bootstrap distribution converges to the true sampling distribution as the sample size increases. However, the validity of the bootstrap relies on having access to independent samples, an assumption violated in the graph case. We thus consider two distinct scenarios, depending on the nature of the graph at hand:

- *For graphs with short-range dependencies*, such as for instance, spatial graphs: we propose to apply a graph-based **block bootstrap** method, inspired by its use in time series and spatial statistics [Castillo-Páez et al. 2019; Politis and Romano 1994]. The block bootstrap is based on the assumption that the dependency structure is well contained within the small neighborhood that we could assume independence among these neighborhoods. We extend the application of the block bootstrap to the graph case here by splitting the graph into smaller (non-overlapping) neighborhoods of size B, and creating new graphs based on replacing each of these neighborhoods by sampling with replacement from the total set of possible neighborhoods (see Algorithm 4). Similar to the spatial setting [Castillo-Páez et al. 2019], the size of the blocks is crucial to the success of the procedure. To guide the choice of the neighborhood, we propose using descriptive graph statistics (see next section) to generate graphs with similar characteristics.
- *For graphs with long-range dependencies*, For non-spatial and homophilic graphs, we propose to use an extension of **network bootstrap** by Levin and Levina [2021]. In this work, Levin and Levina [2021] consider random dot product graphs (RDPG) where the edge connectivity is determined by the inner product of the latent positions H of two nodes: for each edge A_{ij} between node i and j, $A_{ij} \sim \text{Bernouilli}(H_i^T H_j)$. The crux of this method is that by converting an observed network into its latent positions, we can leverage the independence among its latent variables. In our setting, we propose to extend this setting to larger classes of graphs by learning node representations $H_i = GNN(X, A)$ of the graph (see Algorithm 5).

Algorithm 4 Resample Graphs through Block Bootstrap

Input: Spatial coordinates x_coord, y_coord, $grid_size$, $n_samples$.
Output: Block bootstrapped graphs $samples$.
for $i = 1$ to $n_samples$ **do**
  Step 1: Shuffle Data Points
  Create a grid over the spatial domain using coordinates x_coord and y_coord.
  Shuffle the grids to create new patched data, $shuffled_data$.
  Step 2: Convert Shuffled Data to Graphs
  Convert $shuffled_data$ into a graph by the method of choice (e.g. k-NN or radius graph)
  Store the graph $samples[i] = G$
end for

Algorithm 5 Resample Graphs through Network Bootstrap

Input: Graph G, embedding dimension d, $n_samples$, neighborhood size k
Output: Bootstrapped graph samples $samples$
Generate spectral embedding H of adjacency matrix using top d eigenvectors
for $i = 1$ to $n_samples$ **do**
  Sample indices $bootstrap_idx$ from H with replacement
  Generate new graph $\hat{A}$ from bootstrapped latent positions
  Initialize node features x as zeros in the new graph $\hat{G}$
  for each node i in $\hat{G}$ **do**
    Calculate distances from node i to all other nodes in H
    Sort distances and find nearest neighbors (based on neighborhood size k)
    Randomly select a neighbor and assign its features to node i
  end for
  Store the generated graph $\hat{G}$ in the sample set $samples$
end for

Bayesian Bootstrap. Rubin [1981] introduces the Bayesian bootstrap (BB) as a nonparametric alternative to traditional Bayesian inference, sidestepping the need for explicit likelihood functions. Unlike the frequentist bootstrap, which resamples data *with replacement*, the Bayesian bootstrap assigns Dirichlet-distributed random weights to observed data points, generating a posterior distribution for parameters of interest. Specifically, for a dataset $X = \{x_1, x_2, ..., x_n\}$, instead of sampling with replacement as in the frequentist bootstrap, the Bayesian bootstrap draws a random probability vector $p = (p_1, p_2, ..., p_n)$ from a $Dir(1, 1, \ldots, 1)$ distribution, ensuring that $\sum_{i=1}^{n} p_i = 1 \quad \text{and} \quad p_i > 0$. This randomized weighting serves as a Bayesian nonparametric prior, effectively treating the empirical distribution of the data as the prior distribution.

Efron [2012] explores the relationship between Bayesian inference and the parametric bootstrap, demonstrating how frequentist resampling techniques can be adapted to estimate posterior distributions. The key insight of this work is that the parametric bootstrap, traditionally used to approximate sampling distributions, can serve as an efficient computational tool for Bayesian inference when paired with importance sampling. Efron [2012] shows that bootstrap reweighting can be used to transform frequentist confidence intervals into Bayesian credible intervals. This approach provides a bridge between the two paradigms, enabling frequentist methods to yield posterior distributions without relying on Markov Chain Monte Carlo (MCMC) techniques.

The Bayesian bootstrap provides a perspective for interpreting the proposed nonparametric graph rewiring, particularly when edge resampling is guided by shared neighborhood structure. Just as the BB assigns Dirichlet-distributed weights to data points to construct a posterior distribution, the graph rewiring process can be seen as assigning probabilistic weights to edges based on local graph structure, thereby producing alternative realizations of the same graph. In this context, the neighborhood-weighted resampling in LOBSTUR aligns with Bayesian importance sampling, where the rewired edges represent a form of pseudo-posterior distribution over network structures.

Extended VGAE Approach. Inspired by Kipf and Welling [2016], we tried using the Variational Graph Autoencoder(VGAE) as a new graph sampler. The extension was needed as the original method only reconstructed the adjacency matrix. The proposed loss function includes a feature reconstruction component alongside the edge reconstruction and KL divergence losses. With the edge decoder designed by the original work, the feature decoder generates reconstructed node features, and the reconstruction loss for features is based on the sum of squared differences between the original and reconstructed features.

The total loss used for training consists of three parts: the KL divergence loss regularizing the latent variables, the edge reconstruction loss, and the feature reconstruction loss, scaled by a regularization parameter λ. The overall objective is:

$$\text{Total Loss} = \frac{\text{KL}}{n} + \text{loss}_A + \lambda \times \text{loss}_X$$

In our implementation, the parameter λ controls the weight of the feature reconstruction in the loss. This allows the model to focus primarily on learning the graph structure while still incorporating node feature information.

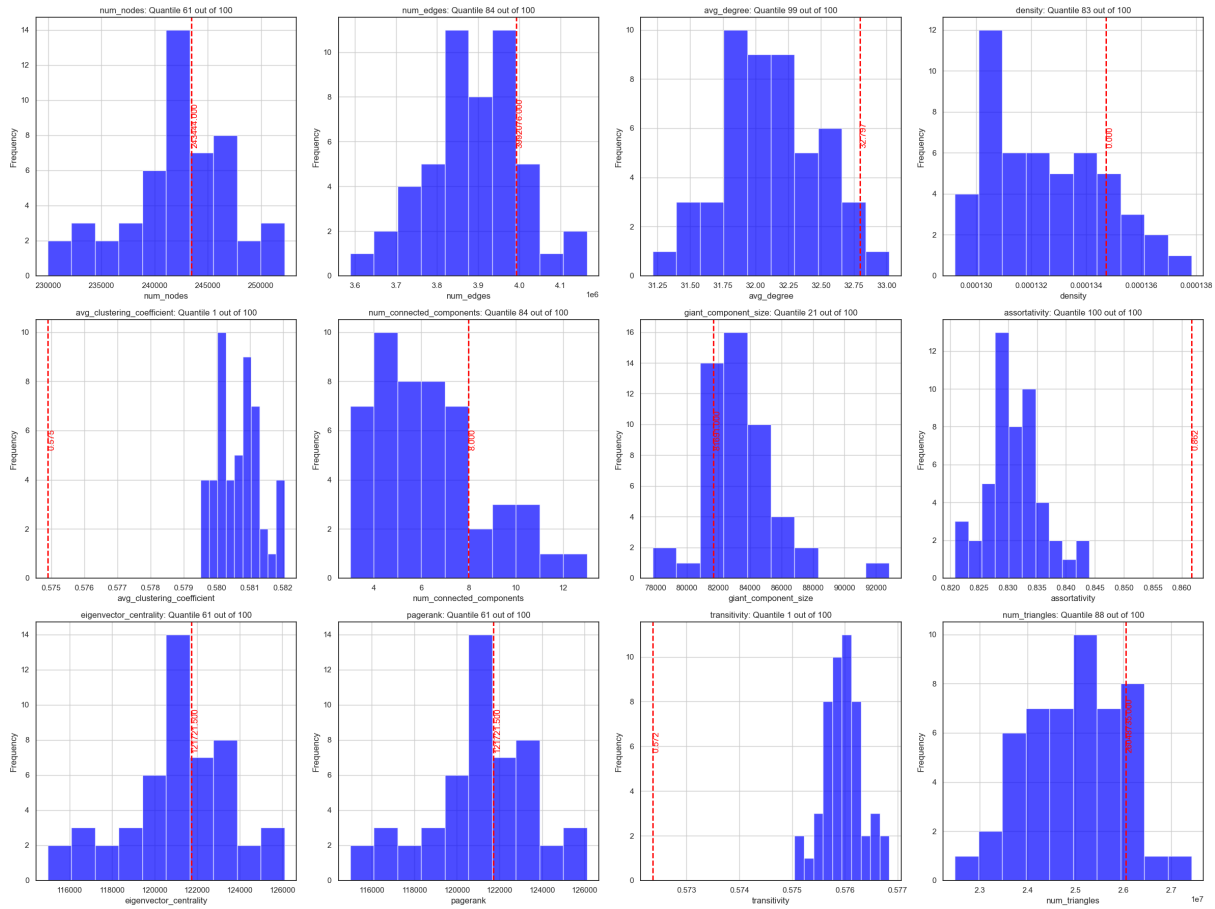


Figure 3: Block Bootstrap for Mouse Spleen data. Distribution of graph statistics of bootstrapped graphs. The principle is to see if the graph statistics of the original graph is within the extremity of the distribution of generated samples. The red dotted line indicates the statistics computed on the original graph. Most of the graph statistics do not lie at the extremity of the distribution of graph statistics by bootstrapped samples.

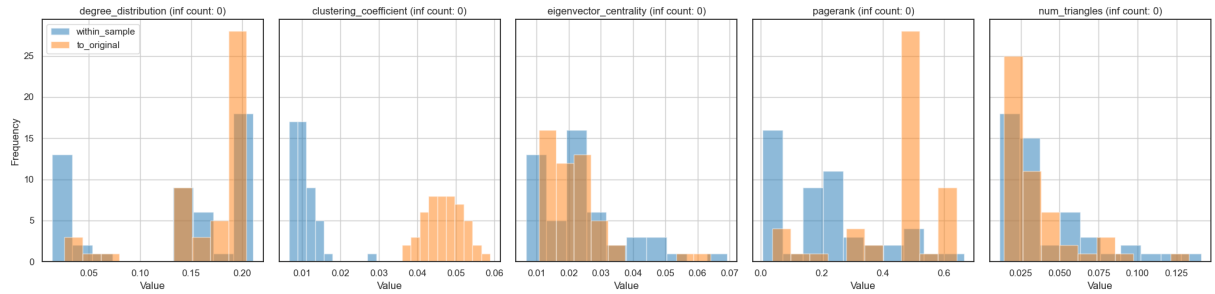


Figure 4: Block Bootstrap for Mouse Spleen data. Distribution of node-level statistics of bootstrapped graphs. The orange-colored distribution represents the JS divergence between the bootstrapped samples and the original graph, and the blue-colored distribution represents among bootstrapped samples divergence. The more the two distributions overlap, the bootstrapped samples 'mimic' the original graph well in terms of node-level statistics.

C.4.1 Experiments: Block Bootstrap.

C.5 Evaluating Embedding Qualities

In Section 4, we propose a stability metric. There are few works proposing metrics to evaluate the quality of unsupervised embeddings, although they are not intended for hyperparameter tuning.

Alignment-based metrics. Our first family of metrics focuses on measuring how well two embeddings align with each other. Suppose we have two embeddings, H_i and H_j, produced by the same learning procedure but on different graph folds. We propose two discretized versions of (2), measuring how much two embeddings align with each other.

(1) **Label Matching**: The first thing we can think of is to make the label from the embedding from each fold, which follows the "converting to the supervised task" convention.
   (a) Determine the clusters on embeddings using simple clustering algorithm such K-Nearest Neighbor or Gaussian Mixture Model(GMM)
   (b) Use widely used clustering evaluation metrics, such as Adjusted Rand Index(ARI) by Hubert and Arabie [1985] or Normalized Mutual Index(NMI), to see the labels from H_i and H_j agree to each other.

(2) **Neighborhood Matching**: If the model is able to extract enough of the latent structure of data, the model trained on the different folds of a graph should be similar. With this reasoning, we can evaluate the model by how much of the neighborhoods in the embedding agree with each other. To avoid the usage of data twice, we will evaluate the neighborhood from H_i and H_j and report the ratio of overlapping neighbors. To construct the neighborhood in the embedding space, we will use the simple k-Nearest algorithm with varying sizes of k. For each node on output embeddings, H_i and H_j, we first find the m-nearest neighbors. Then for node-level neighbor-kept ratio is defined as $N_i(m) = \#$ of overlapped neighbors$/m$, where $m \leq k$ is the neighbor size. Graph-level ratio can be calculated by simply averaging over the nodes, $N(m) = \sum_i N_i(m)$.

Direct Embedding Quality Metrics. Beyond measuring alignment between two embeddings, one can also evaluate an embedding's internal quality or degree of collapse. These methods offer a complementary view: even if two embeddings align with each other, they could both be suffering from dimension collapse or poor distribution of singular vectors.

(1) **RankMe**: Garrido et al. [2023] proposes *RankMe* a metric to measure the effective dimension of embeddings to quantify the embedding collapse in self-supervised learning. To overcome the numerical instability of the exact rank computation, for example, due to round-off error, they propose an alternative to use Shannon entropy of normalized singular values. The formula was originally proposed by Roy and Vetterli [2007] and then applied to dimension collapse context by Garrido et al. [2023]. Formally,

$$\text{RankMe}(H) = \exp\left(-\sum_{k=1}^{\min(n,p)} p_k \log p_k\right), \text{ with } p_k = \frac{\sigma_k(H)}{\|\sigma(H)\|_1} + \epsilon.$$

(2) **Metrics proposed in Tsitsulin et al. [2023]**: Tsitsulin et al. [2023] further extended the approaches and proposed four different metrics to evaluate the embedding quality in terms of embedding collapse and stability perspective. The key differences between their experiment setting and ours are, first, Tsitsulin et al. [2023] only consider the graph structure, not the node features, and second, they do not change the model parameters but change the level of perturbation on the structure (edge dropping or node masking). Let $H \in \mathbb{R}^{n\times p}$ be an embedding obtained from the trained unsupervised model of choice.
   (a) **Coherence**: The coherence metric measures how concentrated the rows of the singular vector matrix U are. A low coherence indicates that the energy is spread more uniformly across all rows (good for compressed sensing), while a high coherence suggests that the energy is concentrated in a few rows, which can indicate a poorly distributed set of singular vectors.

$$\text{Coherence}(H) = \frac{\max_i \|U_i\|_2^2 \cdot n}{p},$$

   where $U \in \mathbb{R}^{n\times p}$ is reduced left singular matrix of $H \in \mathbb{R}^{n\times p}$.
   (b) **Stable Rank**: It is the quantity called a 'numerical rank" (or effective rank) in numerical analysis.

$$\text{Stable Rank}(H) = \frac{\|H\|_F^2}{\|H\|_2^2},$$

   where $\|H\|_F$ denotes the Frobenius norm, $\|H\|_F^2 = \sum \sigma_i^2$, and $\|H\|_2 = \sigma_1$, where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ denotes the singular values of H.
   (c) **Pseudo-condition number**: Let SVD of the embedding H be $H = U\Sigma V^\top$.

$$\kappa_p(H) = \|H\|_p \|H^\dagger\|_p \overset{p=2}{=} \frac{\sigma_1}{\sigma_n}$$

   (d) **SelfCluster**: The idea is to estimate how much the embeddings are clustered in the embedding space compared to random distribution on a sphere. Let $\tilde{H} \in \mathbb{R}^{n\times p}$ be the normalized embeddings.

$$\text{SelfCluster}(H) = \frac{\|\tilde{H}\tilde{H}^\top\|_F - n - n \times (n-1)/d}{n^2 - n - n \times (n-1)/d}$$

D SCALABILITY

While our framework performs well on graphs of moderate size (up to 19k nodes, e.g., the Pubmed citation network), scalability remains a challenge. The bootstrapping procedure and CCA-based evaluation introduce significant additional computation, which can limit applicability to larger graphs. In particular, when applying our method to the OGBN-Arxiv dataset (over 170k nodes), we encountered substantial runtime challenges that made the process very time-consuming.

The main limitation, however, stems from the need to train multiple graph neural networks (GNNs) during the bootstrapping process. This requirement significantly increases the computational cost, but it is essential to ensure robust hyperparameter selection, especially in high-precision applications such as finance or biomedical domains, where reliability and unbiased evaluation are critical.

To address scalability challenges, we have begun exploring two strategies (1) block bootstrapping where the graph is partitioned into smaller subgraphs and bootstrapping is applied within blocks; and (2) approximate rewiring schemes to reduce computational overhead during resampling. Preliminary results for block bootstrapping, presented in C.4, suggest that this direction holds promise.

D.1 Alternative Algorithm for Scalability

Algorithm 6 Approximate Edge Rewiring via A^2

Input: Graph $G = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes, flattened list of edge stems $L = \{u \mid u \in E[:,0]\} \cup \{v \mid v \in E[:,1]\}$, where $E \in \mathbb{R}^{|\mathcal{E}|\times 2}$, and the squared adjacency matrix A^2 representing 2-hop connectivity strengths between nodes.
Initialize an empty graph $G' = (\mathcal{V}, \mathcal{E}')$ with n nodes.
Compute the sparse adjacency matrix A of G and symmetrize it to ensure it is undirected.
Compute the matrix product $A^2 = A \times A$, remove self-loops by setting the diagonal of A^2 to zero, and eliminate any zero entries.
while $\texttt{len}(L) > 0$ **do**
  Sample a source node u uniformly at random from the list L, and remove it from L.
  Retrieve the set of candidate nodes v for u, where each candidate v satisfies $A^2_{uv} > 0$ and $v \neq u$, and where $v \in L$.
  If no such candidate exists, discard u and continue to the next iteration.
  Otherwise, sample a target node v from the set of candidates according to the normalized weights given by A^2_{uv}.
  Remove the sampled node v from the list L.
  Add an undirected edge between u and v in the graph G'.
end while
Output: Bootstrapped graph G' with resampled edge structure.

The original edge rewiring algorithm (Algorithm 2 explores a node's local 1-hop neighborhood at each iteration. For a randomly selected node u, it first identifies its k-nearest neighbors based on some graph-based distance, then for each k-nearest neighbor m, it retrieves all direct neighbors $\mathcal{N}_A(m)$ in the original graph. The node u samples a new connection v from this dynamically built candidate set, with probabilities weighted by the frequency of appearance across different m. This ensures that edge resampling captures local neighborhood information around each node. However, this procedure incurs high computational cost because it needs to explore multiple neighborhoods at every rewiring step.

The approximate algorithm (Algorithm 6) instead precomputes the 2-hop neighborhood connectivity of the graph by squaring the adjacency matrix, yielding A^2. Here, $A \in \mathbb{R}^{n\times n}$ is the (symmetric) adjacency matrix of the graph, and A^2_{ij} counts the number of distinct 2-hop paths between nodes i and j. In this setting, each node u directly samples a target node v from the 2-hop neighbors based on their weighted connection strength given by A^2_{uv}. The sampling probability is proportional to the number of 2-hop paths between u and v, i.e.,

$$P(v|u) = \frac{A^2_{uv}}{\sum_{v' \in C(u)} A^2_{uv'}},$$

where $C(u)$ is the set of candidates for node u with positive 2-hop connectivity and available stems. If no candidates are found, the algorithm discards u and continues.

The relationship between Algorithm 2 and the approximate method (Algorithm 6) depends on the degree of each node and the choice of k for the k-nearest neighbor graph. Specifically, whether the candidate set in the original method is larger or smaller than the set of direct neighbors depends on the comparison between a node's degree and the size of k. If a node u has a low degree, meaning it is connected to only a few nodes in the original graph, then the k-nearest neighbor (k-NN) graph will forcefully connect it to k other nodes based on feature similarity or graph distance, even if u does not have that many direct connections. In this case, the k-NN set can be larger than the direct 1-hop neighbor set. The original algorithm supplements the missing local structure by adding neighbors based on external feature similarity rather than existing edges. Consequently, when $\deg(u) < k$, the original rewiring may result in a broader candidate set than the direct neighborhood.

On the other hand, if a node u has a high degree, meaning it is already connected to many nodes in the adjacency graph, then the k-nearest neighbor graph selects only a subset of its many neighbors. Here, k-NN acts as a filter, choosing the most "important" or closest k neighbors,

possibly ignoring others. In this case, because k is smaller than the degree, the k-NN candidate set becomes smaller than the full direct neighborhood. When $\deg(u) \geq k$, the original algorithm is thus more restrictive compared to simply traversing all direct neighbors.

Therefore, the original algorithm is not always narrower or broader by default; it depends on the relative size of a node's degree and k. This behavior is different from the approximate method using A^2, where no such filtering exists. The approximate method (Algorithm 6) uses all nodes that are reachable in exactly two hops, without considering feature space distances or k-nearest neighbor constraints. As a result, the approximate method includes any node with a 2-hop path from a node u, potentially adding candidates that would never have been explored in the original method, especially when the node's degree is small and the k-NN graph must reach out to faraway nodes.

E GNN EXPERIMENT DETAILS

We use benchmark datasets for node classification, including Cora, Pubmed, and Citeseer, and test our framework on node regression datasets from [Huang et al. 2023]. We summarize the datasets used to demonstrate the entire hyperparameter tuning procedure in Table 4.

We consider various benchmark datasets for node classification tasks, including Cora, Pubmed, Citeseer. Additionally, we have tested our framework on a few datasets for the node regression by [Huang et al. 2023]. To demonstrate our full framework for hyperparameter tuning, we used the following datasets, and their details are summarized in Table 4.

Dataset	Num Nodes	Num Edges	Num Classes	Description	Source
Cora	2708	5429	7	Citation network	PyTorch Geometric
Citeseer	3327	4732	6	Citation network	PyTorch Geometri
Pubmed	19717	44338	3	Citation network	PyTorch Geometric
Amazon Photo	7650	119081	8	Product co-purchasing network	PyTorch Geometric
Amazon Computers	13752	245861	10	Product co-purchasing network	PyTorch Geometric
Coauthor CS	18333	81894	15	Coauthorship network	PyTorch Geometric
Anaheim	914	3881	–	Graph of transportation networks	Conformalized GNN [Huang et al. 2023]
ChicagoSketch	2176	15104	–	Urban traffic network (sketch)	Conformalized GNN [Huang et al. 2023]
County Education	3234	12717	–	County-level education metrics (2012)	Conformalized GNN [Huang et al. 2023]
Twitch PTBR	1912	3170	–	Brazilian Twitch interactions	Conformalized GNN [Huang et al. 2023]

Table 4: Summary of benchmark datasets used for the experiments, including both classification and regression datasets.

The followings are tested combinations of hyperparameters, including different types of unsupervised GNN models.

- model: {CCA-SSG, DGI, BGRL, GRACE}
- feature masking rate (FMR): $\{0.05, 0.25, 0.5, 0.75\}$
- edge dropping rate (EDR): $\{0.05, 0.25, 0.5, 0.75\}$
- λ (CCA-SSG, BGRL) or τ (GRACE): $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1.0, 10.0\}$
- number of layers: 2
- hidden dimension: 256
- output dimension (p): 8
- learning rate: 10^{-3}
- epochs: 500
- number of simulations for each dataset (n_b): 20

E.1 Computer Resources Used

The experiments in this study were conducted using a combination of personal and institutional computational resources. Preliminary analyses and prototyping were performed on a MacBook Pro with an Intel Core i7 processor and 16GB of RAM. For larger-scale experiments, including graph bootstrapping and downstream evaluations, we used high-performance computing resources provided by the institution's research cluster, which includes access to multi-core CPUs and GPU-enabled nodes. While execution time varied by dataset and task, typical runs for clustering and evaluation completed within a few hours. Detailed resource specifications and runtime profiles are available upon request to support reproducibility.

F ADDITIONAL TABLES AND FIGURES

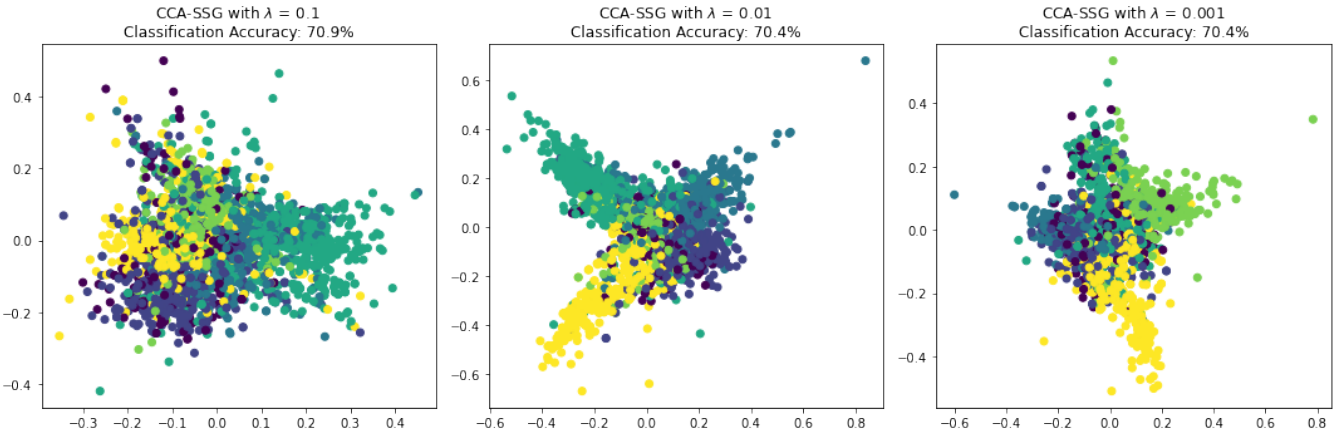
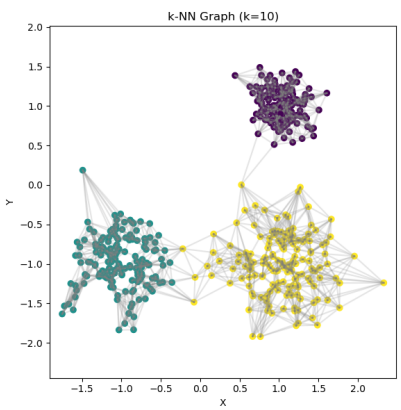


Figure 5: Citeseer: Model trained by different hyperparameters. 2D Visualization through PCA. The learned representations vary by the choice of hyperparameters.

F.1 Validation of Metrics

Synthetic Datasets. The motivation for using synthetic data is that we know the exact data-generating process (DGP), enabling us to replicate the dataset and focus on validating our metric. By controlling the DGP, we remove confounding factors related to real-world data and can better isolate and evaluate the performance of algorithms and metrics.

In this synthetic dataset generation, we create spatially structured data using a simple Gaussian blob. We first define n cluster centers and standard deviations to simulate spatial groupings in a 2D space, which belong to distinct clusters. For each point, we generate a 32-dimensional feature vector, with features generated from Laplacian eigenmap by [Belkin and Niyogi 2001]. The final dataset includes 2D spatial coordinates, cluster labels, and 32-dimensional feature vectors. We generated 15 copies of graphs following the same (and known) data-generating process. We run the procedure (Algorithm 3) and compute the metrics' average and prediction accuracy (Figure 6). Our proposed metric matches the clustering alignments (NMI, ARI) and shows a strong negative correlation with accuracy.



(a)

λ	gmm_nmi3	gmm_ari3	knn_nmi3	knn_ari3	cca_dist	acc
1e-07	0.98 ± 0.01	0.99 ± 0.01	0.95 ± 0.08	0.95 ± 0.08	357.32 ± 163.05	0.72 ± 0.34
1e-06	0.93 ± 0.14	0.92 ± 0.16	0.9 ± 0.18	0.89 ± 0.21	346.87 ± 251.77	0.73 ± 0.35
1e-05	0.91 ± 0.2	0.89 ± 0.23	0.92 ± 0.14	0.9 ± 0.22	343.39 ± 131.06	0.66 ± 0.32
0.0001	0.94 ± 0.05	0.95 ± 0.05	0.92 ± 0.12	0.92 ± 0.15	349.99 ± 119.31	0.73 ± 0.35
0.001	0.97 ± 0.02	0.98 ± 0.01	0.92 ± 0.12	0.9 ± 0.17	342.85 ± 108.51	0.71 ± 0.33
0.01	0.99 ± 0.02	0.99 ± 0.02	0.98 ± 0.02	0.99 ± 0.01	188.53 ± 162.76	0.74 ± 0.35
0.1	0.96 ± 0.06	0.97 ± 0.06	0.95 ± 0.06	0.95 ± 0.07	170.13 ± 79.8	0.74 ± 0.35
1.0	0.97 ± 0.03	0.98 ± 0.02	0.94 ± 0.08	0.96 ± 0.06	219.92 ± 62.21	0.74 ± 0.35
10.0	0.99 ± 0.01	1.0 ± 0.01	0.94 ± 0.05	0.95 ± 0.05	240.02 ± 39.63	0.73 ± 0.35
50.0	0.98 ± 0.01	0.99 ± 0.01	0.97 ± 0.02	0.98 ± 0.02	178.8 ± 36.83	0.74 ± 0.35
100.0	0.98 ± 0.01	0.99 ± 0.01	0.85 ± 0.18	0.83 ± 0.24	251.2 ± 72.93	0.72 ± 0.34

(b)

Figure 6: Summary of synthetic dataset and experiment results. The proposed metric and prediction accuracy show a strong negative Spearman rank correlation (-0.71).

F.1.1 Application to Spatial Single-Cell Datasets. There is a growing demand for robust computational tools that can extract biologically meaningful representations across heterogeneous samples. In such applications, it is crucial to obtain consistent and high-quality embeddings that generalize across samples while preserving fine-grained spatial structure. Our proposed metric is particularly well-suited for this goal, as it evaluates the stability and informativeness of unsupervised embeddings without requiring labeled data. When annotations are available,

we further demonstrate that our method aligns closely with manual labels, exhibiting strong spatial continuity and biological interpretability across a range of datasets.

Mouse Spleen (CODEX). We apply our procedure to a high-resolution spatial proteomics dataset of the mouse spleen from Goltsev et al. [2018]. This dataset, generated using CO-Detection by Indexing (CODEX), provides single-cell spatial and phenotypic profiles of immune cells across intact spleen tissue. With over 30 measured protein markers, it enables precise mapping of cell types, functional states, and spatial interactions at sub-tissue resolution. The dataset preserves key anatomical compartments—including T cell zones, B cell follicles, and red and white pulp—and highlights how spatial arrangement corresponds to immune function, such as structured lymphocyte zones and compartmentalized myeloid populations. We also have an access to the expert annotated lables, which we report the accuracy against it in Table 2. Figure 7 also refelects varying quality of learned embeddings by the choice of λ.

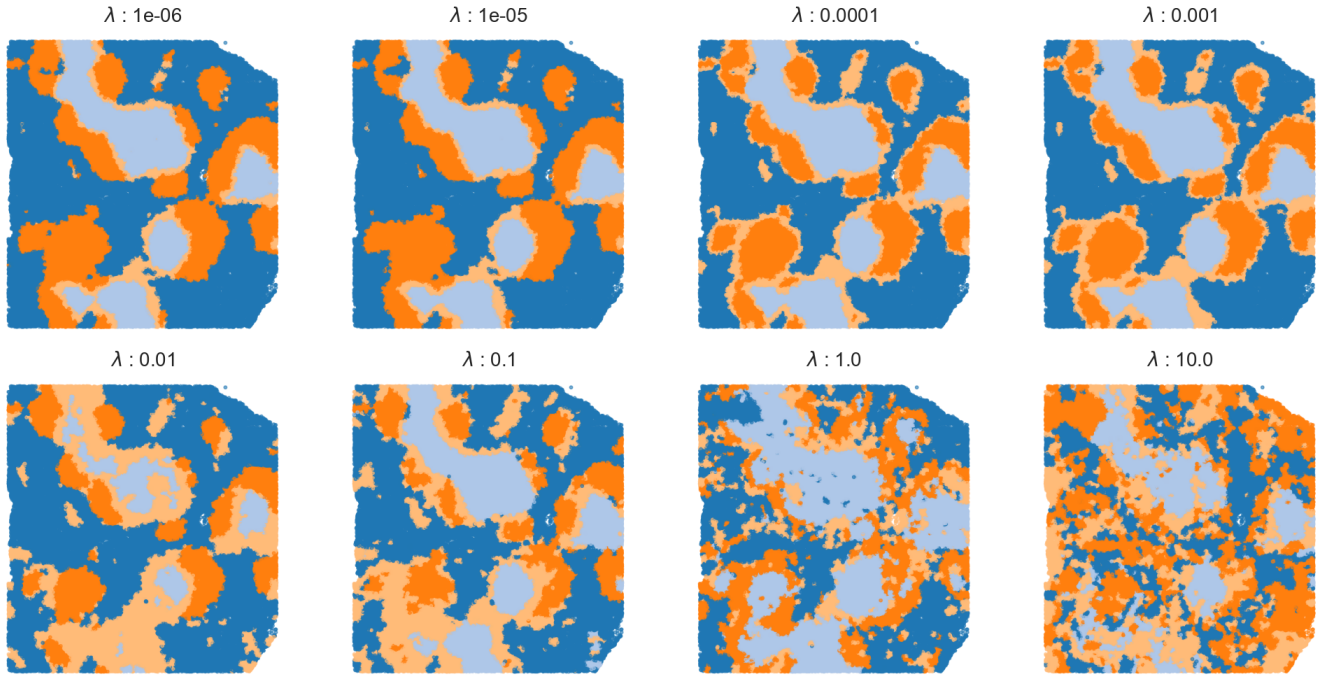


Figure 7: Visualizations of mouse spleen CODEX data based on the output of CCA-SSG model with different λ settings. We can observe that depending on the choice of λ, the quality of expression varies a lot. When λ becomes too large, the learned representation fail to recover the underlying cell environments. See Section 4.2 for the setup.

Triple Negative Breast Cancer Dataset. The dataset from Keren et al. [2018] comprises MIBI-TOF imaging data from 41 TNBC patients, capturing the spatial expression of 36 proteins across tumor, immune, and regulatory markers at subcellular resolution. Tumors are classified into three immune architectures—cold, mixed, and compartmentalized—based on spatial patterns of immune infiltration, cell type organization, and marker expression. Compartmentalized tumors are linked to the best survival outcomes. Mixed tumors, featuring intermingled tumor and immune cells with high CD8+ T cell and checkpoint marker expression, may benefit from immunotherapy. Cold tumors show sparse immune presence and poor prognosis. Among these, the mixed and compartmentalized tumor microenvironments (TMEs) represent favorable immune architectures that the authors aim to recover, as they are identified through a combination of cell type composition, spatial organization, and marker expression profiles. We predict such group (mixed vs. comparatmentalized) based on the learned embeddings. The AUC for the prediction is reported in Table 2.

Colorectal Cancer Dataset. The colorectal cancer (CRC) dataset from Schürch et al. [2020] includes 140 tissue regions from 35 advanced-stage CRC patients, profiled using FFPE-CODEX imaging with 56 protein markers to identify diverse cell populations within the tumor microenvironment (TME). The study identified nine distinct cellular neighborhoods (CNs) through unsupervised clustering of spatial co-occurrence patterns, revealing how the spatial organization of immune and stromal cells shapes immune responses. Two major immune architectures emerged (1) Crohn's-like reaction (CLR), associated with structured immune infiltration and favorable outcomes, and (2) diffuse inflammatory infiltration (DII), marked by disorganized immune presence and poor prognosis. These TMEs, distinguished by differences in cell types, spatial arrangements, and marker expression, represent the patterns the authors aim to recover, as they reflect clinically relevant immune organization associated with patient survival. The AUC for the predicting such group (CLR vs. DII) is reported in Table 2.

F.2 Validation of Bootstrap Samples

	True	Solution 1	Solution 2 (graph k-NN)
Number of Nodes	2708	2708±0	2708±0
Number of Edges	5278	5200.54±9	5171.78±7.34
Average Degree	3.8980	3.84±0.01	3.82±0.01
Density	0.0014	0±0	0±0
Avg Clustering Coefficient	0.2407	0.01±0	0.05±0
Avg Connected Component	78	13.16±3.26	67.91±6.7
Giant Component Size	2485	2684.28±6.51	2620.38±10.78
Assortativity	-0.0659	-0.06±0	-0.07±0
PageRank	1353.5	1353.5±0	1353.5±0
Transitivity	0.0935	0.01±0	0.03±0
Number of Triangles	1630	133.96±11.62	471.48±27.11

Table 5: Graph statistics for Cora illustrating the two solutions suggested in Section 3.2. We see the clear deviation on graph statistics, especially the average connected component and the number of triangles when we follow the *Solution 1*.

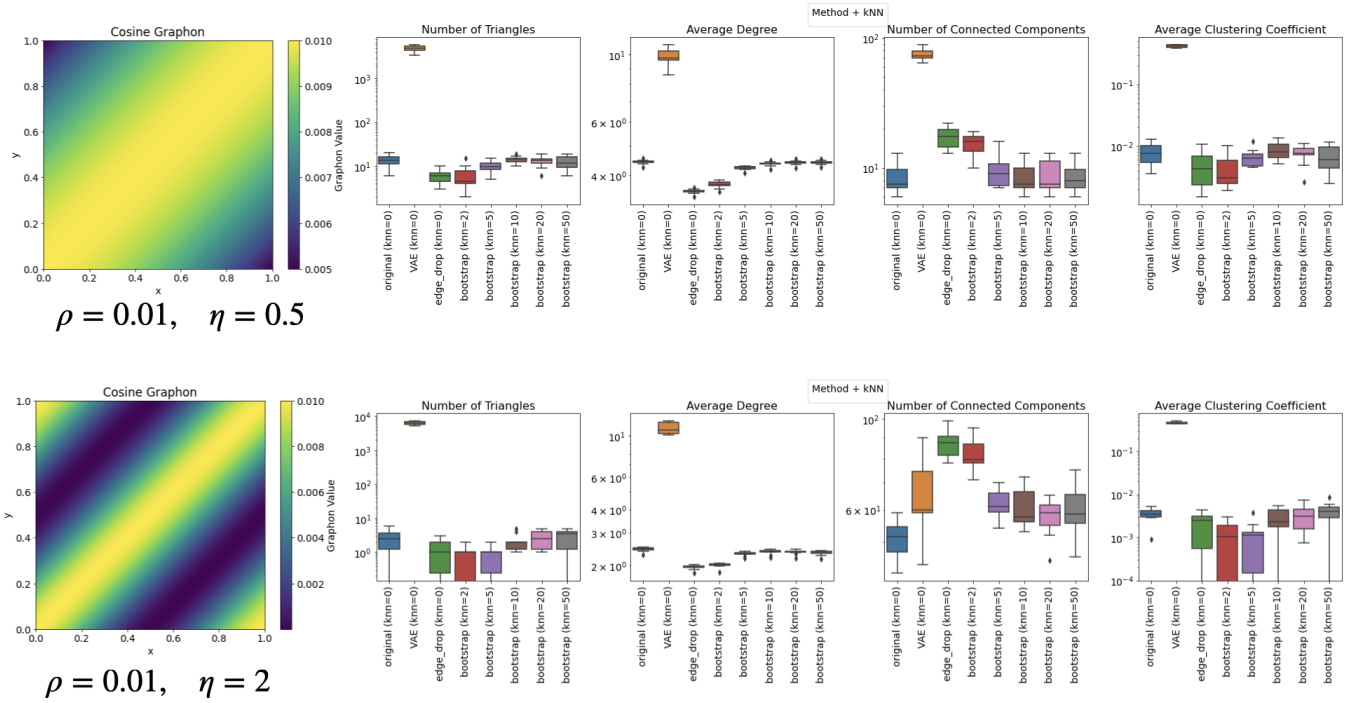


Figure 8: Visualization of the statistics obtained by different methods. The left most plot in each row corresponds to a visualization of the graphon function $W(x, y) = \rho * (1 + \cos(\eta\pi \cdot (x - y)))/2$, for $\rho = 0.01$ and different values of η. Each row presents a visualization of 10 instances of a resampling of a graphon generated according to W using different methods.

		True	Edge Drop	Node Drop	Ours	NB	VAE
	Assortativity	-0.0345	0±0	0±0	0±0	0±0	-0.44±0.05
	Avg ClusterCoefficient	0	0±0	0±0	0±0	0±0	0.12±0.01
	Avg Degree	0.12	0.12±0	0.1±0.01	0.08±0	0.04±0.02	2.63±0.47
	Density	0.0002	0±0	0±0	0±0	0±0	0.01±0
	Giant Component Size	3	2.93±0.26	2.53±0.5	2.79±0.4	4.12±1.36	90.08±8.22
Scenario 1	Num Connected Components	470	471.13±0.87	380.72±2.44	480±1.15	492.81±3.52	410.89±8.21
	Num Edges	30	28.87±0.87	19.28±2.44	20±1.15	8.82±5.12	657.73±118.12
	Num Nodes	500	500±0	400±0	500±0	500±0	500±0
	Num Triangles	0	0±0	0±0	0±0	1.48±2.22	3060.65±1108.48
	PageRank	249.5	249.5±0	199.5±0	249.5±0	249.5±0	249.5±0
	Transitivity	0	0±0	0±0	0±0	0.33±0.31	0.47±0.05
	Assortativity	-0.0227	-0.02±0.01	-0.02±0.04	-0.02±0.03	0.01±0.04	0.04±0.04
	Avg ClusterCoefficient	0.0064	0.01±0	0.01±0	0.01±0	0.02±0.01	0.16±0.01
	Avg Degree	3.224	3.1±0.02	2.58±0.07	3.2±0.01	3.68±0.29	3.13±0.09
	Density	0.0065	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
	Giant Component Size	479	475.02±2.21	360.97±6.3	481.21±2.64	407.42±11.16	70.91±7.09
Scenario 2	Num Connected Components	18	21.41±1.81	33.23±4.71	17.89±1.25	91.93±10.74	391.57±4.03
	Num Edges	806	774.11±4.42	515.92±13.17	801.17±1.81	921±71.47	782.12±21.59
	Num Nodes	500	500±0	400±0	500±0	500±0	500±0
	Num Triangles	7	6.16±0.89	3.59±1.35	5.42±2.36	68.31±21.3	4469.92±189.99
	PageRank	249.5	249.5±0	199.5±0	249.5±0	249.5±0	249.5±0
	Transitivity	0.0083	0.01±0	0.01±0	0.01±0	0.04±0.01	0.69±0.01
	Assortativity	-0.0227	-0.02±0.01	-0.02±0.04	-0.01±0.03	0.01±0.04	-0.58±0.06
	Avg ClusterCoefficient	0.0064	0.01±0	0.01±0	0.01±0	0.02±0.01	0.11±0.02
	Avg Degree	3.224	3.1±0.02	2.58±0.07	3.21±0.01	3.68±0.29	1.28±0.18
	Density	0.0065	0.01±0	0.01±0	0.01±0	0.01±0	0±0
	Giant Component Size	479	475.02±2.21	360.97±6.3	481.2±2.77	407.42±11.16	49.52±10.72
Scenario 3	Num Connected Components	18	21.41±1.81	33.23±4.71	17.84±1.28	91.93±10.74	421.24±11.14
	Num Edges	806	774.11±4.42	515.92±13.17	801.7±1.61	921±71.47	321.11±45.67
	Num Nodes	500	500±0	400±0	500±0	500±0	500±0
	Num Triangles	7	6.16±0.89	3.59±1.35	5.74±2.34	68.31±21.3	798.48±165.04
	PageRank	249.5	249.5±0	199.5±0	249.5±0	249.5±0	249.5±0
	Transitivity	0.0083	0.01±0	0.01±0	0.01±0	0.04±0.01	0.46±0.06
	Assortativity	-0.0833	0±0	0±0	0±0	0±0	-0.02±0.11
	Avg ClusteringCoefficient	0	0±0	0±0	0±0	0±0.01	0.23±0.01
	Avg Degree	0.104	0.1±0	0.08±0.01	0.07±0.01	0.04±0.02	7.39±0.37
	Density	0.0002	0±0	0±0	0±0	0±0	0.01±0
	Giant Component Size	3	3±0.06	2.77±0.42	2.58±0.49	4.28±1.43	81.19±7.89
Scenario 4	Num Connected Components	474	475.1±0.82	383.45±2.4	482.85±1.29	492.15±3.75	350.19±9.04
	Num Edges	26	24.9±0.82	16.55±2.4	17.15±1.29	9.68±5.66	1847.28±91.84
	Num Nodes	500	500±0	400±0	500±0	500±0	500±0
	Num Triangles	0	0±0	0±0	0±0	1.75±2.65	19727.09±959.62
	PageRank	249.5	249.5±0	199.5±0	249.5±0	249.5±0	249.5±0
	Transitivity	0	0±0	0±0	0±0	0.34±0.32	0.75±0.02

Table 6: Comparison of all sampling methods on graphon model as posited in Equation 1. The ground truth graphon is generated with n = 500, p = 150, k = 20. For each methods, 500 (bootstrap) samples are generated. For edge and node drop, we randomly remove 20% of edges or nodes (and corresponding edges). Our proposed nonparametric bootstrap consistently achieves significant similarities with the ground truth graph under different scenarios.

We analyze four scenarios (each in Table 7, 8, 9, and 10) of recovering the underlying dependency by our proposed nonparametric bootstrap method either through graph-knn or feature-knn. The graph is generated by the model posited in Equation 1 with varying graphon function W and feature generator g.

			graph-kNN			feature-kNN		
		Original	**k = 5**	**k=20**	**k=50**	**k = 5**	**k=20**	**k=50**
	Assortativity					0±0	0±0	0±0
	Avg ClusterCoefficient	0	0±0	0±0	0±0	0±0	0±0	0±0
	Avg Degree	0.02	0±0	0±0	0±0	0±0	0±0	0±0
	Density	0.0002	0±0	0±0	0±0	0±0	0±0	0±0
	Giant Component Size	2	1±0	1±0	1±0	1±0	1±0	1±0
n= 100	Num Connected Components	99	100±0	100±0	100±0	100±0	100±0	100±0
	Num Edges	1	0±0	0±0	0±0	0±0	0±0	0±0
	Num Nodes	100	100±0	100±0	100±0	100±0	100±0	100±0
	Num Triangles	0	0±0	0±0	0±0	0±0	0±0	0±0
	PageRank	49.5	49.5±0	49.5±0	49.5±0	49.5±0	49.5±0	49.5±0
	Transitivity	0	0±0	0±0	0±0	0±0	0±0	0±0
	Assortativity	-0.0345	0±0	0±0	0±0	0±0	0±0	0±0
	Avg ClusterCoefficient	0	0±0	0±0	0±0	0±0	0±0	0±0
	Avg Degree	0.12	0±0	0.01±0	0.02±0	0.04±0.01	0.08±0	0.11±0
	Density	0.0002	0±0	0±0	0±0	0±0	0±0	0±0
	Giant Component Size	3	2.04±0.82	2.32±0.47	2.34±0.47	2±0	2.79±0.4	2.94±0.24
n=500	Num Connected Components	470	498.96±0.82	497.07±0.82	495.4±0.96	489.19±1.61	480±1.15	472.94±0.86
	Num Edges	30	1.04±0.82	2.93±0.82	4.6±0.96	10.81±1.61	20±1.15	27.06±0.86
	Num Nodes	500	500±0	500±0	500±0	500±0	500±0	500±0
	Num Triangles	0	0±0	0±0	0±0	0±0	0±0	0±0
	PageRank	249.5	249.5±0	249.5±0	249.5±0	249.5±0	249.5±0	249.5±0
	Transitivity	0	0±0	0±0	0±0	0±0	0±0	0±0
	Assortativity	-0.112	-0.39±0.15	-0.36±0.14	-0.33±0.13	0±0	-0.01±0.11	-0.02±0.08
	Avg ClusterCoefficient	0	0±0	0±0	0±0	0±0	0±0	0±0
	Avg Degree	0.182	0.03±0.01	0.03±0.01	0.04±0.01	0.09±0.01	0.14±0	0.17±0
	Density	0.0002	0±0	0±0	0±0	0±0	0±0	0±0
	Giant Component Size	4	3.45±0.5	3.42±0.49	3.46±0.51	3.37±0.49	4.06±0.6	4.23±0.48
n=1000	Num Connected Components	909	984.21±2.87	983.32±2.82	980.71±2.92	955.1±2.99	928.42±2.25	916.2±1.51
	Num Edges	91	15.79±2.87	16.68±2.82	19.29±2.92	44.9±2.99	71.6±2.24	83.81±1.51
	Num Nodes	1000	1000±0	1000±0	1000±0	1000±0	1000±0	1000±0
	Num Triangles	0	0±0	0±0	0±0	0±0	0.02±0.13	0±0.06
	PageRank	499.5	499.5±0	499.5±0	499.5±0	499.5±0	499.5±0	499.5±0
	Transitivity	0	0±0	0±0	0±0	0±0	0.01±0.04	0±0.01

Table 7: Scenario 1: the graph structure is highly localized ($W(u,v) = \mathbb{1}\{|u-v| < 0.01\}$), leading to disconnected components and the failure of graph-based kNN, while features ($\mathcal{N}(5u, 0.01)$) exhibit a strong correlation with the latent variable u, enabling feature-based kNN success. The greyed-out cells indicate values that are unavailable.

			graph-kNN			feature-kNN		
		Original	**k = 5**	**k=20**	**k=50**	**k = 5**	**k=20**	**k=50**
	Assortativity	0.2884	-0.24±0.16	-0.12±0.18	-0.13±0.17	-0.01±0.19	-0.11±0.17	-0.1±0.17
	Avg ClusterCoefficient	0	0±0	0±0	0±0	0±0	0±0	0±0
	Avg Degree	0.66	0.37±0.03	0.49±0.03	0.6±0.02	0.53±0.03	0.63±0.02	0.65±0.01
	Density	0.0067	0±0	0±0	0.01±0	0.01±0	0.01±0	0.01±0
	Giant Component Size	8	6.15±1.21	6.36±1.4	5.97±1.23	5.22±1.26	5.86±1.25	6.09±1.39
n= 100	Num Connected Components	67	81.9±1.59	75.54±1.45	70.24±1.24	73.58±1.56	68.5±0.79	67.6±0.56
	Num Edges	33	18.27±1.57	24.54±1.44	29.78±1.24	26.44±1.55	31.52±0.79	32.41±0.55
	Num Nodes	100	100±0	100±0	100±0	100±0	100±0	100±0
	Num Triangles	0	0±0	0.02±0.15	0.02±0.13	0.01±0.12	0.01±0.12	0.01±0.12
	PageRank	49.5	49.5±0	49.5±0	49.5±0	49.5±0	49.5±0	49.5±0
	Transitivity	0	0±0	0.01±0.03	0±0.03	0±0.03	0±0.02	0±0.02
	Assortativity	-0.0359	-0.12±0.03	-0.08±0.03	-0.08±0.03	-0.04±0.03	-0.02±0.03	-0.01±0.04
	Avg ClusterCoefficient	0.0052	0±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
	Avg Degree	3.076	2.91±0.02	3.03±0.01	3.02±0.01	2.96±0.02	3.06±0.01	3.07±0
	Density	0.0062	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
	Giant Component Size	471	463.59±3.66	466.91±3.48	468.15±3.02	466.85±3.11	469.38±2.64	469.54±2.58
n=500	Num Connected Components	29	34.37±2.24	31.73±1.94	31.15±1.79	32.07±1.91	29.84±1.27	29.66±1.23
	Num Edges	769	727.53±5.05	757.88±2.53	756.11±2.92	740.32±4.78	764.09±1.75	767.16±1.04
	Num Nodes	500	500±0	500±0	500±0	500±0	500±0	500±0
	Num Triangles	7	4.08±2.01	5.03±2.33	5.09±2.33	4.54±2.05	5.37±2.38	5.48±2.33
	PageRank	249.5	249.5±0	249.5±0	249.5±0	249.5±0	249.5±0	249.5±0
	Transitivity	0.0087	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
	Assortativity	0.0122	-0.05±0.02	-0.03±0.02	-0.04±0.02	-0.01±0.02	0±0.02	0±0.02
	Avg ClusterCoefficient	0.0078	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
	Avg Degree	6.594	6.37±0.02	6.56±0.01	6.59±0	6.5±0.01	6.58±0	6.59±0
	Density	0.0066	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
	Giant Component Size	999	998.74±0.5	998.98±0.13	999±0.06	998.94±0.26	998.99±0.08	998.99±0.15
n=1000	Num Connected Components	2	2.25±0.5	2.02±0.13	2±0.06	2.06±0.25	2.01±0.08	2.01±0.1
	Num Edges	3297	3185.15±9.77	3281.22±3.6	3293.46±1.6	3248.4±6.06	3289.46±2.37	3294.19±1.49
	Num Nodes	1000	1000±0	1000±0	1000±0	1000±0	1000±0	1000±0
	Num Triangles	50	40.6±6.86	47.34±6.66	48.53±6.7	46.97±6.8	50.03±7.75	50.54±7.07
	PageRank	499.5	499.5±0	499.5±0	499.5±0	499.5±0	499.5±0	499.5±0
	Transitivity	0.0069	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0

Table 8: Scenario 2 has a well-structured graph ($W(u,v) = 1 - |u - v|$), ensuring graph kNN success, but highly oscillatory features ($\sin(10u) + \mathcal{N}(0, 0.1)$) disrupt feature-based kNN.

			graph-kNN			feature-kNN		
		Original	**k = 5**	**k=20**	**k=50**	**k = 5**	**k=20**	**k=50**
n= 100	Assortativity	-0.0344	-0.15±0.14	-0.08±0.15	-0.06±0.14	-0.05±0.15	-0.09±0.16	-0.08±0.15
	Avg ClusterCoefficient	0	0±0	0±0.01	0±0.01	0±0.01	0±0.01	0±0.01
	Avg Degree	0.78	0.56±0.03	0.66±0.03	0.74±0.02	0.65±0.03	0.75±0.01	0.77±0.01
	Density	0.0079	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
	Giant Component Size	13	10.37±2.03	10.83±3.2	13.93±3.78	10.35±3.12	13.05±3.86	13.88±3.95
	Num Connected Components	62	72.71±1.53	67.23±1.49	63.09±1.09	67.71±1.65	62.69±0.85	61.78±0.7
	Num Edges	39	27.91±1.45	32.95±1.43	37.19±1	32.46±1.63	37.52±0.73	38.46±0.56
	Num Nodes	100	100±0	100±0	100±0	100±0	100±0	100±0
	Num Triangles	0	0.01±0.09	0.1±0.31	0.12±0.34	0.09±0.31	0.1±0.3	0.11±0.32
	PageRank	49.5	49.5±0	49.5±0	49.5±0	49.5±0	49.5±0	49.5±0
	Transitivity	0	0±0.01	0.01±0.04	0.01±0.03	0.01±0.04	0.01±0.03	0.01±0.03
n=500	Assortativity	-0.009	-0.09±0.03	-0.09±0.03	-0.11±0.03	-0.05±0.03	-0.02±0.04	-0.02±0.03
	Avg ClusterCoefficient	0.00575685	0±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
	Avg Degree	3.228	3.07±0.02	3.19±0.01	3.19±0.01	3.15±0.01	3.21±0.01	3.22±0
	Density	0.0065	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
	Giant Component Size	479	473.05±3	474.67±2.97	476.17±2.19	476.13±2.16	476.77±2.06	476.69±2.09
	Num Connected Components	22	26.44±2	24.65±1.57	23.92±1.34	23.9±1.29	23.14±0.99	23.14±1.01
	Num Edges	807	766.71±5.09	796.85±2.43	796.3±2.54	787.46±3.64	802.61±1.64	805.05±1.07
	Num Nodes	500	500±0	500±0	500±0	500±0	500±0	500±0
	Num Triangles	8	4.08±2.03	5.18±2.31	5.23±2.25	4.94±2.15	5.97±2.44	6.02±2.59
	PageRank	249.5	249.5±0	249.5±0	249.5±0	249.5±0	249.5±0	249.5±0
	Transitivity	0.0094	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
n=1000	Assortativity	0.0014	-0.06±0.02	-0.03±0.02	-0.06±0.02	0±0.02	0.01±0.02	0.01±0.02
	Avg ClusterCoefficient	0.0077	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
	Avg Degree	6.584	6.36±0.02	6.55±0.01	6.58±0	6.53±0.01	6.57±0	6.58±0
	Density	0.0066	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
	Giant Component Size	997	996.68±0.63	996.94±0.31	996.99±0.15	996.97±0.19	996.96±0.28	996.98±0.2
	Num Connected Components	4	4.3±0.57	4.04±0.19	4.01±0.1	4.03±0.19	4.02±0.15	4.01±0.11
	Num Edges	3292	3179.28±10.22	3276.36±3.69	3288.4±1.53	3262.62±4.73	3285.76±2.2	3289.42±1.38
	Num Nodes	1000	1000±0	1000±0	1000±0	1000±0	1000±0	1000±0
	Num Triangles	60	47.84±7.03	49.11±7.17	50.94±7.08	50.74±6.88	55.12±6.99	55.54±7.53
	PageRank	499.5	499.5±0	499.5±0	499.5±0	499.5±0	499.5±0	499.5±0
	Transitivity	0.0082	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0

Table 9: Scenario 3 maintains a structured graph ($W(u, v) = 1 - |u - v|$) and smooth feature variation ($\mathcal{N}(5u, 0.01)$), leading to the success of both methods.

			graph-kNN			feature-kNN		
		Original	**k = 5**	**k=20**	**k=50**	**k = 5**	**k=20**	**k=50**
	Assortativity					0±0	0±0	0±0
	Avg ClusterCoefficient	0	0±0	0±0	0±0	0±0	0±0	0±0
	Avg Degree	0.02	0±0	0±0	0±0	0±0	0±0	0±0
	Density	0.00020202	0±0	0±0	0±0	0±0	0±0	0±0
	Giant Component Size	2	1±0	1±0	1±0	1±0	1±0	1±0
n= 100	Num Connected Components	99	100±0	100±0	100±0	100±0	100±0	100±0
	Num Edges	1	0±0	0±0	0±0	0±0	0±0	0±0
	Num Nodes	100	100±0	100±0	100±0	100±0	100±0	100±0
	Num Triangles	0	0±0	0±0	0±0	0±0	0±0	0±0
	PageRank	49.5	49.5±0	49.5±0	49.5±0	49.5±0	49.5±0	49.5±0
	Transitivity	0	0±0	0±0	0±0	0±0	0±0	0±0
	Assortativity	-0.2	0±0	-0.34±0.19	0±0	0±0	0±0	-0.15±0.1
	Avg ClusterCoefficient	0	0±0	0±0	0±0	0±0	0±0	0±0
	Avg Degree	0.12	0.02±0.01	0.03±0.01	0.04±0.01	0.04±0.01	0.08±0.01	0.11±0
	Density	0.00024048	0±0	0±0	0±0	0±0	0±0	0±0
	Giant Component Size	3	2.89±0.31	3.11±0.32	3.07±0.26	2.8±0.4	3±0.04	3.16±0.37
n=500	Num Connected Components	470	494.14±1.63	491.73±1.63	489.81±1.48	489.6±1.66	480.53±1.37	472.85±1.03
	Num Edges	30	5.86±1.63	8.27±1.63	10.19±1.48	10.4±1.66	19.47±1.37	27.15±1.03
	Num Nodes	500	500±0	500±0	500±0	500±0	500±0	500±0
	Num Triangles	0	0±0	0±0	0±0	0±0	0±0	0±0
	PageRank	249.5	249.5±0	249.5±0	249.5±0	249.5±0	249.5±0	249.5±0
	Transitivity	0	0±0	0±0	0±0	0±0	0±0	0±0
	Assortativity	0.14150943	-0.15±0.16	-0.08±0.15	-0.06±0.13	-0.09±0.08	0±0.1	0±0.1
	Avg ClusterCoefficient	0	0±0	0±0	0±0	0±0	0±0	0±0
	Avg Degree	0.208	0.05±0.01	0.05±0.01	0.06±0.01	0.09±0.01	0.17±0	0.19±0
	Density	0.00020821	0±0	0±0	0±0	0±0	0±0	0±0
	Giant Component Size	7	6.31±0.8	6.18±0.86	6.21±0.86	3.39±0.6	4.58±0.75	5.22±0.98
n=1000	Num Connected Components	896	976.5±2.95	975.11±2.75	970.02±2.58	953.18±3.15	916.91±2.48	904±1.75
	Num Edges	104	23.71±2.94	25.15±2.76	30.18±2.55	46.82±3.15	83.11±2.48	96.01±1.74
	Num Nodes	1000	1000±0	1000±0	1000±0	1000±0	1000±0	1000±0
	Num Triangles	0	0±0	0±0	0±0	0±0	0.02±0.13	0.01±0.09
	PageRank	499.5	499.5±0	499.5±0	499.5±0	499.5±0	499.5±0	499.5±0
	Transitivity	0	0±0	0±0	0±0	0±0	0±0.03	0±0.01

Table 10: Scenario 4 combines a fragmented graph ($W(u,v) = \mathbb{1}\{|u-v| < 0.01\}$) with oscillatory features ($\sin(10u) + \mathcal{N}(0, 0.1)$), making the problem hard for both graph- and feature-based kNN. The greyed-out cells indicate values that are unavailable.

	Statistic	Original	k=3	k=5	k=7	k=10	k=15	k=20	k=50
Cora	Number of Nodes	2708	2708±0	2708±0	2708±0	2708±0	2708±0	2708±0	2708±0
	Number of Edges	5278	4793.52±14.73	4962.71±13.23	5035.19±12.52	5087.69±10.33	5154.92±9.11	5171.78±7.34	5196.15±7.42
	Average Degree	3.90	3.54±0.01	3.67±0.01	3.72±0.01	3.76±0.01	3.81±0.01	3.82±0.01	3.84±0.01
	Density	0.00	0±0	0±0	0±0	0±0	0±0	0±0	0±0
	Avg Clustering Coefficient	0.24	0.1±0.01	0.09±0	0.08±0	0.07±0	0.06±0	0.05±0	0.03±0
	Avg Connected Component	78	136.32±6.99	114.71±6.76	97.29±7.3	97.52±7	62.44±6.95	67.91±6.7	42.54±6.08
	Giant Component Size	2485	2479.92±19.9	2530.81±21.35	2571.64±18.39	2580.51±14.08	2625.44±11.64	2620.38±10.78	2652.33±9.41
	Assortativity	-0.07	-0.09±0	-0.09±0	-0.08±0	-0.08±0	-0.08±0	-0.07±0	-0.08±0
	PageRank	1353	1353.5±0	1353.5±0	1353.5±0	1353.5±0	1353.5±0	1353.5±0	1353.5±0
	Transitivity	0.09	0.04±0	0.04±0	0.04±0	0.03±0	0.03±0	0.03±0	0.02±0
	Number of Triangles	1630	716.03±32.22	664.18±29.42	623.48±29.33	575.85±29.4	512.85±27.32	471.48±27.11	319.15±21.65
Citeseer	Number of Nodes	3327	3327±0	3327±0	3327±0	3327±0	3327±0	3327±0	3327±0
	Number of Edges	4552	3846.33±15.1	3951.45±13.58	3997.51±13.36	4031.65±12.6	4059.15±11.71	4127.78±10.93	4150.58±11.39
	Average Degree	2.74	2.31±0.01	2.38±0.01	2.4±0.01	2.42±0.01	2.44±0.01	2.48±0.01	2.5±0.01
	Density	0.00	0±0	0±0	0±0	0±0	0±0	0±0	0±0
	Avg Clustering Coefficient	0.14	0.05±0	0.05±0	0.04±0	0.04±0	0.04±0	0.03±0	0.03±0
	Avg Connected Component	438	868.13±12.49	848.01±10.31	840.08±11.44	830.07±10.87	795.53±10.75	635.09±11.87	570.92±13.67
	Giant Component Size	2120	1934.25±42.19	2010.02±32.77	2043.56±31.79	2063.28±31.07	2098.62±36.32	2418.12±35.69	2585.67±23.96
	Assortativity	0.05	-0.02±0.01	-0.01±0.01	-0.01±0.01	0±0.01	0.01±0.01	-0.08±0	-0.1±0
	PageRank	1663	1663±0	1663±0	1663±0	1663±0	1663±0	1663±0	1663±0
	Transitivity	0.13	0.07±0	0.07±0	0.06±0	0.05±0	0.05±0	0.04±0	0.03±0
	Number of Triangles	1167	574.04±29.17	550.58±28.97	520.69±27.43	462.24±27.24	431.58±25.85	304.6±19.59	227.62±16.37
ChicagoSketch	Number of Nodes	2176	2176±0	2176±0	2176±0	2176±0	2176±0	2176±0	2176±0
	Number of Edges	15104	14505.33±22.7	14811.91±17.53	14914.4±14.51	14956.5±12.89	14999.55±11.49	15014.23±10.28	15066.34±5.57
	Average Degree	13.88	13.33±0.02	13.61±0.02	13.71±0.01	13.75±0.01	13.79±0.01	13.8±0.01	13.85±0.01
	Density	0.01	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
	Avg Clustering Coefficient	0.57	0.19±0	0.17±0	0.16±0	0.14±0	0.13±0	0.12±0	0.08±0
	Avg Connected Component	1	1±0	1±0	1±0	1±0	1±0	1±0	1±0
	Giant Component Size	2176	2176±0	2176±0	2176±0	2176±0	2176±0	2176±0	2176±0
	Assortativity	0.65	0.31±0.01	0.3±0.01	0.3±0.01	0.31±0.01	0.31±0.01	0.29±0.01	0.13±0.01
	PageRank	1087	1087.5±0	1087.5±0	1087.5±0	1087.5±0	1087.5±0	1087.5±0	1087.5±0
	Transitivity	0.56	0.19±0	0.17±0	0.16±0	0.14±0	0.13±0	0.12±0	0.08±0
	Number of Triangles	38240	11919.95±119.57	11097.95±105.45	10402.11±108.83	9577.65±106.85	8637.74±102.45	8006.58±95.13	5592.24±76.04
Twitch_PTBR	Number of Nodes	1912	1912±0	1912±0	1912±0	1912±0	1912±0	1912±0	1912±0
	Number of Edges	31299	30803.01±27.13	30985.09±25.09	31049.84±24.48	31076.91±23.3	31084.55±23.33	31082.05±22.83	31058.76±25.23
	Average Degree	32.74	32.22±0.03	32.41±0.03	32.48±0.03	32.51±0.02	32.52±0.02	32.51±0.02	32.49±0.03
	Density	0.02	0.02±0	0.02±0	0.02±0	0.02±0	0.02±0	0.02±0	0.02±0
	Avg Clustering Coefficient	0.32	0.17±0	0.17±0	0.17±0	0.17±0	0.17±0	0.17±0	0.17±0
	Avg Connected Component	1.00	1.33±0.55	1.17±0.4	1.2±0.44	1.26±0.5	1.19±0.44	1.14±0.39	1.07±0.25
	Giant Component Size	1912	1911.31±1.19	1911.64±0.84	1911.59±0.9	1911.47±1.01	1911.61±0.88	1911.72±0.77	1911.87±0.5
	Assortativity	-0.23	-0.31±0	-0.3±0	-0.3±0	-0.3±0	-0.29±0	-0.29±0	-0.28±0
	PageRank	955	955.5±0	955.5±0	955.5±0	955.5±0	955.5±0	955.5±0	955.5±0
	Transitivity	0.13	0.08±0	0.08±0	0.08±0	0.08±0	0.08±0	0.08±0	0.08±0
	Number of Triangles	173510	103368.74±1614.98	102572.23±1741.49	103379.78±1759.64	104007.92±1678.98	105301.1±1862.01	105534.51±1904.54	106230.23±1900.76
Education	Number of Nodes	3234	3234±0	3234±0	3234±0	3234±0	3234±0	3234±0	3234±0
	Number of Edges	12717	12449.4±13.31	12567.43±9.6	12593.18±8.39	12606.58±8.34	12616.48±7.7	12615.12±7.79	12608.38±8.42
	Average Degree	7.86	7.7±0.01	7.77±0.01	7.79±0.01	7.8±0.01	7.8±0	7.8±0	7.8±0.01
	Density	0.00	0±0	0±0	0±0	0±0	0±0	0±0	0±0
	Avg Clustering Coefficient	0.43	0.19±0	0.17±0	0.15±0	0.14±0	0.12±0	0.12±0	0.12±0
	Avg Connected Component	17.00	2.24±0.5	1.2±0.44	1.13±0.37	1.14±0.36	1.27±0.55	1.64±0.79	4.14±1.56
	Giant Component Size	3109	3155.56±2.16	3228.04±20.42	3233.87±0.37	3233.86±0.36	3233.73±0.55	3233.36±0.79	3230.86±1.56
	Assortativity	0.14	0.02±0.01	0.02±0.01	0.02±0.01	0.02±0.01	0.02±0.01	0.03±0.01	0.04±0.01
	PageRank	1616.50	1616.5±0	1616.5±0	1616.5±0	1616.5±0	1616.5±0	1616.5±0	1616.5±0
	Transitivity	0.39	0.19±0	0.16±0	0.15±0	0.13±0	0.12±0	0.12±0	0.11±0
	Number of Triangles	6490	5402.03±65.45	4712.47±56.84	4356.54±57.62	3980.89±58.92	3569.63±54.37	3486.89±53.63	3346.53±48.62
Anaheim	Number of Nodes	914	914±0	914±0	914±0	914±0	914±0	914±0	914±0
	Number of Edges	3881	3557.23±17.18	3744.01±10.86	3804.45±8.77	3845.48±5.53	3855.53±4.2	3858.74±4.21	3855.93±4.51
	Average Degree	8.49	7.78±0.04	8.19±0.02	8.32±0.02	8.41±0.01	8.44±0.01	8.44±0.01	8.44±0.01
	Density	0.01	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0	0.01±0
	Avg Clustering Coefficient	0.55	0.19±0.01	0.16±0.01	0.14±0	0.12±0	0.11±0	0.09±0	0.05±0
	Avg Connected Component	1.00	1.17±0.42	1.1±0.33	1.09±0.29	1.05±0.22	1.02±0.15	1.01±0.1	1.01±0.08
	Giant Component Size	914	913.82±0.44	913.89±0.35	913.9±0.36	913.95±0.25	913.98±0.15	913.99±0.1	913.99±0.08
	Assortativity	0.71	0.51±0.01	0.45±0.01	0.41±0.01	0.39±0.01	0.39±0.01	0.39±0.01	0.15±0.01
	PageRank	456	456.5±0	456.5±0	456.5±0	456.5±0	456.5±0	456.5±0	456.5±0
	Transitivity	0.60	0.2±0	0.16±0	0.14±0	0.12±0	0.11±0	0.1±0	0.06±0
	Number of Triangles	7162	1968.36±46.38	1779.38±43.05	1575.97±39.58	1438.32±38.23	1270.53±35.34	1163.42±32.96	768.93±28.73

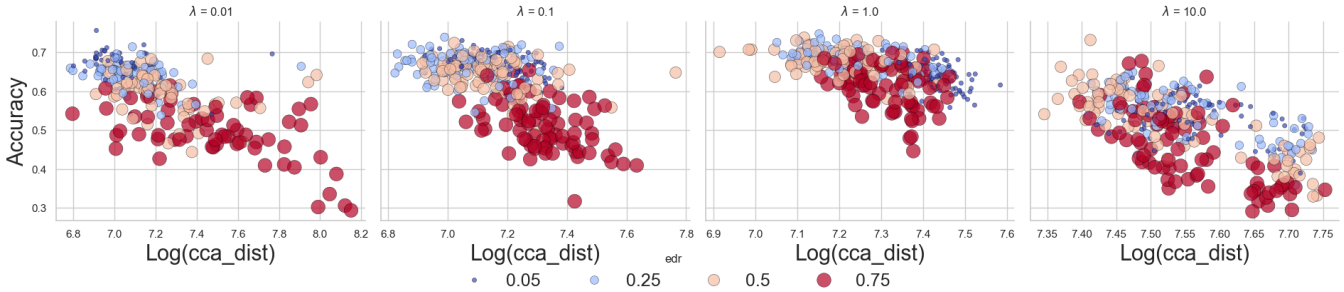
Table 11: Graph statistics of bootstrapped samples generated by Algorithm 2 with varying neighborhood size (k).

		Original	Edge Drop	Node Drop	Ours	Network Bootstrap	VAE
Cora	Assortativity	-0.07	-0.07 ± 0	-0.07 ± 0.01	**-0.07±0**	-0.03 ± 0.03	-0.43 ± 0
	Avg Clustering Coefficient	0.24	0.22 ± 0	0.17 ± 0.01	**0.05±0**	0.02 ± 0.01	0.5 ± 0
	Avg Degree	3.90	3.74 ± 0.01	2.63 ± 0.07	**3.82±0.01**	2.11 ± 0.18	10.04 ± 0.05
	Density	0.00	0 ± 0	0 ± 0	**0±0**	0 ± 0	0 ± 0
	Giant Component Size	2485	2457.77 ± 9.09	1819.98 ± 30.29	**2620.38±10.78**	1090.22 ± 41.05	1931.32 ± 9.06
	Num Connected Components	78	100.12 ± 4.7	587.49 ± 16.74	**67.91±6.7**	1595.93 ± 38.78	777.68 ± 9.06
	Num Edges	5278	5066.1 ± 11.78	3382.72 ± 87.72	**5171.78±7.34**	2857.55 ± 244.83	13598.31 ± 67.26
	Num Nodes	2708	2708 ± 0	2569.6 ± 9.32	2708±0	2708 ± 0	2708 ± 0
	Num Triangles	1630.00	1441.03 ± 20.11	835.13 ± 63.84	471.48±27.11	**801.19 ± 338.24**	78345.91 ± 643.03
	Pagerank	1353.50	1353.5 ± 0	1294.43 ± 4.23	1353.5±0	1353.5 ± 0	1353.5 ± 0
	Transitivity	0.09	0.09 ± 0	0.09 ± 0.01	0.03±0	0.07 ± 0.01	**0.12 ± 0**
TwitchPTBR	Assortativity	-0.23	-0.23±0	-0.23±0.01	**-0.29±0**	0±0	-0.45±0
	Avg Clustering Coefficient	0.32	0.25±0	0.26±0.01	0.17±0	0±0	**0.41±0**
	Avg Degree	32.74	26.19±0	21.87±0.84	**32.51±0.02**	0±0	19.89±0.04
	Density	0.02	0.01±0	0.01±0	**0.02±0**	0±0	0.01±0
	Giant Component Size	1912	1883.96±4.76	1506±5.9	**1911.72±0.77**	1±0	893.77±7.03
	Num Connected Components	1.00	28.12±4.49	325.55±8.23	**1.14±0.39**	1912±0	1016.37±6.87
	Num Edges	31299	25039±0	20029.75±776.17	**31082.05±22.83**	0±0	19010.87±40.09
	Num Nodes	1912	1912±0	1831.36±7.18	1912±0	1912±0	1912±0
	Num Triangles	173510	88756.59±980.58	89048.18±9052.57	**105534.51±1904.54**	0±0	332319.47±623.05
	Pagerank	955.50	955.5±0	921.8±3.18	955.5±0	955.5±0	955.5±0
	Transitivity	0.13	0.1±0	0.13±0	**0.08±0**	0±0	0.31±0
ChicagoSketch	Assortativity	0.65	456.5 ± 0	441.21 ± 2.15	**0.29±0.01**	456.5 ± 0	456.5 ± 0
	Avg Clustering Coefficient	0.57	0 ± 0	0 ± 0	**0.12±0**	0 ± 0	0 ± 0
	Avg Degree	13.88	0.48 ± 0	0.6 ± 0.01	**13.8±0.01**	0 ± 0	0.59 ± 0.01
	Density	0.01	0 ± 0	0 ± 0	**0.01±0**	0 ± 0	0 ± 0
	Giant Component Size	2176	2175.97 ± 0.18	1739.89 ± 0.44	**2176±0**	1 ± 0	703.59 ± 6.77
	Num Connected Components	1.00	1.03 ± 0.16	349.83 ± 7.54	**1±0**	2176 ± 0	1473.41 ± 6.77
	Num Edges	15104	12083 ± 0	9658.73 ± 46.97	**15014.23±10.28**	0 ± 0	9156.6 ± 85.77
	Num Nodes	2176	2176 ± 0	2088.77 ± 7.54	2176±0	2176 ± 0	2176 ± 0
	Num Triangles	38240	19576.4 ± 101.57	19554 ± 325.13	**8006.58±95.13**	0 ± 0	83059.43 ± 1685.2
	Pagerank	1087.50	1087.5 ± 0	1051.19 ± 3.29	1087.5±0	1087.5 ± 0	1087.5 ± 0
	Transitivity	0.56	0.45 ± 0	0.56 ± 0	0.12±0	0 ± 0	**0.51 ± 0**
Eduation	Assortativity	0.14	0.11 ± 0.01	0.33 ± 0.02	**0.03±0.01**	0 ± 0	-0.24 ± 0
	Avg Clustering Coefficient	0.43	0.34 ± 0	0.36 ± 0	0.12±0	0 ± 0	**0.37 ± 0**
	Avg Degree	7.86	6.29 ± 0	5.58 ± 0.02	**7.8±0**	0 ± 0	16.63 ± 0.07
	Density	0.00	0 ± 0	0 ± 0	**0±0**	0 ± 0	0.01 ± 0
	Giant Component Size	3109	3103.63 ± 2.38	2478.84 ± 8.39	**3233.36±0.79**	2.34 ± 0.94	1826.67 ± 7.86
	Num Connected Components	17	25.11 ± 2.6	539.85 ± 8.69	**1.64±0.79**	3231.81 ± 1.71	1408.33 ± 7.86
	Num Edges	12717	10173 ± 0	8654.72 ± 30.07	**12615.12±7.79**	2.94 ± 2.91	26887.73 ± 118.84
	Num Nodes	3234	3234 ± 0	3104.69 ± 8.44	3234±0	3234 ± 0	3234 ± 0
	Num Triangles	6490	3321.39 ± 32.58	3321.02 ± 37.46	**3486.89±53.63**	0.96 ± 2.18	257887.78 ± 2061.14
	Pagerank	1616.50	1616.5 ± 0	1562.58 ± 3.73	1616.5±0	1616.5 ± 0	1616.5 ± 0
	Transitivity	0.39	0.32 ± 0	0.39 ± 0	0.12±0	**0.36 ± 0.48**	0.35 ± 0
Anaheim	Assortativity	0.71	0.6 ± 0.01	0.69 ± 0.02	**0.39±0.01**		-0.15 ± 0.02
	Avg Clustering Coefficient	0.55	0.44 ± 0.01	0.46 ± 0.01	0.09±0		**0.22 ± 0**
	Avg Degree	8.49	6.79 ± 0	5.66 ± 0.08	**8.44±0.01**		5.06 ± 0.07
	Density	0.01	0 ± 0	0 ± 0	**0.01±0**		0 ± 0
	Giant Component Size	914	0.01 ± 0	0.01 ± 0	**913.99±0.1**		0.01 ± 0
	Num Connected Components	1.00	0 ± 0	0 ± 0	**1.01±0.1**		0 ± 0
	Num Edges	3881	911.49 ± 2.43	727.17 ± 3.63	**3858.74±4.21**		121.87 ± 13.67
	Num Nodes	914	2.8 ± 1.45	149.32 ± 5.24	**914±0**		645.81 ± 5.49
	Num Triangles	7162	3104 ± 0	2483.09 ± 37.24	**1163.42±32.96**		2311.09 ± 32.23
	Pagerank	456.50	914 ± 0	877.31 ± 4.94	**456.5±0**		914 ± 0
	transitivity	0.60	3661.41 ± 51.07	3667.47 ± 152.92	**0.1±0**		14770.14 ± 386.77

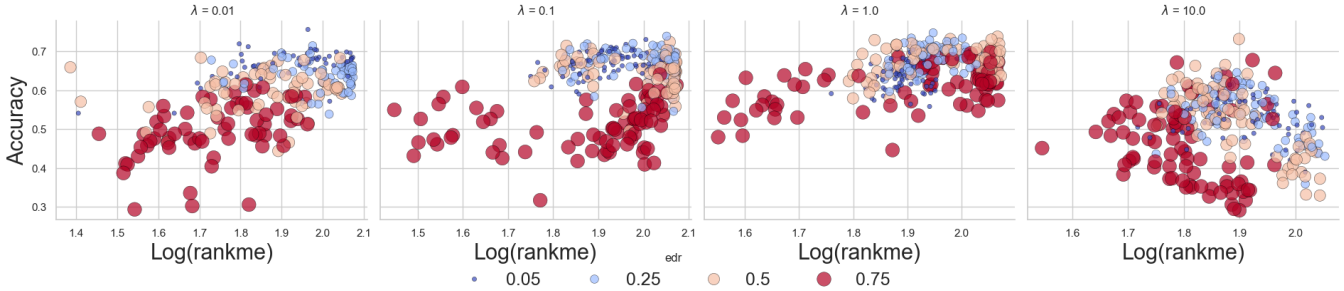
Table 12: Graph statistics of bootstrapped samples generated by Algorithm 2 with $k = 20$, simple generation with Edge or Node Drop (with $p = 0.2$), and network bootstrap (NB) [Levin and Levina 2021] and the extended VAE approach in Appendix-C.4. The best values among our method, NB and VAE are bolded, and the second best values are underlined. The simple split methods largely distort the original connectivity structure, reflected in the Giant Component Size or the Number of Connected Components. Our proposed local bootstrap consistently mimics the original graph in terms of the reported graph statistics. The greyed-out cells indicate values that are unavailable due to instability encountered during training.

Remark F.1. When the graph structure is sufficiently informative for the underlying latent variable distribution (see the differences in Table 7 and Table 9 for the graph-kNN case), the bootstrapped graphs show noticeable robustness to the choice of k (the number of nearest neighbors). For example, with 500 nodes and k=20, the bootstrapped graph retains about 98% of the original edges and recovers approximately 70% of its triangles (see Table 8 for detailed statistics). Increasing k produces progressively denser graphs with more edges and higher average degrees, but it also tends to lower the clustering coefficient and reduce the number of triangles. Conversely, using a very small k leads to sparser graphs that may preserve more local structure—reflected by higher clustering coefficients and relatively more triangles per edge—but can underrepresent global connectivity, often resulting in many small components. On real datasets, setting $k = 20$ typically recovers an edge count close to that of the original graph, though it still underestimates the original triangle count (see Table 11). Nonetheless, our proposed method produces graphs whose triangle counts more closely match the original than those generated by other methods (see Table 12).

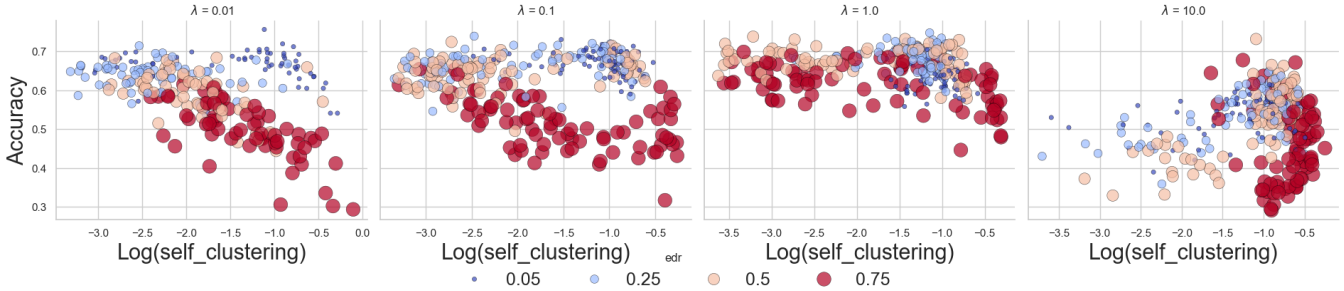
F.3 Validation of the Entire Framework (Algorithm 3)



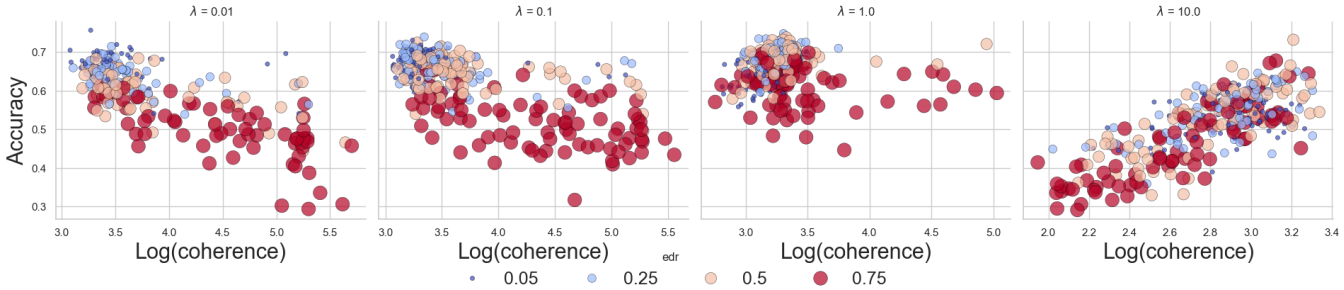
(a) Our CCA-based metrics discussed in Section 4.



(b) RankMe metric by Garrido et al. [2023]



(c) SelfCluster by Tsitsulin et al. [2023]



(d) Coherence by Tsitsulin et al. [2023]

Figure 9: Visualizations of metrics value and classification accuracy on Cora. The CCA-SSG [Zhang et al. 2021] model is trained on set of hyperparameter combinations including λ, edge drop rate (EDR), and feature masking rate (FMR). Each point denotes the each combination of hyperparameters. The color denotes the edge dropping rate with blue dots referring small value ($edr = 0.05$) whereas the red dot referring to the large value ($edr = 0.75$). In this dataset, a clear negative correlation is noted for our metrics across all combinations of hyperparameters.

Dataset	Default	Ours	α-ReQ	pseudo-κ	RankME	NESum	SelfCluster	Stable Rank	Coherence
Cora	0.36	0.4	0.4	**0.57**	0.55	0.4	0.4	**0.57**	0.54
PubMed	0.62	0.68	0.67	**0.74**	0.67	0.67	0.69	**0.74**	0.74
Citeseer	0.32	**0.42**	**0.42**	**0.42**	**0.42**	**0.42**	**0.42**	**0.42**	**0.42**
CS	0.47	0.71	**0.82**	0.75	0.75	**0.82**	**0.82**	0.75	**0.82**
Chicago	**0.39**	0.34	0.35	0.35	0.35	0.35	0.35	0.29	0.4
Anaheim	0.13	**0.23**	0.12	0.18	0.18	0.12	**0.23**	0.18	0.12
Education	0.23	**0.26**	0.18	0.17	0.18	0.16	0.18	**0.26**	0.21
Avg_clf	0.44	0.55	0.58	0.62	0.60	0.58	0.58	0.62	**0.63**
Avg_reg	0.25	**0.28**	0.22	0.23	0.24	0.21	0.25	0.24	0.24

Table 13: Downstream task (classification or regression) performance of the BGRL with hyperparameters chosen by each criteria. We compare to the BGRL [Thakoor et al. 2021] with the default parameters in the left-most column, fmr $= 0.5$, edr $= 0.25$, $\lambda = 10^{-2}$. The best value is bolded and the second best is underlined.

Dataset	Default	Ours	α-ReQ	pseudo-κ	RankME	NESum	SelfCluster	Stable Rank	Coherence
Cora	0.35	0.65	0.66	0.67	0.63	0.63	**0.69**	0.59	0.47
PubMed	0.49	0.81	**0.82**	0.56	0.56	0.56	**0.82**	**0.82**	0.76
Citeseer	0.38	0.51	**0.53**	0.51	**0.53**	**0.53**	**0.53**	0.51	0.22
CS	0.65	0.79	**0.86**	**0.86**	**0.86**	**0.86**	**0.86**	**0.86**	0.76
Photo	0.32	**0.73**	0.58	0.53	0.53	**0.73**	**0.73**	**0.73**	0.69
Computers	0.42	0.57	**0.66**	0.57	**0.66**	**0.66**	**0.66**	0.57	0.65
Anaheim	0.37	**0.38**	**0.38**	**0.38**	**0.38**	**0.38**	**0.38**	**0.38**	**0.38**
Twitch	0.47	**0.52**	0.15	0.15	0.15	0.15	0.46	0.15	0.48
Education	0.29	0.26	0.33	0.33	0.33	0.33	0.33	0.33	0.26
Avg_clf	0.44	0.68	0.69	0.62	0.63	0.66	**0.72**	0.68	0.59
Avg_reg	0.38	**0.39**	0.29	0.29	0.29	0.29	**0.39**	0.29	0.37

Table 14: Downstream task (classification or regression) performance of the CCA-SSG with hyperparameters chosen by each criteria. We compare to the CCA-SSG [Zhang et al. 2021] with the default parameters in the left-most column, fmr $= 0.5$, edr $= 0.25$, $\lambda = 10^{-4}$. The best value is bolded and the second best is underlined.

Dataset	**Default**	**Ours**	**α-ReQ**	**pseudo-κ**	**RankME**	**NESum**	**SelfCluster**	**Stable Rank**	**Coherence**
Cora	0.64	**0.68**	0.54	0.54	0.54	0.54	0.5	0.54	0.42
PubMed		0.78	0.75	0.75	0.75	0.75	0.75	0.75	**0.79**
Citeseer	0.53	**0.55**	0.51	0.51	0.51	0.51	0.48	0.51	0.44
CS		**0.72**	**0.72**	**0.72**	**0.72**	**0.72**	**0.72**	**0.72**	0.61
Photo	0.71	**0.83**	0.79	0.79	0.79	0.79	0.57	0.81	0.41
Computers		**0.45**	**0.45**	**0.45**	**0.45**	0.39	0.39	0.39	0.39
Avg_clf	0.63	**0.67**	0.63	0.63	0.63	0.62	0.57	0.62	0.50

Table 15: Downstream task (classification or regression) performance of the GRACE with hyperparameters chosen by each criteria. We compare to the GRACE [Zhu et al. 2020] with the default parameters in the left-most column, fmr = 0.5, edr = 0.25, $\tau = 1$. The best value is bolded and the second best is underlined. The greyed-out cells indicate values that are unavailable due to instability encountered during training.

	Ours	Literature			Tsitsulin et al. [2023]			
	CCA dist (↓)	α-ReQ (↓)	pseudo-κ (↑)	RankMe (↑)	NEsum (↑)	SelfCluster (↓)	Stable Rank (↑)	Coherence (↓)
Cora	**-0.6596**	-0.2414	0.2036	0.1998	0.2738	0.017	0.1146	0.2914
PubMed	**-0.7702**	-0.2379	0.1422	0.2048	0.4854	-0.0103	0.0532	0.127
Citeseer	-0.2014	**-0.3183**	0.2842	0.2781	0.2518	-0.3156	0.0609	-0.295
CS	0.1875	-0.5459	0.5356	0.5577	**0.5608**	-0.4376	0.1899	-0.3776
Photo	-0.3797	-0.2886	0.274	0.3155	0.2698	**-0.5009**	0.4722	-0.3782
Computers	-0.2433	-0.1943	0.0777	0.2498	0.2875	-0.0714	**0.3322**	-0.283
Chicago	**-0.1225**	0.6618	-0.6887	-0.6667	-0.3284	0.3701	-0.451	0.1446
Anaheim	**-0.1528**	0.352	-0.3273	-0.3091	-0.1757	0.2337	-0.0864	-0.0543
Twitch	**-0.5858**	0.6246	-0.4868	-0.5414	-0.4852	0.3034	-0.2921	0.2839
Education	**-0.3464**	0.4286	-0.4315	-0.3548	-0.0065	0.0171	0.0917	-0.1247
Avg_clf	-0.3445	-0.3044	0.2529	0.3010	**0.3549**	-0.2198	0.2038	-0.1526
Avg_reg	**-0.3019**	0.5168	-0.4836	-0.4680	-0.2490	0.2311	-0.1845	0.0624

Table 16: Spearman correlation between each metric and the node classification accuracy (R^2 if the predicted value is continuous) across different models and sets of hyperparameters. The higher the absolute value is, the better with the sign aligning with the arrow. We note that many other metrics lose their intended direction for some dataset. For example, the high *StableRank* should indicate the better performance but the real relationship turns out to be reversed for Chicago, Aneheim and Twitch dataset.

Received 2 May 2025