CARL-G: Clustering-Accelerated Representation Learning on Graphs

William Shiao
wshia002@ucr.edu
University of California, Riverside
Riverside, CA, USA

Uday Singh Saini
usain001@ucr.edu
University of California, Riverside
Riverside, CA, USA

Yozen Liu
yliu2@snap.com
Snap Inc.
Santa Monica, CA, USA

Tong Zhao
tzhao@snap.com
Snap Inc.
Seattle, WA, USA

Neil Shah
nshah@snap.com
Snap Inc.
Seattle, WA, USA

Evangelos E. Papalexakis
epapalex@cs.ucr.edu
University of California, Riverside
Riverside, CA, USA

ABSTRACT

Self-supervised learning on graphs has made large strides in achieving great performance in various downstream tasks. However, many state-of-the-art methods suffer from a number of impediments, which prevent them from realizing their full potential. For instance, contrastive methods typically require negative sampling, which is often computationally costly. While non-contrastive methods avoid this expensive step, most existing methods either rely on overly complex architectures or dataset-specific augmentations. In this paper, we ask: *Can we borrow from classical unsupervised machine learning literature in order to overcome those obstacles?* Guided by our key insight that the goal of distance-based clustering closely resembles that of contrastive learning: both attempt to pull representations of similar items together and dissimilar items apart. As a result, we propose CARL-G — a novel *clustering-based* framework for graph representation learning that uses a loss inspired by Cluster Validation Indices (CVIs), i.e., internal measures of cluster quality (no ground truth required). CARL-G is adaptable to different clustering methods and CVIs, and we show that with the right choice of clustering method and CVI, CARL-G outperforms node classification baselines on 4/5 datasets with up to a **79**× training speedup compared to the best-performing baseline. CARL-G also performs at par or better than baselines in node clustering and similarity search tasks, training up to **1,500**× faster than the best-performing baseline. Finally, we also provide theoretical foundations for the use of CVI-inspired losses in graph representation learning.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Learning latent representations**; • **Networks** → *Network algorithms.*

KEYWORDS

self-supervised graph representation learning, clustering

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08.
https://doi.org/10.1145/3580305.3599268

ACM Reference Format:
William Shiao, Uday Singh Saini, Yozen Liu, Tong Zhao, Neil Shah, and Evangelos E. Papalexakis. 2023. CARL-G: Clustering-Accelerated Representation Learning on Graphs. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3580305.3599268

1 INTRODUCTION

Graphs can be used to represent many different types of relational data, including chemistry graphs [9, 21], social networks [39, 53], and traffic networks [14, 59]. Graph Neural Networks (GNNs) have been effective in modeling graphs across a variety of tasks, such as for recommendation systems [18, 25, 48, 58, 68, 76], graph generation [17, 56, 70], and node classification [23, 28, 60, 73, 77, 79]. These GNNs are traditionally [33] trained with a supervised loss, which requires labeled data that is often expensive to obtain in real-world scenarios. Graph self-supervised learning (SSL), a recent area of research, attempts to solve this by learning multi-task representations without labeled data [4, 27, 30, 60, 79].

Most of these existing graph SSL methods can be grouped into either contrastive or non-contrastive SSL. Contrastive learning pulls the representations of similar ("positive") samples together and pushes the representations of dissimilar ("negative") samples apart. In the case of graphs, this often means either pulling the representations of a node and its neighbors together [22] or pulling the representations of the same node across two different augmentations together [71]. Graph contrastive learning methods typically use non-neighbors as negative samples [68, 79], which can be costly. Non-contrastive learning [4, 34, 54, 60] avoids this step by only pulling positive samples together while employing strategies to avoid collapse.

However, all of these methods have some key limitations. Contrastive methods rely on a negative sampling step, which has an expensive quadratic runtime [60] and requires careful tuning [66]. Non-contrastive methods often have complex architectures (ex. extra encoder with exponentially updated weights [31, 34, 60]) and/or rely heavily on augmentations [4, 60, 72, 75]. Lee et al. [34] shows that augmentations can change the semantics of underlying graphs, especially in the case of molecular graphs (where perturbing a single edge can create an invalid molecule).

Upon further inspection, we observe that the behavior of contrastive and non-contrastive methods is somewhat similar to that

	Proposed	Baseline Methods				
	CARL-G*	AFGRL [34]	BGRL [60]	G-BT [4]	GRACE [79]	MVGRL [24]
Avoids Negative Sampling	✓	✓	✓	✓	✗	✗
Augmentation-Free	✓	✓	✗	✗	✗	✗
Single Encoder	✓	✗	✗	✓	✗	✗
Single Forward Pass per Epoch	✓	✗	✗	✗	✗	✗

Table 1: Comparison of different self-supervised graph learning methods. *: We use CARL-G_{SIM} as the representative method since it is the best-performing across all of the criteria.

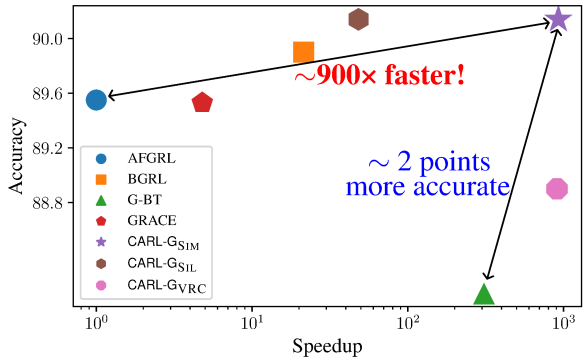


Figure 1: Comparison of our proposed methods with other baselines with respect to node classification accuracy and speedup on the `Amazon-Photos` dataset. See Figure 3 for results on the other datasets.

of distance-based clustering [64]—both attempt to pull together similar nodes/samples and push apart dissimilar ones. The primary advantage of using clustering over negative sampling is that we can work directly in the smaller embedding space, preventing expensive negative sampling over the graph. Furthermore, there have been decades of research exploring the theoretical foundations of clustering methods and many different metrics have been proposed to evaluate the quality of clusters [7, 15, 32, 47]. These metrics have been dubbed Cluster Validation Indices (CVIs) [2]. In this work, we ask the following question: *Can we leverage well-established clustering methods and CVIs to create a flexible, fast, and effective GNN framework to learn node representations without any labels?*

It is worth emphasizing that our goal is *not node clustering* directly—it is self-supervised graph representation learning. The goal is to develop a general framework that is capable of learning node embeddings for various tasks, including node classification, node clustering, and node similarity search. There exists some similar work. DeepCluster [8] trains a Convolutional Neural Network (CNN) with a supervised loss on pseudo-labels generated by a clustering method to learn image embeddings.

AFGRL [34] uses clustering to select positive samples in lieu of augmentations for graph representation learning and applies the general BGRL [60] framework to push those representations together. SCD [57] searches over different hyperparameters to obtain a clustering where the silhouette score is maximized. However, to the best of our knowledge, there is no existing work in self-supervised representation learning that directly optimizes for CVIs, which, as we elaborate below, presents us with tremendous potential in advancing and accelerating the state of the art.

We fill this gap by proposing the novel idea of using Cluster Validation Indices (CVIs) directly as our loss function for a neural network. In conjunction with advances in clustering methods [35, 43, 51, 52], CVIs have been improved over the years as measures of cluster quality after performing clustering and have been shown for almost 5 decades to be effective for this purpose [2, 7, 47, 50].

Our proposed method, CARL-G, has several advantages over existing graph SSL methods by virtue of CVI-inspired losses. First, CARL-G generally outperforms other graph SSL methods in node clustering, clustering, and node similarity tasks (see Tables 2, 5 and 6). Second, CARL-G does not require the use of graph augmentations — which are required by many existing graph contrastive and non-contrastive methods [4, 31, 34, 60, 72, 79] and can inadvertently alter graph semantics [34]. Third, CARL-G has a relatively simple architecture compared to the dual encoder architecture of leading non-contrastive methods [31, 34, 60], drastically reducing the memory cost of our framework. Fourth, we provide theoretical analysis that shows the equivalence of some CVI-based losses and a well-established (albeit expensive) contrastive loss. Finally, CARL-G is sub-quadratic with respect to the size of the graph and much faster than the baselines, with up to a 79× speedup on `Coauthor-CS` over BGRL [60] (the best-performing baseline).

Our contributions can be summarized as follows:

- We propose CARL-G, the first (to the best of our knowledge) framework to use a Cluster Validation Index (CVI) as a neural network loss function.
- We propose 3 variants of CARL-G based on different CVIs, each with its own advantages and drawbacks.
- We extensively evaluate CARL-G_{SIM} — the best all-around performer — across 5 datasets and 3 downstream tasks, where it generally outperforms baselines.
- We provide theoretical insight on CARL-G_{SIM}'s success.
- We benchmark CARL-G_{SIM} against 4 state-of-the-art models and show that it is up to 79 × faster with half the memory consumption (with the same encoder size) compared to the best-performing node classification baseline.

2 PRELIMINARIES

Notation. We denote a graph as $G = (\mathcal{V}, \mathcal{E})$. $\mathcal{V}$ is the set of n nodes (i.e., $n = |\mathcal{V}|$) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. We denote the node-wise feature matrix as $\boldsymbol{X} \in \mathbb{R}^{n \times f}$, where f is the dimension of raw node features, and its i-th row $\boldsymbol{x}_i$ is the feature vector for the i-th node. We denote the binary adjacency matrix as $\boldsymbol{A} \in \{0, 1\}^{n \times n}$ and the learned node representations as $\boldsymbol{H} \in \mathbb{R}^{n \times d}$, where d is the size of latent dimension, and $\boldsymbol{h}_i$ is the representation

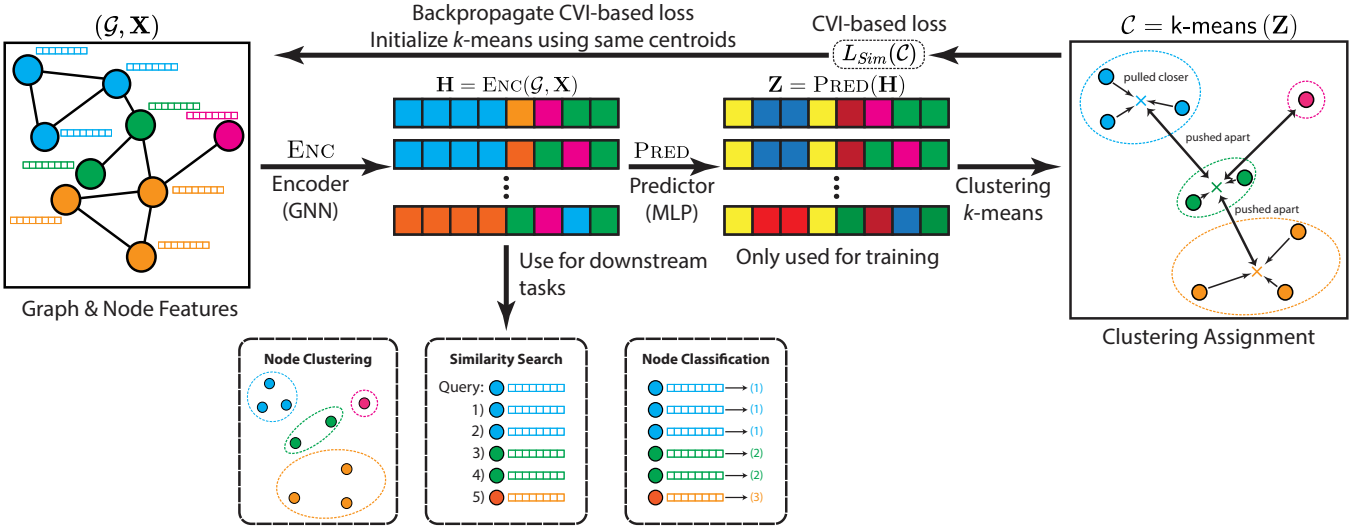


Figure 2: CARL-G architecture diagram. We describe the method in detail in Section 3.

for the i-th node. Let $\mathcal{N}(u)$ be a function that returns the set of neighbors for a given node u (i.e., $\mathcal{N}(u) = \{v | (u, v) \in \mathcal{E}\}$).

Let C be the set of clusters, $C_i \subseteq \mathcal{V}$ be the set of nodes in the i-th cluster, and $c = |C|$ be the number of clusters. For ease of notation, let $\mathcal{U} \in [1, c]^n$ be the set of cluster assignments, where $\mathcal{U}_u$ is the cluster assignment for node u. Let $\boldsymbol{\mu} = \frac{1}{c} \sum_{u \in \mathcal{V}} \boldsymbol{h}_u$ be the global centroid and $\boldsymbol{\mu}_i = \frac{1}{|C_i|} \sum_{u \in C_i} \boldsymbol{h}_u$ be the centroid for the i-th cluster.

2.1 Graph Neural Networks

A Graph Neural Network (GNN) [22, 33, 74] typically performs message-passing along the edges of the graph. Each iteration of the GNN can be described as follows [49]:

$$\boldsymbol{h}_u^{(k+1)} = \textsc{Update}^{(k)} \left(\boldsymbol{h}_u^{(k)}, \textsc{Aggregate}^{(k)}(\{\boldsymbol{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\}) \right), \tag{1}$$

where Update and Aggregate are differentiable functions, and $\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$. In this work, we opt for simplicity and use Graph Convolutional Networks (GCNs) [33] as the default GNN. These are GNNs where Update consists of a single MLP layer, and Aggregate is the mean of a node's representation with its neighbors. Formally, each iteration of the GCN can be written as:

$$\boldsymbol{h}_u^{(k+1)} = \sigma \left(\boldsymbol{W}^{(k+1)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{\boldsymbol{h}_v}{\sqrt{|\mathcal{N}(u)| \cdot |\mathcal{N}(v)|}} \right). \tag{2}$$

2.2 Cluster Validation Indices

Clustering is a class of unsupervised methods that aims to partition the input space into multiple groups, known as clusters. The goal of clustering is generally to maximize the similarity of points within each cluster while minimizing the similarity of points between clusters [64]. In this work, we focus on centroid-based clustering algorithms like k-means [37] and k-medoids [40].

Cluster Validation Indices (CVIs) [2] estimate the quality of a partition (i.e., clustering) by measuring the compactness and separation of the clusters without knowledge of the ground truth clustering. Note that these are different from metrics like Normalized Mutual Information (NMI) [12] or the Rand Index [46], which require ground truth information of cluster labels. Many different CVIs have been proposed over the years and extensively evaluated [2, 50].

Arbelaitz et al. [2] extensively evaluated 30 different CVIs over a wide variety of datasets and found that the Silhouette [47], Davies-Bouldin* [32], and Calinski-Harabasz (also known as the VRC: Variance Ratio Criterion) [7] indices perform best across 720 different synthetic datasets. The VRC has also been shown to be effective in determining the number of clusters for clustering methods [7, 16, 38, 50]. As such, we focus on the silhouette index (the best-performing CVI) and VRC (an effective CVI — especially for choosing the number of clusters) in this work.

2.2.1 Silhouette. The silhouette index computes the ratio of intra-cluster distance with respect to the inter-cluster distance of itself with its nearest neighboring cluster. It returns a value in [-1, 1], where a value closer to 1 signifies more desired and better distinguishable clustering. The silhouette index [47] is defined as $\textsc{Sil}(C) = \frac{1}{n} \sum_{u \in \mathcal{V}} s(u)$, where:

$$s(u) = \frac{b(u) - a(u)}{\max\{a(u), b(u)\}}, \tag{3}$$

and

$$a(u) = \frac{1}{|C_{\mathcal{U}_u}| - 1} \sum_{v \in (C_{\mathcal{U}_u} - \{u\})} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v), \tag{4}$$

$$b(u) = \min_{i \neq \mathcal{U}_u} \frac{1}{|C_i|} \sum_{v \in C_i} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v). \tag{5}$$

The runtime of computing the silhouette index for a given node is $O(n)$, which can be expensive if calculated over all nodes. We discuss this issue and a modified solution later in Section 3.1.

2.2.2 Variance Ratio Criterion. The VRC [7] computes a ratio between its intra-cluster variance and its inter-cluster variance. Its intra-cluster variance is based on the distances of each point to

its centroid, and its inter-cluster variance is based on the distance from each cluster centroid to the global centroid. Formally,

$$\mathrm{VRC}(C) = \frac{n-c}{c-1} \frac{\sum_{C_k \in C} |C_k| \textsc{Dist}(\boldsymbol{\mu}_k, \boldsymbol{\mu})}{\sum_{C_k \in C} \sum_{u \in C_k} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{\mu})} . \tag{6}$$

For the purposes of this paper, we use Euclidean distance, i.e., $\textsc{Dist}(\boldsymbol{a}, \boldsymbol{b}) = \|\boldsymbol{a} - \boldsymbol{b}\|_2$.

3 PROPOSED METHOD

3.0.1 Problem Formulation. Given a graph G and its node-wise feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times f}$, learn node embeddings $\boldsymbol{h}_u \in \mathbb{R}^d$ for each node $u \in \mathcal{V}$ without any additional information (e.g. node class labels). The learned embeddings should be suitable for various downstream tasks, such as node classification and node clustering.

3.0.2 CARL-G. We propose CARL-G, which consists of three main steps (Figure 2). First, a GNN encoder $\textsc{Enc}(\cdot)$ takes the graph as input and produce node embeddings $\boldsymbol{H} = \textsc{Enc}(\boldsymbol{X}, \boldsymbol{A})$. Next, a multilayer perceptron (MLP) predictor network $\textsc{Pred}(\cdot)$ takes the embeddings by GNN and produces a second set of node embeddings $\boldsymbol{Z} = \textsc{Pred}(\boldsymbol{H})$. We then perform a clustering algorithm (e.g., k-means) on $\boldsymbol{Z}$ to produce a set of clusterings C. It is worth noting that the clustering algorithm does not have to be differentiable. Finally, we compute a cluster validation index (CVI) on the cluster assignments and backpropagate to update the encoder's and predictor's parameters. After training, only the GNN encoder $\textsc{Enc}(\cdot)$ and its produced embeddings $\boldsymbol{H}$ are used to perform downstream tasks, and the predictor network $\textsc{Pred}(\cdot)$ is discarded (similar to the prediction heads in non-contrastive learning work [31, 34, 60]).

3.1 Training CARL-G

As aforementioned in Section 2.2, we evaluate the silhouette index [47] and the VRC [7] as learning objectives. In order to use them effectively, we must make slight modifications to the loss functions. First, we must negate the functions since a higher score is better for both CVIs, and we typically want to minimize a loss function. Second, while $\textsc{Sil}(C) = 1$ and $\mathrm{VRC}(C) \to \infty$ are theoretically ideal, we find this is generally not true in practice. This is because the clustering method may miscluster some nodes and fully maximizing the CVIs will push the misclustered representations too close together, negatively impacting a classifier's ability to distinguish them. To bound the maximum values of our loss, we add τ_{SIL} and τ_{VRC} — the target silhouette and VRC indices, respectively. The silhouette-based and VRC-based losses are then defined as follows:

$$L_{\text{SIL}} = |\tau_{\text{SIL}} - \textsc{Sil}(C)| , \qquad L_{\text{VRC}} = |\tau_{\text{VRC}} - \mathrm{VRC}(C)| , \tag{7}$$

where $\tau_{\text{SIL}} \in [-1, 1]$ is the target silhouette index and $\tau_{\text{VRC}} \in [0, \infty)$ is the target VRC.

Upon careful inspection of Equation (3), we can observe that the computational complexity for the silhouette is $O(n^2)$, while the complexity of VRC is only $O(nc)$, where $c \ll n$. This is a critical weakness in using the silhouette, especially when the goal is to avoid a quadratic runtime (the typical drawback of contrastive methods). Backpropagating on this loss function would also result in quadratic memory usage because we have to store the gradients for each operation. To resolve this issue, we leverage the simplified silhouette [26], which instead uses the centroid distance. The simplified silhouette has been shown to have competitive performance with the original silhouette [62] while being much faster — running in $O(nc)$ time. As such, we also try the simplified silhouette, which can be written as:

$$s'(u) = \frac{b'(u) - a'(u)}{\max\{a'(u), b'(u)\}} , \tag{8}$$

where $i = \mathcal{U}_u$ is the cluster assignment for u and

$$a'(u) = \textsc{Dist}(u, \boldsymbol{\mu}_i) , \qquad b'(u) = \min_{C_k \neq C_i} \textsc{Dist}(u, \boldsymbol{\mu}_i) , \tag{9}$$

We use the same loss function as L_{SIL} (Equation (7)), simply substituting $s'(u)$ for $s(u)$ (see Section 2.2.1), and name it L_{SIM}.

3.2 Clustering Method

k-means. We primarily focus on k-means clustering for this framework due to its fast linear runtime (although we do briefly explore using k-medoids in Section 4.3.2 below). The goal of k-means is to minimize the sum of squared errors—also known as the inertia or within-cluster sum of squares. Formally, this can be written as:

$$\arg\min_{C} \sum_{i=1}^{c} \sum_{x \in C_i} \textsc{Dist}(\boldsymbol{x}, \boldsymbol{\mu}_i) . \tag{10}$$

Finding the optimal solution to this problem is NP-hard [13], but efficient approximation algorithms [36, 52] have been developed that return an approximate solution in linear time (see Section 5). While k-means is fast, it is known to be heavily dependent on its initial centroid locations [3, 6], which can be partially solved via repeated re-initialization and picking the clustering that minimizes the inertia.

Poor initialization is typically not a large issue in k-means use cases since the end goal is usually to compute a single clustering so we can simply repeat and re-initialize until we are satisfied. However, since we generate a new clustering once per epoch in CARL-G, poor initialization can result in a large amount of variance between epochs.

To minimize the chance of poor centroid initialization occurring during training, we carry the cluster centroids over between epochs. The centroids will naturally update after running k-means since the embeddings $\boldsymbol{Z}$ changes each epoch (after backpropagation with CVI-based loss).

3.3 Theorectical Analysis

To gain a theoretical understanding of why our framework works, we compare it to Margin loss — a fundamental contrastive loss function that has been shown to work well for self-supervised representation learning [22, 68]. We show that CVI-based loss (especially silhouette loss) has some similarity to Margin based loss, which intuitively explains the success of CVI-based loss. In addition, we show that CVI-based loss has the advantages of (a) lower sensitivity to graph structure, and (b) no negative sampling required.

3.3.1 Similarity analysis of CVI-based loss and Margin loss. Both Margin loss and CVI-based loss fundamentally consist of two terms: one measuring the distance between neighbors/inter-cluster points and one measuring the distance between non-neighbors/inter-cluster points. This similarity allows us to analyze basic versions of our

proposed silhouette loss and margin loss in the context of node classification and show that they are identical in both of their ideal cases. We further analyze the sensitivity of these losses with respect to various parameters of the graph to examine the advantages and disadvantages of our proposed method. To do this, we first define the mean silhouette and margin loss functions:

Definition 3.1 (Mean Silhouette). We define the mean silhouette loss (removing the hyperparameter τ_{SIL}, focusing on the numerator (un-normalizing the index) and replacing min with the mean) as follows:

$$L_{\text{MS}}(u) = -(b_{\text{MS}}(u) - a(u)) = a(u) - b_{\text{MS}}(u)\,, \tag{11}$$

where

$$b_{\text{MS}}(u) = \frac{1}{c-1}\sum_{j\neq i}\frac{1}{|C_j|}\sum_{v\in C_j}\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\,. \tag{12}$$

Definition 3.2 (Margin Loss). We define margin loss as follows:

$$\begin{aligned}\textsc{ml}(u) =& \frac{1}{|\mathcal{N}(u)|}\sum_{v\in\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) \\ &- \frac{1}{|\mathcal{V} - \mathcal{N}(u) - \{u\}|}\sum_{t\notin\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_t)\,.\end{aligned} \tag{13}$$

It is worth noting that this margin loss differs from the max-margin loss traditionally used in graph SSL [68]. We simplify it above in Definition 3.2 by removing the max function for ease of analysis.

Let $\mathcal{L}$ be the set of true class labels, and $\mathcal{L}_u$ be the class label for a node $u \in \mathcal{V}$. We define the expected inter-class and intra-class distances as follows:

$$\mathbb{E}\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\right] = \begin{cases}\alpha, & \text{if } \mathcal{L}_u = \mathcal{L}_v\,;\\ \beta, & \text{otherwise,}\end{cases} \tag{14}$$

where $\alpha, \beta \in \mathbb{R}^+$. Next, let

$$P\left((u,v)\in\mathcal{E}\right) = \begin{cases}p, & \text{if } \mathcal{L}_u = \mathcal{L}_v\,;\\ q, & \text{otherwise,}\end{cases} \tag{15}$$

i.e., $\mathcal{G}$ follows a stochastic block model with a probability matrix $P \in [0,1]^{c\times c}$ of the form:

$$P = \begin{bmatrix} p & q & q & q \\ q & \ddots & \ddots & \ddots \\ q & \ddots & \ddots & q \\ q & \ddots & q & p \end{bmatrix}. \tag{16}$$

Note that q does not necessarily equal $1-p$. Finally, we define the inter-class clustering error rate ϵ and intra-class clustering error rate δ as follows:

$$P(C_u \neq C_v | \mathcal{L}_u = \mathcal{L}_v) = \epsilon\,; \tag{17}$$

$$P(C_u = C_v | \mathcal{L}_u \neq \mathcal{L}_v) = \delta\,. \tag{18}$$

Claim 1. *Given the above assumptions, the expected value of the simplified silhouette loss approaches that of the margin loss as $p \to 1, q \to 0$, and $\epsilon, \delta \to 0$.*

Proof. To find $\mathbb{E}\left[L_s(u)\right]$, we first find $\mathbb{E}\left[a(u)\right]$ and $\mathbb{E}\left[b_s(u)\right]$:

$$\mathbb{E}\left[a(u)\right] = \frac{\alpha}{c} - \frac{\epsilon\alpha}{c} + \delta\beta - \frac{\delta\beta}{c}\,; \tag{19}$$

$$\mathbb{E}\left[b_s(u)\right] = \frac{\epsilon\alpha}{c} + \beta - \beta\delta - \frac{\beta}{c} + \frac{\delta\beta}{c}\,; \tag{20}$$

$$\begin{aligned}\mathbb{E}\left[L_s(u)\right] &= \mathbb{E}\left[a(u)\right] - \mathbb{E}\left[b_s(u)\right] \\ &= -\frac{2\epsilon\alpha}{c} - \frac{2\delta\beta}{c} + \frac{\beta}{c} + \frac{\alpha}{c} - \beta + 2\delta\beta\,.\end{aligned} \tag{21}$$

Next, we take its limit as $\epsilon, \delta \to 0$:

$$\lim_{\epsilon,\delta\to 0}\left(-\frac{2\epsilon\alpha}{c} - \frac{2\delta\beta}{c} + \frac{\beta}{c} + \frac{\alpha}{c} - \beta + 2\delta\beta\right) = \frac{\beta}{c} + \frac{\alpha}{c} - \beta\,. \tag{22}$$

To find $\mathbb{E}\left[\textsc{ml}(u)\right]$, we first find the expected value of its left and right sides:

$$\mathbb{E}\left[\frac{1}{\mathcal{N}(u)}\sum_{v\in\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\right] = \frac{\alpha p}{c} + \beta q - \frac{\beta q}{c}\,; \tag{23}$$

$$\mathbb{E}\left[\frac{1}{|\mathcal{V} - \mathcal{N}(u) - \{u\}|}\sum_{t\notin\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_t)\right] \tag{24}$$

$$= \frac{\alpha}{c} - \frac{\alpha p}{c} + \beta - \beta q - \frac{\beta}{c} + \frac{\beta q}{c}\,. \tag{25}$$

Substituting them back in, we get

$$\mathbb{E}\left[\textsc{ml}(u)\right] = \frac{2\alpha p}{c} + 2\beta q - \frac{2\beta q}{c} - \frac{\alpha}{c} + \frac{\beta}{c} - \beta\,. \tag{26}$$

Taking its limit as $p \to 1, q \to 0$, we find

$$\lim_{p\to 1, q\to 0}\left(\frac{2\alpha p}{c} + 2\beta q - \frac{2\beta q}{c} - \frac{\alpha}{c} + \frac{\beta}{c} - \beta\right) \tag{27}$$

$$= \frac{2\alpha}{c} - \frac{\alpha}{c} + \frac{\beta}{c} - \beta = \frac{\alpha}{c} + \frac{\beta}{c} - \beta\,. \tag{28}$$

$$\therefore \lim_{p\to 1, q\to 0}\mathbb{E}\left[\textsc{ml}(u)\right] = \lim_{\epsilon,\delta\to 0}\mathbb{E}\left[L_s(u)\right]\,. \tag{29}$$

□

Since the two loss functions are identical in their ideal cases, one may wonder: *Why not use margin loss instead?* Well, the silhouette-based loss has two key advantages:

3.3.2 Lower sensitivity to graph structure. The margin loss is minimized as $p \to 1$ and $q \to 0$. However, p and q are attributes of the graph itself, making it difficult for a user to directly improve the performance of a model using that loss function. On the other hand, the mean silhouette depends on ϵ and δ, the inter/intra-class clustering error rates, instead. Even on the same graph, a silhouette-based loss can likely be improved by either choosing a more suitable clustering method or distance metric. This greatly increases the flexibility of this loss function.

3.3.3 No negative sampling. Negative sampling is required for most graph contrastive methods and often requires either many samples [24] or carefully chosen samples [67, 69]. This is costly, often costing quadratic time [60]. The primary advantage of non-contrastive methods is that they avoid this step [4, 60]. The simplified silhouette avoids this issue by only working in the $n \times d$ embedding space instead of the $n \times n$ graph. It also contrasts node

representations against centroid representations instead of against other nodes directly.

4 EXPERIMENTAL EVALUATION

We evaluate 3 variants of CARL-G: (a) **CARL-G_{SIL}** — based on the silhouette loss in Equation (7), (b) **CARL-G_{VRC}** — based on the VRC loss in Equation (7), and (c) **CARL-G_{SIM}** — based on the simplified silhouette loss in Equation (8). We evaluate these variants on 5 datasets on node classification and thoroughly benchmark their memory usage and runtime. We then select the best-performing variant, CARL-G_{SIM}, and evaluate its performance across 2 additional tasks: node clustering, and embedding similarity search.

4.0.1 Node Classification. A common task for GNNs is to classify each node into one of several different classes. In the supervised setting, this is often explicitly optimized for during the training process since the GNN is typically trained with cross-entropy loss over the labels [22, 33] but this is not possible for graph SSL methods where we do not have the labels during the training of the GNN. As such, the convention [4, 34, 60, 61, 79] is to train a logistic regression classifier on the frozen embeddings produced by the GNN.

Following previous works, we train a logistic regression model with ℓ_2 regularization on the frozen embeddings produced by our encoder model. We compare against a variety of self-supervised baselines, including both GNN-based and non-GNN-based: DeepWalk [45], RandomInit [61], DGI [61], GMI [44], MVGRL [24], GRACE [79], G-BT [4], AFGRL [34], and BGRL [60]. We also evaluate our method against two supervised models: GCA [80] and a GCN [33]. We follow [34, 60] and use an 80/10/10 train/validation/test, early stopping on the validation accuracy. We re-run AFGRL and BGRL using their published code and weights (where possible) on that split[1]. Finally, we use node2vec results from [34] and the reported results of the other baseline methods from their respective papers. See Section 4.4 for implementation details.

4.0.2 Node Clustering. Following previous graph representation learning work [24, 34, 41], we also evaluate CARL-G on the task of node clustering. We fit a k-means model on the generated embeddings $\boldsymbol{H}$ using the evaluation criteria from [34] — NMI and cluster homogeneity. Following [34], we re-run our model with different hyperparameters (the embeddings are not the same as node classification) and report the highest clustering scores. Due to computational resource constraints, we choose to only evaluate CARL-G_{SIM}, the overall best-performing model. We report the scores of the baselines models from [34].

4.0.3 Similarity Search. Following [34], we evaluate our model on node similarity search. The goal of similarity search is to, given a query node u, return the k nearest neighbors. In our setting, the goal is to return other nodes belonging to the same class as the query node. We evaluate the performance of each method by computing its Hits@k — the percentage of the top k neighbors that belong to the same class. Similar to [34], we evaluate our model every epoch and report the highest similarity search scores. We use $k \in \{5, 10\}$ and use the scores from [34] as baseline results.

[1]The official BGRL implementation online uses an 80/0/20 split compared to the 80/10/10 split mentioned in [60], so we re-run their trained models on an 80/10/10 split. Most results are similar but we get slightly different results on `Amazon-Computers`.

		GRACE	GCA	BGRL	AFGRL	CARL-G_{SIM}
`WikiCS`	NMI	0.428	0.337	0.397	0.413	**0.471**
	Hom.	0.442	0.353	0.416	0.430	**0.491**
`Computers`	NMI	0.479	0.528	0.536	0.552	**0.558**
	Hom.	0.522	0.582	0.587	0.604	**0.607**
`Photo`	NMI	0.651	0.644	0.684	0.656	**0.701**
	Hom.	0.666	0.658	0.700	0.674	**0.718**
`Co.CS`	NMI	0.756	0.762	0.773	0.786	**0.790**
	Hom.	0.791	0.797	0.804	**0.816**	0.815
`Co.Physics`	NMI	OOM	OOM	0.557	0.729	**0.771**
	Hom.	OOM	OOM	0.602	0.735	**0.776**

Table 2: Node clustering performance in terms of cluster NMI and homogeneity. CARL-G_{SIM} outperforms the baselines on 4/5 datasets.

4.1 Evaluation Results

4.1.1 Node Classification Performance. We show the node classification accuracy of our three proposed methods along with the baseline results in Table 5. CARL-G_{SIL} generally performs the best of all the evaluated methods, with the highest accuracy on 4/5 of the datasets (all except `Wiki-CS`). CARL-G_{SIM} generally performs similarly to CARL-G_{SIL}, with similar performance on all datasets except `Wiki-CS` and `Amazon-Photos`, and still outperforms the baselines on 4/5 datasets. CARL-G_{VRC} is the weakest-performing method of the three methods. It only outperforms baselines on 2/5 datasets. Since CARL-G_{SIM} is much faster than CARL-G_{SIL} (see Section 3.1 and Figure 4a) without sacrificing much performance, we focus on CARL-G_{SIM} for the remainder of the evaluation tasks.

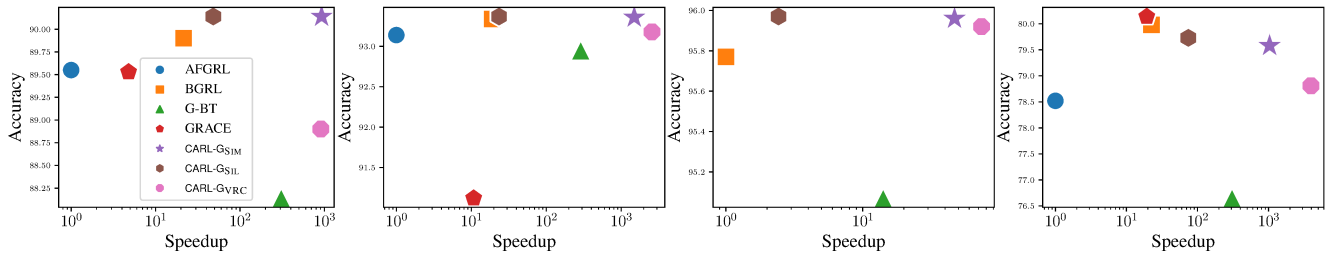
4.1.2 Node Clustering Performance. We evaluate CARL-G_{SIM} on node clustering and display the results in Table 2. We find that it generally outperforms its baselines on 5/5 datasets in terms of NMI and 4/5 datasets in terms of homogeneity. CARL-G_{SIM} and AFGRL [34] both encourage a clusterable representations by utilizing k-means clustering as part of their respective training pipelines.

4.1.3 Similarity Search Performance. We evaluate CARL-G_{SIM} on similarity search in Table 6, where it roughly performs on par with AFGRL, the best-performing baseline. This is surprising, as AFGRL specifically optimizes for the similarity search task by using k-NN as one of the criteria to sample neighbors.

4.2 Resource Benchmarking

We benchmark the 3 variants of our proposed method against BGRL [60] (the best-performing baseline), AFGRL [34] (the most recent baseline), G-BT [4] (the fastest baseline), and GRACE [79] (a strong contrastive baseline). We time the amount of time it takes to train each of the best-performing node classification models. We remove all evaluation code and purely measure the amount of time it takes to train each method, taking care to synchronize all asynchronous GPU operations. We use the default values in the respective papers for AFGRL and BGRL: 5,000 epochs for AFGRL and 10,000 epochs for BGRL. We use 50 epochs for CARL-G, although our method converges much faster in practice.

We also measure the GPU memory usage of each method. We use the hyperparameters by the respective paper authors for each



(a) Amazon-Computers (b) Coauthor-CS (c) Coauthor-Physics (d) Wiki-CS

Figure 3: Runtime v.s. accuracy plots. CARL-G_{SIM}, CARL-G_{SIL}, and CARL-G_{VRC} are our proposed methods. Speedup is relative to the slowest baseline (AFGRL). AFGRL and GRACE run out of memory on Coauthor-Physics.

dataset, which is why the methods use different encoder sizes. Note that the encoder sizes greatly affect the runtime and memory usage of each of the models, so we report the layer sizes used in Table 3. Our benchmarking results can be found in Figures 4a and 4b.

	Computers	Photos	Co-CS	Co-Phy	Wiki
AFGRL	[512]	[512]	[1024]	OOM	[1024]
BGRL	[256,128]	[256,128]	[512,256]	[256,128]	[512,256]
G-BT	[256,128]	[512,256]	[512,256]	[256,128]	[512,256]
GRACE	[256,128]	[256,128]	[512,256]	[256,128]	[512,256]
CARL-G_{SIM}	[512,256]	[512,256]	[512,256]	[512,256]	[512,256]
CARL-G_{SIL}	[512,256]	[512,256]	[512,256]	[512,256]	[512,256]
CARL-G_{VRC}	[512,256]	[512,256]	[512,256]	[512,256]	[512,256]

Table 3: GCN layer sizes used by the encoder for each method. The layer sizes greatly affect the amount of memory used by each model (shown in Figure 4b).

4.2.1 CARL-G is fast. In Figure 4a, we show that CARL-G_{SIM} is much faster than competing baselines, even in cases where the encoder is larger (see Table 3). BGRL is the best-performing node classification baseline, and CARL-G_{SIM} is about 79× faster on Coauthor-CS, and 57× faster on Coauthor-Physics. AFGRL is by far the slowest method, requiring much longer to train.

4.2.2 CARL-G works with a fixed encoder size. We find that CARL-G works well with a fixed encoder size (see Table 3). Unlike AFGRL, BGRL, GRACE, and G-BT, we fix the encoder size for CARL-G across all datasets. This has practical advantages by allowing a user to fix the model size across datasets, thereby reducing the number of hyperparameters in the model. We observed that increasing the embedding size also increases the performance of our model across all datasets. This is not true for all of our baselines — for example, [34] found that BGRL, GRACE, and GCA performance will often decrease in performance as embedding sizes increase. We limited our model embedding size to 256 for a fair comparison with other models.

4.2.3 CARL-G_{SIM} uses much less memory for the same encoder size. When the encoder sizes of baseline methods are the same, CARL-G_{SIM} uses much less memory than the baselines. The GPU memory usage of CARL-G_{SIM} is also much lower than (about half) the memory usage of a BGRL model of the same size. This is because BGRL stores two copies of the encoder with different weights. The second encoder's weights gradually approach that of the first encoder during training but still takes up twice the space compared to single-encoder models like CARL-G_{SIM} or G-BT [4].

4.2.4 CARL-G_{SIM}'s runtime is linear with respect to the number of neighbors. In Section 3.1, we mention that the runtime of CARL-G_{SIL}, the silhouette-based loss, is $O(n^2)$. This was the motivation for us to propose CARL-G_{SIM} — the simplified-silhouette-based loss which has an $O(nc)$ runtime instead.

4.3 Ablation Studies

We perform an ablation and sensitivity analysis on several aspects of our model. First, we examine the sensitivity of our model with respect to c—the number of clusters. Second, we examine the effect of using k-medoids instead of k-means. Finally, we try to inject more graph structural information during the clustering stage to see if we are losing any information.

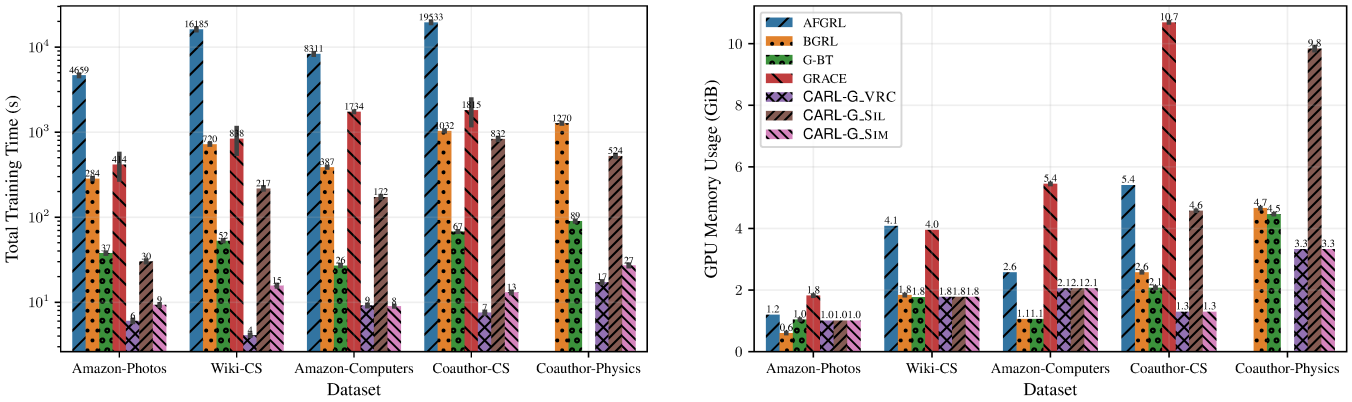
4.3.1 Effect of the number of clusters. We perform sensitivity analysis on c — the number of clusters (see Figures 5a and 5b). We find that, generally, the accuracy of our method goes up as the number of clusters increases. As the number of clusters continues to increase, the accuracy begins to drop. This implies that, much like traditional clustering [50], there is some "sweet spot" for c. However, it is worth noting that this number does not directly correspond to the number of classes in the data and is much higher than c for all of the datasets. DeepCluster [8] also makes similar observations, where they find 10,000 clusters is ideal for ImageNet, despite there only being 1,000 labeled classes.

4.3.2 k-medoids v.s. k-means. We study the effect of using k-medoids instead of k-means as our clustering algorithm. Both algorithms are partition-based clustering methods [64] and have seen optimizations in recent years [51, 52]. We find that the k-means-based CARL-G_{SIM} generally performs better across all 4 of the evaluated datasets. The differences in node classification accuracy are shown in Table 4.

Table 4: k-medoids w/ CARL-G_{SIM}.

Dataset	Accuracy Δ
Computers	-1.41
Co.CS	-0.33
Co.Phy	-0.07
Photos	-0.11

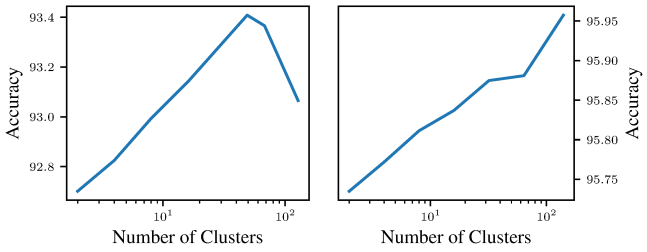
4.3.3 Does additional information help? It may appear as if we are losing graph information by working only with the embeddings. If this is the case, we should be able to improve the performance of our method by injecting additional information into the clustering



(a) Mean total training time. The y-axis is on a log. scale.

(b) Max GPU memory usage.

Figure 4: Mean total training time (left) and max GPU usage (right) for each model. CARL-G_{VRC} is the fastest with generally the least amount of memory used. CARL-G_{SIM} uses the same amount of memory but is slightly slower. Note that not all of the baselines use the same encoder size—see Table 3 for encoder sizes.



(a) Acc. on `Amazon-Photos`.

(b) Acc. on `Coauthor-Physics`.

Figure 5: Node classification accuracy of CARL-G_{SIM} on `Amazon-Photos` and `Coauthor-Physics` with a different number of clusters.

step. We can do this by modifying the distance function of our clustering algorithm to the following:

$$\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) = \lambda \left\|\boldsymbol{h}_u - \boldsymbol{h}_v\right\|_2 + (1-\lambda)\boldsymbol{D}_{u,v}\,, \tag{30}$$

where $\boldsymbol{D}$ is the all-pairs shortest path (APSP) length matrix of $\mathcal{G}$. This allows us to inject node neighborhood information into the clustering algorithm on top of the aggregation performed by the GNN. However, we find there is no significant change in performance for low λ and performance decreases for high λ. This helps confirm the hypothesis that the GNN encoder is able to successfully embed a sufficient amount of structural data in the embedding.

4.4 Implementation Details

For fair evaluation with other baselines, we elect to use a standard GCN [33] encoder. Our focus is on the overall framework rather than the architecture of the encoder. All of our baselines also use GCN layers. Following [34, 60], we use two-layer GCNs for all datasets and use a two-layer MLP for the predictor network.We implement our model with PyTorch [42] and PyTorch Geometric [19]. A copy of our code is publically available at https://github.com/willshiao/carl-g. We adapt the code from [55], which contains implementations of BGRL [60], GRACE [79], and GBT [4] to use the split and downstream tasks from [34]. We also use the official implementation of AFGRL [34]. We perform 50 runs of Bayesian hyperparameter optimization on each dataset and task for each of our 3 methods. The hyperparameters for our results are available at that link. All of our timing experiments were conducted on Google Cloud Platform using 16 GB NVIDIA V100 GPUs.

4.5 Limitations & Future Work

While our proposed framework has been shown to be highly effective in terms of both training speed and performance across the 3 tasks, there are also some limitations to our approach. One such limitation is that we use hard clustering assignments, i.e., each node is assigned to exactly one cluster. This can pose issues for multi-label datasets like the Protein-Protein Interaction (PPI) [81] graph dataset. One possible solution to this problem is to perform soft clustering and use a weighted average of CVIs for second/tertiary cluster assignments, but this would require major modifications to our method, and we reserve an exploration of this for future work.

5 ADDITIONAL RELATED WORK

Deep Clustering. A related, but distinct, area of work is deep clustering, which uses a neural network to directly aid in the clustering of data points [1]. However, the fundamental goal of deep clustering differs from graph representation learning in that the goal is to produce a clustering of the graph nodes rather than just representations of them. An example of this is DEC [63], which uses a deep autoencoder with KL divergence to learn cluster centers, which are then used to cluster points with k-means.

Clustering for Representation Learning. There exists work that uses clustering to learn embeddings [8, 65, 78]. Notably, DeepCluster [8] trains a CNN with standard cross-entropy loss on pseudo-labels produced by k-means. Similarly, [65] simultaneously performs clustering and dimensionality reduction with a deep neural network. The key difference between those models and our proposed framework is that we use graph data and CVI-based losses instead of traditional supervised losses.

	Method	Wiki-CS	Amazon-Computers	Amazon-Photos	Coauthor-CS	Coauthor-Physics
Traditional	Raw Features	71.98 ± 0.00	73.81 ± 0.00	78.53 ± 0.00	90.37 ± 0.00	93.58 ± 0.00
	node2vec [20]	71.79 ± 0.05	84.39 ± 0.08	89.67 ± 0.12	85.08 ± 0.03	91.19 ± 0.04
	DeepWalk [45]	74.35 ± 0.06	85.68 ± 0.06	89.44 ± 0.11	84.61 ± 0.22	91.77 ± 0.15
	DeepWalk [45] + Feat.	77.21 ± 0.03	86.28 ± 0.07	90.05 ± 0.08	87.70 ± 0.04	94.90 ± 0.09
GNN SSL	Random-Init [61]	78.95 ± 0.58	86.46 ± 0.38	92.08 ± 0.48	91.64 ± 0.29	93.71 ± 0.29
	DGI [61]	75.35 ± 0.14	83.95 ± 0.47	91.61 ± 0.22	92.15 ± 0.63	94.51 ± 0.09
	GMI [44]	74.85 ± 0.08	82.21 ± 0.31	90.68 ± 0.17	OOM	OOM
	MVGRL [24]	77.52 ± 0.08	87.52 ± 0.11	91.74 ± 0.07	92.11 ± 0.12	95.33 ± 0.03
	GRACE [79]	**80.14** ± 0.48	89.53 ± 0.35	92.78 ± 0.45	91.12 ± 0.20	OOM
	G-BT [4]	76.65 ± 0.62	88.14 ± 0.33	92.63 ± 0.44	92.95 ± 0.17	95.07 ± 0.17
	AFGRL [34]	78.52 ± 0.72	89.55 ± 0.36	92.91 ± 0.26	93.14 ± 0.23	OOM
	BGRL [60]	<u>79.98</u> ± 0.10	<u>89.90</u> ± 0.19	93.17 ± 0.30	93.34 ± 0.13	95.77 ± 0.05
Proposed	CARL-G_{VRC}	78.81 ± 0.49	88.90 ± 0.39	93.31 ± 0.36	93.18 ± 0.31	95.92 ± 0.14
	CARL-G_{SIM}	79.58 ± 0.60	**90.14** ± 0.33	<u>93.37</u> ± 0.37	<u>93.36</u> ± 0.39	<u>95.96</u> ± 0.09
	CARL-G_{SIL}	79.73 ± 0.44	**90.14** ± 0.34	**93.44** ± 0.32	**93.37** ± 0.33	**95.97** ± 0.14
Supervised	GCA [80]	78.35 ± 0.05	88.94 ± 0.15	92.53 ± 0.16	93.10 ± 0.01	95.73 ± 0.03
	Supervised GCN [33]	77.19 ± 0.12	86.51 ± 0.54	92.42 ± 0.22	93.03 ± 0.31	95.65 ± 0.16

Table 5: Table of node classification accuracy. Bolded entries indicate the highest accuracy for that dataset. Underlined entries indicate the second-highest accuracy. OOM indicates out-of-memory.

		GRACE	GCA	BGRL	AFGRL	CARL-G_{SIM}
WikiCS	Hits@5	0.775	0.779	0.774	0.781	**0.789**
	Hits@10	0.765	0.767	0.762	0.766	**0.775**
Computers	Hits@5	0.874	0.883	0.895	**0.897**	0.881
	Hits@10	0.864	0.874	0.886	**0.889**	0.871
Photo	Hits@5	0.916	0.911	**0.925**	0.924	0.922
	Hits@10	0.911	0.905	**0.920**	0.917	0.917
Co.CS	Hits@5	0.910	0.913	0.911	**0.918**	0.916
	Hits@10	0.906	0.910	0.909	**0.914**	**0.914**
Co.Physics	Hits@5	OOM	OOM	0.950	**0.953**	**0.953**
	Hits@10	OOM	OOM	0.946	0.949	**0.950**

Table 6: Performance on similarity search. Surprisingly, CARL-G performs fairly well on this task, despite not being explicitly optimized for this task (unlike AFGRL, which uses KNN during training).

Clustering for Efficient GNNs. There also exists work that uses clustering to speed up GNN training and inference. Cluster-GCN [11] samples node blocks produced by graph clustering algorithms and speeds up GCN layers by limiting convolutions within each block for training and inference. However, it is worth noting that it computes a fixed clustering, rather than updating the clustering jointly with our model (unlike CARL-G). FastGCN [10] does not explicitly cluster nodes but uses Monte Carlo importance sampling to similarly reduce neighborhood size and improve the speed of GCNs.

Efficient k-means. Over the years, many variants and improvements to k-means have been proposed. The original method proposed to solve the k-means assignment problem was Lloyd's algorithm [36]. Since then, several more efficient algorithms have been developed. Bottou and Bengio [5] propose using stochastic gradient descent for finding a solution. Sculley [52] further builds on this work by proposing a k-means variant that uses mini-batching to dramatically speed up training. Finally, approximate nearest-neighbor search libraries like FAISS [29] allow for efficient querying of nearest neighbors, further speeding up training.

6 CONCLUSION

In this work, we are the first to introduce Cluster Validation Indexes in the context of graph representation learning. We propose a novel CVI-based framework and investigated trade-offs between different CVI variants. We find that the loss function based on the simplified silhouette achieves the best overall performance to runtime ratio. It outperforms all baselines across 4/5 datasets in node classification and node clustering tasks, training up to 79× faster than the best-performing baseline. It also performs on-par with the best performing node similarity search baseline while training 1,500× faster. Moreover, to more comprehensively understand the effectiveness of CARL-G, we establish a theoretical connection between the silhouette and the well-established margin loss.

ACKNOWLEDGMENTS

The authors would like to thank UCR Research Computing and Ursa Major for the Google Cloud resources provided to support this research. This research was also supported by the National Science Foundation under CAREER grant no. IIS 2046086 and CREST Center for Multidisciplinary Research Excellence in Cyber-Physical Infrastructure Systems (MECIS) grant no. 2112650, by the Agriculture and Food Research Initiative Competitive Grant no. 2020-69012-31914 from the USDA National Institute of Food and Agriculture and by the Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes not withstanding any copyright notation here on.

REFERENCES

[1] Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, and Daniel Cremers. 2018. Clustering with Deep Learning: Taxonomy and New Methods. arXiv:1801.07648 http://arxiv.org/abs/1801.07648

[2] Olatz Arbelaitz, Ibai Gurrutxaga, Javier Muguerza, Jesús M Pérez, and Iñigo Perona. 2013. An extensive comparative study of cluster validity indices. *Pattern recognition* 46, 1 (2013), 243–256.

[3] David Arthur and Sergei Vassilvitskii. 2006. *k-means++: The advantages of careful seeding*. Technical Report. Stanford.

[4] Piotr Bielak, Tomasz Kajdanowicz, and Nitesh V Chawla. 2022. Graph Barlow Twins: A self-supervised representation learning framework for graphs. *Knowledge-Based Systems* 256 (2022), 109631.

[5] Leon Bottou and Yoshua Bengio. 1994. Convergence properties of the k-means algorithms. *Advances in neural information processing systems* 7 (1994).

[6] Paul S Bradley and Usama M Fayyad. 1998. Refining initial points for k-means clustering.. In *ICML*, Vol. 98. Citeseer, Morgan Kaufmann, San Francisco, CA, USA, 91–99.

[7] Tadeusz Caliński and Jerzy Harabasz. 1974. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods* 3, 1 (1974), 1–27.

[8] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. 2018. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European conference on computer vision (ECCV)*. ECCV, ECCV 2018, 132–149.

[9] Chi Chen, Weike Ye, Yunxing Zuo, Chen Zheng, and Shyue Ping Ong. 2019. Graph networks as a universal machine learning framework for molecules and crystals. *Chemistry of Materials* 31, 9 (2019), 3564–3572.

[10] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling.

[11] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. Association for Computing Machinery, New York, NY, USA, 257–266.

[12] Leon Danon, Albert Diaz-Guilera, Jordi Duch, and Alex Arenas. 2005. Comparing community structure identification. *Journal of statistical mechanics: Theory and experiment* 2005, 09 (2005), P09008.

[13] Sanjoy Dasgupta. 2008. *The hardness of k-means clustering*. Department of Computer Science and Engineering, University of California, San Diego, San Diego, CA, USA.

[14] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. 2021. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. ACM, Queensland, Australia, 3767–3776.

[15] J. C. Dunn. 1974. Some Recent Investigations of a New Fuzzy Partitioning Algorithm and its Application to Pattern Classification Problems. *Journal of Cybernetics* 4, 2 (1974), 1–15. https://doi.org/10.1080/01969727408546062 arXiv:https://doi.org/10.1080/01969727408546062

[16] Brian. Everitt. 2001. *Cluster analysis /*. Heinemann Educational for the Social Science Research Council,, London :.

[17] Shuangfei Fan and Bert Huang. 2019. Labeled Graph Generative Adversarial Networks. *CoRR* abs/1906.03220 (2019), 1–10. arXiv:1906.03220 http://arxiv.org/abs/1906.03220

[18] Wenqi Fan, Xiaorui Liu, Wei Jin, Xiangyu Zhao, Jiliang Tang, and Qing Li. 2022. Graph Trend Filtering Networks for Recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, Madrid, Spain, 112–121.

[19] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric.

[20] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*. ACM, New York, NY, USA, 855–864.

[21] Zhichun Guo, Chuxu Zhang, Wenhao Yu, John Herr, Olaf Wiest, Meng Jiang, and Nitesh V Chawla. 2021. Few-shot graph learning for molecular property prediction. In *Proceedings of the Web Conference 2021*. ACM, New York, NY, USA, 2559–2567.

[22] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. NeurIPS, Long Beach, CA, 1024–1034.

[23] Xiaotian Han, Tong Zhao, Yozen Liu, Xia Hu, and Neil Shah. 2022. MLPInit: Embarrassingly Simple GNN Training Acceleration with MLP Initialization.

[24] Kaveh Hassani and Amir Hosein Khas Ahmadi. 2020. Contrastive Multi-View Representation Learning on Graphs. In *ICML (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, Cambridge, MA, 4116–4126.

[25] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. ACM, New York, NY, USA, 639–648.

[26] Eduardo R Hruschka, Leandro Nunes de Castro, and Ricardo JGB Campello. 2004. Evolutionary algorithms for clustering gene-expression data. In *Fourth IEEE International Conference on Data Mining (ICDM'04)*. IEEE, IEEE, New York, 403–406.

[27] Wei Jin, Xiaorui Liu, Xiangyu Zhao, Yao Ma, Neil Shah, and Jiliang Tang. 2021. Automated Self-Supervised Learning for Graphs. *CoRR* abs/2106.05470 (2021), 1–20. arXiv:2106.05470 https://arxiv.org/abs/2106.05470

[28] Wei Jin, Tong Zhao, Jiayuan Ding, Yozen Liu, Jiliang Tang, and Neil Shah. 2022. Empowering graph representation learning with test-time graph transformation.

[29] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[30] Mingxuan Ju, Tong Zhao, Qianlong Wen, Wenhao Yu, Neil Shah, Yanfang Ye, and Chuxu Zhang. 2023. Multi-task Self-supervised Graph Neural Networks Enable Stronger Task Generalization.

[31] Zekarias T. Kefato and Sarunas Girdzijauskas. 2021. Self-supervised Graph Neural Networks without explicit negative sampling. *CoRR* abs/2103.14958 (2021), 1–8. arXiv:2103.14958 https://arxiv.org/abs/2103.14958

[32] Minho Kim and RS Ramakrishna. 2005. New indices for cluster validity assessment. *Pattern Recognition Letters* 26, 15 (2005), 2353–2363.

[33] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*. ICLR, Toulon, France, 1–14.

[34] Namkyeong Lee, Junseok Lee, and Chanyoung Park. 2022. Augmentation-free self-supervised learning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. AAAI Press, Virtual, 7372–7380.

[35] Lars Lenssen and Erich Schubert. 2022. Clustering by Direct Optimization of the Medoid Silhouette. In *Similarity Search and Applications: 15th International Conference, SISAP 2022, Bologna, Italy, October 5–7, 2022, Proceedings*. Springer, Springer, New York, NY, USA, 190–204.

[36] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[37] J MacQueen. 1965. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symposium on Math., Stat., and Prob.* University of California Press, CA, USA, 281.

[38] Glenn W Milligan and Martha C Cooper. 1985. An examination of procedures for determining the number of clusters in a data set. *Psychometrika* 50 (1985), 159–179.

[39] M. E. J. Newman, D. J. Watts, and S. H. Strogatz. 2002. Random graph models of social networks. *Proceedings of the National Academy of Sciences* 99, suppl_1 (2002), 2566–2572. https://doi.org/10.1073/pnas.012582999 arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.012582999

[40] Hae-Sang Park and Chi-Hyuck Jun. 2009. A simple and fast algorithm for K-medoids clustering. *Expert systems with applications* 36, 2 (2009), 3336–3341.

[41] Jiwoong Park, Minsik Lee, Hyung Jin Chang, Kyuewang Lee, and Jin Young Choi. 2019. Symmetric graph convolutional autoencoder for unsupervised graph representation learning. In *Proceedings of the IEEE/CVF international conference on computer vision*. IEEE, New York, NY, USA, 6519–6528.

[42] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019), 1–12.

[43] Dan Pelleg and Andrew Moore. 1999. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, New York, NY, USA, 277–281.

[44] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. 2020. Graph representation learning via graphical mutual information maximization. In *Proceedings of The Web Conference 2020*. ACM, New York, NY, USA, 259–270.

[45] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, New York, NY, USA, 701–710.

[46] William M Rand. 1971. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association* 66, 336 (1971), 846–850.

[47] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53–65. https://doi.org/10.1016/0377-0427(87)90125-7

[48] Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. 2021. Graph neural networks for friend ranking in large-scale social platforms. In *Proceedings of the Web Conference 2021*. ACM, New York, NY, USA, 2535–2546.

[49] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80. https://doi.org/10.1109/TNN.2008.2005605

[50] Erich Schubert. 2022. Stop using the elbow criterion for k-means and how to choose the number of clusters instead.

[51] Erich Schubert and Peter J Rousseeuw. 2019. Faster k-medoids clustering: improving the PAM, CLARA, and CLARANS algorithms. In *Similarity Search and Applications: 12th International Conference, SISAP 2019, Newark, NJ, USA, October 2–4, 2019, Proceedings 12*. Springer, Springer, New York, NY, USA, 171–187.

[52] David Sculley. 2010. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*. ACM, New York, NY, USA, 1177–1178.

[53] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[54] William Shiao, Zhichun Guo, Tong Zhao, Evangelos E Papalexakis, Yozen Liu, and Neil Shah. 2022. Link Prediction with Non-Contrastive Learning.

[55] William Shiao, Zhichun Guo, Tong Zhao, Evangelos E Papalexakis, Yozen Liu, and Neil Shah. 2022. Link Prediction with Non-Contrastive Learning. *arXiv preprint arXiv:2211.14394* abs/2211.14394 (2022), 1–19.

[56] William Shiao and Evangelos E Papalexakis. 2021. Adversarially Generating Rank-Constrained Graphs. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, IEEE, New York, NY, USA, 1–8.

[57] Blaž Škrlj, Jan Kralj, and Nada Lavrač. 2020. Embedding-based Silhouette community detection. *Machine Learning* 109 (2020), 2161–2193.

[58] Xianfeng Tang, Yozen Liu, Xinran He, Suhang Wang, and Neil Shah. 2022. Friend Story Ranking with Edge-Contextual Local Graph Convolutions. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. ACM, Singapore, 1007–1015.

[59] Xianfeng Tang, Yozen Liu, Neil Shah, Xiaolin Shi, Prasenjit Mitra, and Suhang Wang. 2020. Knowing your fate: Friendship, action and temporal explanations for user engagement prediction on social apps. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. ACM, San Diego, CA, 2269–2279.

[60] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L. Dyer, Rémi Munos, Petar Velickovic, and Michal Valko. 2022. Large-Scale Representation Learning on Graphs via Bootstrapping. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, Virtual, 1–18. https://openreview.net/forum?id=0UXT6PpRpW

[61] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2018. Deep Graph Infomax. https://doi.org/10.48550/ARXIV.1809.10341

[62] Fei Wang, Hector-Hugo Franco-Penya, John D Kelleher, John Pugh, and Robert Ross. 2017. An analysis of the application of simplified silhouette to the evaluation of k-means clustering validity. In *Machine Learning and Data Mining in Pattern Recognition: 13th International Conference, MLDM 2017, New York, NY, USA, July 15-20, 2017, Proceedings 13*. Springer, Springer, New York, NY, USA, 291–305.

[63] Junyuan Xie, Ross Girshick, and Ali Farhadi. 2016. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*. PMLR, PMLR, Cambridge, MA, USA, 478–487.

[64] Dongkuan Xu and Yingjie Tian. 2015. A comprehensive survey of clustering algorithms. *Annals of Data Science* 2 (2015), 165–193.

[65] Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. 2017. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *international conference on machine learning*. PMLR, PMLR, Cambridge, MA, USA, 3861–3870.

[66] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. 2020. Understanding negative sampling in graph representation learning. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. ACM, New York, NY, USA, 1666–1676.

[67] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. 2020. Understanding Negative Sampling in Graph Representation Learning. https://doi.org/10.48550/ARXIV.2005.09863

[68] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. ACM, London, UK, 974–983. https://doi.org/10.1145/3219819.3219890

[69] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. *Advances in Neural Information Processing Systems* 2018-December (6 2018), 4800–4810. http://arxiv.org/abs/1806.08804

[70] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*. PMLR, JMLR.org, Cambridge, MA, 5708–5717.

[71] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems* 33 (2020), 5812–5823.

[72] Hengrui Zhang, Qitian Wu, Junchi Yan, David Wipf, and Philip S Yu. 2021. From canonical correlation analysis to self-supervised graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 76–89.

[73] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*. Curran Associates, New York, NY, USA, 5165–5175.

[74] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2018. Deep Learning on Graphs: A Survey. *CoRR* abs/1812.04202 (2018), 1–24. arXiv:1812.04202 http://arxiv.org/abs/1812.04202

[75] Tong Zhao, Gang Liu, Stephan Günnemann, and Meng Jiang. 2022. Graph Data Augmentation for Graph Machine Learning: A Survey. https://doi.org/10.48550/ARXIV.2202.08871

[76] Tong Zhao, Gang Liu, Daheng Wang, Wenhao Yu, and Meng Jiang. 2022. Learning from counterfactual links for link prediction. In *International Conference on Machine Learning*. PMLR, PMLR, Cambridge, MA, 26911–26926.

[77] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. 2021. Data Augmentation for Graph Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. AAAI Press, Cambridge, MA, 11015–11023.

[78] Tong Zhao, Xianfeng Tang, Danqing Zhang, Haoming Jiang, Nikhil Rao, Yiwei Song, Pallav Agrawal, Karthik Subbian, Bing Yin, and Meng Jiang. 2022. Auto-GDA: Automated Graph Data Augmentation for Node Classification. In *The First Learning on Graphs Conference*. PMLR, Cambridge, MA, 1–17.

[79] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2020. Deep Graph Contrastive Representation Learning. *CoRR* abs/2006.04131 (2020), 1–17. arXiv:2006.04131 https://arxiv.org/abs/2006.04131

[80] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*. ACM, New York, NY, USA, 2069–2080.

[81] Marinka Zitnik and Jure Leskovec. 2017. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* 33, 14 (2017), i190–i198.

A APPENDIX

A.1 Full Proof of Equivalency to Margin Loss

Proof. For ease of analysis, we work with the simplified silhouette loss (Definition 3.1) and the non-max margin loss (Definition 3.2). Let $\mathcal{L}$ be the set of class labels, and $\mathcal{L}_u$ be the class label for node u. Let C_u be the cluster assignment for node u, and $c = |C|$ be the number of clusters/classes. We define the expected inter-class and intra-class distances as follows:

$$\mathbb{E}\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\right] = \begin{cases} \alpha & \text{if } \mathcal{L}_u = \mathcal{L}_v \\ \beta & \text{otherwise} \end{cases}, \tag{31}$$

where $\alpha, \beta \in \mathbb{R}^+$. Next, let

$$P\left((u,v) \in \mathcal{E}\right) = \begin{cases} p & \text{if } \mathcal{L}_u = \mathcal{L}_v \\ q & \text{otherwise} \end{cases}, \tag{32}$$

i.e., $\mathcal{G}$ follows a stochastic block model with a probability matrix $P \in [0,1]^{c \times c}$ of the form:

$$P = \begin{bmatrix} p & q & q & q \\ q & \ddots & \ddots & \ddots \\ q & \ddots & \ddots & q \\ q & \ddots & q & p \end{bmatrix}. \tag{33}$$

Note that q does not necessarily equal $1 - p$. We define the inter-class clustering error rate ϵ and intra-class clustering error rate δ as follows:

$$P(C_u \neq C_v | \mathcal{L}_u = \mathcal{L}_v) = \epsilon \tag{34}$$

$$P(C_u = C_v | \mathcal{L}_u \neq \mathcal{L}_v) = \delta\,. \tag{35}$$

To find $\mathbb{E}\left[s_s(u)\right]$, we first find $\mathbb{E}\left[a(u)\right]$ and $\mathbb{E}\left[b_s(u)\right]$:

$$\mathbb{E}\left[a(u)\right] = \mathbb{E}\left[\frac{1}{|C_i| - 1} \sum_{v \in (C_i - \{u\})} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\right] \tag{36}$$

$$= \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | C_u = C_v\right] \tag{37}$$

$$= P(\mathcal{L}_u = \mathcal{L}_v) \cdot P(C_u = C_v | \mathcal{L}_u = \mathcal{L}_v) \tag{38}$$
$$\cdot\, \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | C_u = C_v \wedge \mathcal{L}_u = \mathcal{L}_v\right]$$
$$+ P(\mathcal{L}_u \neq \mathcal{L}_v) \cdot P(C_u = C_v | \mathcal{L}_u \neq \mathcal{L}_v)$$
$$\cdot\, \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | C_u = C_v \wedge \mathcal{L}_u \neq \mathcal{L}_v\right]$$

$$= \left(\frac{1}{c}\right)(1 - \epsilon)\alpha + \left(1 - \frac{1}{c}\right)\delta\beta \tag{39}$$

$$= \frac{\alpha}{c} - \frac{\epsilon\alpha}{c} + \delta\beta - \frac{\delta\beta}{c} \tag{40}$$

and

$$\mathbb{E}\left[b_s(u)\right] = \mathbb{E}\left[\frac{1}{c-1} \sum_{j \neq i} \frac{1}{|C_j|} \sum_{v \in C_j} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\right] \tag{41}$$

$$= \mathbb{E}_{j \neq i}\left[\frac{1}{|C_j|} \sum_{v \in C_j} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\right] \tag{42}$$

$$= \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v | C_u \neq C_v)\right] \tag{43}$$

$$= P(\mathcal{L}_u = \mathcal{L}_v) \cdot P(C_u \neq C_v | \mathcal{L}_u = \mathcal{L}_v) \cdot \tag{44}$$
$$\mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | C_u \neq C_v \wedge \mathcal{L}_u = \mathcal{L}_v\right]$$
$$+ P(\mathcal{L}_u \neq \mathcal{L}_v) \cdot P(C_u \neq C_v | \mathcal{L}_u \neq \mathcal{L}_v)$$
$$\cdot\, \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | C_u \neq C_v \wedge \mathcal{L}_u \neq \mathcal{L}_v\right] \quad = \left(\frac{1}{c}\right)\epsilon\alpha + \left(1 - \frac{1}{c}\right)(1 - \delta)\beta$$

$$= \frac{\epsilon\alpha}{c} + \beta\left(1 - \delta - \frac{1}{c} + \frac{\delta}{c}\right) \tag{45}$$

$$= \frac{\epsilon\alpha}{c} + \beta - \beta\delta - \frac{\beta}{c} + \frac{\delta\beta}{c}\,. \tag{46}$$

Now, we can find $\mathbb{E}\left[s_s(u)\right]$:

$$\mathbb{E}\left[s_s(u)\right] = \mathbb{E}\left[b_s(u)\right] - \mathbb{E}\left[a(u)\right] \tag{47}$$

$$= \frac{\epsilon\alpha}{c} + \beta - \beta\delta - \frac{\beta}{c} + \frac{\delta\beta}{c} - \left(\frac{\alpha}{c} - \frac{\epsilon\alpha}{c} + \delta\beta - \frac{\delta\beta}{c}\right) \tag{48}$$

$$= \frac{\epsilon\alpha}{c} + \beta - \beta\delta - \frac{\beta}{c} + \frac{\delta\beta}{c} - \frac{\alpha}{c} + \frac{\epsilon\alpha}{c} - \delta\beta + \frac{\delta\beta}{c} \tag{49}$$

$$= \frac{2\epsilon\alpha}{c} + \frac{2\delta\beta}{c} - \frac{\beta}{c} - \frac{\alpha}{c} + \beta - 2\delta\beta\,. \tag{50}$$

Taking its limit as $\epsilon, \delta \to 0$, we find

$$\lim_{\epsilon,\delta \to 0}\left(-\frac{2\epsilon\alpha}{c} - \frac{2\delta\beta}{c} + \frac{\beta}{c} + \frac{\alpha}{c} - \beta + 2\delta\beta\right) = \frac{\beta}{c} + \frac{\alpha}{c} - \beta\,. \tag{51}$$

We similarly break down the margin loss into two terms:

$$\mathbb{E}\left[\frac{1}{\mathcal{N}(u)} \sum_{v \in \mathcal{N}(u)} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\right] \tag{52}$$

$$= \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | (u,v) \in \mathcal{E}\right] \tag{53}$$

$$= P(\mathcal{L}_u = \mathcal{L}_v) \cdot P((u,v) \in \mathcal{E} | \mathcal{L}_u = \mathcal{L}_v) \tag{54}$$
$$\cdot\, \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | (u,v) \in \mathcal{E} \wedge \mathcal{L}_u = \mathcal{L}_v\right]$$
$$+ P(\mathcal{L}_u \neq \mathcal{L}_v) \cdot P((u,v) \in \mathcal{E} | \mathcal{L}_u \neq \mathcal{L}_v)$$
$$\cdot\, \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | (u,v) \in \mathcal{E} \wedge \mathcal{L}_u \neq \mathcal{L}_v\right]$$

$$= \left(\frac{1}{c}\right)\alpha p + \left(1 - \frac{1}{c}\right)\beta q \tag{55}$$

$$= \frac{\alpha p}{c} + \beta q - \frac{\beta q}{c} \tag{56}$$

and

$$\mathbb{E}\left[\frac{1}{|\mathcal{V} - \mathcal{N}(u) - \{u\}|} \sum_{t \notin \mathcal{N}(u)} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_t)\right] \tag{57}$$

$$= \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | (u,v) \notin \mathcal{E}\right] \tag{58}$$

$$= P(\mathcal{L}_u = \mathcal{L}_v) \cdot P((u,v) \notin \mathcal{E} | \mathcal{L}_u = \mathcal{L}_v) \tag{59}$$
$$\cdot\, \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | (u,v) \notin \mathcal{E} \wedge \mathcal{L}_u = \mathcal{L}_v\right]$$
$$+ P(\mathcal{L}_u \neq \mathcal{L}_v) \cdot P((u,v) \notin \mathcal{E} | \mathcal{L}_u \neq \mathcal{L}_v)$$
$$\cdot\, \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | (u,v) \notin \mathcal{E} \wedge \mathcal{L}_u \neq \mathcal{L}_v\right]$$

$$= \left(\frac{1}{c}\right)(1 - p)\alpha + \left(1 - \frac{1}{c}\right)(1 - q)\beta \tag{60}$$

$$= \frac{\alpha}{c} - \frac{\alpha p}{c} + \beta - \beta q - \frac{\beta}{c} + \frac{\beta q}{c}\,. \tag{61}$$

Re-substituting the terms into the margin loss equation, we get

$$\mathbb{E}\left[\frac{1}{|\mathcal{N}(u)|}\sum_{v\in\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_v) - \frac{1}{|\mathcal{V}-\mathcal{N}(u)-\{u\}|}\sum_{t\notin\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_t)\right] \tag{62}$$

$$= \mathbb{E}\left[\frac{1}{|\mathcal{N}(u)|}\sum_{v\in\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_v)\right] - \mathbb{E}\left[\frac{1}{|\mathcal{V}-\mathcal{N}(u)-\{u\}|}\sum_{t\notin\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_t)\right] \tag{63}$$

$$= \frac{\alpha p}{c} + \beta q - \frac{\beta q}{c} - \left(\frac{\alpha}{c} - \frac{\alpha p}{c} + \beta - \beta q - \frac{\beta}{c} + \frac{\beta q}{c}\right) \tag{64}$$

$$= \frac{\alpha p}{c} + \beta q - \frac{\beta q}{c} - \frac{\alpha}{c} + \frac{\alpha p}{c} - \beta + \beta q + \frac{\beta}{c} - \frac{\beta q}{c} \tag{65}$$

$$= \frac{2\alpha p}{c} + 2\beta q - \frac{2\beta q}{c} - \frac{\alpha}{c} + \frac{\beta}{c} - \beta\,. \tag{66}$$

Taking its limit as $p \to 1, q \to 0$:

$$\lim_{p\to1,q\to0}\left(\frac{2\alpha p}{c} + 2\beta q - \frac{2\beta q}{c} - \frac{\alpha}{c} + \frac{\beta}{c} - \beta\right) \tag{67}$$

$$= \frac{2\alpha}{c} - \frac{\alpha}{c} + \frac{\beta}{c} - \beta = \frac{\alpha}{c} + \frac{\beta}{c} - \beta \tag{68}$$

$$\therefore \lim_{p\to1,q\to0}\mathbb{E}\left[\textsc{ml}(u)\right] = \lim_{\epsilon,\delta\to0}\mathbb{E}\left[L_S(u)\right]\,. \tag{69}$$

□

A.2 Meaning of Ideal Conditions

Our analysis in Appendix A.1 aims to show that the more traditional margin-based losses and silhouette-based losses are sensitive to different parameters and their equivalence in the best-case scenario. Here, we briefly summarize what each of those ideal conditions means:

- $p \to 1$: We approach the case where an edge exists between each node of the same class.
- $q \to 0$: We approach the case where an edge never exists between nodes of different classes.
- $\epsilon \to 0$: We approach the case where we always place two nodes in the same cluster if they are the same class.
- $\delta \to 0$: We approach the case where we never place two nodes in the same cluster if they are in different classes.

Essentially, the ideal case for a margin-loss GNN is $p \to 1$ and $q \to 0$. Conversely, the ideal case for CARL-G is $\epsilon \to 0$, $\delta \to 0$. As we mentioned in Section 3.3, silhouette-based loss relies on the clustering error rate rather than the inherent properties of the graph. We show that a margin-loss GNN is exactly equivalent to a mean-silhouette-loss GNN under the above conditions; however, it also follows that some equivalence can also be drawn between them for different non-ideal values of p, q, ϵ, and δ, but we feel such analysis is out of the scope of this work.

A.3 Additional Experiment Details

We ran our experiments on a combination of local and cloud resources. All non-timing experiments were run on an NVIDIA RTX A4000 or V100 GPU, both with 16 GB of VRAM. All timing experiments were conducted on a Google Cloud Platform (GCP) instance with 12 CPU Intel Skylake cores, 64 GB of RAM, and a 16 GB V100 GPU. Accuracy means and standard deviations are computed by re-training the classifier on 5 different splits. The code and exact hyperparameters for this paper can be found online at https://github.com/willshiao/carl-g.

Method	Dataset	Max GPU Memory	Mean CPU Memory	Training Time	Layer Sizes
AFGRL	Amazon-Computers	2,637	2,671	8,311.94	[512]
	Amazon-Photos	1,221	2,615	4,659.91	[512]
	Coauthor-CS	5,537	3,038	19,533.74	[1024]
	Wiki-CS	4,177	2,647	16,185.56	[1024]
BGRL	Amazon-Computers	1,081	2,289	387.03	[256,128]
	Amazon-Photos	615	2,267	284.29	[256,128]
	Coauthor-CS	2,637	2,722	1,032.14	[512,256]
	Coauthor-Physics	4,769	3,362	1,270.93	[256,128]
	Wiki-CS	1,877	2,240	720.37	[512,256]
CARL-G_{SIM}	Amazon-Computers	2,100	2,910	8.97	[512,256]
	Amazon-Photos	1,032	2,875	9.29	[512,256]
	Coauthor-CS	1,325	3,352	13.07	[512,256]
	Coauthor-Physics	3,405	3,998	27.11	[512,256]
	Wiki-CS	1,816	2,857	15.64	[512,256]
CARL-G_{SIL}	Amazon-Computers	2,100	3,435	172.26	[512,256]
	Amazon-Photos	1,032	3,320	30.13	[512,256]
	Coauthor-CS	4,682	4,273	832.25	[512,256]
	Coauthor-Physics	10,074	4,882	524.27	[512,256]
	Wiki-CS	1,816	3,392	217.63	[512,256]
CARL-G_{VRC}	Amazon-Computers	2,100	2,875	9.15	[512,256]
	Amazon-Photos	1,032	2,843	6.04	[512,256]
	Coauthor-CS	1,325	3,320	7.57	[512,256]
	Coauthor-Physics	3,405	3,964	17.22	[512,256]
	Wiki-CS	1,816	2,826	4.08	[512,256]

Table 7: Performance of various methods.

A.4 Dataset Statistics

Dataset	Nodes	Edges	Features	Classes
Wiki-CS	11,701	216,123	300	10
Coauthor-CS	18,333	163,788	6,805	15
Coauthor-Physics	34,493	495,924	8,415	5
Amazon-Computers	13,752	491,722	767	10
Amazon-Photos	7,650	238,162	745	8

Table 8: Statistics for the datasets used in our work.

A.5 Training Time

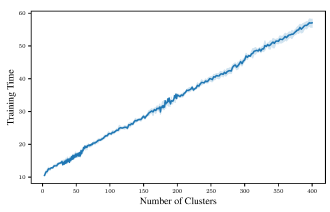


Figure 6: Training time versus number of clusters for CARL-G_{SIM} on Coauthor-Physics. As expected (see Section 3.1), the training time is linear with respect to the number of clusters.