TUNEUP: A TRAINING STRATEGY FOR IMPROVING GENERALIZATION OF GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite many advances in Graph Neural Networks (GNNs), their training strategies simply focus on minimizing a loss over nodes in a graph. However, such simplistic training strategies may be sub-optimal as they neglect that certain nodes are much harder to make accurate predictions on than others. Here we present TUNEUP, a curriculum learning strategy for better training GNNs. Crucially, TUNEUP trains a GNN in two stages. The first stage aims to produce a strong base GNN. Such base GNNs tend to perform well on head nodes (nodes with large degrees) but less so on tail nodes (nodes with small degrees). So, the second stage of TUNEUP specifically focuses on improving prediction on tail nodes. Concretely, TUNEUP synthesizes many additional supervised tail node data by dropping edges from head nodes and reusing the supervision on the original head nodes. TUNEUP then minimizes the loss over the synthetic tail nodes to finetune the base GNN. TUNEUP is a general training strategy that can be used with any GNN architecture and any loss, making TUNEUP applicable to a wide range of prediction tasks. Extensive evaluation of TUNEUP on two GNN architectures, three types of prediction tasks, and both inductive and transductive settings shows that TUNEUP significantly improves the performance of the base GNN on tail nodes, while often even improving the performance on head nodes, which together leads up to 58.5% relative improvement in GNN predictive performance. Moreover, TUNEUP significantly outperforms its variants without the two-stage curriculum learning, existing graph data augmentation techniques, as well as other specialized methods for tail nodes.

1 INTRODUCTION

Graph Neural Networks (GNNs) are one of the most successful and widely used paradigms for representation learning on graphs, achieving state-of-the-art performance in a variety of prediction tasks, such as semi-supervised node classification (Kipf & Welling, 2017; Velickovic et al., 2018), link prediction (Hamilton et al., 2017; Kipf & Welling, 2016), and recommender systems (Ying et al., 2018; He et al., 2020). There has been a surge of work on improving GNN model architectures (Velickovic et al., 2018; Xu et al., 2019; 2018; Shi et al., 2020; Klicpera et al., 2019; Wu et al., 2019; Zhao & Akoglu, 2019; Li et al., 2019; Chen et al., 2020; Li et al., 2021) and task-specific losses (Kipf & Welling, 2016; Rendle et al., 2012; Verma et al., 2021; Huang et al., 2021). Despite all these advances, strategies for training a GNN on a given supervised loss remain largely simplistic. Existing work has focused on simply minimizing the given loss over nodes in a graph. While such a simplistic default strategy already gives a strong performance, the strategy may still be sub-optimal as it neglects that some nodes are much harder to make accurate predictions on than others. Consequently, a GNN trained with the default strategy may significantly under-perform on those hard nodes, resulting in overall sub-optimal predictive performance.

Here we present TUNEUP to better train a GNN on a given supervised loss. The key motivation behind TUNEUP is that GNNs tend to under-perform on tail nodes, *i.e.*, nodes with a small number of neighbors (Liu et al., 2021). In practice, performing well on tail nodes is important since they are prevalent in real-world scale-free graphs (Clauset et al., 2009) and newly-arriving cold-start nodes (Lika et al., 2014). To better train a GNN on those hard-to-predict tail nodes, the key idea of TUNEUP is to use a curriculum learning strategy (Bengio et al., 2009); TUNEUP first trains a GNN to perform well on relatively easy head nodes, *i.e.*, nodes with a large number of neighbors. It then proceeds to improve the performance on the hard tail nodes.

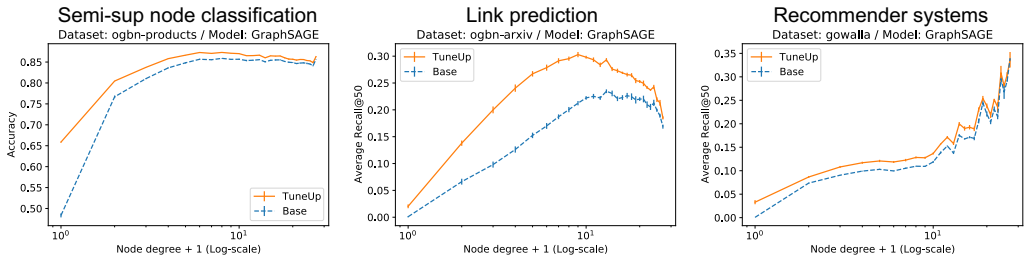

Figure 1: Degree-specific generalization performance of the base GNN and TuneUp in the transductive setting. The x-axis represents the node degrees in the training graph, and the y-axis is the generalization performance averaged over nodes with the specific degrees. We see from the dotted blue curves that the base GNN tends to perform poorly on tail nodes, *i.e.*, nodes with small degrees. Our TuneUp (denoted by the solid orange curves) improves or at least maintains the base GNN performance on almost all node degrees. The improvement is more significant on tail nodes.

Specifically, TuneUp uses the two-stage strategy to train a GNN. In the first stage, TuneUp employs the default training strategy, *i.e.*, simply minimizing the given supervised loss, to produce a strong base GNN to start with. The base GNN tends to perform well on head nodes, but poorly on tail nodes (see the dotted blue curves in Figure 1). To mitigate this issue, the second stage of TuneUp focuses on improving the performance on the tail nodes. Specifically, TuneUp synthesizes many additional tail node inputs by dropping edges from head nodes. TuneUp then adds target supervision (*e.g.*, class labels for node classification, edges for link prediction) on the synthetic tail nodes by reusing the supervision on the original head nodes (before dropping edges). Finally, TuneUp finetunes the base GNN by minimizing the loss over the increased supervised tail node data. The dedicated training on the synthetic tail nodes allows the resulting GNN to perform much better on the real tail nodes, while often even improving the performance on head nodes. TuneUp is simple to implement on top of the default training pipeline of GNNs, as shown in Algorithm 1. Moreover, TuneUp can be used to train any GNN model with any supervised loss, making it generally applicable to a broad range of node and edge-level prediction tasks.

We extensively evaluate TuneUp on a wide range of settings. We consider two representative GNN architectures (GraphSAGE (Hamilton et al., 2017) and GCN (Kipf & Welling, 2017)), three types of key prediction tasks (semi-supervised node classification, link prediction, and recommender systems) with a total of eight datasets, as well as both transductive (*i.e.*, prediction on nodes seen during training) and inductive (*i.e.*, prediction on new nodes never seen during training) settings. For the inductive setting, we additionally consider the challenging cold-start scenario (*i.e.*, limited edge connectivity from new nodes) by randomly removing certain portions of edges from new nodes.

Across all settings, TuneUp produces consistent improvement on the generalization performance of GNNs. In the transductive setting, TuneUp significantly improves the performance of base GNNs on tail nodes, while oftentimes even improving the performance on head nodes (see Figure 1). Moreover, our ablation study shows that the two-stage curriculum training strategy of TuneUp is critical and gives significantly improved performance over its variant strategy without curriculum learning. Finally, we extensively compare our TuneUp against recent graph augmentation techniques (Rong et al., 2020; Liu et al., 2022) and specialized methods for tail nodes (Liu et al., 2021; Zheng et al., 2022; Zhang et al., 2022; Kang et al., 2022). Our TuneUp outperforms all these methods in all settings, while being simpler and more general. Overall, our work demonstrates that training strategies can play an important role in improving generalization performance of GNNs.

2 General Setup and TuneUp

TuneUp is a curriculum learning strategy to train any GNN model with any supervised loss to solve node or edge-level prediction tasks over graphs. We first provide a general task setup for machine learning on graphs and review the default training strategy of GNNs to solve the task. We then present TuneUp, which adds a few simple components to the default training strategy. Finally, we discuss assumptions TuneUp exploits to improve generalization performance of GNNs and why TuneUp even improves the performance on head nodes.

2.1 GENERAL TASK SETUP

We are given a graph $G = (V, E)$, with a set of nodes V and edges E with potentially some features associated with them. GNN F_θ, parameterized by θ, takes the graph G as input and makes prediction $\widehat{Y}$ for the task of interest. The loss function L measures the discrepancy between the GNN's prediction $\widehat{Y}$ and the target supervision Y. In the default training, GNN parameter θ is learned to minimize the loss $L(\widehat{Y}, Y)$ using gradient descent. The setup is general to cover most graph machine learning scenarios. Below, we describe three representative scenarios under the general task setup, which we also consider in our experiments.

Semi-supervised node classification. The task is to predict class labels of unlabeled nodes given a small set of labeled nodes in a graph, which can be formalized as follows.

- **Graph** G: A graph with input node features.
- **Supervison** Y: Class labels of labeled nodes $V_{\text{labeled}} \subset V$.
- **GNN** F_θ: A model that takes G as input and predicts class probabilities over V.
- **Prediction** $\widehat{Y}$: The GNN's prediction over V_{labeled}.
- **Loss** L: Cross-entropy loss.

Since input node features are available, the GNN F_θ can make not only transductive predictions, *i.e.*, prediction over $V_{\text{unlabeled}} \equiv V \setminus V_{\text{labeled}}$, but also inductive predictions (Hamilton et al., 2017), *i.e.*, prediction over new nodes V_{new} that are not in V but connected to V via new edges E_{new}.

Link prediction. The task is to predict new links in a graph given existing links. We consider the node-centric formulation (You et al., 2021): given a source node, predict target nodes that the source node is linked to.

- **Graph** G: A graph with input node features.
- **Supervison** Y: Whether node $s \in V$ is linked to node $t \in V$ in G (positive) or not (negative).
- **GNN** F_θ: A model that takes G as input and predicts the score for a pair of nodes $(s, t) \in V \times V$. Specifically, the model generates embedding $\boldsymbol{z}_v$ for each node in $v \in V$ and uses an MLP over the concatenation of $\boldsymbol{z}_s$ and $\boldsymbol{z}_t$ to predict the score for the pair (s, t) (He et al., 2017).
- **Prediction** $\widehat{Y}$: The GNN's predicted scores over $V \times V$.
- **Loss** L: The Bayesian Personalized Ranking (BPR) loss (Rendle et al., 2012), which encourages the predicted score for the positive pair (s, t_{pos}) to be higher than that for the negative pair (s, t_{neg}) for each source node $s \in V$.

As input node features are available, the GNN F_θ can naturally make inductive link prediction by generating node embeddings on a new graph with new nodes and edges.

Recommender systems. A recommender system can be modeled as a bipartite graph between user nodes V_{user} and item nodes V_{item}, where edges represent user-item interactions. The task is essentially link prediction, *i.e.*, given a user node $u \in V_{\text{user}}$, predict a set of item nodes that u is likely to interact with. In recommender systems, the most successful paradigm is collaborative filtering (Schafer et al., 2007), where shallow embeddings (learnable embeddings for each node) instead of input node features are used to achieve state-of-the-art performance (Wang et al., 2019; He et al., 2020). As input node features are not available in many public recommender system datasets anyway, we focus on the feature-less setting.

- **Graph** G: User-item bipartite graph *without* input node features.
- **Supervison** Y: Whether a user node u has interacted with an item node v in G (positive) or not (negative).
- **GNN** F_θ: A model that takes G as input and predicts the score for a pair of nodes $(u, v) \in V_{\text{user}} \times V_{\text{item}}$. Following Wang et al. (2019), GNN parameter θ contains the input shallow embeddings in addition to the original message passing GNN parameter. To produce the score for the pair of nodes (u, v), we generate the user and item embeddings, $\boldsymbol{z}_u$ and $\boldsymbol{z}_v$, and take the inner product $\boldsymbol{z}_u^\top \boldsymbol{z}_v$ to compute the score (Wang et al., 2019).
- **Prediction** $\widehat{Y}$: The GNN's predicted scores over $V_{\text{user}} \times V_{\text{item}}$.
- **Loss** L: The BPR loss (Rendle et al., 2012).

As we learn the shallow embedding for each node, it is non-trivial to make inductive predictions on new nodes. Therefore, we only consider the transductive setting for recommender systems.

Algorithm 1 TUNEUP. Compared to the default training of a GNN (L2–5), TUNEUP introduces the two-stage training and only adds two components (L8 and L12) that are straightforward to implement.

Given: GNN F_θ, graph G, loss L, supervision Y, DropEdge ratio α.

1: **# First stage: Default training to obtain a base GNN.**
2: **while** θ not converged **do**
3: Make prediction $\widehat{Y} = F_\theta(G)$
4: Compute loss $L(\widehat{Y}, Y)$, compute gradient $\nabla_\theta L$, and update parameter θ.
5: **end while**
6: **# Set up for the second stage.**
7: **if** task is semi-supervised node classification **then**
8: Use F_θ to predict pseudo labels on non-isolated, unlabeled nodes. Add the pseudo labels into Y.
9: **end if**
10: **# Second stage: Fine-tuning the base GNN with increased tail supervision.**
11: **while** θ not converged **do**
12: Synthesize tail nodes, *i.e.*, randomly drop α of edges: $G \xrightarrow{\text{DropEdge}} \widetilde{G}$.
13: Make prediction $\widehat{Y} = F_\theta(\widetilde{G})$.
14: Compute loss $L(\widehat{Y}, Y)$, compute gradient $\nabla_\theta L$, and update parameter θ.
15: **end while**

2.2 DEFAULT GNN TRAINING

Given the graph G, supervision Y, GNN F_θ, its prediction $\widehat{Y} = F_\theta(G)$, and the loss function $L(\widehat{Y}, Y)$, it is straightforward to train the GNN F_θ using gradient descent to minimize the loss. The default training procedure of a GNN is described in L2–5 of Algorithm1.

Remark on mini-batch training. In practice, the prediction $\widehat{Y}$ and the loss computation $L(\widehat{Y}, Y)$ can be made in a mini-batch manner for scalable training. For instance, in the case of semi-supervised node classification, we can predict and compute the loss on a mini-batch of labeled nodes (Hamilton et al., 2017; Zeng et al., 2020). In the case of link prediction and recommender systems, to compute the BPR loss, the score prediction only needs to be made over positive links and randomly sampled negative links. Moreover, the BPR loss can be computed in a mini-batch manner by subsampling the source nodes and keeping only one positive link per source node. These mini-batch training tricks are hidden in Algorithm 1 for simplicity, but should be implemented in practice. Our TUNEUP, which we explain next, is fully compatible with mini-batching training.

2.3 TUNEUP

We are ready to present TUNEUP that uses a two-stage curriculum learning strategy (Bengio et al., 2009) to better train a GNN. At high level, TUNEUP first trains a GNN to perform well on relatively easy head nodes and then proceeds to finetune the GNN to also perform well on hard tail nodes. Specifically, in the first stage (L2–5 in Algorithm 1), TUNEUP uses the default training of GNNs to obtain a strong base GNN model. The base GNN model tends to perform well on head nodes, but poorly on tail nodes. To remedy this issue, in the second training stage, TUNEUP finetunes the base GNN with increased supervision on tail nodes (L7–L15 in Algorithm 1). TUNEUP increases the supervised tail node data in two steps: (1) synthesizing additional tail node inputs and (2) adding target supervision on the synthetic tail nodes, which we detail below.

(1) Synthesizing tail node inputs. TUNEUP synthesizes many additional tail nodes by removing edges from the head nodes. Specifically, in this work, we directly adopt DropEdge (Rong et al., 2020) for simplicity, where a certain portion (given by hyperparameter α) of edges are randomly removed from the original graph G to obtain $\widetilde{G}$ (L12 in Algorithm 1). The resulting $\widetilde{G}$ contains more nodes with low degrees, *i.e.*, tail nodes, than the original graph G does. Hence, the GNN sees more (synthetic) tail nodes as input during training. More advanced strategies to synthesize tail nodes (*e.g.*, dropping more ratio of edges from head nodes) are left for future work.

(2) Adding supervision on the synthetic tail nodes. After synthesizing the tail node inputs, TUNEUP adds target supervision (*e.g.*, class labels for node classification, edges for link prediction) on them so that the supervised loss can be computed over the synthetic tail nodes.

For link prediction tasks, TUNEUP directly reuses the original edges E in G (before dropping) as the target supervision on the synthetic tail nodes. To describe the effectiveness of the approach, suppose we have a node v with six neighbors in the original training graph G. After dropping $\alpha = 0.5$ of edges in L12 of Algorithm 1, this node becomes a synthetic tail node $\widetilde{v}$ with three neighbors in $\widetilde{G}$. Nevertheless, in the loss computation of L14, TUNEUP still reuses *all original six edges* from v in G as target supervision on this synthetic node $\widetilde{v}$. Therefore, this synthetic tail node $\widetilde{v}$ has twice as much edge supervision as any degree-three real tail node (in the original graph G) has.

Similarly, for semi-supervised node classification, TUNEUP can also reuse the target labels of labeled nodes in G for synthetic tail nodes in $\widetilde{G}$. Specifically, for a labeled node $v \in V_{\text{labeled}}$ with ground-truth class label y_v, TUNEUP can reuse y_v for the corresponding synthetic tail node $\widetilde{v}$ in $\widetilde{G}$. However, in the semi-supervised setting, the number of labeled nodes V_{labeled} is often very small, *e.g.*, 1%–5% of all nodes V, limiting the amount of target label supervision TUNEUP can reuse.

To resolve this issue, TUNEUP utilizes pseudo labels (Lee et al., 2013) in addition to the limited ground-truth labels on V_{labeled}. Specifically, TUNEUP applies the base GNN (obtained in the first training stage) over G to predict pseudo labels on non-isolated (*i.e.*, positive-degree) nodes in $V_{\text{unlabeled}}$. TUNEUP then includes the predicted pseudo labels as supervision Y in the second stage (L8 in Algorithm 1). This would significantly increase the size of the supervision Y, *e.g.*, by a factor of $\approx$100 if only 1% of nodes are labeled. While the predicted pseudo labels are noisy in general, they are "best guesses" in the sense that the base GNN uses the *full graph information* G to predict the labels. In the second stage, TUNEUP essentially forces the base GNN to maintain its best guesses *given sparser graph* $\widetilde{G}$ *with limited neighborhood as input*. This in turn allows the resulting GNN to make more accurate prediction on *real* tail nodes with limited neighborhood in the original graph G.

2.4 ASSUMPTIONS TUNEUP EXPLOITS TO IMPROVE GENERALIZATION PERFORMANCE

As the no-free-lunch theorem suggests (Wolpert, 1996), improving generalization performance involves exploiting additional assumptions on real-world prediction tasks, which may not be satisfied by all possible tasks. Here we discuss three key assumptions TUNEUP exploits to improve generalization performance, which we believe to be satisfied by many real-world tasks. We also discuss scenarios when those assumptions may not be satisfied, in which case TUNEUP may not be as effective.

Tail nodes can be synthesized by dropping edges from head nodes. This assumption holds for many real-world graph datasets, as head nodes often start off as tail nodes, *e.g.*, well-cited paper nodes are not cited at the beginning in a paper citation network, and warm users (users with many item interactions) start off as cold-start users in recommender systems. Nonetheless, this assumption may not be satisfied when the characteristics of the node features or neighborhood are very different between head nodes and tail nodes. In such a case, dropping edges from head nodes would give synthetic tail nodes that have different characteristics from real tail nodes. As a result, there is no guarantee that training on the synthetic tail nodes would improve the performance on real tail nodes.

Target supervision on head nodes can be reused for synthetic tail nodes. This assumption holds for tasks where prediction to be made on a given node is more or less a static property of the node, *i.e.*, does not correlate well with the node degrees. For instance, papers' subject areas in a paper citation network, products' categories in a product co-purchasing network, and users' taste in recommender systems stay (mostly) the same regardless of the number of edges we observe on the nodes. However, the assumption may not be satisfied when the target labels to be predicted correlate heavily with the node degrees. As an extreme example, if the target labels we wish to predict are the node degrees, directly reusing the labels on head nodes for the synthetic tail nodes would simply be wrong; consequently, TUNEUP would not work.

Edge information helps for GNNs to make prediction. TUNEUP assumes that edges are useful for GNNs to make accurate prediction, and tail nodes are harder to predict due to the lack of edges. This assumption is likely to hold for many tasks as GNNs can aggregate more neighboring information with more edges. Nevertheless, the assumption may not be satisfied if neighborhood information is detrimental for GNNs to make accurate prediction, *e.g.*, heterophily graph datasets (Lim et al., 2021), where neighboring nodes often have different target labels.

2.5 WHY TUNEUP IMPROVES PERFORMANCE ON HEAD NODES

It is counter-intuitive that TUNEUP improves performance not only on tail nodes but also on head nodes, as seen in Figure 1. One reason may be that best-performing GNNs for node/edge-level tasks, including our experimented ones, use (roughly) *average-based* schemes to aggregate neighboring node features (Hamilton et al., 2017; Kipf & Welling, 2017; Wu et al., 2019; Velickovic et al., 2018). With the average-based GNNs, node embeddings obtained on the sparsified graph $\widetilde{G}$ can be thought of as the *noisy version* of the node embeddings obtained on the full graph G. If the base GNN is finetuned to perform well on many realizations of the noisy embeddings (with different realizations of $\widetilde{G}$ in L12 of Algorithm 1), then the resulting GNN would most likely still perform well on the noise-free embeddings (computed over the full graph G). Moreover, training with the noisy embeddings can even improve the generalization performance on head nodes. We leave in-depth empirical/theoretical investigation for future work.

3 RELATED WORK

Data augmentation for GNNs. The second stage of TUNEUP can be regarded as data augmentation over graphs, on which there has been rich body of work (Ding et al., 2022). Some are specifically designed for semi-supervised node classification (Zhao et al., 2021; Feng et al., 2020; Verma et al., 2021), while others are designed for recommender systems (Verma et al., 2021) and graphs with input node features (Liu et al., 2022). Different from these methods, TUNEUP is generally applicable to any prediction tasks over nodes and edges. The general nature of TUNEUP also allows it to be combined with any of the task-specific data augmentation techniques. As a general graph augmentation technique, Kong et al. (2020) proposed FLAG, which adversarially perturbs input node features (Shafahi et al., 2019). This is complementary to TUNEUP, which perturbs the edge connectivity of the graph. DropEdge (Rong et al., 2020) randomly drops edges from graphs. It was originally developed to overcome the over-smoothing issue of GNNs (Li et al., 2018) in semi-supervised node classification. In contrast, in this work, we adopt DropEdge as a way to synthesize additional tail node inputs for a wide range of prediction tasks over graphs. Methodologically, TUNEUP is distinct from DropEdge in that it employs the two-stage curriculum learning strategy and uses pseudo labels to add supervision on the synthetic tail nodes, both of which are important to yield substantially better performance than the original DropEdge.

Curriculum learning for GNNs. A few works have explored curriculum learning for GNNs. Wang et al. (2021) developed a curriculum learning approach for graph classification, while our work focuses on node/edge-level prediction tasks. Ying et al. (2018) presented a curriculum learning for negative sampling in link prediction, and Li et al. (2022) developed a curriculum learning for tackling imbalanced class labels in node classification. TUNEUP is complementary to both of these approaches while being more broadly applicable to any node/edge-level prediction tasks.

Pre-training GNNs. Pre-training GNNs has attracted huge attention (Veličković et al., 2019; Hu et al., 2020b; Qiu et al., 2020; Hu et al., 2020c; You et al., 2020). This line of work develops *task-agnostic* strategies to pre-train a GNN such that the resulting GNN can be finetuned with task-specific supervised losses to improve performance on diverse downstream tasks. Our work focuses on the downstream stage and presents a strategy for training a GNN on a *task-specific supervised loss*.

Specialized methods for tail nodes. Recently, many methods have been developed for improving generalization performance of GNNs on tail nodes (Liu et al., 2021; Zheng et al., 2022; Kang et al., 2022; Zhang et al., 2022). These methods require augmenting a GNN with tail-node-specific architectural components, while our work does not require any architectural modification and focuses purely on a strategy for *training a GNN* that performs well on *both tail and head nodes*.

4 EXPERIMENTS

Here we extensively evaluate TUNEUP under a wide range of settings. We consider the two representative GNNs (the mean-pooling variant of GraphSAGE (Hamilton et al., 2017) and GCN) as the backbone architectures, and test them on the three prediction tasks described in Section 2.1.

4.1 DATASETS

We present datasets for the three tasks. The statistics are summarized in Table 4 in Appendix.

Semi-supervised node classification.

- **ogbn-arxiv** (Hu et al., 2020a): Given a paper citation network, the task is to predict the subject areas of the papers. Each paper has abstract words as its feature.
- **ogbn-products** (Hu et al., 2020a): Given a product co-purchasing network, the task is to predict the categories of the products. Each product has the product description as its feature.

Link prediction.

- **ppi** (Chandak et al., 2022): Given an incomplete protein-protein interaction network, the task is to predict new interactions. Each protein feature is generated with ESM protein language model (Rives et al., 2021) applied to the protein sequence.
- **flickr** (Zeng et al., 2020): Given an incomplete image-image common-property (*e.g.*, same geographic location, same gallery, comments by the same user, etc.) network, the task is to predict the new common-property links between images. Each image has its description has its feature.
- **ogbn-arxiv** (Hu et al., 2020a): Given an incomplete paper citation network, the task is to predict the additional citation links. Each paper has words in its abstract as its feature.

Recommender systems.

- **gowalla** (Liang et al., 2016; Wang et al., 2019): Given an user-location check-in bipartite graph, the task is to predict new check-in of users.
- **yelp2018** (Wang et al., 2019): Given user-restaurant review graph, the task is to predict new reviews by users.
- **amazon-book** (He & McAuley, 2016; Wang et al., 2019): Given user-product reviews, the task is to predict new reviews by users.

4.2 Data Split and Evaluation Metric

We describe how we split the data to create the transductive and inductive prediction settings. We also present the evaluation metric for each prediction task type.

Semi-supervised node classification. Given the entire nodes in the original dataset, we randomly selected 95% of the nodes and used their subgraph induced as the graph $G = (V, E)$ to train GNNs. The remaining 5% of the entire nodes, V_{new}, are used for inductive prediction. Within V, 10% and 2% of the nodes are used as labeled nodes V_{labeled} for ogbn-arxiv and ogbn-products, respectively. A half of V_{labeled} is used for computing the loss for supervised training, and another half is used as the transductive validation set for tuning hyper-parameters. For the evaluation metric, we used the standard classification accuracy. For the transductive performance, we report the accuracy on the unlabeled nodes $V_{\text{unlabeled}} \equiv V \setminus V_{\text{labeled}}$, while for the inductive performance, we report the accuracy on V_{new}. For the inductive prediction, we also consider the *cold-start* scenario, where certain portions (30%, 60%, and 90%) of edges are randomly removed from the new nodes.

Link prediction. We randomly split the edges in the original graph into training and validation edges with the ratio of 50%/50% (no test split for simplicity (Wang et al., 2019; He et al., 2020)). We follow the same protocol as semi-supervised node classification to obtain nodes for transductive and inductive settings. For the evaluation metric, we used the recall@50 averaged over nodes (Wang et al., 2019). For the transductive performance, we report the recall@50 computed over validation edges within V, while for inductive setting, we report the recall@50 over validation edges from V_{new}. For the inductive setting, we also consider the cold-start scenario.

Recommender systems. For recommender systems, we notice that widely-used benchmark datasets are heavily processed to eliminate all tail nodes, *e.g.*, via the 10-core algorithm (Wang et al., 2019), which does not reflect the realistic use case that involves many cold-start users and items (Lika et al., 2014). To reflect the realistic use case, we use the small training edge ratio on top of the existing benchmark datasets. Specifically, we randomly split the edges in the original graph into training and validation edges with 10%/90% ratio. For the evaluation metric, we used the recall@50 averaged over users (Wang et al., 2019). For the transductive performance, we report the recall@50 computed over the validation edges. We do not consider the inductive setting for recommender systems.

4.3 Baselines and Ablations

We compare our TuneUp against the following strong baselines.

Table 1: Semi-supervised node classification performance with GraphSAGE as the backbone architecture. The metric is classification accuracy. For the "Inductive (cold)", 90% of edges are randomly removed from new nodes. For the results with other edge removal ratios, refer to Table 6 in Appendix. Refer to Table 5 in Appendix for the performance with GCN, where a similar trend is observed.

Method	**ogbn-arxiv**			**ogbn-products**		
	Transductive	**Inductive**	**Inductive (cold)**	**Transductive**	**Inductive**	**Inductive (cold)**
Base	0.6736±0.0005	0.6691±0.0010	0.4738±0.0055	0.8408±0.0008	0.8424±0.0007	0.7223±0.0011
DropEdge	0.6753±0.0013	0.6679±0.0013	0.5456±0.0064	0.8464±0.0005	0.8470±0.0004	0.7713±0.0013
LocalAug	0.6824±0.0005	**0.6763±0.0013**	0.4962±0.0045	0.8446±0.0005	0.8460±0.0007	0.7255±0.0007
ColdBrew	0.6723±0.0007	0.6480±0.0020	0.5095±0.0088	0.8378±0.0008	0.8386±0.0007	0.7422±0.0030
GraphLessNN	0.6075±0.0009	0.5470±0.0028	0.5470±0.0028	0.6671±0.0011	0.6642±0.0006	0.6642±0.0006
Tail-GNN	0.6612±0.0018	0.6555±0.0024	0.5384±0.0022	OOM	OOM	OOM
TUNEUP w/o curriculum	0.6755±0.0014	0.6684±0.0027	0.5473±0.0133	0.8456±0.0005	0.8463±0.0008	0.7569±0.0019
TUNEUP w/o syn-tails	0.6783±0.0011	**0.6754±0.0002**	0.4891±0.0049	0.8433±0.0002	0.8450±0.0002	0.7251±0.0008
TUNEUP w/o pseudo-labels	0.6745±0.0008	0.6672±0.0021	0.5306±0.0116	0.8461±0.0006	0.8471±0.0008	0.7609±0.0040
TUNEUP (ours)	**0.6864±0.0010**	**0.6774±0.0028**	**0.6007±0.0011**	**0.8555±0.0002**	**0.8564±0.0004**	**0.8051±0.0004**
Relative gain over base	+1.9%	+1.2%	+26.8%	+1.7%	+1.7%	+11.5%

Table 2: Link prediction performance with GraphSAGE as the backbone architecture. The metric is recall@50. For the "Inductive (cold)", 60% of edges are randomly removed from new nodes. For other edge removal ratios, refer to Table 9 in Appendix, where TUNEUP consistently outperforms the baselines. Refer to Table 8 in Appendix for the performance with GCN, where we see a similar trend.

Method	**ppi**			**flickr**			**ogbn-arxiv**		
	Transductive	**Inductive**	**Inductive (cold)**	**Transductive**	**Inductive**	**Inductive (cold)**	**Transductive**	**Inductive**	**Inductive (cold)**
Base	0.1165±0.0028	0.1061±0.0076	0.0835±0.0028	0.1100±0.0020	0.1030±0.0025	0.0606±0.0020	0.1326±0.0037	0.1255±0.0047	0.0722±0.0031
DropEdge	0.1750±0.0017	0.1492±0.0073	0.1282±0.0042	0.1437±0.0008	0.1297±0.0011	0.1007±0.0017	0.2015±0.0036	0.1753±0.0040	0.1204±0.0027
LocalAug	0.1265±0.0050	0.1186±0.0041	0.0950±0.0063	0.1117±0.0025	0.1058±0.0027	0.0619±0.0035	0.1365±0.0037	0.1294±0.0028	0.0750±0.0033
ColdBrew	0.1011±0.0031	0.0934±0.0047	0.0761±0.0033	0.0697±0.0174	0.0611±0.0178	0.0413±0.0073	0.1161±0.0048	0.1095±0.0058	0.0630±0.0042
Tail-GNN	0.1150±0.0011	0.1002±0.0025	0.0929±0.0030	0.0950±0.0014	0.0804±0.0014	0.0746±0.0018	0.1081±0.0025	0.0866±0.0039	0.0643±0.0039
TUNEUP w/o curriculum	0.1688±0.0013	0.1458±0.0027	0.1215±0.0023	0.1477±0.0011	0.1333±0.0020	0.1024±0.0015	0.1995±0.0031	0.1760±0.0030	0.1159±0.0040
TUNEUP w/o syn-tails	0.1172±0.0031	0.1057±0.0071	0.0824±0.0021	0.1101±0.0021	0.1029±0.0024	0.0603±0.0017	0.1341±0.0028	0.1273±0.0031	0.0738±0.0021
TUNEUP (ours)	**0.1819±0.0017**	**0.1577±0.0066**	**0.1328±0.0043**	**0.1508±0.0023**	**0.1379±0.0037**	**0.1092±0.0039**	**0.2101±0.0041**	**0.1837±0.0031**	**0.1252±0.0044**
Relative gain over base	+56.1%	+48.7%	+59.1%	+37.2%	+33.8%	+80.1%	+58.5%	+46.4%	+73.3%

- **Base**: Trains a GNN with the default strategy, *i.e.*, L2–5 of Algorithm 1.
- **DropEdge** (Wang et al., 2019): Randomly drops edges during training, *i.e.*, L11–15 of Algorithm 1.
- **Local augentation (LocalAug)** (Liu et al., 2022): Uses a conditional generative model to generate neighboring node features and use them as additional input to a GNN.
- **ColdBrew** (Zheng et al., 2022): Distills head node embeddings computed by the base GNN into an MLP. Uses the resulting MLP to obtain higher-quality tail node embeddings.
- **GraphLessNN** (Zhang et al., 2022): Distills the pseudo labels predicted by the base GNN into an MLP. Uses the resulting MLP to make prediction.
- **Tail-GNN** (Liu et al., 2021): Adds a tail-node specific component inside the original GNN.
- **RAWLS-GCN** (Kang et al., 2022): Modifies the GCN's adjacency matrix to be doubly-stochastic (*i.e.*, all rows and columns sum to 1).

Note that GraphLessNN is only applicable for node classification. LocalAug and ColdBrew require input node features to be available; hence, not applicable to recommender systems. RAWLS-GCN is only applicable to the GCN architecture.

In addition to the existing baselines, we consider the following three direct ablations of TUNEUP.

- **TUNEUP w/o curriculum**: Interleaves the first stage prediction (L3 in Algorithm 1) and the second stage prediction (L12–13 in Algorithm 1) in every parameter update. It is close to TUNEUP except that it does not follow the two-stage curriculum learning strategy.
- **TUNEUP w/o syn-tails**: No L12 in Algorithm 1.
- **TUNEUP w/o pseudo-labels**: No L8 in Algorithm 1.

Another possible ablation, TUNEUP w/o the first stage training (*i.e.*, only performing the second stage training of L2–5 in Algorithm 1), is essentially covered as DropEdge in our experiments.

4.4 HYPER-PARAMETERS

We used 3-layer GNNs and the Adam optimizer (Kingma & Ba, 2015) for all GNN models and datasets, which we found to perform well in our preliminary experiments. For all methods, we

Table 3: Transductive performance on the recommender systems datasets. The metric is recall@50.

Method	gowalla		yelp2018		amazon-book	
	SAGE	GCN	SAGE	GCN	SAGE	GCN
Base	0.0856±0.0005	0.0911±0.0006	**0.0864±0.0005**	0.0791±0.0003	0.0559±0.0003	0.0540±0.0002
DropEdge	0.0835±0.0004	0.0827±0.0012	0.0814±0.0003	0.0719±0.0013	0.0543±0.0003	0.0556±0.0004
Tail-GNN	0.0806±0.0005	0.0815±0.0016	0.0713±0.0004	0.0713±0.0005	0.0562±0.0002	0.0542±0.0003
RAWS-GCN	–	0.0632±0.0006	–	0.0684±0.0004	–	0.0476±0.0002
TuneUp w/o curriculum	0.0849±0.0004	0.0874±0.0038	0.0837±0.0006	0.0755±0.0021	0.0549±0.0001	0.0539±0.0004
TuneUp w/o syn-tails	0.0859±0.0006	0.0917±0.0005	**0.0866±0.0004**	0.0797±0.0003	0.0561±0.0003	0.0542±0.0002
TuneUp (ours)	**0.1043±0.0020**	**0.1102±0.0013**	**0.0866±0.0004**	**0.0885±0.0006**	**0.0577±0.0005**	**0.0630±0.0004**
Relative gain over base	+21.8%	+21.0%	+0.2%	+11.9%	+3.1%	+16.7%

performed the early stopping and tuned their hyper-parameters based on the transductive validation performance. We used the resulting models for both transductive and inductive prediction. For the drop edge ratio α, we tuned it from [0.25, 0.5, 0.75] for all the datasets. We repeated all the experiments with 5 different training seeds to report the mean and the standard deviation. More details are described in Appendix A.

4.5 Results

We first compare TuneUp against the base GNNs that are trained with the default strategy. The last rows of Tables 1, 2, and 3 highlight the relative improvement of TuneUp over the base GNNs. TuneUp improves over the base GNNs across the transductive settings, giving up to 1.9%, 58.5%, and 21.8% relative improvement in the semi-supervised node classification, link prediction, and recommender systems, respectively. TuneUp gives even larger improvement on the challenging cold-start inductive prediction setting, yielding up to 26.8% and 80.1% relative improvement on the node classification and link prediction, respectively. In Appendix, we provide Tables 6, 7, 9, and 10 to show the full results on the cold-start inductive prediction with the different edge removal ratios from new nodes. The larger the ratio is, the more cold-start the setting becomes. We observe that TuneUp provides larger relative gain on larger edge removal ratios, demonstrating its effectiveness on the highly cold-start prediction setting.

We also analyze the degree-specific generalization improvement and highlight the results in Figure 1. The full results (two GNN architectures times the eight datasets) are available in Figures 2, 3, and 4 in Appendix. Across all datasets, architectures, and node degrees, TuneUp produces consistent improvement over the base GNNs. Not surprisingly, improvement is most significant on tail nodes.

Finally, we compare TuneUp against the strong baselines and ablation methods described in Section 4.3. Tables 1, 2, and 3 show the results. We summarize our findings below.

- TuneUp outperforms the graph augmentation methods (DropEdge and LocalAug) as well as the specialized methods for tail nodes (ColdBrew, GraphLessNN, and Tail-GNN), establishing its superior performance against the existing strong baseline methods.
- TuneUp outperforms TuneUp w/o curriculum, which highlights the importance of the two-stage curriculum learning strategy in TuneUp.
- TuneUp also outperforms TuneUp w/o syn-tails and TuneUp w/o pseudo-labels, which suggests that both of the ablated components are necessary for TuneUp to achieve the high performance.

5 Conclusions

In this paper, we presented TuneUp, a curriculum learning strategy to train a GNN to improve their generalization performance. TuneUp first trains a GNN to produce a strong base GNN that performs well on easy head nodes. It then proceeds to improve the prediction over hard tail nodes by finetuning the base GNN with additional synthetic tail nodes. TuneUp is a general strategy that can be used to train any GNN model with any supervised loss. Through extensive experiments, we demonstrated the effectiveness of TuneUp on a wide range of settings, including two GNN architectures, three types of prediction tasks, and both transductive and inductive settings. Overall, our work suggests that training strategies matter in improving generalization of GNNs and can be complementary to advances in model architectures and task-specific losses.

REFERENCES

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, pp. 41–48, 2009.

Payal Chandak, Kexin Huang, and Marinka Zitnik. Building a knowledge graph to enable precision medicine. *bioRxiv*, 2022.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning (ICML)*, pp. 1725–1735. PMLR, 2020.

Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. Data augmentation for deep graph learning: A survey. *arXiv preprint arXiv:2202.08235*, 2022.

Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:22092–22103, 2020.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1025–1035, 2017.

Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the International World Wide Web Conference (WWW)*, pp. 507–517, 2016.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the International World Wide Web Conference (WWW)*, pp. 173–182, 2017.

Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *ACM SIGIR conference on Research and development in Information Retrieval (SIGIR)*, pp. 639–648, 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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020a.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2020b.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. Gpt-gnn: Generative pre-training of graph neural networks. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1857–1867, 2020c.

Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. Mixgcf: An improved training method for graph neural network-based recommender systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 665–674, 2021.

Jian Kang, Yan Zhu, Yinglong Xia, Jiebo Luo, and Hanghang Tong. Rawlsgcn: Towards rawlsian difference principle on graph convolutional network. In *Proceedings of the International World Wide Web Conference (WWW)*, pp. 1214–1225, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

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

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations (ICLR)*, 2019.

Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. Flag: Adversarial data augmentation for graph neural networks. *arXiv preprint arXiv:2010.09891*, 2020.

Dong-Hyun Lee et al. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, pp. 896, 2013.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9267–9276, 2019.

Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. Training graph neural networks with 1000 layers. In *International Conference on Machine Learning (ICML)*, pp. 6437–6449. PMLR, 2021.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *arXiv preprint arXiv:1801.07606*, 2018.

Xiaohe Li, Lijie Wen, Yawen Deng, Fuli Feng, Xuming Hu, Lei Wang, and Zide Fan. Graph neural network with curriculum learning for imbalanced node classification. *arXiv preprint arXiv:2202.02529*, 2022.

Dawen Liang, Laurent Charlin, James McInerney, and David M Blei. Modeling user exposure in recommendation. In *Proceedings of the International World Wide Web Conference (WWW)*, pp. 951–961, 2016.

Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert systems with applications*, 41(4):2065–2073, 2014.

Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 20887–20902, 2021.

Songtao Liu, Rex Ying, Hanze Dong, Lanqing Li, Tingyang Xu, Yu Rong, Peilin Zhao, Junzhou Huang, and Dinghao Wu. Local augmentation for graph neural networks. In *International Conference on Machine Learning (ICML)*, pp. 14054–14072. PMLR, 2022.

Zemin Liu, Trung-Kien Nguyen, and Yuan Fang. Tail-gnn: Tail-node graph neural networks. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1109–1119, 2021.

Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1150–1160, 2020.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15):e2016239118, 2021.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations (ICLR)*, 2020.

J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web*, pp. 291–324. Springer, 2007.

Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations (ICLR)*, 2019.

Vikas Verma, Meng Qu, Kenji Kawaguchi, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. Graphmix: Improved training of gnns for semi-supervised learning. In *AAAI Conference on Artificial Intelligence*, volume 35, pp. 10024–10032, 2021.

Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *ACM SIGIR conference on Research and development in Information Retrieval (SIGIR)*, pp. 165–174, 2019.

Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. Curgraph: Curriculum learning for graph classification. In *Proceedings of the International World Wide Web Conference (WWW)*, pp. 1238–1248, 2021.

David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning (ICML)*, 2019.

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 (ICML)*, pp. 5453–5462, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 974–983, 2018.

Jiaxuan You, Jonathan M Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. In *AAAI Conference on Artificial Intelligence*, volume 35, pp. 10737–10745, 2021.

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

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-Saint: Graph sampling based inductive learning method. In *International Conference on Learning Representations (ICLR)*, 2020.

Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old mlps new tricks via distillation. In *International Conference on Learning Representations (ICLR)*, 2022.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. 2019.

Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. In *AAAI Conference on Artificial Intelligence*, volume 35, pp. 11015–11023, 2021.

Wenqing Zheng, Edward W Huang, Nikhil Rao, Sumeet Katariya, Zhangyang Wang, and Karthik Subbian. Cold brew: Distilling graph node representations with incomplete or missing neighborhoods. In *International Conference on Learning Representations (ICLR)*, 2022.

Table 4: Statistics of nodes used for the transductive evaluation. For link prediction and the recommender system graphs (user-item bipartite graphs), we only evaluate on nodes/users that have at least one edge in the validation set.

Task	Dataset	#Nodes	Avg deg.	Feat. dim
Node cls.	ogbn-arxiv	143,941	12.93	128
	ogbn-products	2,277,597	48.01	100
Link pred.	ppi	15,390	17.30	1,280
	flickr	82,981	4.81	500
	ogbn-arxiv	141,917	7.20	128
Recsys	gowalla	29,858	3.44	–
	yelp2018	31,668	4.93	–
	amazon-book	52,643	5.67	–

Table 5: Semi-supervised node classification performance with GCN as the backbone architecture. The evaluation metric is classification accuracy. For the "Inductive (cold)", 90% of edges are randomly removed from new nodes. For the results with other edge removal ratios, refer to Table 7 in Appendix.

Method	ogbn-arxiv			ogbn-products		
	Transductive	Inductive	Inductive (cold)	Transductive	Inductive	Inductive (cold)
Base	0.6921±0.0001	0.6884±0.0012	0.5490±0.0039	0.8432±0.0008	0.8449±0.0008	0.7469±0.0033
DropEdge	0.6958±0.0006	0.6936±0.0010	0.5621±0.0011	0.8486±0.0003	0.8497±0.0007	0.7664±0.0017
LocalAug	0.6986±0.0013	**0.6959±0.0025**	0.5665±0.0015	0.8488±0.0004	0.8502±0.0005	0.7521±0.0009
ColdBrew	0.6863±0.0007	0.6727±0.0024	0.5368±0.0044	0.8399±0.0009	0.8408±0.0006	0.7377±0.0024
GraphLessNN	0.6125±0.0008	0.5467±0.0029	0.5467±0.0029	0.6668±0.0007	0.6646±0.0007	0.6646±0.0007
RAWLS-GCN	0.6703±0.0009	0.6690±0.0020	0.5328±0.0026	0.8210±0.0010	0.8222±0.0012	0.7086±0.0055
Tail-GNN	0.6440±0.0006	0.6421±0.0022	0.5369±0.0047	OOM	OOM	OOM
TUNEUP w/o curriculum	0.6958±0.0006	0.6921±0.0030	0.5578±0.0017	0.8480±0.0005	0.8490±0.0003	0.7619±0.0031
TUNEUP w/o syn-tails	0.6938±0.0004	0.6915±0.0011	0.5597±0.0034	0.8452±0.0006	0.8468±0.0007	0.7553±0.0027
TUNEUP w/o pseudo-labels	0.6964±0.0007	0.6936±0.0017	0.5528±0.0042	0.8487±0.0008	0.8498±0.0003	0.7675±0.0026
TUNEUP (ours)	**0.6992±0.0001**	**0.6968±0.0012**	**0.5949±0.0017**	**0.8521±0.0006**	**0.8533±0.0002**	**0.7918±0.0052**
Relative gain over base	+1.0%	+1.2%	+8.4%	+1.0%	+1.0%	+6.0%

A DETAILS OF HYPER-PARAMETERS

Here we present the details of hyper-parameters we used in our experiments.

Semi-supervised node classification. We used the hidden dimensionality of 256 and 64 for ogbn-arxiv and ogbn-products, respectively. We trained GNNs in a full-batch manner, and for ogbn-products, we used the reduced dimensionality of 64 so that the entire graph can be fit into the limited GPU memory of 45GB. Mini-batch training is left for future work. We used 1500 epochs for both the default training and finetuning. The learning rate is set to 0.001.

Link prediction. We used the hidden dimensionality of 256 for all datasets. We added the L2 regularization on the node embeddings and tuned its weight for each dataset and GNN architecture. For both the default training and finetuning, we used 1000 epochs and the learning rate of 0.0001.

Recommender systems. We used the shallow embedding dimensionality of 64 and the hidden embedding dimensionality of 256. Similar to the link prediction, we added the L2 regularization to the node embeddings and tuned its weight for each dataset and GNN architecture. For the default training, we trained the model for 2000 epochs with the initial learning rate of 0.001, which is multiplied by 0.1 at the 1000th and 1500th epoch. For finetuning, we used 500 epochs with the learning rate of 0.0001.

For training strategies without curriculum learning, we used the same configuration as the default training.

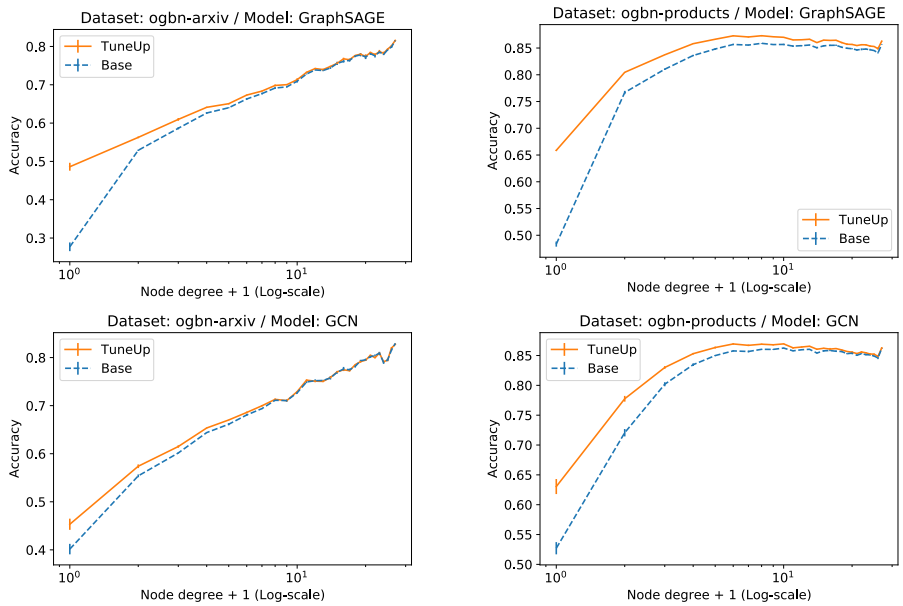

Figure 2: Degree-specific generalization performance of the base GNN and TuneUp in transductive semi-supervised node classification. The evaluation metric is classification accuracy.

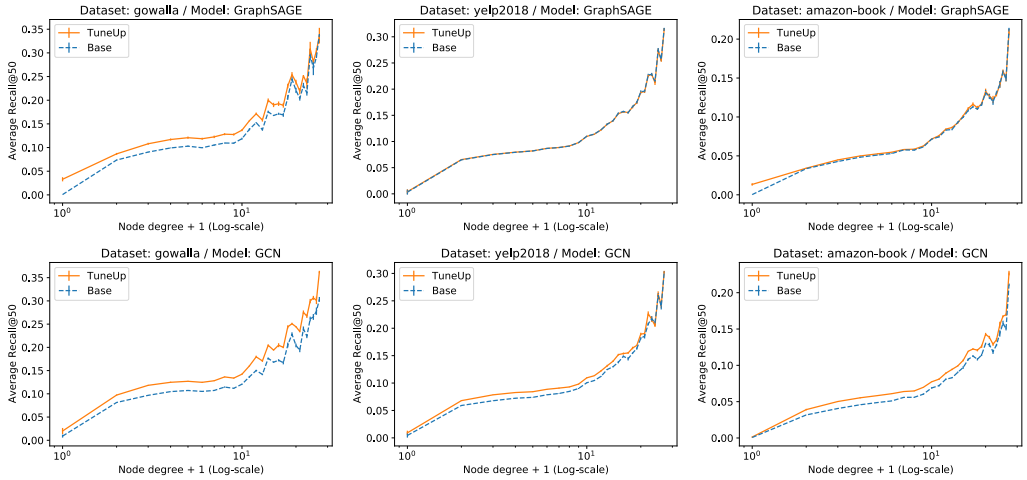

Figure 3: Degree-specific generalization performance of the base GNN and TuneUp in transductive link prediction. The evaluation metric is recall@50.

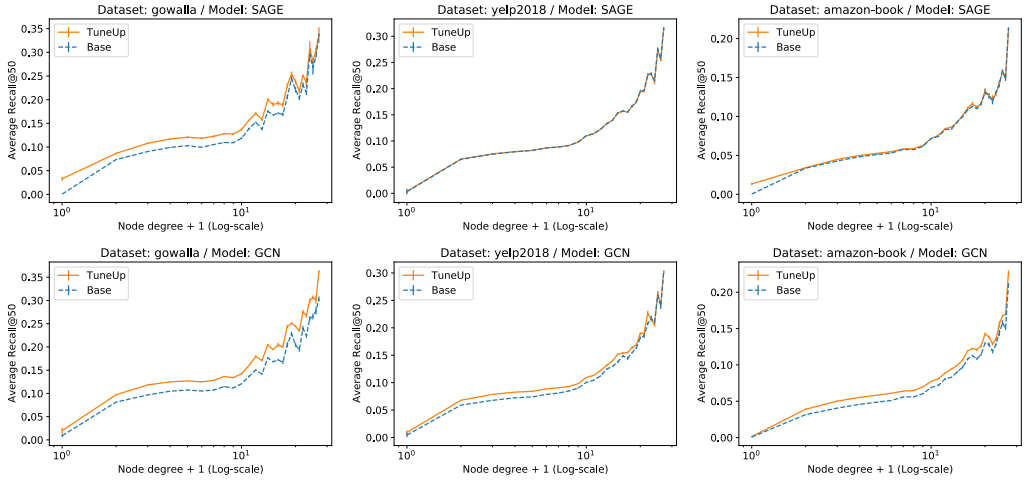

Figure 4: Degree-specific generalization performance of the base GNN and TuneUp in transductive recommender systems. The evaluation metric is recall@50.

Table 6: Cold-start inductive node classification performance with GraphSAGE as the backbone architecture. The larger the edge removal ratio is, the more cold-start the prediction task becomes. The evaluation metric is classification accuracy.

Method	**ogbn-arxiv**			**ogbn-products**		
	Edge removal ratio			**Edge removal ratio**		
	30%	60%	90%	30%	60%	90%
Base	0.6448±0.0017	0.5997±0.0011	0.4738±0.0055	0.8331±0.0010	0.8128±0.0009	0.7223±0.0011
DropEdge	0.6530±0.0020	0.6244±0.0044	0.5456±0.0064	0.8407±0.0004	0.8281±0.0005	0.7713±0.0013
LocalAug	0.6550±0.0026	0.6155±0.0014	0.4962±0.0045	0.8370±0.0009	0.8166±0.0009	0.7255±0.0007
ColdBrew	0.6291±0.0024	0.5945±0.0047	0.5095±0.0088	0.8314±0.0010	0.8143±0.0011	0.7422±0.0030
GraphLessNN	0.5470±0.0028	0.5470±0.0028	0.5470±0.0028	0.6642±0.0006	0.6642±0.0006	0.6642±0.0006
Tail-GNN	0.6388±0.0023	0.6128±0.0017	0.5384±0.0022	OOM	OOM	OOM
TuneUp w/o curriculum	0.6529±0.0029	0.6270±0.0041	0.5473±0.0133	0.8393±0.0005	0.8241±0.0004	0.7569±0.0019
TuneUp w/o syn-tails	0.6517±0.0018	0.6100±0.0026	0.4891±0.0049	0.8360±0.0004	0.8159±0.0003	0.7251±0.0008
TuneUp w/o pseudo-labels	0.6502±0.0016	0.6180±0.0041	0.5306±0.0116	0.8403±0.0010	0.8255±0.0014	0.7609±0.0040
TuneUp (ours)	**0.6667±0.0015**	**0.6504±0.0030**	**0.6007±0.0011**	**0.8522±0.0004**	**0.8434±0.0004**	**0.8051±0.0004**
Relative gain over base	+3.4%	+8.4%	+26.8%	+2.3%	+3.8%	+11.5%

Table 7: Cold-start inductive node classification performance with GCN as the backbone architecture. The larger the edge removal ratio is, the more cold-start the prediction task becomes. The evaluation metric is classification accuracy.

Method	**ogbn-arxiv**			**ogbn-products**		
	Edge removal ratio			**Edge removal ratio**		
	30%	60%	90%	30%	60%	90%
Base	0.6708±0.0013	0.6387±0.0015	0.5490±0.0039	0.8376±0.0009	0.8212±0.0012	0.7469±0.0033
DropEdge	0.6749±0.0022	0.6461±0.0014	0.5621±0.0011	0.8436±0.0005	0.8298±0.0009	0.7664±0.0017
LocalAug	0.6786±0.0020	0.6493±0.0023	0.5665±0.0015	0.8428±0.0003	0.8265±0.0003	0.7521±0.0009
ColdBrew	0.6504±0.0022	0.6177±0.0041	0.5368±0.0044	0.8331±0.0010	0.8158±0.0011	0.7377±0.0024
GraphLessNN	0.5467±0.0029	0.5467±0.0029	0.5467±0.0029	0.6646±0.0007	0.6646±0.0007	0.6646±0.0007
RAWLS-GCN	0.6481±0.0010	0.6114±0.0029	0.5328±0.0026	0.8124±0.0016	0.7915±0.0024	0.7086±0.0055
Tail-GNN	0.6275±0.0011	0.6037±0.0007	0.5369±0.0047	OOM	OOM	OOM
TuneUp w/o curriculum	0.6757±0.0012	0.6437±0.0007	0.5578±0.0017	0.8426±0.0005	0.8284±0.0007	0.7619±0.0031
TuneUp w/o syn-tails	0.6737±0.0018	0.6456±0.0012	0.5597±0.0034	0.8397±0.0007	0.8249±0.0010	0.7553±0.0027
TuneUp w/o pseudo-labels	0.6754±0.0021	0.6437±0.0015	0.5528±0.0042	0.8439±0.0006	0.8299±0.0011	0.7675±0.0026
TuneUp (ours)	**0.6809±0.0021**	**0.6604±0.0008**	**0.5949±0.0017**	**0.8487±0.0006**	**0.8381±0.0013**	**0.7918±0.0052**
Relative gain over base	+1.5%	+3.4%	+8.4%	+1.3%	+2.1%	+6.0%

Table 8: Link prediction performance with the GCN as the backbone architecture. The evaluation metric is recall@50. For the "Inductive (cold)", 60% of edges are randomly removed from new nodes. For the results with other edge removal ratios, refer to Table 10 in Appendix.

Method	ppi			flickr			ogbn-arxiv		
	Transductive	**Inductive**	**Inductive (cold)**	**Transductive**	**Inductive**	**Inductive (cold)**	**Transductive**	**Inductive**	**Inductive (cold)**
Base	0.1491±0.0020	0.1178±0.0033	0.0909±0.0037	0.1431±0.0011	0.1369±0.0022	0.0882±0.0022	0.2152±0.0014	0.2077±0.0014	0.1192±0.0019
DropEdge	0.1903±0.0009	0.1712±0.0096	**0.1364±0.0066**	0.1543±0.0007	0.1425±0.0033	0.1006±0.0020	0.2306±0.0012	0.2116±0.0017	0.1346±0.0010
LocalAug	0.1460±0.0030	0.1321±0.0062	0.0920±0.0046	0.1487±0.0017	0.1445±0.0020	0.0936±0.0030	0.2210±0.0010	0.2134±0.0008	0.1216±0.0015
ColdBrew	0.1133±0.0082	0.0383±0.0272	0.0376±0.0223	0.1201±0.0019	0.1154±0.0019	0.0686±0.0035	0.1783±0.0017	0.1723±0.0017	0.0927±0.0015
Tail-GNN	0.1564±0.0016	0.1294±0.0088	0.0971±0.0053	0.1282±0.0007	0.1230±0.0019	0.0830±0.0030	0.1253±0.0005	0.1144±0.0014	0.0811±0.0014
RAWLS-GCN	0.1280±0.0017	0.0653±0.0045	0.0453±0.0010	0.0704±0.0015	0.0448±0.0018	0.0430±0.0017	0.1010±0.0009	0.0816±0.0010	0.0408±0.0014
TUNEUP w/o curriculum	0.1827±0.0020	0.1511±0.0048	0.1241±0.0039	0.1571±0.0007	0.1466±0.0011	0.1048±0.0016	0.2341±0.0007	0.2169±0.0008	0.1356±0.0017
TUNEUP w/o syn-tails	0.1553±0.0012	0.1240±0.0091	0.0916±0.0053	0.1458±0.0010	0.1398±0.0016	0.0883±0.0015	0.2197±0.0009	0.2123±0.0010	0.1195±0.0014
TUNEUP (ours)	**0.2009±0.0012**	**0.1857±0.0036**	**0.1430±0.0073**	**0.1637±0.0015**	**0.1517±0.0008**	**0.1080±0.0027**	**0.2417±0.0018**	**0.2254±0.0022**	**0.1440±0.0017**
Relative gain over base	+34.8%	+57.7%	+57.4%	+14.4%	+10.8%	+22.5%	+12.3%	+8.5%	+20.9%

Table 9: Cold-start inductive link prediction performance with GraphSAGE. The evaluation metric is recall@50. The larger the edge removal ratio is, the more cold-start the prediction task becomes.

Method	ppi			flickr			ogbn-arxiv		
	Edge removal ratio			**Edge removal ratio**			**Edge removal ratio**		
	30%	60%	90%	30%	60%	90%	30%	60%	90%
Base	0.0990±0.0058	0.0835±0.0028	0.0708±0.0035	0.0830±0.0021	0.0606±0.0020	0.0186±0.0026	0.1021±0.0046	0.0722±0.0031	0.0240±0.0025
DropEdge	0.1401±0.0072	0.1282±0.0042	**0.0999±0.0043**	0.1159±0.0015	0.1007±0.0017	0.0602±0.0037	0.1531±0.0045	0.1204±0.0027	**0.0579±0.0019**
LocalAug	0.1110±0.0049	0.0950±0.0063	0.0717±0.0162	0.0855±0.0028	0.0619±0.0035	0.0185±0.0026	0.1051±0.0031	0.0750±0.0033	0.0261±0.0045
ColdBrew	0.0846±0.0038	0.0761±0.0033	0.0597±0.0077	0.0497±0.0129	0.0413±0.0073	0.0356±0.0086	0.0887±0.0042	0.0630±0.0042	0.0318±0.0045
Tail-GNN	0.0939±0.0032	0.0929±0.0030	0.0842±0.0072	0.0767±0.0011	0.0746±0.0018	0.0521±0.0018	0.0771±0.0037	0.0643±0.0039	0.0410±0.0034
TUNEUP w/o curriculum	0.1374±0.0017	0.1215±0.0023	0.0905±0.0053	0.1192±0.0015	0.1024±0.0015	0.0624±0.0023	0.1516±0.0038	0.1159±0.0040	**0.0567±0.0067**
TUNEUP w/o syn-tails	0.0988±0.0046	0.0824±0.0021	0.0710±0.0031	0.0829±0.0020	0.0603±0.0017	0.0184±0.0026	0.1036±0.0024	0.0738±0.0021	0.0273±0.0041
TUNEUP (ours)	**0.1513±0.0070**	**0.1328±0.0043**	**0.1045±0.0074**	**0.1247±0.0032**	**0.1092±0.0039**	**0.0715±0.0041**	**0.1593±0.0041**	**0.1252±0.0044**	**0.0594±0.0046**
Relative gain over base	+52.8%	+59.1%	+47.5%	+50.3%	+80.1%	+285.1%	+56.0%	+73.3%	+147.5%

Table 10: Cold-start inductive link prediction performance with GCN. The evaluation metric is recall@50. The larger the edge removal ratio is, the more cold-start the prediction task becomes.

Method	ppi			flickr			ogbn-arxiv		
	Edge removal ratio			**Edge removal ratio**			**Edge removal ratio**		
	30%	60%	90%	30%	60%	90%	30%	60%	90%
Base	0.1083±0.0030	0.0909±0.0037	0.0381±0.0031	0.1144±0.0023	0.0882±0.0022	0.0261±0.0012	0.1717±0.0017	0.1192±0.0019	0.0376±0.0015
DropEdge	0.1610±0.0067	**0.1364±0.0066**	**0.0765±0.0037**	0.1241±0.0027	0.1006±0.0020	0.0604±0.0047	0.1822±0.0011	0.1346±0.0010	0.0597±0.0018
LocalAug	0.1151±0.0027	0.0920±0.0046	0.0400±0.0044	0.1222±0.0026	0.0936±0.0030	0.0282±0.0024	0.1772±0.0011	0.1216±0.0015	0.0378±0.0012
ColdBrew	0.0388±0.0265	0.0376±0.0223	0.0328±0.0142	0.0937±0.0023	0.0686±0.0035	0.0483±0.0028	0.1399±0.0015	0.0927±0.0015	0.0415±0.0018
Tail-GNN	0.1189±0.0071	0.0971±0.0053	0.0560±0.0018	0.1059±0.0015	0.0830±0.0030	0.0541±0.0050	0.1001±0.0009	0.0811±0.0014	0.0468±0.0016
RAWLS-GCN	0.0575±0.0031	0.0453±0.0010	0.0394±0.0033	0.0437±0.0020	0.0430±0.0017	0.0421±0.0019	0.0623±0.0005	0.0408±0.0014	0.0239±0.0013
TUNEUP w/o curriculum	0.1437±0.0046	0.1241±0.0039	**0.0754±0.0053**	0.1272±0.0022	0.1048±0.0016	**0.0661±0.0021**	0.1852±0.0015	0.1356±0.0017	0.0557±0.0023
TUNEUP w/o syn-tails	0.1110±0.0078	0.0916±0.0053	0.0347±0.0036	0.1166±0.0021	0.0883±0.0015	0.0284±0.0010	0.1737±0.0008	0.1195±0.0014	0.0368±0.0010
TUNEUP (ours)	**0.1714±0.0051**	**0.1430±0.0073**	**0.0786±0.0090**	**0.1308±0.0021**	**0.1080±0.0027**	**0.0661±0.0028**	**0.1933±0.0018**	**0.1440±0.0017**	**0.0627±0.0015**
Relative gain over base	+58.2%	+57.4%	+106.4%	+14.4%	+22.5%	+153.2%	+12.6%	+20.9%	+67.0%