UGSL: A Unified Framework for Benchmarking Graph Structure Learning

Bahare Fatemi [1] **Sami Abu-El-Haija** [1] **Anton Tsitsulin** [1] **Mehran Kazemi** [1] **Dustin Zelle** [1] **Neslihan Bulut** [1] **Jonathan Halcrow** [1] **Bryan Perozzi** [1]

Abstract

Graph neural networks (GNNs) demonstrate outstanding performance in a broad range of applications. While the majority of GNN applications assume that a graph structure is given, some recent methods substantially expanded the applicability of GNNs by showing that they may be effective even when no graph structure is explicitly provided. The GNN parameters and a graph structure are jointly learned. Previous studies adopt different experimentation setups, making it difficult to compare their merits. In this paper, we propose a benchmarking strategy for graph structure learning using a unified framework. Our framework, called Unified Graph Structure Learning (UGSL), reformulates existing models into a single model. We conduct extensive analyses using our proposed framework. Our results provide a clear and concise understanding of the different methods in this area as well as their strengths and weaknesses.

1. Introduction

Graph Representation Learning (GRL) is a rapidly-growing field applicable in domains where data can be represented as a graph (Chami et al., 2022). The allure of GRL models is both obvious and well deserved – there are many examples in the literature where graph structure information can increase task performance (Abu-El-Haija et al., 2019). However, recent results show that the success of graph-aware machine learning models, such as Graph Neural Networks (GNNs), is limited by the quality of the input graph structure (Palowitch et al., 2022). In fact, when the graph structure does not provide an appropriate inductive bias for the task, GRL methods can perform worse than similar models without graph information (Chami et al., 2022).

As a result, the field of Graph Structure Learning (GSL) has emerged to investigate the design and creation of optimal graph structures to aid in graph representation learning tasks. The relational biases found through GSL typically use multiple different sources of information and can offer significant improvements over the kinds of 'in vivo' graph structure found by measuring a single real-world process (*e.g.*, friend formation in a social network). To this end, GSL has been found to be especially important in real-world settings where the observed graph structure might be noisy, incomplete, or even unavailable.

In this paper, we aim to provide the first holistic examination of Graph Structure Learning. We propose a benchmarking strategy for GSL using a unified framework, which we call Unified Graph Structure Learning (UGSL). The framework reformulates ten existing models into a single architecture, allowing for the first comprehensive comparison of methods. We implement a wide range of existing models in our framework and conduct extensive analyses of the effectiveness of different components in the framework. Our results provide a clear and concise understanding of the different methods in this area as well as their strengths and weaknesses.

Specifically, our contributions are: **UGSL**, our unified framework for benchmarking GSL which encompasses over ten existing methods and four thousand different architectures in the same model. **GSL benchmarking study**, the results of our GSL Benchmarking study, a first-of-its-kind effort that compared over *four thousand architectures* across six different datasets in twenty-two different settings, giving insights into the general effectiveness of the components and architectures. **Open source code**, upon the acceptance of our paper, we will open-source our code. This will allow other researchers to reproduce our results, build on our work, and develop their own GSL models. We believe that open-sourcing our code will accelerate the development of GSL research and lead to new and innovative applications.

2. Preliminaries

2.1. Notation

Lowercase letters (*e.g.*, n) denote scalars. Bold uppercase (*e.g.*, $\boldsymbol{A}$) denotes matrices. Calligraphic letters (*e.g.*, $\mathcal{X}$)

[1]Google Research. Correspondence to: Bahare Fatemi <baharef@google.com>.

Presented at the 2^{nd} Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (TAG-ML) at the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. 2023.

denote sets. Sans-serif (*e.g.*, MyFunc) denotes functions. $\boldsymbol{I}$ is the identity matrix. For a matrix $\boldsymbol{M}$, we represent its i^{th} row as $\boldsymbol{M}_i$ and the element at the i^{th} row and j^{th} column as $\boldsymbol{M}_{ij}$. Further, $\odot$ denotes Hadamard product, $\circ$ denotes function composition, Cos is cosine similarity of the input vectors, σ denotes element-wise non-linearity, $^\top$ a transposition operation, and $||$ as a concatenation operation. We let $|\mathcal{M}|$ represent the number of elements in $\mathcal{M}$ and $||\boldsymbol{M}||_{\mathsf{F}}$ indicate the Frobenius norm of matrix $\boldsymbol{M}$. Finally, $[n] = \{1, 2, \ldots, n\}$.

Let graph $\mathtt{G} = (\boldsymbol{X}, \boldsymbol{A})$ with n nodes, feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, and adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$. Let in-degree diagonal matrix $\overleftarrow{\boldsymbol{D}}$ with $\overleftarrow{\boldsymbol{D}}_{ii}$ counting the in-degrees of node $i \in [n]$, and $\overrightarrow{\boldsymbol{D}}_{ii}$ counting its out-degree. Let $\mathcal{G} = \mathcal{X} \times \mathcal{A}$ denote the space of graphs with n nodes. Let $\mathtt{G}^{(0)} \in \mathcal{G}$ be an **input** graph $\mathtt{G}^{(0)} = (\boldsymbol{X}^{(0)}, \boldsymbol{A}^{(0)})$ with feature matrix $\boldsymbol{X}^{(0)} \in \mathcal{X} \subseteq \mathbb{R}^{n \times d_0}$ and adjacency matrix $\boldsymbol{A}^{(0)} \in \mathcal{A} \subseteq \mathbb{R}^{n \times n}$. In most-cases, $\boldsymbol{X}^{(0)} = \boldsymbol{X}$ (and $d_0 = d$).

2.2. Problem Formulation

The **Graph Structured Learning (GSL) Problem** is defined as follows: *Given* $\mathtt{G}^{(0)}$ *and a task* T *find an adjacency matrix* $\mathbf{A}$ *which provides the best graph inductive bias for* T. Our proposed method UGSL captures functions of the form: $\mathsf{f} : \mathcal{G} \to \mathcal{G}$, where f denotes a graph generator model. Specifically, we are interested in methods that (iteratively) output graph structures as:

$$(\boldsymbol{X}^{(\ell)}, \boldsymbol{A}^{(\ell)}) = \mathsf{f}^{(\theta_\ell, \ell)}(\boldsymbol{X}^{(\ell-1)}, \boldsymbol{A}^{(\ell-1)}) \text{ for } \ell \in [L]$$
$$\text{with } \mathsf{f}^{(\theta)} = \mathsf{f}^{(\theta_L, L)} \circ \cdots \circ \mathsf{f}^{(\theta_2, 2)} \circ \mathsf{f}^{(\theta_1, 1)}$$

Here, $\mathsf{f}^{(\theta_\ell, \ell)}$ represents the ℓ-th graph generator model with parameters θ_l. A variety of methods fit this framework. A class of these methods does not process an adjacency matrix as input. As such, they can be written as $\mathsf{f}^{(\theta)} : \mathcal{X} \to \mathcal{X} \times \mathcal{A}$. Another class does not output node features, *i.e.*, $\mathsf{f}^{(\theta)} : \mathcal{X} \times \mathcal{A} \to \mathcal{A}$. Nonetheless, we only consider f with $\mathcal{A} \in \mathsf{range}(\mathsf{f})$. Specifically, methods that output an adjacency matrix. Here, we are interested in methods where the adjacency matrix produced by f is utilized in a downstream model, which can be trained in a supervised, unsupervised, or self-supervised, end-to-end manner.

3. UGSL: A Unified Framework for Graph Structure Learning Models

The objective of this section is to present a comprehensive unified framework for models designed for graph structure learning. We first describe our proposed unified framework using UGSL layers, a general layer for graph structure learning. The ℓ-th UGSL layer defines function $\mathsf{f}^{(\ell, \theta^\ell)}$ as:

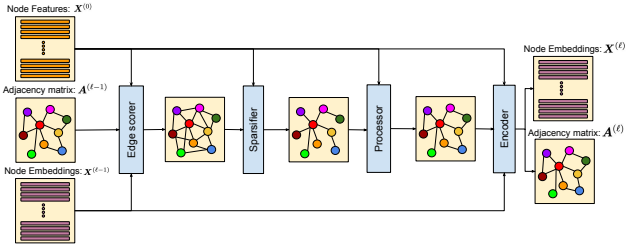


Figure 1: Overview of the ℓ-th GSL layer, $\mathsf{f}^{(\ell, \theta^\ell)}$.

$$\mathsf{f}^{(\ell, \theta^\ell)} = \mathsf{Encoder}^{(\theta^\ell_{\mathsf{E}})} \circ \mathsf{Processor}^{(\theta^\ell_{\mathsf{P}})} \circ \mathsf{Sparsifier}^{(\theta^\ell_{\mathsf{S}})} \circ \mathsf{EdgeScorer}^{(\theta^\ell_{\mathsf{ES}})}, \quad (1)$$

the composition of 4 trainable modules. Multiple UGSL layers can be combined to create a UGSL model as in Equation 1. Next, we summarize the role of each module. Then, we show that many methods can be cast into the UGSL framework by specifying these modules (**??**) .

* Input: Each UGSL layer ℓ takes as input the output graph of the $(\ell-1)$-th layer $\mathtt{G}^{(\ell-1)} = (\boldsymbol{X}^{(\ell-1)}, \boldsymbol{A}^{(\ell-1)})$, and the input graph $\mathtt{G}^{(0)} = (\boldsymbol{X}^{(0)}, \boldsymbol{A}^{(0)})$

* EdgeScorer (w. parameters θ_{ES}) scores every node-pair, producing output $\in \mathbb{R}^{n \times n}$.

$$\boldsymbol{A}^{(\mathsf{ES}, \ell)} = \mathsf{EdgeScorer}(\mathtt{G}^{(0)}, \mathtt{G}^{(\ell-1)}; \theta_{\mathsf{ES}}) \quad (2)$$

* Sparsifier (w. parameters θ_{S}) Sparsifies the graph, e.g., via top-k or thresholding:

$$\boldsymbol{A}^{(\mathsf{S}, \ell)} = \mathsf{Sparsifier}(\mathtt{G}^{(0)}, \mathtt{G}^{(\ell-1)}, \boldsymbol{A}^{(\mathsf{ES}, \ell)}; \theta_{\mathsf{S}}) \quad (3)$$

* Processor (w. parameters θ_{P}) takes the output of the sparsifier and output a processed graph as:

$$\boldsymbol{A}^{(\mathsf{P}, \ell)} = \mathsf{Processor}(\mathtt{G}^{(0)}, \mathtt{G}^{(\ell-1)}, \boldsymbol{A}^{(\mathsf{S}, \ell)}; \theta_{\mathsf{P}}) \quad (4)$$

* Encoder (w. parameters θ_{E}) generates updated node embeddings $\boldsymbol{A}^{(\mathsf{E}, \ell)}$ as:

$$(\boldsymbol{A}^{(\mathsf{E}, \ell)}, \boldsymbol{X}^{(\ell)}) = \mathsf{Encoder}(\mathtt{G}^{(0)}, \mathtt{G}^{(\ell-1)}, \boldsymbol{A}^{(\mathsf{P}, \ell)}; \theta_{\mathsf{E}}) \quad (5)$$

* Output: The above modules are invoked for $\ell \in [L]$ giving final UGSL output of:

$$\mathtt{G}^{(L)} = (\boldsymbol{A}^{(\mathsf{E}, L)}, \boldsymbol{X}^{(L)}). \quad (6)$$

4. Experimental Design

This section explores the key components of different UGSL modules and experiments with them across various datasets.

Edge scorer	Formula	Description
FP	$\boldsymbol{A}_{ij}^{(\mathsf{ES},\ell)} = \boldsymbol{V}_{ij}^{(\mathsf{ES})}$	Each possible edge in the graph has a separate parameter learned directly. Initialization $\boldsymbol{V}_{ij}^{(\mathsf{ES})} = \mathsf{Cos}(\boldsymbol{X}_i^{(0)}, \boldsymbol{X}_j^{(0)})$ (cosine similarity of features) is significantly better than random.
ATT	$\boldsymbol{A}_{ij}^{(\mathsf{ES},\ell)} = \frac{1}{m}\sum_{p=1}^{m} \mathsf{Cos}(\boldsymbol{X}_i^{(l-1)} \odot \boldsymbol{V}_p^{(\mathsf{ES})}, \boldsymbol{X}_j^{(l-1)} \odot \boldsymbol{V}_p^{(\mathsf{ES})})$	Learning a multi-head version of a weighted cosine similarity.
MLP	$\boldsymbol{A}_{ij}^{(\mathsf{ES},\ell)} = \mathsf{Cos}(\mathsf{MLP}(\boldsymbol{X}_i^{(l-1)}), \mathsf{MLP}(\boldsymbol{X}_j^{(l-1)}))$	A cosine similarity function on the output of an MLP model on the input.

Table 1: An overview of different edge scorers.

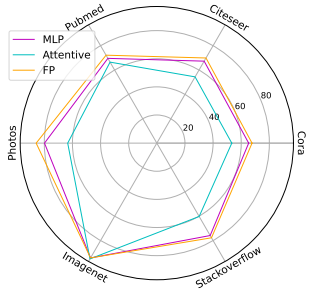


Figure 2: Edge scorers.

Sparsifier	Description
kNN	Define $N(i) = \arg\mathrm{sort}_j \boldsymbol{A}_{ij}^{(\mathsf{ES},\ell)}$. Then, for each node, keep the top k edges with the highest weights.
d-kNN	Dilated convolutions (Yu & Koltun, 2015) were introduced in the context of graph learning by (Li et al., 2019) to build graphs for very deep graph networks. We adapt dilated nearest neighbor operator to GSL. d-kNN adds a dilation with the rate d to kNN to increase the receptive field of each node.
ϵNN	Only keeping the edges with weights greater than ϵ.
Bernoulli	Applying a relaxation of Bernoulli and then keeping the edges with weights greater than ϵ.

Table 2: An overview of different sparsifiers.

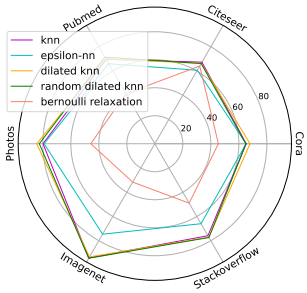


Figure 3: Sparsifiers.

Base model. A base model is a UGSL model used as a reference for comparisons. The objective of such a model is to be minimal and have popular components. Our base model uses only raw features in the input, has an MLP as edge scorer, k-nearest neighbors as sparsifier, no processor, and a GCN as an encoder. The base model is trained with a supervised classification loss. The components of the base model will be explained as we explore different options. In the rest of this section, we explain different components of the framework and experiment with them. For analyzing different components, we consider the base model and only change the corresponding component to be able to measure the effectiveness of one component at a time.

Edge scorer. Most edge scorers in the literature are variants of one of the following: Full parameterization (FP), Attentive (ATT), and multilayer perceptron (MLP). Table 1 shows a summary of edge scorers. Figure 2 compares the edge scorers across the datasets. MLP and FP outperform ATT across the six datasets. FP is the general winner of the three. However, as FP learns one single parameter for each edge, it does not scale to large graphs and is not applicable to an inductive setup with a growing graph at inference time. MLP performs competitively with FP on most of the datasets and has the advantage that it can scale to larger datasets and to the inductive setup.

Sparsifier. Sparsifiers take a dense graph generated by the edge scorer and return a sparse version of that. Here, we experiment with the following sparsifiers: k-nearest neighbors (kNN), dilated k-nearest neighbors (d-kNN), ϵ-nearest neighbors (ϵNN), and the concrete relaxation of the Bernoulli distribution (Bernoulli). Table 2 contains a summary of these methods. In the case of ϵNN, since the graph is not fixed and the weights are being learned, the number of edges whose weight surpasses ϵ may be large, and so running ϵNN on accelerators may cause out-of-memory (OOM) error. To avoid this issue, we tried ϵNN on an extensive memory setup. The results comparing the sparsifiers are summarized in Figure 3. The relaxation of Bernoulli does not generalize well to the test set due to its large fluctuation of loss at train time. The kNN variants outperform ϵNN on our datasets. Among the kNN variants, dilated kNN works slightly better than the other variants.

Processor. Many works in the literature have explored the use of different forms of processing on the output of sparsifiers. These processing techniques can be broadly classified into three categories: (1) applying non-linearities on the edge weights (e.g., to remove negative weights), (2) symmetrizing the output of sparsifiers, and (3) applying both non-linearity and symmetriz functions. The processors we used are listed in Table 3 and the results of applying these processors are shown in Figure 4. According to the results, except in the case of Citeseer, both activation and symmetrization help improve the results, and their combination performs best in several of the datasets.

Encoder. We experiment with GCN (Kipf & Welling, 2017) and GIN (Xu et al., 2019) as encoder layers that can incorporate the learned graphs. As a baseline, we also add an MLP model that ignores the generated graph and only uses the features to make predictions. The results comparing

Processor	Formula	Description
symmetrize	$\boldsymbol{A}_{ij}^{(\mathrm{P},\ell)} = \frac{\boldsymbol{A}_{ij}^{(\mathrm{S},\ell)} + \boldsymbol{A}_{ji}^{(\mathrm{S},\ell)}}{2}$	Making a symmetric version from the output of sparsifier.
activation	$\boldsymbol{A}_{ij}^{(\mathrm{P},\ell)} = \sigma(\boldsymbol{A}_{ij}^{(\mathrm{S},\ell)})$	A non-linear function σ is applied on the output of sparsifier.
activation-symmetrize	$\boldsymbol{A}_{ij}^{(\mathrm{P},\ell)} = \frac{\sigma(\boldsymbol{A}_{ij}^{(\mathrm{S},\ell)}) + \sigma(\boldsymbol{A}_{ji}^{(\mathrm{S},\ell)})}{2}$	First applying a non-linear transformation and then symmetrizing the output.

Table 3: An overview of different processors.

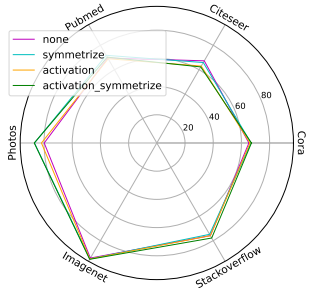


Figure 4: Processors.

the encoders are shown in Figure 5. On some of the datasets, the GNN variants GCN and GIN outperform MLP by a large margin. In some other datasets, MLP performs on par with the UGSL models. On those with a strong MLP baseline, GIN outperforms GCN, which shows the importance of self-loops in these classification tasks.

References

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Steeg, G. V., and Galstyan, A. MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *ICML 2019*, volume 97 of *Proceedings of Machine Learning Research*, pp. 21–29. PMLR, 09–15 Jun 2019.

Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., and Murphy, K. Machine learning on graphs: A model and comprehensive taxonomy. *JMLR*, 23(89):1–64, 2022.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

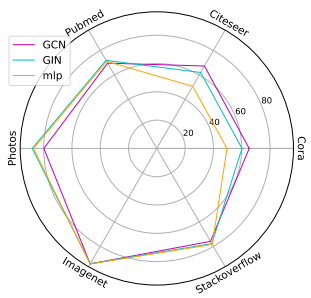


Figure 5: Encoders.

Fatemi, B., El Asri, L., and Kazemi, M. Slaps: Self-supervision improves structure learning for graph neural networks. In *NeurIPS 2021*, volume 34, pp. 22667–22681, 2021.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Li, G., Muller, M., Thabet, A., and Ghanem, B. Deepgcns: Can gcns go as deep as cnns? In *CVPR*, pp. 9267–9276, 2019.

McAuley, J., Targett, C., Shi, Q., and Van Den Hengel, A. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pp. 43–52, 2015.

Palowitch, J., Tsitsulin, A., Mayer, B., and Perozzi, B. Graphworld: Fake graphs bring real insights for gnns. In *KDD*, KDD '22, pp. 3691–3701, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393850. doi: 10.1145/3534678.3539203. URL https://doi.org/10.1145/3534678.3539203.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Xu, J., Xu, B., Wang, P., Zheng, S., Tian, G., and Zhao, J. Self-taught convolutional neural networks for short text clustering. *Neural Networks*, 88:22–31, 2017.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *ICLR*, 2019.

Yu, F. and Koltun, V. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

Zhang, J., Zhang, H., Xia, C., and Sun, L. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.

A. Experimental Design

Datasets. To benchmark using our proposed UGSL framework, we experiment across six different datasets from various domains. The first class of datasets consists of the three established benchmarks in the GNN literature namely Cora, Citeseer, and Pubmed (Sen et al., 2008). Another dataset is Amazon Photos (or Photos for brevity) (Shchur et al., 2018) which is a segment of the Amazon co-purchase graph (McAuley et al., 2015). The other two datasets are used extensively for classification with no structure known in advance. One is an Image classification dataset Imagenet-20 (or Imagenet for brevity), a subset of Imagenet containing only 20 classes (totalling 10,000 train examples). We used pre-trained Vision Transformer feature extractor (Dosovitskiy et al., 2021). The last dataset is a text classification dataset Stackoverflow (Xu et al., 2017).

Implementation. The framework and all components are implemented in Tensorflow using the TF-GNN library for learning with graphs.

A.1. Input and Loss Components

A.1.1. Input.

To make our proposed framework applicable to a wide range of applications and conduct coherent experiments on multiple datasets, we assume $\boldsymbol{A}^{(0)}$ is an empty matrix. When a non-empty $\boldsymbol{A}^{(0)}$ is required (*e.g.*, for computing positional encodings), we create a kNN graph from the raw features $\boldsymbol{X}$. For $\boldsymbol{X}^{(0)}$, many models assume $\boldsymbol{X}^{(0)} = \boldsymbol{X}$, *i.e.*, the input embeddings are simply the given node features. However, in some architectures, $\boldsymbol{X}^{(0)}$ is defined differently to contain information from the input adjacency matrix $\boldsymbol{A}^{(0)}$ too. Here, we explore two approaches one based on positional encodings of Weisfeiler-Lehman (WL) absolute role of the nodes (Zhang et al., 2020) and another based on positional encodings of top k eigenvectors of the graph Laplacian (Zhang et al., 2020). We explain the two variants in Table 4 and compare their results in Figure 6(a). As the results show, adding WL and spectral roles to the raw features does not yield more effective results compared to using raw features across the six datasets. This is even valid on datasets with less expressive raw features (*e.g.*, bag-of-words in Cora and Citeseer).

A.1.2. Loss Functions and Regularizers

In our framework, we experiment with a supervised classification loss, four different regularizers, and two unsupervised loss functions. See Table 5 for a summary of descriptions. Figure 6(b) compares the regularizers. The results show that the log-barrier regularizer alone does not achieve effective results and this is mainly because this regularizer alone only encourages a denser adjacency matrix. The sparse-connect regularizer outperforms the rest of the regularizers with a small margin almost on all datasets. Figure 6(c) compares the two unsupervised losses in the UGSL framework. The contrastive loss is the most effective across most of the datasets. The unsupervised losses are one of those components that increased the performance of the base model the most which might mainly be because of the supervision starvation problem studied in the GSL literature (Fatemi et al., 2021).

A.2. Random Search Over All Components

In this section, we perform a random search over all components of the UGSL framework, including their combinations and hyperparameters. For each dataset, we run 30,000 trials.

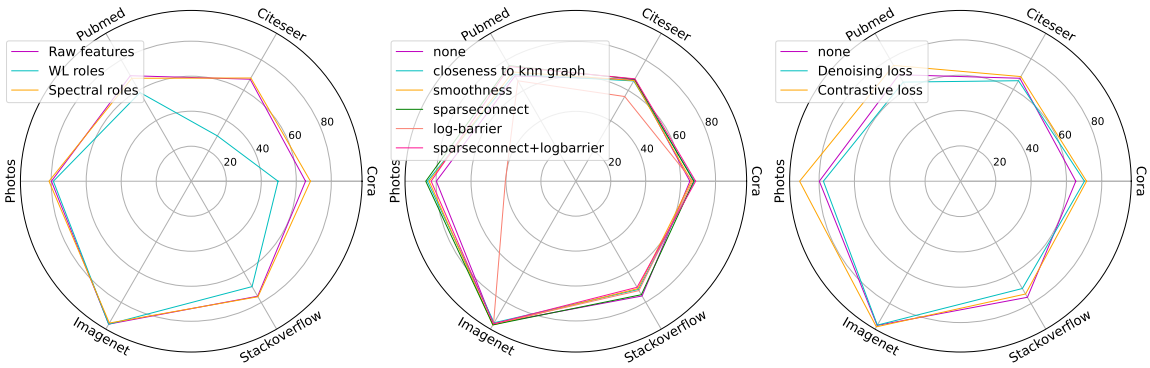
Best results obtained. The trials with the best validation accuracy are reported in Table 6, along with the corresponding test accuracy and the components in the architecture of the corresponding model. The results show that combining different combinations from different models further improves the base model. As we are running many trials for each dataset, we excluded the sparsifiers with ϵ variants to be able to run all trials on accelerators. We can see that the best-performing

Positional encoding	Explanation
WL role	Positional encoding based on the Weisfeiler-Lehman absolute role (Shervashidze et al., 2011).
Spectral role	Positional encoding based on the top k eigenvectors of the graph laplacian.

Table 4: An Overview of Different Positional Encodings.

Name	Formula	Description
Supervised loss	$\sum_i \mathsf{CE}(\text{label}_i, \text{pred}_i)$	The supervision from the classification task as a categorical cross-entropy (CE represents the cross-entropy loss and the sum is over labeled nodes).
Closeness	$\|\|\boldsymbol{A}^{(0)} - \boldsymbol{A}\|\|_F^2$	Discourages deviating from the initial graph.
Smoothness	$\sum_{i,j} \boldsymbol{A}_{ij} \mathsf{dist}(v_i, v_j)$	Discourages connecting (or putting a high weight on) pairs of nodes with dissimilar initial features.
sparse-connect	$\|\|\boldsymbol{A}\|\|_F^2$	Discourages large edge weights.
Log-Barrier	$-\mathbf{1}^T \log(\boldsymbol{A}\mathbf{1})$	Discourages low-degree nodes, with an infinite penalty for singleton nodes.
Denoising Auto-Encoder	$\sum_{i,j \in \mathcal{F}} \mathsf{CE}(\boldsymbol{X}_{ij}, \mathsf{GNN}_{\mathsf{DAE}}(\tilde{\boldsymbol{X}}, \boldsymbol{A}))$	Selects a subset of the node features $\mathcal{F}$, adds noise to them to create $\tilde{\boldsymbol{X}}$, and then trains a separate GNN to denoise $\tilde{\boldsymbol{X}}$ based on the learned graph.
Contrastive	$\frac{1}{2n}\big(\sum_i^n \log \frac{\left(\exp(\mathsf{sim}(\boldsymbol{X}_i^L, \boldsymbol{Y}_i^L)/\tau)\right)}{\sum_{j=1}^n \left(\exp(\mathsf{sim}(\boldsymbol{X}_i^L, \boldsymbol{Y}_j^L)/\tau)\right)} + \log \frac{\left(\exp(\mathsf{sim}(\boldsymbol{Y}_i^L, \boldsymbol{X}_i^L)/\tau)\right)}{\sum_{j=1}^n \left(\exp(\mathsf{sim}(\boldsymbol{Y}_i^L, \boldsymbol{X}_j^L)/\tau)\right)}\big)$	Let $\mathtt{G}_1 = (\boldsymbol{X}, \boldsymbol{A})$ and $\mathtt{G}_2 = (\boldsymbol{X}, \mathsf{combine}(\boldsymbol{A}^{(0)}, \boldsymbol{A}))$ ($\boldsymbol{A}$ is the learned structure and $\boldsymbol{A}^{(0)}$ is the initial – $\boldsymbol{I}$ if no initial structure). The combine function is a slow-moving weighted sum of the original and the learned graphs. Then, Let $\tilde{\mathtt{G}_1}$ and $\tilde{\mathtt{G}_2}$ be variants of $\mathtt{G}_1$ and $\mathtt{G}_2$ with noise added to the graph and features. The two views are fed into a GNN followed by an MLP to obtain node features $\boldsymbol{X}^{(L)}$ and $\boldsymbol{Y}^{(L)}$, on which the loss is computed.

Table 5: A summary of loss functions (supervised and unsupervised) and regularizers.



(a) Input: positional encodings (b) Regularizers. (c) Unsupervised loss functions.

Figure 6: my caption

components vary across datasets. However, there are some general trends. For example, using raw features in the input with no positional encoding is frequently used in the top-performing models. Also, the ATT edge scorer is accompanied by a GIN encoder in the architectures where it participated.

Top-performing components. To get more insights from the different trials, in addition to the best results reported above, we analyze the top 5% performing trials for each component and visualize their results. Figures 7 to 12 show the box charts for different components across for Pubmed, Photo, Imagenet, and Stackoverflow (the rest in the appendix).

Dataset	Val Accuracy	Test Accuracy	Input Features	Edge scorer	Sparsifier	Processor	Encoder	Regularizers	Unsupervised Losses
Cora (base)	68.20	65.30	features	MLP	kNN	none	GCN	none	none
Cora (best)	70.80	72.30	features	FP	d-kNN	activation	GCN	none	denoising and contrastive
Citeseer (base)	69.20	67.30	features	MLP	kNN	none	GCN	none	none
Citeseer (best)	72.00	71.20	features	FP	kNN	activation-sym	GCN	closeness	none
Pubmed (base)	72.60	69.60	features	MLP	kNN	none	GCN	none	none
Pubmed (best)	80.80	76.00	features	MLP	d-kNN	activation	GIN	sparse-connect	contrastive
Photo (base)	80.81	79.87	features	MLP	kNN	none	GCN	none	none
Photo (best)	92.55	89.84	spectral	ATT	d-kNN	activation	GIN	sparse-connect	denoising and contrastive
Imagenet (base)	96.80	94.25	features	MLP	kNN	none	GCN	none	none
Imagenet (best)	96.95	94.25	features	FP	kNN	activation	GIN	smoothness	none
Stackoverflow (base)	76.28	75.86	features	MLP	kNN	none	GCN	none	none
Stackoverflow (best)	77.48	77.82	features	ATT	kNN	none	GIN	sparse-connect	none

Table 6: Comparison of best random search models vs. base models

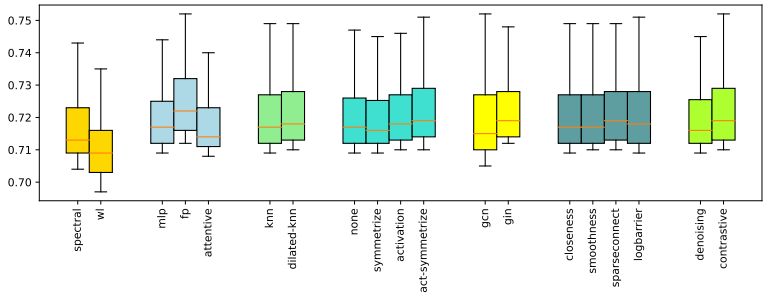


Figure 7: Results of the top 5% performing UGSL models in a random search on Pubmed.

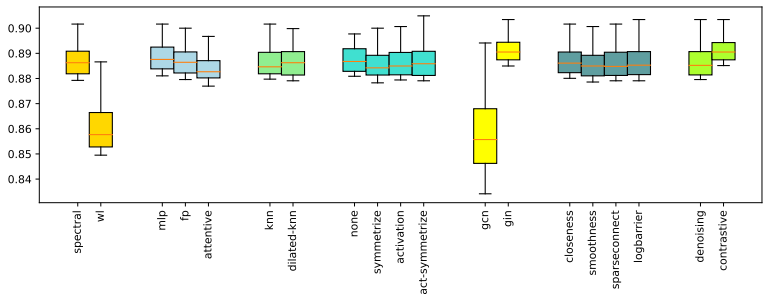


Figure 8: Results of the top 5% performing UGSL models in a random search on Photo.

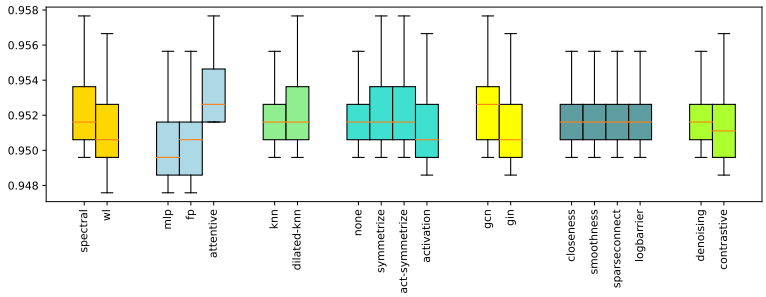


Figure 9: Results of the top 5% performing UGSL models in a random search on Imagenet.

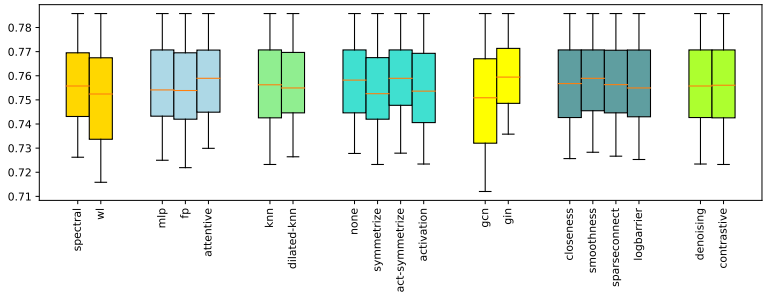


Figure 10: Results of the top 5% performing UGSL models in a random search on Stackoverflow.

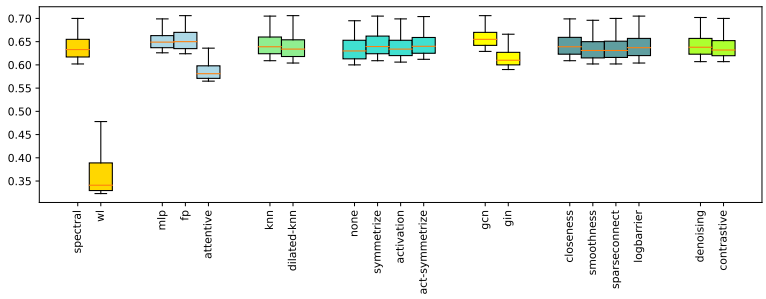


Figure 11: Results of the top 5% performing UGSL models in a random search on Cora.

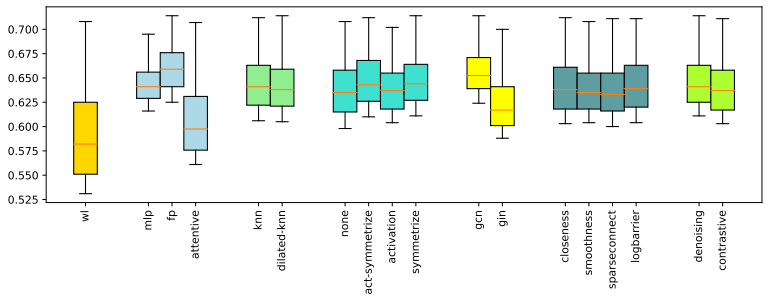


Figure 12: Results of the top 5% performing UGSL models in a random search on Citeseer.