Appendices of:
Violin: Virtual Overbridge Linking for Enhancing Semi-supervised Learning on Graphs with Limited Labels

Siyue Xie , Da Sun Handason Tam , Wing Cheong Lau
The Chinese University of Hong Kong
{xs019, tds019, wclau}@ie.cuhk.edu.hk,

A More Discussions on Violin

In this part, we review more related literatures and compare them with our proposed framework Violin.

A.1 Comparisons with Graph Contrastive Learning Algorithms

Graph contrastive learning (GCL) is found effective for graph unsupervised/ self-supervised learning in recent researches. In general, GCL models usually include an encoder to generate node/ graph representations and a discriminator to distinguish different kinds of representations. Representations with the same semantics or highly-correlated are paired as positive samples, while unrelated representations are paired as negative samples. By training the discriminator to distinguish positive pairs from the negatives, the encoder can learn to generate a general embedding for each node without labels.

A key step of GCL models is to construct positive and negative samples. In most cases, contrastive samples are generated based on handcrafted graph augmentations. In Deep Graph Infomax (DGI) [Veličković *et al.*, 2018], a node representation (local view) and the graph representation (global view) learned from the original graph are paired as a positive sample. A node representation learned from a corrupted graph is paired with the graph representation for a negative sample. The model is trained by maximizing (minimizing) the mutual information between the positive (negative) global and local views. MVGRL [Hassani and Khasahmadi, 2020] follows the same idea of DGI by optimizing the mutual information of the global-local view. However, it uses a graph diffusion method to generate multi-view augmented graphs and contrasts among different views. GRACE [Zhu *et al.*, 2020] and GCA [Zhu *et al.*, 2021] generate augmented graphs by applying feature masking and edge masking. Their models are learned by contrasting nodes both in the same graph and across different augmented graphs. Graph augmentations methods used in BGRL [Thakoor *et al.*, 2021] are similar to that of GRACE. However, the model is trained only based on positive pairs generated by a online encoder and a target encoder.

Although GCL methods make great progress on unsupervised/ self-supervised graph learning, there are still some common drawbacks of GCL methods. For example, the performance of GCL methods heavily relies on the quality of graph augmentations, which are usually determined by human prior knowledge. However, it is not easy for experts to design an augmentation technique that can be generalized to all graphs. In addition, to apply the learned node embeddings to downstream tasks, we usually need to train another model separately, e.g., an individual classifier for node classification tasks. Such a two-stage paradigm disable GCL models to make use of the feedback information from the downstream tasks during the first stage's training.

In practice, the semantics-consistent graph of Violin can also be viewed as an augmentation of the original graph. However, different from many GCL methods that rely on handcrafted graph augmentations, such as random edge add/ drop, the way to build virtual overbrdiges (VOs) for the semantic-consistent graph is data-driven as the labels of the two end-points are determined by the prediction results. Furthermore, Violin always works in an end-to-end manner. The model can therefore learn to purposefully adjust itself based on the feedback from the downstream task, which guides it to adaptively modify the graph augmentation process and results in generating more task-oriented node representations. Such advantages make Violin outperform previous methods on GCL.

A.2 Comparisons with Self-training Methods

Self-training is a method demonstrated to be effective to train models when labels are limited. Generally, each unlabeled instance will be assigned a pseudo-label, which is estimated through some techniques, e.g., by the predictions of a pretrained model. A set of confident unlabeled instances accompanied with their pseudo-labels will be merged into the labeled set, which is used to train the model in a supervised manner.

There are some previous methods that apply self-training techniques to graph semi-supervised learning. Li *et al.* [Li *et al.*, 2018] propose a framework to improve GCN by introducing self-training. A pretrained GCN model is used to generate estimated labels for all nodes and a set of confident unlabeled nodes are collected to extend the labeled set. Furthermore, they additionally introduce a random walk-based method to collect confident unlabeled nodes around the labeled instances. The two sets of collected unlabeled nodes will be used to update the training set to help train GCN. M3S [Sun *et al.*, 2020] proposes a multi-stage self-training mecha-

nism by jointing with K-means clustering. In M3S, by applying the K-means algorithm, each unlabeled node is assigned to a unique cluster, which aligns with one of the classes of the classification task. A confident unlabeled node will then be added to the training set only when its estimated pseudo-label is consistent with its cluster label.

However, it should be noted that there usually be noisy estimated labels even for highly-confident nodes. Noises will inevitably be introduced into the training stage, which may mislead the learning process. In addition, when updating the training/ labeled nodes set, previous methods usually follow some naïve mechanisms based on confidence value, e.g., pick the top-K most confident nodes. However, the selected nodes may not be helpful as some of them can be the so-called easy samples that are less informative to improving the training. The model can also be biased if selected nodes are mostly from the same class, which happens in the cases when a specific class is more distinguishable than others.

In Violin, we also make use of the estimated labels provided by a trained model. However, instead of directly taking the estimated labels as ground-truth for training, Violin learns from estimated labels indirectly by adding VOs to encourage the model to collect information from those trusted unlabeled nodes. Therefore, the supervised signals from the training set are guaranteed to be always true but simultaneously the model is able to learn from more nodes that are unobserved for the pretrained model. This makes Violin more robust to noises (i.e., wrong predictions) but capable of learning more general patterns from the entire graph. In addition, we propose an adaptive thresholding mechanism for picking nodes to add VOs. Any node expected to be correctly predicted has chances to serve for the model, even if its predicted labels are not with high-confidence. As observed in the work of [Wang *et al.*, 2021], common GNN models usually yield under-estimated predictions. With the adaptive thresholding mechanism, Violin is able to learn from more diverse nodes at different confidence levels. Such an operation is especially effective in the beginning stage of the training where the predictions of most nodes are under-estimated. Violin is therefore more likely to learn from those hard examples that are more informative for training, which results in generating more discriminative representations for nodes.

A.3 Comparisons with Enhancement Techniques for Graph Semi-supervised Learning

Considering that GNN models can only learn limited knowledge from a few-labeled graph under semi-supervised scenarios, many previous works attempt to enhance GNNs by various techniques.

Some works resort to integrating semi-supervised GNNs with contrastive learning. For example, CG^3 [Wan *et al.*, 2020] introduces a hierarchical graph convolution branch to generate a 'global view' representation of the graph for each node. The node representations learned from the original graph can be viewed as a 'local view'. A unsupervised regularization term can then be formed to contrast between two nodes from two different views. CoCoS [Xie *et al.*, 2022] creates a context-consistent graph by shuffling node features within the same class of nodes. Contrastive pairs of the positive and negative can be formed for nodes in the original and the context-consistent graph, which derives a regularization term for semi-supervised training. Some of the works make effort to design sophisticated graph augmentations to enhance semi-supervised learning. GRAND [Feng *et al.*, 2020] generates multiple graph augmentations by randomly dropping nodes and applying random feature propagations. NodeAug [Wang *et al.*, 2020b] designs specific node attributes replacement, edge removal and edge adding strategies to the local neighborhood of each node, which yields an individual augmented subgraph for each node. A consistency loss can therefore be derived by regularizing between the node representations from the original graph and the augmented graph(s). C&S [Huang *et al.*, 2020] proposes a two stage enhancement framework for semi-supervised graph learning. In the first stage, a model is trained on the partially-labeled graph to generate label predictions for all nodes. In the second stage, it elaborately propagates 'error correlations' and 'prediction correlations' in the graph, which corrects the predictions in the first stage.

Motivated by previous methods on learning enhancement, in Violin, we also adopt the consistent learning paradigm and techniques equivalent to graph augmentations to improve the learning of common GNNs. However, Violin are different from them in many perspectives. Compared with previous methods resorting to contrastive learning or consistent training, the augmented graphs derived by Violin are totally data-driven based on task-related labels instead of being manually designed. Compared with C&S, our framework can be trained in an end-to-end manner in both the two learning stages (the pretraining stage of the vanilla GNN and the fine-tuning stage by applying Violin to the vanilla GNN). In Violin, the estimated labels of each node can be updated as the training goes, while C&S fixes the estimated labels in its propagation stage. This allows Violin to correct itself on time in the training process. With these properties, models equipped with Violin can therefore be more task-oriented, resulting in generating more discriminative representations for downstream tasks.

However, we should also note that our proposed Violin framework is orthogonal to the above methods. Combining Violin with some of the aforementioned techniques will also be one of our future directions.

A.4 Comparisons with Data-Driven Augmentation Methods on Graph Topology

In the proposed Violin framework, a key step is to learn to add VOs to the original graph so as to generate a semantic-consistent graph. The operation of adding VOs can be regarded as a data-driven augmentation technique targeting on graph topology. Many previous methods also attempt to augment a graph from the view of topology but most of them are subject to heuristics, e.g., random edge-add, edge-drop [Wang *et al.*, 2020b] or subgraph random sampling [Hassani and Khasahmadi, 2020]. Among all, to the best of our knowledge, the works closest to our method are AdaEdge [Chen *et al.*, 2020a] and GAUG [Zhao *et al.*, 2021].

AdaEdge proposes to purposefully modify the graph topology based on a pretrained model's predictions. Following the

self-training paradigm, AdaEdge adds an edge to two confident nodes that predicted to be in the same class, and remove an existed edge from two less-confident nodes that are estimated from different classes. A GNN model will then be trained on the new graph until the prediction results get stable. GAUG adapts the idea from link-prediction task. By learning an edge predictor, GAUG augments a graph by adding/ removing 'missing'/ 'should not exit' edges to/ from the original graph. A link prediction loss derived from the edge predictor is added as a regularization term to the objective function for semi-supervised node classifications.

Similar to the work of AdaEdge [Chen *et al.*, 2020a], Violin modifies the graph topology based on predicted labels. However, AdaEdge still focuses on learning from the few labeled nodes and their neighbors since it still follows the basic graph semi-supervised learning paradigm we specified in Section 3.2 in the main text. In other words, although the graph topology is optimized, most unlabeled nodes are still under-exploited in a few-labeled graph. In contrast, in Violin, we proactively incorporate all available nodes into the learning process by integrating with the consistency regularized training paradigm. Violin can therefore be able to learn more general patterns by observing all nodes. Similar to previous methods, AdaEdge sets a static confidence threshold to pick trusted nodes. The node with higher confidence will more likely to be added more edges. Such an operation may bias the training if the selected nodes are mostly easy samples and tend to overfit to the limited observed high-confidence nodes. Instead, Violin adopts an adaptive thresholding method to select unlabeled nodes, which covers a wider range of confidence levels. Such a strategy encourages Violin to learn from diverse nodes, which can be more robust and effective. Furthermore, AdaEdge needs to be trained from scratch every time the graph topology is changed, while Violin can reuse the pretrained model and fine-tune on it. From the perspective of training efficiency, Violin can also be more computationally-friendly.

GAUG and Violin both attempt to augment a graph from the view of topology, but the motivation and effects are different. GAUG attempts to recover the 'missing edges' in the graph based on a fully unsupervised edge predictor, which is blind to the supervised signals provided in the graph. The edge added by GAUG between two nodes are essentially close in distance/ proximity, but may not be informative/ semantically-related to the downstream classification task. Instead, the VOs added by Violin are driven by supervised signals. The two connected nodes can not only topologically-related but also semantics-consistent from two remote regions. Such characteristics make sure that messages propagated through VOs can be more specific and informative to dealing with the downstream task. In addition, GAUG has to learn an additional edge predictor, while our framework can handle all actions by one single GNN model without adding any learnable module. Violin can therefore be more efficient in terms of memory consumption and computational cost.

#layers	Cora	Citeseer	Pubmed
2	82.76	71.87	77.74
3	80.70	70.01	77.50
4	80.51	67.93	77.01
5	79.30	67.11	77.01

Table 1: The classification accuracy (%) of GAT with different number of GNN layers.

B Empirical Experiments on Deep Attention-Based GNN Models

Neural networks can usually learn better representations from the input by stacking a more deeper model. However, such a trick does not make sense when it comes to GNNs. This is because GNN models usually observe a performance degradation when building a deeper network architecture. Some researchers think the problem is rooted in the phenomenon of over-smoothing, where the representations of different nodes become similar as each other when stacking a deeper GNN [Li *et al.*, 2018; Wu *et al.*, 2020; Rong *et al.*, 2019]. Nodes are hard to be distinguished from each other and therefore degrades the performance in downstream tasks.

In recent years, the attention mechanism [Vaswani *et al.*, 2017] is shown effective in capturing long-distance interactions and also introduced to enhance GNN models [Vaswani *et al.*, 2017; Zhang *et al.*, 2018]. A typical method is the Graph Attention Network (GAT) [Veličković *et al.*, 2017], where attention mechanism is integrated into the aggregation operation of each GNN layer. GAT is therefore expected to distinguish different neighbors of a target by assigning each an importance weight.

In our main text, we propose to extend the receptive field of a GNN model so as to improve the data utilization in a graph with few labeled data. Stacking a deep GNN model can be a straightforward solution to extend receptive field. But as mentioned above, merely stacking multiple layers can lead to over-smoothing, which degrades the performance. One may expect that the attention-based GNNs can be an exception as the attention mechanism helps distinguish different neighbors and capture long-distance interactions. However, we empirically show in the following that the commonly-used attention mechanism can still hardly prevent a GNN model from performance degrading when stacking multiple layers.

We take GAT as the representative. Experiments are conducted on three publicly-recognized datasets (Cora, Citeseer and Pubmed) for evaluating graph semi-supervised learning. We equip GAT with different number of layers to evaluate its performance on node classification task. All other settings follow the work of [Veličković *et al.*, 2017].

Experimental results are shown in Table 1. It can be observed that the classification performance of GAT drops significantly as the model goes deeper. Although GAT adopts the attention mechanism to selectively aggregate information from neighbors, the network is still a hierarchical structure, where error can be cumulated during multiple times of aggregation. In addition, nodes remote from the target node are usually low-correlated with the target node. Stacking multiple layers just enlarges the receptive field omnidirectionally,

model	Cora	Citeseer	Pubmed
GCN	82.52	71.02	79.16
GCN-O, $m = 1$	95.66(+13.14)	89.97(+18.95)	93.20(+14.04)
GCN-O, $m = 2$	98.53(+16.01)	95.94(+24.92)	97.65(+18.59)
GCN-O, $m = 3$	99.39(+16.87)	98.77(+27.75)	98.88(+19.72)

Table 2: The classification accuracy (%) of GCN enhanced by Violin (evaluated in the semantic-consistent graph $\tilde{\mathcal{G}}$).

model	Cora	Citeseer	Pubmed
GCN	82.52	71.02	79.16
GCN-O, $m = 1$	85.62(+3.10)	75.52(+4.50)	81.52(+2.36)
GCN-O, $m = 2$	85.44(+2.92)	76.16(+5.14)	82.47(+3.31)
GCN-O, $m = 3$	85.37(+2.85)	76.64(+5.62)	82.65(+3.49)

Table 3: The classification accuracy (%) of GCN enhanced by Violin (evaluated in the original graph $\mathcal{G}$).

which forces the model to consider more unrelated nodes when making decisions. The attention mechanism may not be affordable since the number of involved nodes grows exponentially when the model goes deeper. On the other hand, in a graph with few labeled nodes, stacking more layers may risk overfitting as the training relies heavily on the labeled data. Therefore, from the observations of such an empirical experiment, we argue that stacking more layers for GNN is not a good solution to learning from more unlabeled nodes. Instead, we propose Violin, which extends the receptive field by adding VOs purposefully so that we can circumvent the problems mentioned above.

C Empirical Studies of Violin with Oracles

To make it clear that when Violin can help and how much it can help for GNN model enhancement, we conduct a series of experiments based on ideal settings. Suppose that we know the oracles of a graph, i.e., the class label of each node. Then based on the oracles, we can always add absolutely correct virtual overbridges (VOs) to the original graph $\mathcal{G}$ for generating semantics-consistent graphs $\tilde{\mathcal{G}}$ for training and evaluations. In the followings, we take GCN [Kipf and Welling, 2016] as a representative to investigate the potential and properties of Violin.

C.1 The Effect of Adding VOs With Oracles

In this part, we investigate the basic power of adding VOs for enhancing a GNN model. Following the notations of Section 3.2 in the main text, let m be the number of VOs added to each node. We can then train and evaluate the GCN model on the semantic-consistent graph $\tilde{\mathcal{G}}$, which we denote it as GCN-O. Note that we randomly choose another intra-class node for a target node to build a VO, we can therefore generate different $\tilde{\mathcal{G}}$ in each training epoch. All settings are the same as that of [Kipf and Welling, 2016] except that we use $\tilde{\mathcal{G}}$ to replace $\mathcal{G}$.

The results are shown in Table 2. The classification accuracy of GCN-O gets a striking improvement over the vanilla GCN by more than 10% in each dataset even if we just add one single VO to each node in $\mathcal{G}$. When assigning three VOs to each node, the prediction accuracy nearly hits the performance upper bound. We owe such a great improvement to the use of semantic-consistent graph. By adding VOs, nodes with similar semantics can share information with each other, even if they are remote from each other. All unlabeled nodes are expected to be incorporated for training, since each node has the chances to be linked to or included in the receptive field of an intra-class labeled node. This is effective for semi-supervised scenarios, where data utilization is usually low if we train the model following the basic learning paradigm mentioned in Section 3.2 in our main text. Such a strategy enables a GNN model to learn more general but task-related patterns from different regions of a graph, which yields more discriminative representations for nodes.

We also conduct another group of experiments, where we still train the GCN model on the semantics-consistent graph $\tilde{\mathcal{G}}$ but evaluate it on the original graph $\mathcal{G}$. This setting fits some real-world cases where sometimes we may not have enough evidence to determine which pair of nodes should be linked by a VO in the test stage. The results are shown in Table 3. Obviously, GCN-O still observes a significant improvement even if it is not allowed to make use of the messages propagated through VOs in the test time. Such a result verifies that GCN-O actually learns to capture more general and discriminative patterns during the training, instead of merely relying on the messages propagated through oracle VOs. This also demonstrates that Violin is promising for enhancing the learning capability of a GNN model.

C.2 Investigations on VOs: Quantity and Quality

We conduct a series of experiments to investigate the characteristics of Violin about VOs with the following concerns:

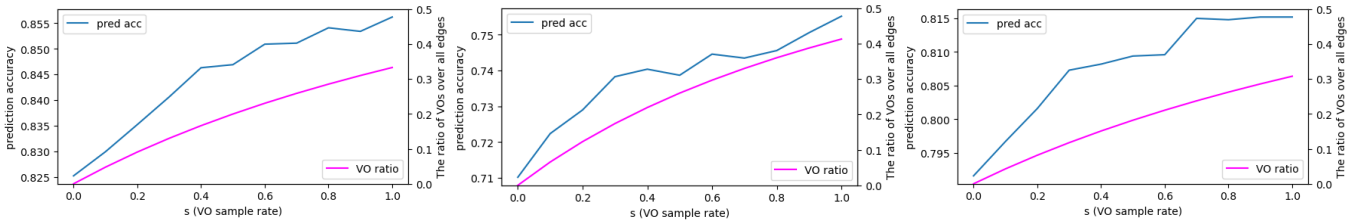
1. How many VOs added to the graph will Violin help?
2. How robust of Violin against noisy VOs?
3. Will Violin still work if the quantity and quality of VOs are both limited?

Since we concentrate on studying the 'limitations' of Violin in this subsection, we only evaluate the trained model on the original graph $\mathcal{G}$, which is a more demanding setting.

The Effect of Adding Different Numbers of VOs

The first question comes from such a concern: if we are unable to find absolutely true VOs for each node to create the semantics-consistent graphs, then how good will Violin be? Therefore, in this experiment, we control the number of VOs added to the original graph. Specifically, we collect only one VO for each node in the graph. For the collected VO set $\tilde{\mathcal{E}}_{vo}$, we will only sample $s|\tilde{\mathcal{E}}_{vo}|$ VOs from it and add them to the original graph $\mathcal{G}$. **Therefore, $s \in [0, 1]$ indicates the proportion of VOs being sampled, or in other words, the proportion of nodes that will be added an VO.** Other settings follow Appendix C.1.

Results are shown in Figure 1. When each node is only allowed to add at most one VO, it is clear that the performance of GCN-O improves as more VOs are added. When 40% (s=0.4) of nodes are added VOs for training, the performance has already been improved by a large margin. At this point, the number of added VOs accounts for around 20% of total edges in the graph. **Therefore, it is not required to add VOs**



(a) GCN-O on Cora. (b) GCN-O on Citeseer. (c) GCN-O on Pubmed.

Figure 1: Evaluations of GCN-O with different values of VO sample rate s on Cora, Citeseer and Pubmed. $s = 0$ indicates the vanilla GCN.

for every node. When only limited numbers of VOs are added, Violin can still work to enhance the vanilla model.

The Influence of the Quality of VOs

One may be curious about the performance of Violin if we add some uninformative VOs to the graph. Therefore, we also investigate the robustness of Violin against noisy VOs. In our experiment, a noisy VO is an VO that the two endpoints of it come from different classes. In fact, any two nodes that are not connected in the original graph and come from different classes can be a candidate of a noisy VO. Let $r \in [0, 1]$ be the proportion of noisy VOs. Similarly, we collect one ground-truth VO for each node to form the VO set $\tilde{\mathcal{E}}_{vo}$. We can therefore control the ratio of noisy VOs by replacing $r|\tilde{\mathcal{E}}_{vo}|$ ground-truth VOs by noisy VOs randomly sampled from the original graph. Then we train GCN on the semantic-consistent graph with noises. Other experimental settings are the same as Appendix C.1.

We show the results of GCN-O with varying r in Figure 2. As expected, the performance of GCN-O drops as the ratio of noisy VOs increases. It is reasonable as noisy VOs are not guaranteed to propagate information related to the target node. Even though, GCN-O still observes a significant improvement over the vanilla GCN when there are 30% (r=0.3) added VOs are incorrect or unexpected. In Cora, Citeseer and Pubmed, GCN-O can afford up to nearly 60%, 70% and 50% noisy VOs, respectively. **Based on such observations, we can expect Violin to be robust to noisy VOs to a certain extent.**

Concerns on Both Quantity and Quality

With the previous two experiments, we can roughly find the 'sweet spot' of Violin. We therefore conduct an experiment to further investigate the potentials of Violin by considering the concerns on both the quantity and quality of VOs. Specifically, we vary s from 0.5 to 1 and r from 0.1 to 0.4, by a step 0.1. Such a setting can be more challenging but helps simulate different conditions of real-world scenarios.

Experimental results can be found in Figure 3. GCN-O still outperforms the vanilla GCN significantly in all cases even if we impose some limitations on the quantity and quality of VOs. **This observation also provides some concrete evidences that Violin is robust in different cases. The different combinations of s and r shown in Figure 3 have already simulated many scenarios that a real-world dataset will be. Therefore, it is promising to apply Violin to real-world settings without oracles.** We also observe that the quality of VOs may take more effect when there are enough VOs added. For example, the performance of GCN-O at $s = 0.9$ is improved marginally compared with the $s = 0.8$ variant, with the same r. However, the performance difference can be more significant among variants with different settings of r but the same s. Such a finding implies that we can pay more attention on selecting qualified VOs when we have enough VO candidates, which motivates us to propose the adaptive confidence thresholding mechanism in our main text.

D Detailed Experimental Settings

D.1 Datasets

Our experiments are conducted on six node classification datasets that are publicly-recognized for graph semi-supervised learning, including Cora [Sen *et al.*, 2008], Citeseer [Sen *et al.*, 2008], Pubmed [Namata *et al.*, 2012], Amazon-Photo (AmzP for short) [Shchur *et al.*, 2018], Coauthor-CS (CoCS for short) [Shchur *et al.*, 2018] and a large-scale dataset Ogbn-arxiv (Ogbna for short) [Hu *et al.*, 2020]. Some basic statistics of each of these datasets are shown in 4.

Cora and Citeseer

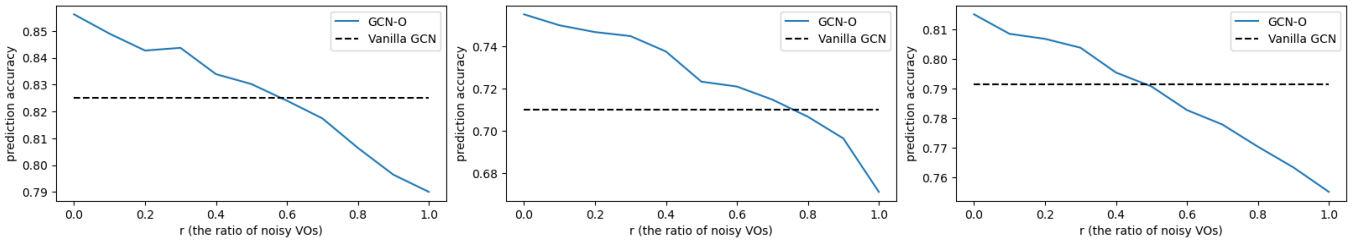
Cora and Citeseer are two citation networks. A node in a graph is corresponding to a scientific document while an edge between two nodes indicate a citation/ reference relationship between the two papers. Each node is associated with a sparse bag-of-word feature vector to represent the paper. The class label of each node reflects the research domain of the paper.

Pubmed and Ogbn-arxiv

Pubmed and Ogbn-arxiv are other two citation networks with a larger size. Pubmed is collected from the PubMed database while Ogbn-arxiv is a subset of the MAG database [Wang *et al.*, 2020a] that collected from arXiv. The definition of nodes and edges are the same as that of Cora and Citeseer. In Pubmed, a node is associated with a 500-dim TF/IDF feature vector while in Ogbn-arxiv, a 128-dim vector generated from skip-gram model is used as node features.

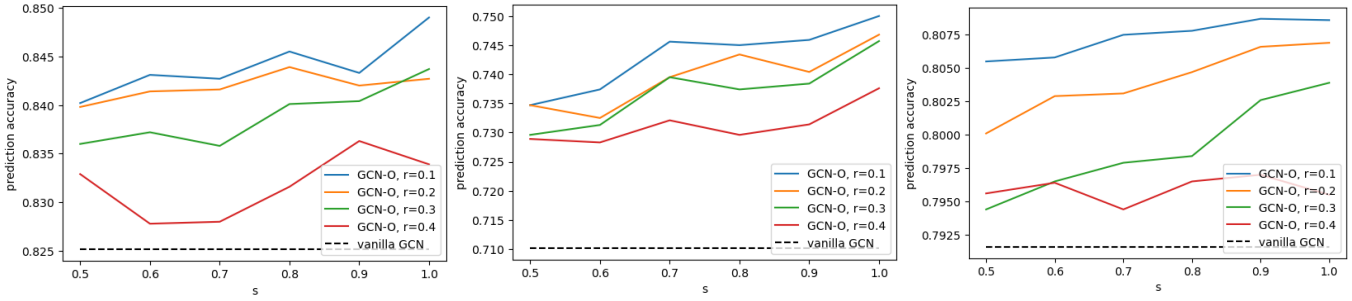
Amazon-Photo

Amazon-Photo is a subset of the Amazon co-purchase graph [McAuley *et al.*, 2015], where each node is corresponding to a good/ product and an edge between two nodes indicates that these two goods are frequently purchased together. A bag-of-word vector encoding product reviews is used to represent



(a) GCN-O on Cora. (b) GCN-O on Citeseer. (c) GCN-O on Pubmed.

Figure 2: Evaluations of GCN-O with different values of noisy VO ratio r on Cora, Citeseer and Pubmed.



(a) GCN-O on Cora. (b) GCN-O on Citeseer. (c) GCN-O on Pubmed.

Figure 3: Evaluations of GCN-O by varying different values of s and r on Cora, Citeseer and Pubmed.

each node and class labels of nodes are given by the product category.

Coauthor-CS

Coauthor-CS is a dataset that describes the academic co-authorships among researchers. A node in a graph is an author and an edge connects two authors indicate that the two nodes co-author a paper. The node feature vector represents the keywords of each author's paper and the class label of each node reflects the most active research domain of the author.

D.2 Methods for Comparisons

In our experiments, we compare Violin with 20 models, which can be categorized into:

1. Models learned without graph: Multi-layer Perceptron (MLP);
2. Unsupervised graph learning models without GNN: the node2vec [Grover and Leskovec, 2016] (n2v);
3. Semi-supervised GNN baseline models: GCN [Kipf and Welling, 2016], GAT [Veličković *et al.*, 2017], GraphSAGE (SAGE for short) [Hamilton *et al.*, 2017], SGC [Wu *et al.*, 2019], JK-Net [Xu *et al.*, 2018b], APPNP [Klicpera *et al.*, 2018], PNA [Corso *et al.*, 2020], GIN [Xu *et al.*, 2018a] and GCNII [Chen *et al.*, 2020b];
4. Unsupervised or self-supervised GNN models: DGI [Veličković *et al.*, 2018], MVGRL [Hassani and Khasahmadi, 2020] and DIMP [Yang *et al.*, 2022];
5. State-of-the-art enhancement methods for semi-supervised GNNs: CG^3 [Wan *et al.*, 2020], GAM [Stretcu *et al.*, 2019], GRAND [Feng *et al.*, 2020], GAUG [Zhao *et al.*, 2021], AdaEdge [Chen *et al.*, 2020a] and CoCoS [Xie *et al.*, 2022];

Note that some models mentioned in the related works are not included in the list as their settings are different from us or unable to be reproduced. We therefore do not compare with them for fairness. In the following, we give a brief introduction of each of the listed models.

MLP Multi-Layer Perceptron (MLP) is a basic neural network that stacks multiple fully-connected layers.

Node2vec [Grover and Leskovec, 2016] is a unsupervised node embedding algorithm. Node2vec applies random walk in a graph to sample a series of node sequences. Following the ideas from the domain of Neural Language Processing (NLP), it learns an embedding for each node using word embedding techniques based on the sampled sequences. Correlations of nodes can therefore be encoded into a multi-dimensional vector.

GCN [Kipf and Welling, 2016] is a semi-supervised framework for graph learning. GCN simplifies the spectral graph convolution operation through one-order approximations. By stacking multiple graph convolutional layers, information from multi-hop away neighbors can be iteratively aggregated to the target node and encoded into the node representation.

GraphSAGE [Hamilton *et al.*, 2017] adopts a spatial-based message passing framework for neighborhood aggregation. Messages are first aggregated from direct neighbors and then combined with the target node itself for the next layer. GraphSAGE additionally proposes a mini-batch training scheme

Dataset	#Node	#Edge	#Class	#Feature	# Train/Val/Test	Label Rate
Cora	2708	5429	7	1433	$140/500/1000$	5.17%
Citeseer	3312	4732	6	3703	$120/500/1000$	3.62%
Pubmed	19717	44338	3	500	$60/500/1000$	0.30%
Ogbn-arxiv	$169K$	$2.5M$	40	128	$91K/29k/48K$	53.70%
Amazon-Photo	7650	119081	10	745	10 per class/ 100 per class/ rest	1.31%
Coauthor-CS	18333	84894	15	6805	10 per class/ 100 per class/ rest	0.82%

Table 4: The statistics of different datasets.

with neighbor sampling, which makes it more efficient to learn from large-scale graphs.

GAT [Veličković *et al.*, 2017] proposes a multi-head attention mechanism for neighborhood aggregation for GNNs. Specifically, different attention weights are assigned to different neighbors when conducting aggregation. Messages propagated from more important neighbors can therefore be highlighted, which results in generating a more discriminative representation for the target node.

JK-Net [Xu *et al.*, 2018b] proposes a jump-connection manner for each GNN layer. With the jump-connection bypasses, messages from each layer can be collected in the final layer, which allows the model to reuse the knowledge learned from different hops and yield a more comprehensive representation for a node.

SGC [Wu *et al.*, 2019] simplifies the aggregation process of common GNN models by removing all non-linear transformation operations in each layer. Such a simplification step significantly speeds up the training of a GNN and leads to a light-weight GNN architecture but still keep comparable performance.

PNA [Corso *et al.*, 2020] proposes the principle neighborhood aggregation mechanism, which adopts a set of different aggregators combining with different scalers for aggregations. With different combinations of aggregators and scalers, PNA is theoretically able to distinguish some graph structure patterns that some previous GNNs fail to.

APPNP [Klicpera *et al.*, 2018] is a learning framework that separates the (non)linear transformation operations and message propagation process for GNN models. It additionally introduces the idea from personal page-rank to guide the message propagation, which is empirically demonstrated to yield node representations with better performance.

GIN [Xu *et al.*, 2018a] develops a new aggregator for common GNNs by adding a learnable parameter or a fixed scalar to scale the target node features before aggregations. The simple yet effective modification is proved to make GIN highly expressive and have as large discriminative power as the Weisfeiler-Lehman test, which enables GIN to distinguish different graph structures.

GCNII [Chen *et al.*, 2020b] proposes a new graph convolutional operator by introducing a initial residual connection and identity mapping for each layer. The initial residual provides a skip-connection to propagate initial information from the input, while the identity mapping ensures that the representation power will not degrade for stacking multiple layers. Empirical experiments show that the influence of oversmoothing can be suppressed in some extent.

DGI [Veličković *et al.*, 2018] learns node embeddings based on a graph contrastive learning paradigm. By maximizing (minimizing) the mutual information between the positive (negative) local view and global view, a GNN model can be trained in a self-supervised manner.

MVGRL [Hassani and Khasahmadi, 2020] is a graph contrastive learning method for node embeddings. It uses a graph diffusion method and a sub-graph sampling method to generate multiple views of augmentations of the original graph. Instead of contrasting the representations within the same view, MVGRL proposes to contrast the node and graph representations across different views. Such a multi-view contrasting scheme makes it outperform many previous graph contrastive learning algorithms [1].

DIMP [Yang *et al.*, 2022] proposes a diverse and interactive message passing framework for self-supervised learning, which aims to solve the problems of blind messages and uniform passing in existing GNN frameworks. The key idea is to generate messages based on the two connected nodes when conducting neighborhood aggregation, and assign different weights for different channels of attributes. By integrating with graph contrastive learning paradigm, DIMP can achieve comparable or even better performance against semi-supervised methods.

CG^3 [Wan *et al.*, 2020] introduces a hierarchical convolution branch to generate a global view for each node. An additional contrastive loss and a generative loss can therefore be formed based on the node representations across these two views to regularize the training process.

GAM [Stretcu *et al.*, 2019] is a self-training algorithm, which introduces a Graph Agreement module to predict the label agreement, i.e., whether the class labels are the same, between any two adjacent nodes. Node pairs with different label agreement types will be assigned different weights and be included in different regularization terms for the objective function. Unlabeled nodes that with high prediction confidence, together with their predicted labels, will be selected to extend the labeled nodes set [2].

[1] However, some of the reported numbers in their paper are complaint to be not reproducible. Details refer to their code repository at https://github.com/kavehhassani/mvgrl and the corresponding Issue panel.

[2] The performance reported in their published paper are found to be wrong due to a bug. The corrected number is reported in their

GRAND [Feng *et al.*, 2020] proposes to generate multiple graph augmentations by introducing node drop and random propagations. A consistency loss derived from the original graph and each augmented graph is added to the final objective to enhance the training of a GNN model.

GAUG-O [Zhao *et al.*, 2021] learns an additional edge predictor to find 'missing edges' in the graph, which can be used to generate an augmentation on graph topology. A GNN will be trained on the augmented graph and regularized by a link prediction loss derived from the edge predictor.

AdaEdge [Chen *et al.*, 2020a] proposes to add edges from intra-class nodes and remove edges from inter-class nodes based on their predicted labels. It applies a multi-round self-training framework to train the model.

CoCoS [Xie *et al.*, 2022] proposes to randomly shuffle node features within intra-class nodes, which generates a context-consistent graph as the augmentation of the original graph. Contrastive loss derived from the node representations across these two graphs is added as an additional term to regularize the model.

D.3 Implementation Details

In our experiments, we implement or reproduce some of the methods mentioned above, including the models listed in type 1, 2, 3 of Appendix D.2. For the models listed in type 4 and 5, we directly compare with their reported performance or the results reproduced by other comparable papers. In the following, we specify the implementation details of our experiments.

Model Architectures

We implement different GNN models and use them as the backbone to validate our proposed Violin framework. In our experiments, the network architecture of most implemented models follow the suggested settings in their published papers, e.g., GCN [Kipf and Welling, 2016], GAT [Veličković *et al.*, 2017] and GCNII [Chen *et al.*, 2020b]. For other models that are not evaluated in node classification task, e.g., GIN [Xu *et al.*, 2018a] and PNA [Corso *et al.*, 2020], or the experimental settings (such as the train/ val/ test split) in their published paper are different from us, e.g., JK-Net [Xu *et al.*, 2018b], we align their network architecture to the implemented GCN model so that the size of learnable parameters is comparable. To fairly compare with models in type (5) of Section D.2, we additionally implement variants of Violin, whose model parameters' size is aligned with them. For experiments in the Ogbn-arxiv datasets, the architectures of GCN [Kipf and Welling, 2016] and GCNII [Chen *et al.*, 2020b] directly follow the implementations provided in the leaderboard [3], while implementations of other models are aligned with GCN. Note that in our evaluations, we would like to investigate the performance improvement that is only raised by Violin. Therefore, some popular tricks that are commonly used for Ogbna in the leaderboard (e.g., knowledge distillation or introducing external node features) are eliminated from our implementations. The architecture details of each model on each dataset are summarized in Table 5.

Hyperparameters for Training

In the training stage, the learning rate for all implemented GNN backbones (except SGC) as well as their corresponding Violin variants is 0.01. For SGC, we follow the suggestion in their published paper [Wu *et al.*, 2019] to train it with a learning rate 0.2 and weight-decay 5×10^{-5}. For each Violin variants, we let $\gamma = 0.4$ (for Ogbn-arxiv) and $\gamma = 0.6$ (for other datasets). We search α in the range of $[0.2, 1]$ by an interval of 0.2 and m (the number of VOs per node) in $\{1, 2, 3, 4, 5\}$. The detailed hyperparameter settings regarding the reported performance of each Violin variant are listed in Table 6 for reproduction. In the experiments, we first train the vanilla GNN backbone in Cora, Citeseer and Pubmed for 300 epochs, while this number is 500 in Ogbn-arxiv and 150 in Amazon-Photo and Coauthor-CS. For the model enhancement stage, we fine-tune the GNN backbone using Violin with the same number of training epochs.

Evaluation Schemes

In the training stage, we add VOs to the original graph $\mathcal{G}$ to generate the semantics-consistent graph $\tilde{\mathcal{G}}$. $\mathcal{G}$ and $\tilde{\mathcal{G}}$ are both utilized by Violin to train a given GNN model. In the evaluation stage, we assess the Violin-enhanced model on $\mathcal{G}$.

For the experiments on Cora, Citeseer, Pubmed and Ogbn-arxiv, we follow the standard (official) train/ validation/ test splits provided by the datasets. Since there are no official train/ validation/ test splits for Amazon-Photo and Coauthor-CS, we randomly sample 10 nodes per class to form the training set, and 100 nodes per class to build the validation set. The remaining nodes are collected for test set. Such a split scheme is similar to the split used by Cora/ Citeseer/ Pubmed and keep the dataset in a low label rate that matches the scenarios we focusing on. The numbers of nodes in each split of each dataset are listed in Table 4.

We run experiments ten times with ten different random seeds. For Amazon-Photo and Coauthor-CS, the nodes used for the train/ validation/ test split is different for each round due to different seeds being used. We record the test accuracy corresponding to the best validation loss for each round and report the averaged test accuracy over ten rounds.

Other Reproduction Information

Our implementations are based on the PyTorch Geometric framework [Fey and Lenssen, 2019]. All our experiments are conducted on a single Linux machine, with an AMD Ryzen Threadripper 3970X CPU @ 3.7GHz, 256 GB RAM, and a NVIDIA RTX TITAN GPU with 24 GB RAM. The code implementation is provided in: *https://github.com/xslangley/violin*.

E Supplementary Experimental Results

In this section, we provide more evaluation results on the proposed Violin algorithm.

code repository at https://github.com/tensorflow/neural-structured-learning/tree/master/research/gam.

[3] https://ogb.stanford.edu/docs/leader_nodeprop/#ogbn-arxiv

Model	Cora, Citeseer, Pubmed	Ogbn-arxiv	Amazon-Photo, Coauthor-CS
GCN	#layer: 2, hidden dim: 16, dropout: 0.6, BatchNorm: False;	#layer: 3, hidden dim: 256, dropout: 0.5, BatchNorm: True;	#layer: 2, hidden dim: 64, dropout: 0.6, BatchNorm: True;
GraphSAGE	#layer: 2, hidden dim: 16, dropout: 0.6, BatchNorm: False;	#layer: 3, hidden dim: 256, dropout: 0.5, BatchNorm: True;	#layer: 2, hidden dim: 64, dropout: 0.6, BatchNorm: True;
GAT	#layer: 2, hidden dim: 8, #heads: 8 dropout: 0.6, BatchNorm: False;	#layer: 3, hidden dim: 75, #heads: 3 dropout: 0.5, BatchNorm: True;	#layer: 2, hidden dim: 8, #heads: 8 dropout: 0.6, BatchNorm: True;
JK-Net	#layer: 2, hidden dim: 16, layer aggregator: concatenation; dropout: 0.6, BatchNorm: False;	#layer: 3, hidden dim: 256, layer aggregator: concatenation; dropout: 0.5, BatchNorm: True;	#layer: 2, hidden dim: 64, layer aggregator: concatenation; dropout: 0.6, BatchNorm: True;
SGC	#layer: 2, hidden dim: N/A, dropout: N/A, BatchNorm: N/A;	#layer: 3, hidden dim: N/A, dropout: N/A, BatchNorm: N/A;	#layer: 2, hidden dim: N/A, dropout: N/A, BatchNorm: N/A;
PNA	#layer: 2, hidden dim: 16, aggregators: mean, max, min; scalers: identity, amplification, attenuation; dropout: 0.6, BatchNorm: False;	#layer: 3, hidden dim: 64, aggregators: mean, max, min; scalers: identity, amplification, attenuation; dropout: 0.5, BatchNorm: True;	Not implemented;
APPNP	#layer: 2, hidden dim: 16, K(#iteration): 10, alpha: 0.1; dropout: 0.5, BatchNorm: False;	#layer: 3, hidden dim: 256, K(#iteration): 10, alpha: 0.1; dropout: 0.5, BatchNorm: True;	Not implemented;
GIN	#layer: 2, hidden dim: 16, ϵ: trainable dropout: 0.6, BatchNorm: False;	#layer: 3, hidden dim: 256, ϵ: trainable dropout: 0.5, BatchNorm: True;	#layer: 2, hidden dim: 64, ϵ: trainable dropout: 0.6, BatchNorm: True;
GCNII	#layer: 64, hidden dim: 64, $\alpha : 0.1, \theta : 0.5$; dropout: 0.6, BatchNorm: False;	#layer: 8, hidden dim: 256, $\alpha : 0.1, \theta : 0.5$; dropout: 0.1, BatchNorm: True;	#layer: 64, hidden dim: 64, $\alpha : 0.1, \theta : 0.5$; dropout: 0.6, BatchNorm: True;

Table 5: The network architectures of each implemented GNN backbone.

Model	Cora	Citeseer	Pubmed	Ogbn-arxiv	Amazon-Photo	Coauthor-CS
Violin(GCN)	$\gamma = 0.6, \alpha = 0.8$ $m = 1, \delta = 0.9$	$\gamma = 0.6, \alpha = 0.2$ $m = 1, \delta = 0.8$	$\gamma = 0.6, \alpha = 0.4$ $m = 2, \delta = 0.9$	$\gamma = 0.4, \alpha = 0.6$ $m = 2, \delta = 0.8$	$\gamma = 0.6, \alpha = 1$ $m = 1, \delta = 0.95$	$\gamma = 0.6, \alpha = 0.4$ $m = 1, \delta = 0.95$
Violin(GraphSAGE)	$\gamma = 0.6, \alpha = 0.8$ $m = 1, \delta = 0.9$	$\gamma = 0.6, \alpha = 0.2$ $m = 1, \delta = 0.8$	$\gamma = 0.6, \alpha = 0.4$ $m = 1, \delta = 0.9$	$\gamma = 0.4, \alpha = 0.8$ $m = 2, \delta = 0.8$	$\gamma = 0.6, \alpha = 0.8$ $m = 1, \delta = 0.95$	$\gamma = 0.6, \alpha = 1$ $m = 1, \delta = 0.95$
Violin(GAT)	$\gamma = 0.6, \alpha = 0.6$ $m = 1, \delta = 0.9$	$\gamma = 0.6, \alpha = 0.2$ $m = 2, \delta = 0.8$	$\gamma = 0.6, \alpha = 0.4$ $m = 2, \delta = 0.9$	$\gamma = 0.4, \alpha = 1$ $m = 2, \delta = 0.8$	$\gamma = 0.6, \alpha = 0.8$ $m = 1, \delta = 0.95$	$\gamma = 0.6, \alpha = 1$ $m = 1, \delta = 0.95$
Violin(JK-Net)	$\gamma = 0.6, \alpha = 0.4$ $m = 1, \delta = 0.9$	$\gamma = 0.6, \alpha = 1$ $m = 1, \delta = 0.8$	$\gamma = 0.6, \alpha = 1$ $m = 1, \delta = 0.9$	$\gamma = 0.4, \alpha = 0.8$ $m = 2, \delta = 0.8$	$\gamma = 0.6, \alpha = 1$ $m = 1, \delta = 0.95$	$\gamma = 0.6, \alpha = 1$ $m = 1, \delta = 0.95$
Violin(GIN)	$\gamma = 0.6, \alpha = 1$ $m = 2, \delta = 0.9$	$\gamma = 0.6, \alpha = 0.2$ $m = 1, \delta = 0.8$	$\gamma = 0.6, \alpha = 0.2$ $m = 2, \delta = 0.9$	$\gamma = 0.4, \alpha = 0.8$ $m = 2, \delta = 0.75$	$\gamma = 0.6, \alpha = 1$ $m = 1, \delta = 1$	$\gamma = 0.6, \alpha = 0.8$ $m = 1, \delta = 0.9$
Violin(GCNII)	$\gamma = 0.6, \alpha = 0.8$ $m = 1, \delta = 0.9$	$\gamma = 0.6, \alpha = 0.6$ $m = 1, \delta = 0.9$	$\gamma = 0.6, \alpha = 1$ $m = 2, \delta = 0.9$	$\gamma = 0.4, \alpha = 0.4$ $m = 2, \delta = 0.8$	$\gamma = 0.6, \alpha = 1$ $m = 1, \delta = 0.85$	$\gamma = 0.6, \alpha = 1$ $m = 1, \delta = 0.85$

Table 6: The hyperparameter settings of each Violin variant with regard to the reported performances.

E.1 Node Classification Results on Two Other Datasets

The results on Amazon-Photo and Coauthor-CS are shown in Table 7. We can observe that the accuracies of all listed GNN backbones are significantly improved, with at least 2% of performance gain across these two datasets. Note that the label rates of these two datasets are both in a relatively low level (1.31% for Amazon-Photo and 0.82% for Coauthor-CS), some GNN models with more sophisticated or advanced architectures (which also means more learnable parameters) suffer from overfitting, such as GAT and GCNII. However, being enhanced by Violin, overfitting can be well alleviated. Violin(GCNII) can even achieve a performance gain by over 15% on the Amazon-Photo dataset. Such an significant improvement can be owed to the advantage that Violin is able to learn from all available data. By adding virtual overbridges (VOs) to the graph, some unlabeled nodes that are untouched before can now be accessible by the target nodes, which enables the model to learn more general patterns from the graph. The introduction of the consistency regularized training paradigm additionally forces the model to be adaptive to different forms of graph contexts, which yields more robust representations for nodes.

E.2 Evaluations on Ogbn-arxiv with Limited Labels

We additionally conduct experiments to investigate the performance of different Violin variants on Ogbn-arxiv with limited labels. Specifically, we randomly sample 10% of nodes for validation, 5%, 1%, 0.5% respectively to form three different training set. The rest of the data is used for testing. Other settings and evaluation schemes are kept the same.

The experimental results are shown in Table 8. All different variants are observed to achieve better performances across different split settings. Such results reflect the great effectiveness of Violin for a wide range of GNN models. Moreover, the performance gain is more significant when fewer labels (lower label rate) are provided for training. The classification accuracies of all Violin variants in the 0.5/10/89.5 split are ven better than or comparable to that of their corre-

Model	Amazon-Photo	Coauthor-CS
GCN	82.89	88.76
GAT	77.10	87.89
SAGE	86.04	86.02
JK-Net	84.63	84.75
SGC	66.57	90.47
GIN	77.76	83.48
GCNII	68.10	84.85
Violin(GCN)	87.77(+4.88)	91.82(+3.06)
Violin(GAT)	87.77(+10.67)	90.59(+2.70)
Violin(SAGE)	88.04(+2.00)	90.03(+4.01)
Violin(JK-Net)	88.22(+3.59)	87.73(+2.98)
Violin(GIN)	82.88(+5.12)	89.77(+6.18)
Violin(GCNII)	83.51(+15.41)	91.55(+6.70)

Table 7: The classification accuracies of each GNN backbone and their corresponding Violin-enhanced variants (%)

Splits: Train/Val/Test (%)	5/10/85	1/10/89	0.5/10/89.5
GCN	69.32	63.85	60.89
GAT	69.50	65.34	62.96
SAGE	68.42	63.21	60.54
JK-Net	68.23	62.79	59.53
GCNII	67.50	62.43	59.85
Violin(GCN)	70.35(+1.03)	66.82(+2.97)	64.88(+3.99)
Violin(GAT)	70.20(+0.70)	67.07(+1.73)	65.23(+2.27)
Violin(SAGE)	70.12(+1.70)	66.56(+3.35)	63.89(+3.35)
Violin(JK-Net)	70.04(+1.81)	65.90(+3.11)	62.91(+3.38)
Violin(GCNII)	70.19(+2.69)	66.13(+3.70)	63.72(+3.87)

Table 8: The performances of different GNNs and their Violin variants with label rates on Ogbn-arxiv (%)

sponding vanilla GNN backbones in the 1/10/90 split, even though the number of available labeled nodes of the formers are only half of the latter split. This demonstrates that Violin can be more effective when only limited labels are given.

E.3 Experiments on Extremely Labeled Settings

We additionally evaluate Violin under more extremely labeled settings. In order to be comparable with other methods, we align our settings with CGPN-GCN [Wan *et al.*, 2021] and GraFN [Lee *et al.*, 2022] for fairness concern. Specifically, for each dataset, we randomly sample 4 nodes and 80 nodes from each class to form the training split and the validation split, respectively. The remaining nodes are all left for the testing split. We also implement another similar setting, where only 2 nodes per class are sampled from the Amazon-Photo dataset for building the training set and 40 nodes per class are sampled for validation. All other settings, e.g., the model architecture, training details, etc., are kept the same as the above.

Experimental results are shown in Table 9. In the four labels/ class setting, Violin achieves the best performance on Cora, Citeseer and AmzP and it is the runner-up in Pubmed. In the two labels/ class setting, Violin outperforms all other state-of-the-art methods. In particular, the advantages over other SOTAs are more significant in AmzP. It clearly shows that Violin is still effective when there are extremely few labeled instances for training. Note that CGPN-GCN and GraFN do not evaluate on less extreme settings (as shown in Table 3 in the main text), while Violin performs well for both cases. Such results further verify that Violin is more powerful and robust for different scenarios.

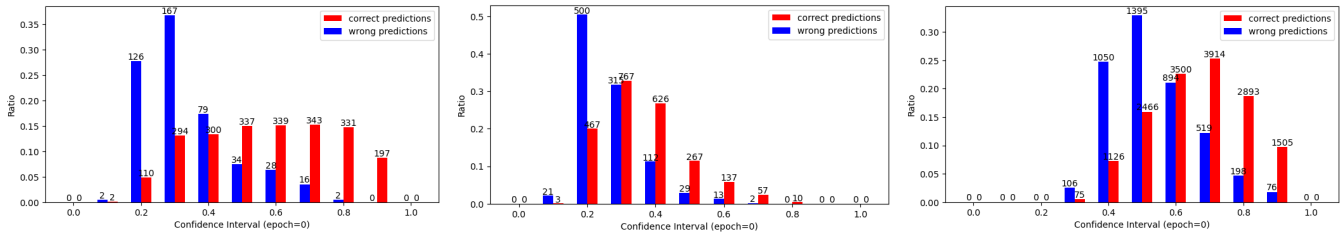
Dataset	Cora	Citeseer	Pubmed	AmzP	AmzP{2}
GCN	67.53	56.64	69.07	83.09	77.22
MVGRL	73.79	63.70	67.69	76.74	71.35
GRAND	70.92	58.40	61.25	N/A	75.83
GRACE	68.69	58.00	68.35	N/A	71.89
GraFN	72.50	66.47	68.41	N/A	80.87
CGPN-GCN	74.43	64.79	**70.94**	84.43	79.54
Violin(GCN)	**75.49**	**67.36**	69.52	**86.12**	**83.09**

Table 9: Accuracies(%) on extremely limited label settings: 4 labels/ class for training. (AmzP{2}: 2 labels/ class)

E.4 Analyses on Prediction Confidence

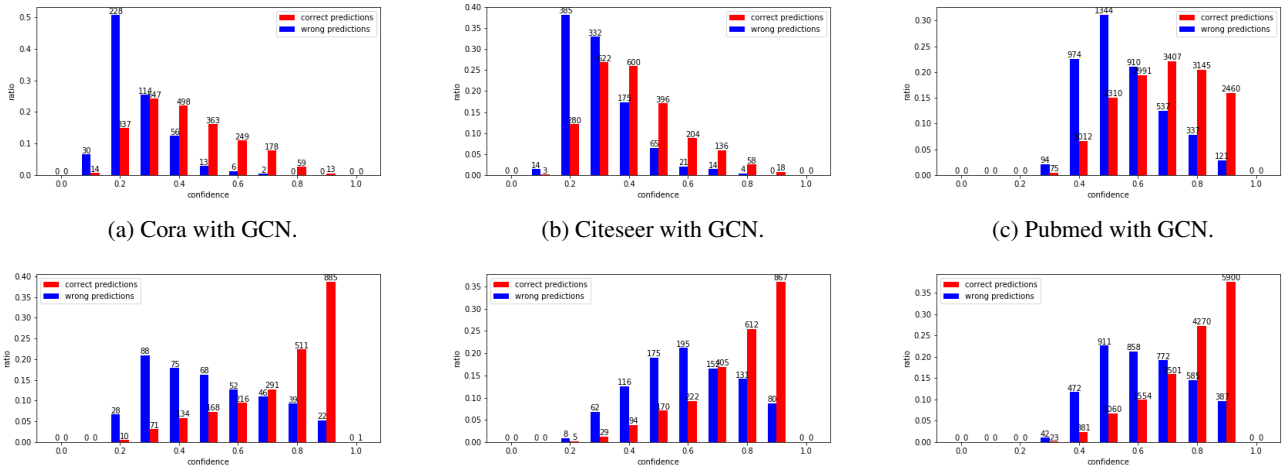
In Figure 4, we visualize the confidence distributions of Violin(GCN) on three typical datasets at different epochs of the training process. (Please click on the corresponding subfigure to show the evolution.) At the beginning of the training, the peak of the confidence distribution (both for correct predictions and wrong predictions) commonly locates in low-value intervals. As the training goes, correctly-predicted nodes shift quickly towards the intervals of high confidence value. This is because the introduction of VOs changes the way of message propagation in the original graph. Class-related patterns extracted from the neighborhood of a remote unlabled node are now available for the aggregation process of a labeled node if they are linked by a VO. Such an operation induces the model to learn more discriminative features from different parts of the graph and be more confident to make a decision. On the other hand, the consistency regularized training mechanism proactively incorporates all nodes into training. The rise of data utilization forces the model to learn from more diverse knowledge, which helps it collect more evidence for generating a more confident prediction. However, the confidence distribution varies at different stages of the training. In other words, it is hard (or inappropriate) to set a static threshold that can always be effective all the time for selecting qualified nodes for adding VOs. Such observations also motivate us to propose the adaptive threshold mechanism for Violin, which helps identify qualified nodes at different training stages.

We additionally compare the distributions of prediction confidence before and after applying Violin to enhance the GCN model. The visualization results are shown in Figure 5. From Figure 5 (a), (b) and (c), we can observe that the confidence distribution of correctly-predicted samples is entangled or largely overlapped with that of the wrong predictions. Most of the predictions are located in the low confidence intervals, which means that GCN is not confident to make decisions on these samples. These instances can be hard samples for GCN under the basic semi-supervised learning paradigm. On the contrary, by enhancing GCN with Violin, the confidence distributions (Figure 5 (d), (e) and (f)) of correct predictions shift towards high-confidence intervals. Compared with the vanilla GCN, Violin(GCN) is more confident to make decisions for most samples. In other words, Violin(GCN) can be more discriminative to some of the hard samples of the vanilla GCN, which significantly improves the



(a) Cora (b) Citeseer (c) Pubmed

Figure 4: Animation examples to show the evolving distributions of prediction confidence of Violin(GCN) on Cora, Citeseer and Pubmed during the training process. (From the beginning of the training till the epoch of the best performance on the validation set.) Animation figures are best viewed via Acrobat Reader on a desktop. Click on the image to start the animation.



(a) Cora with GCN. (b) Citeseer with GCN. (c) Pubmed with GCN.

(d) Cora with Violin(GCN). (e) Citeseer with Violin(GCN). (f) Pubmed with Violin(GCN).

Figure 5: An example of the comparisons of prediction confidence distribution between GCN and Violin(GCN) on Cora, Citeseer and Pubmed datasets (at the epoch with the best prediction performance on the validation set).

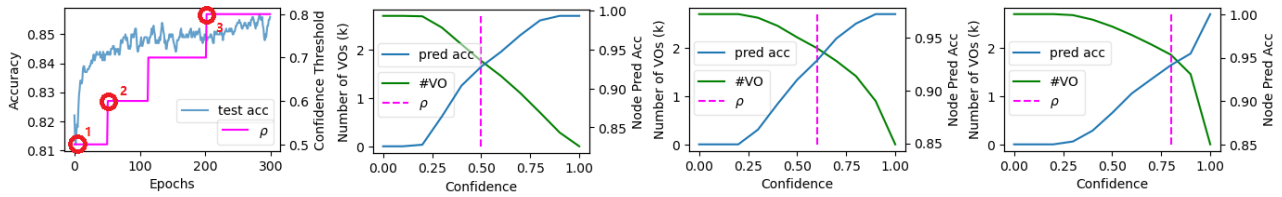
classification accuracies. Although the confidence distribution of wrong predictions is also shifted, most of the wrongly-predicted instances are still concentrated on the middle-level confidence intervals, which can still be differentiated from the distribution of correct predictions. For real-world applications that are sensitive to such low-confidence hard samples, we can manually annotate them or introduce human-in-the-loop techniques such as active learning to further improve the performance. Note that manual annotation can be expensive if based on the vanilla GCN's prediction as most nodes locate in low-confidence intervals. However, the cost can be much more controllable or acceptable for Violin(GCN) since low-confidence nodes are the minority in the predictions. We also believe that distinguishing these hard/ misclassified instances from other samples can be a promising direction to further improve the performance of Violin, which will be left as our future work.

E.5 More Visualization Results on the Adaptive Threshold

We visualize more results on the statuses of prediction accuracies and confidence thresholds during training.

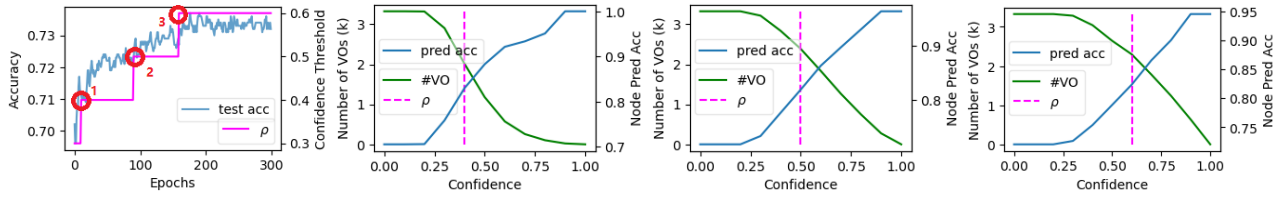
Figure 6 to 8 show three examples on Cora, Citeseer and Pubmed respectively. From the subfigure (a) of these examples, we can see that the confidence threshold (ρ) is dynamically changing as the training goes. More specifically, the value of ρ tends to increase gradually. This is reasonable as the distribution of prediction confidence is shifting during training, as shown in Appendix E.4. Since Violin uses the trained vanilla backbone as the starting point, the prediction confidence at the beginning of the training is underestimated, as revealed by [Wang *et al.*, 2021]. At such epochs, it is reasonable to set a relatively relaxed confidence threshold. As the training goes, the model is more confident on yielding predictions for most of nodes. Violin is therefore increasing the value of ρ to match such a trend of the prediction confidence. Such a dynamic adjustment mechanism enables Violin to monitor the training status of the model and proactively recalibrate itself timely, which improves the prediction performance step by step.

We additionally capture some snapshots of the training statuses at different epochs. Prediction accuracies and the cumulative number of VOs truncated at different confidence levels are shown in subfigures (b), (c) and (d) of Figure 6 to 8.



(a) Accuracies and ρ. (b) Status at 1. (c) Status at 2. (d) Status at 3.

Figure 6: The status of Violin(GCN) on Cora during training. (a) shows the value of prediction accuracies and the confidence threshold at each epoch. (b) - (d) show the prediction accuracies v.s. the cumulative number of qualified nodes/ VOs at each confidence interval, with respect to the points marked in (a).



(a) Accuracies and ρ. (b) Status at 1. (c) Status at 2. (d) Status at 3.

Figure 7: The status of Violin(GCN) on Citeseer during training. (a) shows the value of prediction accuracies and confidence threshold at each epoch. (b) - (d) show the prediction accuracies v.s. the cumulative number of qualified nodes/ VOs at each confidence interval, with respect to the points marked in (a).

As we increase the confidence level, the prediction accuracy of qualified nodes rises. However, the number of qualified nodes also decreases, which equivalently results in less VO candidates for Violin. In other words, the quality (prediction accuracy) and quantity (the number of qualified nodes) act towards different directions. It can also be observed that the curves of prediction accuracies and the cumulative number of VOs are dynamically changing at different training epochs. This is because the distribution of prediction confidence also shifts during training. In other words, although a static confidence threshold, which is commonly applied by many previous works [Stretcu *et al.*, 2019; Chen *et al.*, 2020a; Sun *et al.*, 2020], fits the model at the beginning stage of training, it will be inappropriate for the following stages, vice versa. As we have shown in the preliminary validations in Appendix C.2, a confidence threshold that can make a balance between the quality and quantity makes more sense. Different from most previous works, Violin adopts an adaptive mechanism to derive the confidence threshold at each training epoch. As shown in the subfigures (b), (c) and (d) of Figure 6 to 8, the adaptive confidence threshold locates near the intersection of the two curves. This means that Violin always tries to make a balance between the quantity and quality of added VOs. By continually monitoring the training status, Violin is able to recalibrate itself and yield more reasonable VOs for the model, which is more generalizable and therefore effectively improves the performances.

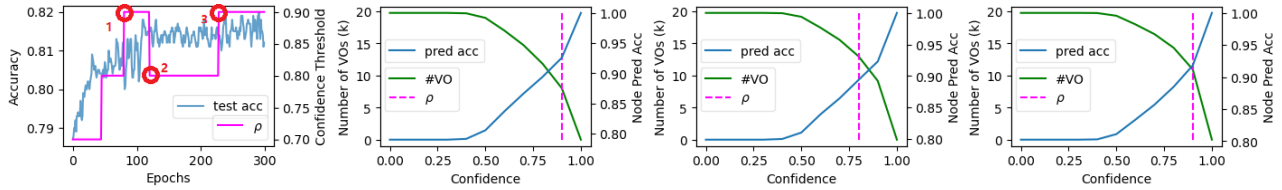
E.6 The Performance of Different Hyperparameter Settings

In this section, we investigate the performance of Violin under different settings of hyperparameters.

The Effect of the Number of Added Virtual Overbridges

In the proposed Violin framework, we can assign multiple VOs to a trusted node to create the semantic-consistency graph. To investigate the influence of the number of added VOs per node (m), we conduct an ablation study on the Cora, Citeseer, Pubmed and Ogbn-arxiv datasets based on the Violin(GCN) variant. Specifically, we assign different values of m to train Violin(GCN) and keep all other settings the same as listed in Table 6.

The experimental results are shown in Figure 9. We can observe that the value of m of best performance can be different for different datasets. There is a common trend that the accuracy of Violin(GCN) first increases as m grows, and then slightly drops. This is reasonable as adding VOs can create more paths for the target node to collect informative messages from a remote but (expected) high-correlated node, which helps extract more discriminative patterns. However, adding too many VOs for a node can also disrupt the graph topology. Too dramatic changes on the graph structure may deviate the semantics of the new graph from the original graph, which degrades the generalization ability of the model. In addition, we can also find that the best value of m tends to be larger in datasets with larger scale. This is because in larger dataset, there are more choices of qualified VOs for each target node. A GNN model also requires to learn diverse patterns so as to be generalizable to such a large-scale graph. Adding more VOs allows the model to extract patterns not only from the local neighborhood but also from remote regions that may have different contexts with the target node, which enables it to learn more generalizable knowledge from the graph.



(a) Accuracies and ρ. (b) Status at 1. (c) Status at 2. (d) Status at 3.

Figure 8: The status of Violin(GCN) on Pubmed during training. (a) shows the value of prediction accuracies and confidence threshold at each epoch. (b) - (d) show the prediction accuracies v.s. the cumulative number of qualified nodes/ VOs at each confidence interval, with respect to the points marked in (a).

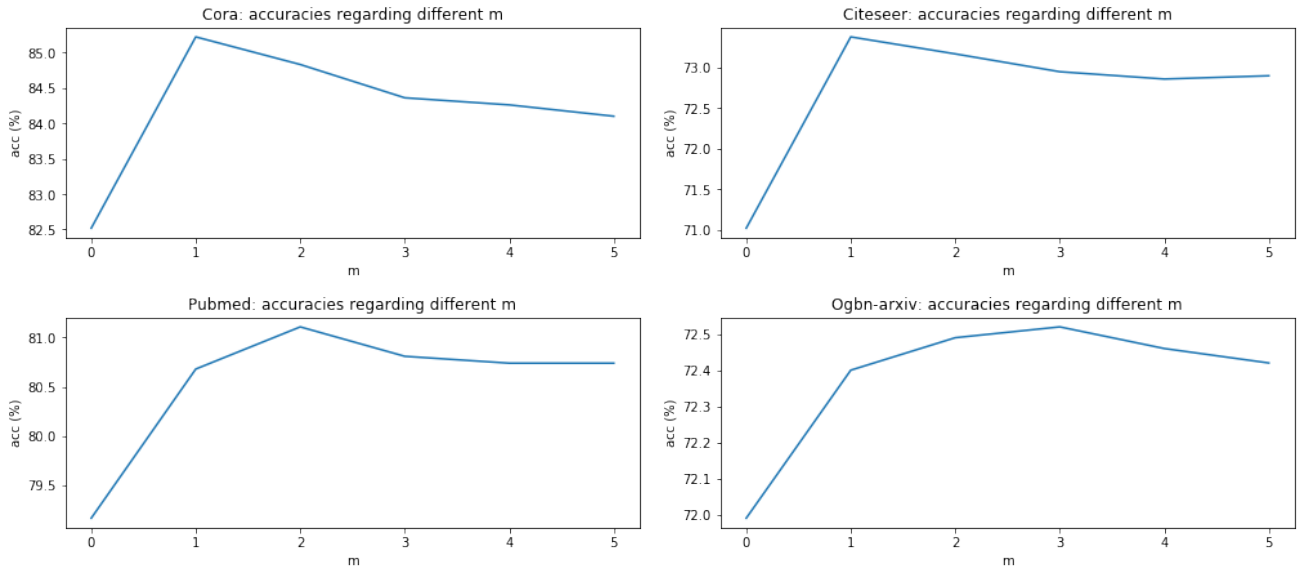


Figure 9: The accuracies of Violin(GCN) by varying the number of virtual overbridges per node (m). The $m = 0$ variant is corresponding to the vanilla GCN.

The Effect of the Regularization Coefficients

In this section, we investigate the performance of Violin when varying the values of the two coefficients for regularization terms, i.e., α and γ. Our experiments are conducted based on Violin(GCN) over the Cora, Citeseer, Pubmed and Ogbn-arxiv datasets. Except for varying the corresponding coefficient value, all other settings are kept the same as that shown in Table 6.

The experimental results are shown in Figure 10. Obviously, with different settings of α or γ, the performance of Violin(GCN) does not change dramatically. The results show that Violin can work steadily in terms of different hyperparameter settings. Such a property can be important since the performance of a model may get unstable when training with fewer labeled data. This also demonstrates the robustness and generalizability of Violin.

More Results with Static Thresholds

We provide more detailed results of Violin(GCN) when trained with static confidence thresholds (ρ). Such an variant is denoted as Violin(GCN)-S. Results are shown in Figure 11. Compared with the vanilla GCN backbone, Violin(GCN)-S performs better in most cases. This is reasonable as VOs can still be added to augment the original graph even if it is assigned a static ρ, which benefits the model by propagating more informative messages among nodes. A small static ρ indicates that Violin(GCN)-S applies a relatively loose rule to select qualified nodes, resulting in adding considerable but noisy VOs to the graph. But as we shown in Appendix C.2, Violin is robust to noises to some extent. Such a property ensures Violin(GCN)-S to work better than the vanilla GCN. On the contrary, a large static ρ implements a stringent bar for qualified nodes and therefore only limited VOs can be added. Under such a condition, qualified nodes are those that have high confidence predicted labels, which results in adding VOs with less noises. Such VOs are efficient for delivering informative messages and are benefitial to the learning even though they are very limited.

However, Violin(GCN) still performs better than Violin(GCN)-S in most cases. In fact, there doesn't exist a consistent ρ that can be generalized to different datasets, e.g., the value of ρ corresponding to the best performance of Violin(GCN)-S on Cora, Citeseer and Pubmed are 0.4, 0.5 and 0.8, respectively. This is reasonable as the data distributions as well as the classification targets, e.g., the

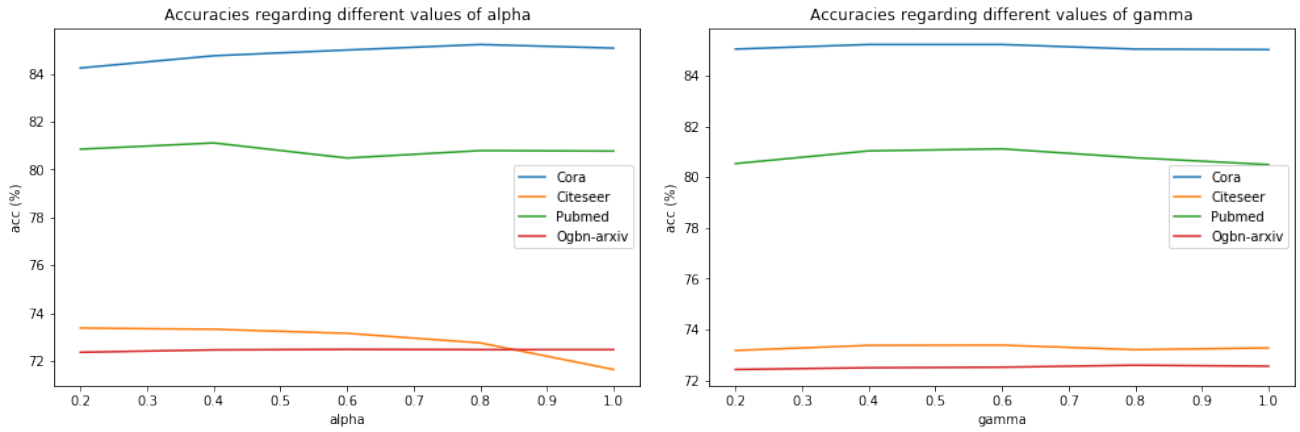
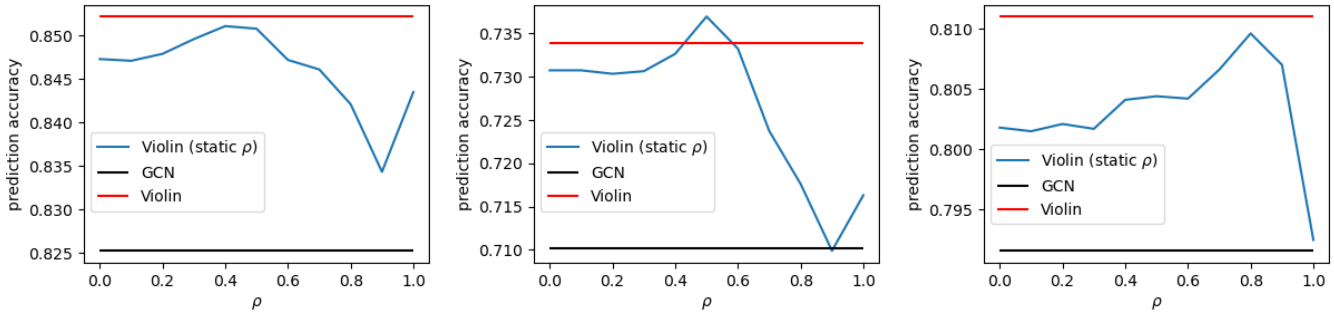


Figure 10: The accuracies of Violin(GCN) by varying α and γ.



(a) Results on Cora. (b) Results on Citeseer. (c) Results on Pubmed.

Figure 11: Ablation studies on Violin that are trained using different static confidence thresholds ρ on Cora, Citeseer and Pubmed.

total number of classes to be classified, are different. It is hard to find a single ρ that can work appropriately for all. On the other hand, Violin(GCN)-S observes a steep performance drop if the value of ρ exceeds a certain extent. When ρ gets larger, fewer VOs are added to form the semantic-consistent graph in the beginning stage of training. As our preliminary results shown in Figure 1, the benefits of introducing VOs become marginal when there are not enough VOs, even if the added VOs are all correct. In contrast, Violin(GCN) can dynamically change the threshold by monitoring the training status of the model. As Figure 6 to 8 show, Violin(GCN) usually starts with a moderate confidence threshold, which adds enough VOs for ignition. As the training goes, it gradually increases the value of ρ so as to keep the pace with the shifting distributions of prediction confidence (as shown in Figure 4). In this way, the adaptive thresholding mechanism can automatically make a balance between the quantity and quality of added VOs. Such superiorities enable the model to learn from a reasonable semantic-consistent graph all the way along the training and also be readily generalized to different datasets for different scenarios, which in turn eases the work for hyperparameter tuning.

E.7 Complexity Analysis

The key idea of Violin is to purposefully modify the graph topology so as to improve information propagation in the graph. From the view of training complexity, the computational overheads caused by Violin is mainly from the message propagation process.

Suppose there is a message-passing GNN model, which stacks L layers, each with a fixed feature dimension value F. Suppose the graph has N nodes and M edges. The time complexity of the message propagation process is approximated to be $O(LMF)$, while the time complexity of dimension transformation is $O(LNF^2)$. Therefore, the time complexity of a common GNN on a forward-pass is:

$$T_{GNN} = O(LMF + LNF^2) \tag{1}$$

When applying Violin to enhance a GNN model, suppose there are μN qualified nodes after thresholding and we plan to add m VOs for each qualified nodes, where $\mu \in [0, 1]$. The number of added VOs is $m\mu N$ and the total number of edges will be $m\mu N + M = \tau M$, where $\tau > 0$. To create the semantics-consistent graph, it additionally takes $O(mN)$ for intra-class node shuffling to generate the required VO candi-

Model	Cora	Citeseer	Pubmed	Ogbn-arxiv
GCN	1.01	1.96	1.82	36.32
Violin(GCN)	1.53	2.23	3.05	88.40

Table 10: The averaged time cost (seconds/ 100 training epochs).

dates. Accordingly, the time complexity of Violin is:

$$T_{Violin} = O\big(\tau LMF + LNF^2 + mN\big) \quad (2)$$

$$\approx O\big(\tau LMF + LNF^2\big) \quad (3)$$

where $LNF^2 \gg mN$ applies almost for all cases.

In practical experiments, the value of MF is smaller than NF^2 and we usually have $\tau < 2$, referring to the statistics in Table 4, model architecture details in Table 5 and the best results shown in Figure 9. Therefore, the time complexity of applying Violin is approximately in the same order of (or linear to) the vanilla GNN model.

We record the averaged time cost (over ten rounds) of empirical experiments of GCN and Violin(GCN) for every 100 training epochs. The settings of GCN and Violin(GCN) on each dataset follow Table 6. The results are shown in Table 10. We can observe that the time cost of Violin(GCN) is around 1.5 times of GCN. It may take more time on larger datasets, e.g., Ogbn-arxiv, as more virtual overbridges are added to the graph, which results in a larger τ, compared with that of in Cora and Citeseer. Even though, the increment in the training time of Violin(GCN) against GCN is still linear, which is acceptable and follows our expectations.

Since Violin does not introduce any additional learnable parameters but only several hyperparameters for tuning, the size of the model enhanced by Violin is the same as that of the vanilla backbone. During the training process, we will need to apply message propagation in both the original graph and the semantics-consistent graph, which costs roughly two times of (GPU) memories of the vanilla backbone. In the inference stage, the enhanced model is applied only to the original graph, which consumes strictly the same resources as the vanilla backbone. Compared with other methods learned from multi-view augmentations [Hassani and Khasahmadi, 2020] or node-wise universe [Wang *et al.*, 2020b], the memory consumption of Violin is much less.

In summary, compared with previous enhancement methods, Violin can be both computationally efficient and memory efficient.

References

[Chen *et al.*, 2020a] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445, 2020.

[Chen *et al.*, 2020b] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pages 1725–1735. PMLR, 2020.

[Corso *et al.*, 2020] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems*, 2020.

[Feng *et al.*, 2020] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural network for semi-supervised learning on graphs. In *NeurIPS'20*, 2020.

[Fey and Lenssen, 2019] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[Hamilton *et al.*, 2017] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[Hassani and Khasahmadi, 2020] Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*, pages 4116–4126. PMLR, 2020.

[Hu *et al.*, 2020] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[Huang *et al.*, 2020] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R Benson. Combining label propagation and simple models out-performs graph neural networks. *arXiv preprint arXiv:2010.13993*, 2020.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[Klicpera *et al.*, 2018] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

[Lee *et al.*, 2022] Junseok Lee, Yunhak Oh, Yeonjun In, Namkyeong Lee, Dongmin Hyun, and Chanyoung Park. Grafn: Semi-supervised node classification on graph with few labels via non-parametric distribution assignment. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2243–2248, 2022.

[Li *et al.*, 2018] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

[McAuley *et al.*, 2015] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 43–52, 2015.

[Namata *et al.*, 2012] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, page 1, 2012.

[Rong *et al.*, 2019] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

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

[Shchur *et al.*, 2018] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[Stretcu *et al.*, 2019] Otilia Stretcu, Krishnamurthy Viswanathan, Dana Movshovitz-Attias, Emmanouil Platanios, Sujith Ravi, and Andrew Tomkins. Graph agreement models for semi-supervised learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[Sun *et al.*, 2020] Ke Sun, Zhouchen Lin, and Zhanxing Zhu. Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5892–5899, 2020.

[Thakoor *et al.*, 2021] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Rémi Munos, Petar Veličković, and Michal Valko. Bootstrapped representation learning on graphs. *arXiv preprint arXiv:2102.06514*, 2021.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Veličković *et al.*, 2017] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[Veličković *et al.*, 2018] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

[Wan *et al.*, 2020] Sheng Wan, Shirui Pan, Jian Yang, and Chen Gong. Contrastive and generative graph convolutional networks for graph-based semi-supervised learning. *arXiv preprint arXiv:2009.07111*, 2020.

[Wan *et al.*, 2021] Sheng Wan, Yibing Zhan, Liu Liu, Baosheng Yu, Shirui Pan, and Chen Gong. Contrastive graph poisson networks: Semi-supervised learning with extremely limited labels. *Advances in Neural Information Processing Systems*, 34:6316–6327, 2021.

[Wang *et al.*, 2020a] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1:396–413, 2020.

[Wang *et al.*, 2020b] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, Juncheng Liu, and Bryan Hooi. Nodeaug: Semi-supervised node classification with data augmentation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 207–217, 2020.

[Wang *et al.*, 2021] Xiao Wang, Hongrui Liu, Chuan Shi, and Cheng Yang. Be confident! towards trustworthy graph neural networks via confidence calibration. *Advances in Neural Information Processing Systems*, 34, 2021.

[Wu *et al.*, 2019] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[Wu *et al.*, 2020] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[Xie *et al.*, 2022] Siyue Xie, Wing Cheong Lau, et al. Cocos: Enhancing semi-supervised learning on graphs with unlabeled data via contrastive context sharing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 4272–4280, 2022.

[Xu *et al.*, 2018a] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[Xu *et al.*, 2018b] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462. PMLR, 2018.

[Yang *et al.*, 2022] Liang Yang, Cheng Chen, Weixun Li, Bingxin Niu, Junhua Gu, Chuan Wang, Dongxiao He, Yuanfang Guo, and Xiaochun Cao. Self-supervised graph neural networks via diverse and interactive message passing. 2022.

[Zhang *et al.*, 2018] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv preprint arXiv:1803.07294*, 2018.

[Zhao *et al.*, 2021] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. In *Proceedings of*

the AAAI Conference on Artificial Intelligence, volume 35, pages 11015–11023, 2021.

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

[Zhu *et al.*, 2021] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, pages 2069–2080, 2021.