GRAND+: Scalable Graph Random Neural Networks

Wenzheng Feng\textsuperscript{1}, Yuxiao Dong\textsuperscript{1}, Tinglin Huang\textsuperscript{1}, Ziqi Yin\textsuperscript{2}, Xu Cheng\textsuperscript{3}\textsuperscript{13} Evgeny Kharlamov\textsuperscript{4}, Jie Tang\textsuperscript{1}
\textsuperscript{1}Tsinghua University, \textsuperscript{2}Beijing Institute of Technology
\textsuperscript{3}Tencent Inc., \textsuperscript{4}Bosch Center for Artificial Intelligence
fwz17@mails.tsinghua.edu.cn,\{yuxiaod,tl091,jietang\}@tsinghua.edu.cn,ziqiyin18@gmail.com
alexcheng@tencent.com,evgeny.kharlamov@de.bosch.com

1 INTRODUCTION

Graph structure is a commonplace of both our physical and virtual worlds, such as social relationships, chemical bonds, and information diffusion. The inherent incompleteness of the real-world graph data sparks enormous interests in the problem of semi-supervised learning on graphs [19,33]. To date, graph neural networks (GNNs) have been considered by many as the de facto way to address this problem [1,11,19,28,29]. Briefly, GNNs leverage the graph structure among data samples to facilitate model predictions, enabling them to produce prominent performance improvements over traditional semi-supervised learning methods [32].

However, there are remaining challenges for GNN-based semi-supervised learning solutions. Notably, the generalization of GNNs usually does not form their strengths, as most of them only use a supervised loss to learn parameters [9,19,28,29]. This setup makes the model prone to overfit the limited labeled samples, thereby degrading the prediction performance over unseen samples. To overcome this issue, the graph random neural network (GRAND) [12] designs graph data augmentation and consistency regularization strategies for GNNs. These designs enable it to bring significant performance gains over existing GNNs for semi-supervised learning on Cora, Citeseer and Pubmed.

Specifically, GRAND develops the random propagation operation to generate effective structural data augmentations. It is then trained with both the supervised loss on labeled nodes and the consistency regularization loss on different augmentations of unlabeled nodes. To achieve a good graph augmentation, random propagation in GRAND proposes to use a mixed-order adjacency matrix to propagate the feature matrix. The propagation essentially requires the power iteration of the adjacency matrix at every training step, making it computationally challenging to scale GRAND to large-scale graphs.

To address this issue, we present the GRAND+ framework for large-scale semi-supervised learning on graphs. GRAND+ is a scalable GNN consistency regularization method. In GRAND+, we introduce efficient approximation techniques to perform random propagation in a mini-batch manner, addressing the scalability limitation of GRAND. Furthermore, we improve GRAND by adopting a confidence-aware loss for regulating the consistency between different graph data augmentations. This design stabilizes the training process and provides GRAND+ with good generalization. Specifically, GRAND+ comprises the following techniques:

- **Generalized feature propagation:** We propose a generalized mixed-order matrix to perform random feature propagation. Such matrix...
offers a set of tunable weights to control the importance of different orders of neighborhoods and thus offers a flexible mechanism for dealing with complex real-world graphs.

• **Efficient approximation**: Inspired by recent matrix approximation based GNNs [5, 8], GRAND+ adopts an approximation method—Generalized Forward Push (GFPush)—to efficiently calculate the generalized propagation matrix. This enables GRAND+ to perform random propagation and model learning in a mini-batch manner, offering the model with significant scalability.

• **Confidence-aware loss**: We design a confidence-aware loss for the GRAND+ regularization framework. This helps filter out potential noises during the consistency training by ignoring highly uncertain unlabeled samples, thus improving the generalization performance of GRAND+.

We conduct comprehensive experiments on seven public graph datasets with different genres and scales to demonstrate the performance of GRAND+. Overall, GRAND+ yields the best classification results compared to ten GNN baselines on three benchmark datasets and surpasses five representative scalable GNNs on the other four relatively large datasets with efficiency benefits. For example, GRAND+ achieves a state-of-the-art accuracy of 85.0% on Pubmed. On MAG-Scholar-C with 12.4 million nodes, GRAND+ is about 10 fold faster than FastGCN and GraphSAINT, and offers a 4.9% accuracy gain over PPRGo—previously the fastest method on this dataset—with a comparable running time.

## 2 SEMI-SUPERVISED GRAPH LEARNING

### 2.1 Problem

In graph-based semi-supervised learning, the data samples are organized as a graph \( G = (V, E) \), where each node \( v \in V \) represents a data sample and \( E \subset V \times V \) is a set of edges that denote the relationships between each pair of nodes. We use \( A \in \{0, 1\}^{\mid V \mid \times \mid V \mid} \) to represent \( G \)'s adjacency matrix, with each element \( A(s, v) = 1 \) indicating that there exists an edge between \( s \) and \( v \), otherwise \( A(s, v) = 0 \). \( D \) is the diagonal degree matrix where \( D(s, s) = \sum_{A(s, v)} \). \( \tilde{G} \) is used to denote the graph \( G \) with added self-loop connections. The corresponding adjacency matrix is \( \tilde{A} = A + I \) and the degree matrix is \( D = D + I \).

In this work, we focus on the classification problem, in which each sample \( s \) is associated with 1) a feature vector \( X_s \in X \in \mathbb{R}^{\mid V \mid \times df} \) and 2) a label vector \( Y_s \in Y \in \{0, 1\}^{\mid V \mid \times C} \) with \( C \) representing the number of classes. In the semi-supervised setting, only limited nodes \( L \in V \) have observed labels \( 0 < \mid L \mid \ll \mid V \mid \), and the labels of remaining nodes \( U = V - L \) are unseen. The objective of semi-supervised graph learning is to infer the missing labels \( Y_U \) for unlabeled nodes \( U \) based on graph structure \( G \), node features \( X \), and the observed labels \( Y_L \).

### 2.2 Related Work

Graph neural networks (GNNs) have been widely adopted for addressing the semi-supervised graph learning problem. In this part, we review the progress of GNNs with an emphasis on their large-scale solutions to semi-supervised graph learning.

### Graph Convolutional Network

The graph convolutional network (GCN) [19] generalizes the convolution operation into graphs. Specifically, the \( l \)-th GCN layer is defined as:

\[
H^{(l+1)} = \sigma(\hat{A}H^{(l)}W^{(l)}),
\]

where \( H^{(l)} \) denotes the hidden node representations of the \( l \)-th layer, \( \hat{A} = D^{-\frac{1}{2}}\hat{D}^{-\frac{1}{2}} \) is the symmetric normalized adjacency matrix of \( \tilde{G} \), \( W^{(l)} \) denotes the weight matrix of the \( l \)-th layer, and \( \sigma(\cdot) \) denotes the activation function. In practice, this graph convolution procedure would be repeated multiple times, and the final representations are usually fed into a logistic regression layer for classification.

### Simplified Graph Convolution

By taking a closer look at Equation 1, we can observe that graph convolution consists of two operations: feature propagation \( \hat{A}H^{(l)} \) and non-linear transformation \( \sigma(\cdot) \). Wu et al. [29] simplify this procedure by removing the non-linear transformations in hidden layers. The resulting simplified graph convolution (SGC) is formulated as:

\[
\hat{Y} = \text{softmax}(\hat{X}^N\hat{W}),
\]

where \( \hat{X}^N \) is considered as a simplified \( N \)-layer graph convolution on \( X, W \) refers to the learnable weight matrix for classification, and \( \hat{Y} \) denotes the model's predictions.

### GNNs with Mixed-Order Propagation

As pointed by Li et al. [23], \( \hat{X}^N \) will converge to a fix point as \( N \) increases according to the Markov chain convergence theorem, namely, the smoothing issue. To address it, a typical kind of methods [5, 20, 21] suggest to use a more complex mixed-order matrix for feature propagation. For example, APPNP [20] adopts the truncated personalized PageRank (ppr) matrix \( \hat{H}^{ppr} = \sum_{n=0}^{N} \alpha(1 - \alpha)^n \hat{A}^n \), where the hyperparameter \( \alpha \in (0, 1) \) denotes the teleport probability, allowing the model to preserve the local information even when \( N \to +\infty \).

### Scalable GNNs

Broadly, there are three categories of methods proposed for making GNNs scalable: 1) The node sampling methods employ sampling strategies to speed up the recursive feature aggregation procedure. The representative methods include GraphSAGE [14], FastGCN [7], and LADIES [34]; 2) The graph partition methods attempt to divide the original large graph into several small sub-graphs and run GNNs on sub-graphs. This category consists of Cluster-GCN [10] and GraphSAINt [31]; 3) The matrix approximation methods follow the design of SGC [29] to decouple feature propagation and non-linear transformation, and to utilize some approximation methods to accelerate feature propagation. The proposed GRAND+ framework is highly related to matrix approximation based methods such as PPRGo [5] and GBP [8]. We will analyze their differences in Section 3.6.

## 3 THE GRAND+ FRAMEWORK

In this section, we briefly review the graph random neural network (GRAND) and present its scalable solution GRAND+ for large-scale semi-supervised graph learning.

### 3.1 The Graph Random Neural Network

Recently, Feng et al. [12] introduce the graph neural network (GRAND) for semi-supervised node classification. GRAND is a GNN
consistency regularization framework that optimizes the prediction consistency of unlabeled nodes in different augmentations. Specifically, it designs random propagation—a mixed-order propagation strategy—to achieve graph data augmentations. First, the node features $X$ are randomly dropped with DropNode—a variant of dropout. Then the resultant corrupted feature matrix is propagated over the graph with a mixed-order matrix. Instead of the PPR matrix, GRAND uses an average pooling matrix $\Pi^{\text{avg}} = \sum_{n=0}^{N} \tilde{A}^n / (N + 1)$ for propagation. Formally, the random propagation strategy is formulated as:

$$X = \Pi^{\text{avg}} \cdot \text{diag}(z) \cdot X, \quad z_i \sim \text{Bernoulli}(1 - \delta),$$

(3)

where $z \in \{0, 1\}^{|V|}$ denotes the random DropNode masks drawn from Bernoulli(1 - \delta), and \delta represents DropNode probability. In doing so, the dropped information of each node is compensated by its neighborhoods. Under the homophily assumption of graph data, the resulting matrix $X$ can be seen as an effective data augmentation of the original feature matrix $X$. Owing to the randomness of DropNode, this method can in theory generate exponentially many augmentations for each node.

In each training step of GRAND, the random propagation procedure is performed for $M$ times, leading to $M$ augmented feature matrices $(\tilde{X}^m)$ for $1 \leq m \leq M$. Then all the augmented feature matrices are fed into an MLP to get $M$ predictions. During optimization, GRAND is trained with both the standard classification loss on labeled data and an additional consistency regularization loss [4] on the unlabeled node set $U$, that is,

$$\frac{1}{M \cdot |U|} \sum_{i \in U} \sum_{m=1}^{M} \| \Phi^m_s(z) - Y_s \|^2, \quad Y_s = \sum_{i=1}^{M} \frac{1}{M} \Phi_s^m,$$

(4)

where $\Phi^m_s(z)$ is MLP’s prediction probability for node $s$ when using $\tilde{X}^m_s$ as input. The consistency loss provides an additional regularization effect by enforcing the neural network to give similar predictions for different augmentations of unlabeled data. With random propagation and consistency regularization, GRAND achieves better generalization capability over conventional GNNs [12].

**Scalability of GRAND.** In practice, the $n$-th power of the adjacency matrix $\tilde{A}^n$ is computationally infeasible when $n$ is large [25]. To avoid this issue, GRAND adopts the power iteration to directly calculate the entire augmented feature matrix $\tilde{X}$ (in Equation 3), i.e., iteratively calculating and summing up the product of $A$ and $A^n \cdot \text{diag}(z) \cdot X$ for $0 \leq n < N$. This procedure is implemented with the sparse-dense matrix multiplication and has a linear time complexity of $O(|V| + |E|)$. However, it needs to be performed for $M$ times at every training step to generate different feature augmentations. Thus the total complexity of $T$ training steps becomes $O(T \cdot M \cdot (|V| + |E|))$, which is prohibitively expensive when dealing with large graphs.

**3.2 Overview of GRAND+**

We present GRAND+ to achieve both scalability and accuracy for graph based semi-supervised learning. It follows the general consistency regularization principle of GRAND and comprises techniques to make it scalable to large graphs while maintaining GRAND’s flexibility and generalization capability.

Briefly, instead of propagating features with power iteration, we develop an efficient approximation algorithm—generalized forward push (GFPush)—in GRANT+ to pre-compute the required row vectors of propagation matrix and perform random propagation in a mini-batch manner. The time complexity of this procedure is controlled by a predefined hyperparameter, avoiding the scalability limitation faced by GRAND. Furthermore, GRAND+ adopts a new confidence-aware loss for consistency regularization, which makes the training process more stable and leads to better generalization performance than GRAND.

**Propagation Matrix.** In GRAND+, we propose the following generalized mixed-order matrix for feature propagation:

$$\Pi = \sum_{n=0}^{N} w_n \cdot P^n, \quad P = \tilde{D}^{-1} \tilde{A},$$

(5)

where $\sum_{n=0}^{N} w_n = 1$ and $w_n \geq 0$. $P$ is the row-normalized adjacency matrix. Different from the propagation matrices used in GRAND
and other GNNs, the form of \( \Pi \) adopts a set of tunable weights \( \{ w_n | 0 \leq n \leq N \} \) to fuse different orders of adjacency matrices. By adjusting \( w_n \), GRAND+ can flexibly manipulate the importance of different orders of neighborhoods to suit the diverse graphs of distinct structural properties in the real world.

**Training Pipeline.** To achieve fast training, GRAND+ abandons the power iteration method which directly calculates the entire augmented feature matrix \( \tilde{X} \), and instead computes each augmented feature vector separately for each node. Ideally, the augmented feature vector \( \tilde{X}_n \) of node \( n \) is calculated by:

\[
\tilde{X}_n = \sum_{v \in N^2_n} x_v \cdot \Pi(s, v) \cdot X_v, \quad x_v \sim \text{Bernoulli}(1 - \delta). (6)
\]

Here we use \( \Pi \) to denote the row vector of \( \Pi \) corresponding to node \( n \). \( N^2_n \) is used to represent the indices of non-zero elements of \( \Pi \). \( \Pi(s, v) \) denotes the \( v \)-th element of \( \Pi \). This paradigm allows us to generate augmented features for only a batch of nodes in each training step, and thus enables us to use efficient mini-batch gradient descent for optimization.

However, it is difficult to calculate the exact form of \( \Pi \) in practice. To address this problem, we develop several efficient methods to approximate \( \Pi \) in GRAND+. The approximation procedure consists of two stages. In the first stage, we propose an efficient method called *Generalized Forward Push (GFPush)* to compute an error-bounded approximation \( \Pi(k) \) for the row vector \( \Pi \). In the second stage, we adopt a top-k sparsification strategy to truncate \( \Pi(k) \) to only contain the top \( k \) largest elements. The obtained sparsified row approximation \( \Pi(k) \) is used to calculate \( \tilde{X}_n \) as a substitute of \( \Pi \) (Eq. 6). For efficiency, it is required to pre-compute the corresponding row approximations for all nodes used in training. In addition to labeled nodes, GRAND+ also requires unlabeled nodes to perform consistency regularization during training. To further improve efficiency, instead of using the full set of \( U \), GRAND+ samples a smaller subset of unlabeled nodes \( U' \subseteq U \) for consistency regularization. As illustrated in Figure 1, the training pipeline of GRAND+ consists of three steps:

- **Sub-matrix approximation.** We obtain a sparsified row approximation \( \Pi(k) \) for each node \( n \in U \cup U' \) through GFPush and top-k sparsification. The resultant sparsified sub-matrix is used to support random propagation.

- **Mini-batch random propagation.** At each training step, we sample a batch of nodes from \( U \cup U' \) and generate multiple augmentations for each node in the batch with the approximated row vector.

- **Confidence-aware consistency training.** We feed the augmented features into an MLP to get corresponding predictions and optimize the model with both supervised loss and confidence-aware consistency loss.

### 3.3 Sub-Matrix Approximation

**Generalized Forward Push (GFPush).** It can be observed that the row-normalized adjacency matrix \( \mathbf{P} = \mathbf{D}^{-1/2}\mathbf{L} \) is also the reverse random walk transition probability matrix \( \mathbf{P}_r \) on \( \mathcal{G} \), where row vector \( \mathbf{P}_r \) denotes random walk transition probabilities starting from node \( s \). Based on this fact, we propose an efficient algorithm called *Generalized Forward Push (GFPush)* to approximate row vector \( \Pi(s) = \sum_{v \in N^2_n} w_n \mathbf{P}^v \) with a bounded error. GFPush is inspired by the *Forward Push* [2] algorithm for approximating personalized PageRank vector, while has much higher flexibility with the ability to approximate the generalized mixed-order matrix \( \Pi \). The core idea of GFPush is to simulate an \( n \)-step random walk probability diffusion process from \( s \) with a series of pruning operations for acceleration. To achieve that, we should maintain a pair of vectors at each step \( n (0 \leq n \leq N) \):

1. Reserve vector \( \mathbf{q}^{(n)} \in \mathbb{R}^{|V|} \), denoting the probability masses reserved at step \( n \); 2. Residue vector \( \mathbf{r}^{(n)} \in \mathbb{R}^{|V|} \), representing the probability masses to be diffused beyond step \( n \).

Algorithm 1 shows the pseudo-code of GFPush. At beginning, \( \mathbf{r}^{(0)} \) and \( \mathbf{q}^{(0)} \) are both initialized as the indicator vector \( \mathbf{e}^{(s)} \) where \( e^{(s)}_s = 1 \) and \( e^{(s)}_v = 0 \) for \( v \neq s \), meaning the random walk starts from \( s \) with the probability mass of 1. Other reserve and residue vectors (i.e., \( \mathbf{r}^{(n)} \) and \( \mathbf{q}^{(n)} \), \( 1 \leq n \leq N \)) are set to \( \mathbf{0} \). Then the algorithm iteratively updates reserve and residue vectors with \( N \) steps. In the \( n \)-th iteration, the algorithm conducts a *push* operation (Line 5-9 of algorithm 1) for node \( v \) which satisfies \( e^{(s)}_v \geq 0 \).

Here \( d_v = \mathbf{D}(v, v) \) represents the degree of \( v \), \( r_{max} \) is a predefined threshold. In the push operation, the residue \( r^{(n-1)}_v \) of \( v \) is evenly distributed to its neighbors, and the results are stored into the \( n \)-th residue vector \( \mathbf{r}^{(n)} \). Meanwhile, the reserve vector \( \mathbf{q}^{(n)} \) is also updated to be identical with \( \mathbf{r}^{(n)} \). After finishing the push operation on \( v \), we reset \( r^{(n-1)}_v \) to 0 to avoid duplicate updates.

To gain more intuition of this procedure, we could observe that \( r^{(n-1)}_v \) is the conditional probability that a random walk moves from \( s \) to a neighboring node \( u \), conditioned on it reaching \( v \) with probability \( r^{(n-1)}_v \) at the previous step. Thus each push operation on \( v \) can be seen as a one-step random walk probability diffusion process from \( v \) to its neighborhoods. To ensure efficiency, GFPush only conducts push operations for node \( v \) whose residue value is greater than \( d_v \cdot r_{max} \). Thus when the \( n \)-th iteration is finished, \( \mathbf{q}^{(n)} \) can be seen as an approximation of the \( n \)-step random walk transition vector \( \mathbf{P}_r^n \). And \( \Pi_n = \sum_{v \in \mathcal{V}} w_n \mathbf{P}^{(n)}_r \) is accordingly considered as the approximation of \( \Pi \) as returned by the algorithm.

**Theoretical Analysis.** We have the following theorem about the bounds of time complexity, memory complexity, and approximation error of GFPush.

**Theorem 1.** Algorithm 1 has \( O(N/r_{max}) \) time complexity and \( O(N/r_{max}) \) memory complexity, and returns \( \Pi_n \) as an approximation of \( \Pi \) with the \( L_1 \) error bound \( || \Pi_n - \Pi ||_1 \leq N \cdot (2|E| + |V|) \cdot r_{max} \).

**Proof.** See Appendix A.2. □

Theorem 1 suggests that the approximation precision and running cost of GFPush are negatively correlated with \( r_{max} \). In practice, we could use \( r_{max} \) to control the trade-off between efficiency and approximation precision.

**Top-k Sparsification.** To further reduce training cost, we perform top-k sparsification for \( \Pi \). In this procedure, only the top-k largest elements of \( \Pi \) are preserved and other entries are set to 0. Hence the resultant sparsified transition vector \( \Pi(k) \) has at most \( k \) non-zero elements. In this way, the model only considers the \( k \) most
We will empirically examine the effects of employing multiple augmentations of unlabeled data, which is shown to be effective in improving generalization capability. GRAND+ also follows this idea, while adopts a new confidence-aware consistency loss to further improve effectiveness.

Specifically, for node $s \in U_t$, we first calculate the distribution $\hat{Y}_s$ for $s$ in $L_t \cup U_t$ by:

$$\hat{Y}_s = \frac{1}{|L_t| \cdot M} \sum_{s \in L_t} \sum_{m=1}^{M} Y_{s,m} \cdot \log(\hat{Y}_{s,m}).$$

(10)

Confidence-Aware Consistency Loss. Inspired by recent advances in semi-supervised learning [4], GRAND adopts an additional consistency loss to optimize the prediction consistency of multiple augmentations of unlabeled data, which is shown to be effective in improving generalization capability. GRAND+ also follows this idea, while adopts a new confidence-aware consistency loss to further improve effectiveness.

In Equation 7, the augmented feature vector $\hat{X}_s$ is calculated with raw features $X$. However, in some real applications (e.g., image or text classification), the dimension of $X$ might be extremely large, which will incur a huge cost for calculation. To mitigate this issue, we can employ a linear layer to transform each $X_v$ to a low-dimensional hidden representation $H_v \in \mathbb{R}^{d_h}$ firstly, and then perform random propagation with $H$:

$$\hat{X}_s = \sum_{v \in \mathcal{N}_s} z_v \cdot \hat{H}_v \cdot H_v = X_v \cdot W^{(0)},$$

where $W^{(0)} \in \mathbb{R}^{d_f \times d_h}$ denotes learnable transformation matrix. In this way, the computational complexity of this procedure is reduced to $O(k \cdot b \cdot d_h)$, where $d_h \ll d_f$ denotes the dimension of $H_v$.

Prediction. During training, the augmented feature vector $\hat{X}_s$ is fed into an MLP model to get the corresponding outputs:

$$\hat{Y}_s = \text{MLP}(X_v; \Theta),$$

(9)

where $\hat{Y}_s \in [0, 1]^C$ denotes the prediction probabilities of $s$. $\Theta$ represents MLP’s parameters.

3.5 Confidence-Aware Consistency Training

GRAND+ adopts both supervised classification loss and consistency regularization loss to optimize model parameters during training. The supervised loss is defined as the average cross-entropy over multiple augmentations of labeled nodes:

$$L_{sup} = \frac{1}{|L_t|} \sum_{s \in L_t} \sum_{m=1}^{M} Y_{s,m} \cdot \log(\hat{Y}_{s,m}).$$

Confidence-Aware Consistency Loss. Inspired by recent advances in semi-supervised learning [4], GRAND adopts an additional consistency loss to optimize the prediction consistency of multiple augmentations of unlabeled data, which is shown to be effective in improving generalization capability. GRAND+ also follows this idea, while adopts a new confidence-aware consistency loss to further improve effectiveness.

Specifically, for node $s \in U_t$, we first calculate the distribution center by taking the average of its $M$ prediction probabilities, i.e., $\bar{Y}_s = \frac{1}{M} \sum_{m=1}^{M} \hat{Y}_{s,m} / M$. Then we apply sharpening [17] trick over $\bar{Y}_s$ to “guess” a pseudo label $\hat{Y}_s$ for node $s$. Formally, the guessed probability on the $j$-th class of node $s$ is obtained via:

$$\hat{Y}(s,j) = \bar{Y}(s,j) + \frac{1}{\sum_{c=0}^{C-1} \bar{Y}(s,c)} \cdot \Theta,$$

(11)

where $0 < \tau \leq 1$ is a hyperparameter to control the sharpness of the guessed pseudo label. As $\tau$ decreases, $\hat{Y}_s$ is enforced to become sharper and converges to a one-hot distribution eventually.

Then the confidence-aware consistency loss on unlabeled node batch $U_t$ is defined as:

$$L_{con} = \frac{1}{|U_t| \cdot M} \sum_{s \in U_t} \sum_{m=1}^{M} \mathbb{I}(\max(\bar{Y}_s) \geq \gamma) \cdot \sum_{m=1}^{M} D(\bar{Y}_s, \hat{Y}_s^{(m)}).$$

(12)

where $\mathbb{I}(\max(\bar{Y}_s) \geq \gamma)$ is an indicator function which outputs 1 if $\max(\bar{Y}_s) \geq \gamma$ holds, and outputs 0 otherwise. $0 \leq \gamma < 1$ is a predefined threshold. $D(p, q)$ is a distance function which measures the distribution discrepancy between $p$ and $q$. Here we mainly consider two options for $D$: $L_2$ distance and KL divergence.

Compared with the consistency loss used in GRAND (Cf. Equation 4), the biggest advantage of $L_{con}$ is that it only considers “highly confident” unlabeled nodes determined by threshold $\tau$ in optimization. This mechanism could reduce the potential training noise by filtering out uncertain pseudo-labels, further improving model’s performance in practice. Combining $L_{con}$ and $L_{sup}$, the final loss for model optimization is defined as:

$$L = L_{sup} + \lambda(t) \cdot L_{con}.$$  

(13)
where $\lambda(t)$ is a linear warmup function [13] which increases linearly from 0 to the maximum value $\lambda_{\text{max}}$ as training step $t$ increases.

Model Inference. After training, we need to infer the predictions for unlabeled nodes. GRAND+ adopts power iteration to calculate the exact prediction results for unlabeled nodes during inference:

$$\hat{Y}^{(inf)} = \text{MLP}(\Pi \cdot (1 - \delta) \cdot X, \Theta),$$

where we rescale $X$ with $(1 - \delta)$ to make it identical with the expectation of the DropNode perturbed features used in training. Note that unlike GRAND, the above power iteration process only needs to be performed once in GRAND+, and the computational cost is acceptable in practice. Compared with obtaining predictions with GPush as done in training, this inference strategy could provide more accurate predictions in theory. Algorithm 2 shows the entire training and inference procedure of GRAND+.

### 3.6 Model Analysis

**Complexity Analysis.** We provide detailed analyses for the time complexities of GRAND+’s different learning stages. According to Theorem 1, the complexity of approximation stage (Line 2–5 of Algorithm 2) is $O((|U|^2 + |L|) \cdot N / \tau_{\text{max}})$. As for the training stage (Line 6–15 of Algorithm 2), the total complexity of $T$ training steps is $O(k \cdot b \cdot M \cdot T)$, which is practically efficient for large graphs since $b$ and $k$ are usually much smaller than the graph size. The complexity of inference stage (Line 17 of Algorithm 2) is $O((|V| + |E|) \cdot N)$, which is linear with the sum of node and edge counts.

**GRAND+ vs. PPRGo and GBP.** Similar with GRAND+ PPRGo [5] and GBP [8] also adopt matrix approximation methods to scale GNNs. However, GRAND+ differs from the two methods in several key aspects. PPRGo scales up APPNP by using Forward Push [2] to approximate the ppr matrix. Compared with PPRGo, GRAND+ is more flexible in real applications thanks to the adopted generalized propagation matrix $\Pi$ and GPush algorithm. GBP also owns this merit by using the generalized PageRank matrix [22] for feature propagation. However, it directly approximates the propagation results of raw features through bidirectional propagation [3], whose computational complexity is linear with the raw feature dimension, rendering it difficult to handle datasets with high-dimensional features. Moreover, different from PPRGo and GBP design for the general supervised classification problem, GRAND+ makes significant improvements for semi-supervised setting by adopting random propagation and consistency regularization to enhance generalization capability.

### 4 EXPERIMENTS

#### 4.1 Experimental Setup

**Baselines.** In our experiments, we compare GRAND+ with five state-of-the-art full-batch GNNs—GCN [19], GAT [28], APPNP [20], GCNNII [9] and GRAND [12], as well as five representative scalable GNNs—FastGCN [7], GraphSAINT [31], SGC [29], GBP [8] and PPRGo [5]. For GRAND+, we implement three variants with different settings for propagation matrix $\Pi$ (Cf. Equation 5):

- **GRAND+ (P):** Truncated ppr matrix $\Pi_{\text{trunc}} = \sum_{n=0}^{N} \alpha(1 - \alpha)^n \Pi^n$.
- **GRAND+ (A):** Average pooling matrix $\Pi_{\text{avg}} = \sum_{n=0}^{N} \Pi^n / (N + 1)$.
- **GRAND+ (S):** Single order matrix $\Pi_{\text{single}} = \Pi^N$.

**Datasets.** The experiments are conducted on seven public datasets of different scales, including three widely adopted benchmark graphs—Cora, Citeseer and Pubmed [30], and four relatively large graphs—AMiner-CS [12], Reddit [14], Amazon2M [10] and MAG-Scholar-C [5]. For Cora, Citeseer and Pubmed, we use public data splits [19, 28, 30]. For AMiner-CS, Reddit, Amazon2M and MAG-Scholar-C, we use 20×classes nodes for training, 30×classes nodes for validation and the remaining nodes for test. The corresponding statistics are summarized in Table 1. More details for the setup and reproducibility can be found in Appendix A.1.

### 4.2 Results on Benchmark Datasets

To evaluate the effectiveness of GRAND+, we compare it with 10 GNN baselines on Cora, Citeseer and Pubmed. Following the community convention, the results of baseline models on the three benchmarks are taken from the previous works [9, 12, 28]. For GRAND+, we conduct 100 trials with random seeds and report the average accuracy and the corresponding standard deviation over
Table 2: Classification Accuracy (%) on Benchmarks.

<table>
<thead>
<tr>
<th>Category</th>
<th>Method</th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-batch GNNs</td>
<td>GCN</td>
<td>81.5</td>
<td>71.3</td>
<td>79.1</td>
</tr>
<tr>
<td></td>
<td>GAT</td>
<td>83.0</td>
<td>72.5</td>
<td>79.0</td>
</tr>
<tr>
<td></td>
<td>APPNP</td>
<td>84.1</td>
<td>71.6</td>
<td>79.7</td>
</tr>
<tr>
<td></td>
<td>GCNII</td>
<td>85.5</td>
<td>73.4</td>
<td>80.3</td>
</tr>
<tr>
<td></td>
<td>GRAND+ (P)</td>
<td>85.4</td>
<td>75.4</td>
<td>82.7</td>
</tr>
<tr>
<td></td>
<td>GRAND+ (A)</td>
<td>85.5</td>
<td>75.4</td>
<td>82.8</td>
</tr>
<tr>
<td></td>
<td>GRAND+ (S)</td>
<td>86.0</td>
<td>75.6</td>
<td>83.5</td>
</tr>
<tr>
<td>Scalable GNNs</td>
<td>FastGCN</td>
<td>81.4</td>
<td>68.8</td>
<td>77.6</td>
</tr>
<tr>
<td></td>
<td>GraphSAIN</td>
<td>81.3</td>
<td>70.5</td>
<td>78.2</td>
</tr>
<tr>
<td></td>
<td>SGC</td>
<td>81.0</td>
<td>71.8</td>
<td>79.0</td>
</tr>
<tr>
<td></td>
<td>GBP</td>
<td>83.9</td>
<td>72.9</td>
<td>80.6</td>
</tr>
<tr>
<td></td>
<td>PPRGo</td>
<td>82.4</td>
<td>71.3</td>
<td>80.0</td>
</tr>
<tr>
<td>Our Methods</td>
<td>GRAND+ (P)</td>
<td>85.8</td>
<td>75.6</td>
<td>84.5</td>
</tr>
<tr>
<td></td>
<td>GRAND+ (A)</td>
<td>85.5</td>
<td>75.5</td>
<td>85.0</td>
</tr>
<tr>
<td></td>
<td>GRAND+ (S)</td>
<td>85.0</td>
<td>74.5</td>
<td>84.2</td>
</tr>
</tbody>
</table>

Table 3: Accuracy (%) and Running Time (s) on Large Graphs.

<table>
<thead>
<tr>
<th>Method</th>
<th>AMiner-CS</th>
<th>Reddit</th>
<th>Amazon2M</th>
<th>MAG-Scholar-C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc</td>
<td>RT</td>
<td>Acc</td>
<td>RT</td>
</tr>
<tr>
<td>GRAND</td>
<td>53.1 ± 1.1</td>
<td>750</td>
<td>OOM</td>
<td>OOM</td>
</tr>
<tr>
<td>FastGCN</td>
<td>48.9 ± 1.6</td>
<td>69</td>
<td>89.6 ± 0.6</td>
<td>158</td>
</tr>
<tr>
<td>GraphSAIN</td>
<td>51.8 ± 1.3</td>
<td>39</td>
<td>92.1 ± 0.5</td>
<td>39</td>
</tr>
<tr>
<td>SGC</td>
<td>50.2 ± 1.2</td>
<td>9</td>
<td>92.5 ± 0.2</td>
<td>31</td>
</tr>
<tr>
<td>GBP</td>
<td>52.7 ± 1.7</td>
<td>21</td>
<td>88.7 ± 1.1</td>
<td>370</td>
</tr>
<tr>
<td>PPRGo</td>
<td>51.2 ± 1.4</td>
<td>11</td>
<td>91.3 ± 0.2</td>
<td>233</td>
</tr>
<tr>
<td>GRAND+ (P)</td>
<td>53.9 ± 1.8</td>
<td>17</td>
<td>93.3 ± 0.2</td>
<td>183</td>
</tr>
<tr>
<td>GRAND+ (A)</td>
<td>54.2 ± 1.7</td>
<td>14</td>
<td>93.5 ± 0.2</td>
<td>174</td>
</tr>
<tr>
<td>GRAND+ (S)</td>
<td>54.2 ± 1.6</td>
<td>10</td>
<td>92.8 ± 0.2</td>
<td>62</td>
</tr>
</tbody>
</table>

Figure 2: Effects of $\lambda_{max}$ and $\gamma$ on Pubmed.

On Cora and Citeseer, GRAND+ (P) achieves better results than GRAND+ (A) and GRAND+ (S); On Pubmed, Reddit and MAG-Scholar-C, GRAND+ (A) surpasses the other two variants; On AMiner-CS and Amazon2M, GRAND+ (S) gets the best classification results. This indicates that the propagation matrix plays a critical role in this task, and further suggests that GRAND+ could flexibly deal with different graphs by adjusting the generalized mixed-order matrix II.

Second, we observe GRAND+ consistently surpasses all baseline methods in accuracy and gets efficient running time on the four datasets. Importantly, on the largest graph MAG-Scholar-C, GRAND+ could succeed in training and making predictions in around 10 minutes, while SGC and GBP require more than 24 hours to finish, because the two methods are designed to directly propagate the high-dimensional raw features in pre-processing step. Compared with FastGCN and GraphSAIN, GRAND+ (S) achieves 8× and 12× acceleration respectively. When compared with PPRGo, the fastest model on this dataset in the past, GRAND+ (S) gets 4.9% improvement in accuracy while with a comparable running time. These results indicate GRAND+ scales well on large graphs and further emphasize its excellent performance.

We also report the accuracy and running time of GRAND+ on AMiner-CS. Note that it can not be executed on the other three large datasets due to the out-of-memory error. As we can see, GRAND+ achieves over 40× acceleration in terms of running time over GRAND on AMiner-CS, demonstrating the effectiveness of the proposed approximation techniques in improving efficiency.

4.4 Generalization Improvements

In this section, we quantitatively investigate the benefits of the proposed confidence-aware consistency loss $L_{con}$ to model’s generalization capability. In GRAND+, $L_{con}$ is mainly dominated by two hyperparameters: confidence threshold $\gamma$ (Equation 12) and maximum consistency loss weight $\lambda_{max}$ (Equation 13).

We first analyze the effects of $\gamma$ and $\lambda_{max}$ on GRAND+’s classification performance. Specifically, we adjust the values of $\gamma$ and $\lambda_{max}$ separately with other hyperparameters fixed, and observe how GRAND+’s accuracy changes on test set. Figure 2 illustrates the results on Pubmed dataset. From Figure 2 (a), it can be seen that the accuracy is significantly improved as $\lambda_{max}$ increases from 0 to 0.8. When $\lambda_{max}$ is greater than 0.8, the accuracy tends to be stable. This indicates that the consistency loss could really contribute to GRAND+’s performance. From Figure 2 (b), we observe model’s performance benefits from the enlargement of $\gamma$ when $\gamma$ is...
Figure 3: Training and Validation Losses on Pubmed.

Figure 4: GRAND+ w.r.t. \( k \) and \( r_{\text{max}} \) on MAG-Scholar-C.

Figure 5: Effects of propagation order \( N \) on MAG-Scholar-C.

4.5 Parameter Analysis

Threshold \( r_{\text{max}} \) and Neighborhood Size \( k \). GRAND+ uses GF-Push and top-\( k \) sparsification to approximate multiple row vectors of \( \Pi \) to perform mini-batch random propagation (Cf. Section 3.4). The approximation error of this process is mainly influenced by two hyperparameters—threshold \( r_{\text{max}} \) of GFpush and maximum neighborhood size \( k \) for sparsification. We conduct detailed experiments to better understand the effects of \( k \) and \( r_{\text{max}} \) on model’s accuracy and running time. Figure 4 illustrates the corresponding results of GRAND+ (S) w.r.t. different values of \( k \) and \( r_{\text{max}} \) on MAG-Scholar-C. As we can see, both the accuracy and running time increase when \( r_{\text{max}} \) becomes smaller, which is coincident with the conclusion of Theorem 1. While \( k \) has an opposite effect—the accuracy and running time are enlarged with the increase of \( k \). Interestingly, as \( k \) decreases from 128 to 32, the running time is cut in half with only \( \approx 2\% \) performance drop in accuracy. This demonstrates the effectiveness of the top-\( k \) sparsification strategy, which could achieve significant acceleration at little cost of accuracy.

Propagation Order \( N \). We study the influence of propagation order \( N \) on GRAND+ when using different propagation matrices. Figure 5 presents the classification performance and running time of three GRAND+ variants on MAG-Scholar-C w.r.t. different values of \( N \). As we can see, when \( N = 2 \), GRAND+ (S) achieves better accuracy and faster running time than GRAND+ (P) and GRAND+ (A). However, as \( N \) increases, the accuracy of GRAND+ (S) drops dramatically because of the over-smoothing issue, while GRAND+ (P) and GRAND+ (A) do not suffer from this problem and benefit from a larger propagation order. On the other hand, increasing \( N \) will enlarge models’ running time. In real applications, we can flexibly adjust the propagation matrix and the value of \( N \) to make desired efficiency and effectiveness.

5 CONCLUSION

We propose GRAND+, a scalable and high-performance GNN framework for graph-based semi-supervised learning. The advantages of GRAND+ include both the scalability and generalization capability while the existing state-of-the-art solutions typically feature only one of the two. To this effect, we follow the consistency regularization principle of GRAND in achieving the generalization performance, while significantly extend it to achieve scalability and retain and even exceed the flexibility and generalization capability of GRAND. To achieve these, GRAND+ utilizes a generalized mixed-order matrix for feature propagation, and uses our approximation method generalized forward push (GFPush) to calculate it efficiently. In addition, GRAND+ adopts a new confidence-aware consistency loss to achieve better consistency training. Extensive experiments show that GRAND+ not only gets the best performance on benchmark datasets, but also achieves performance and efficiency superiority over existing scalable GNNs on datasets with millions of nodes. In the future, we would like to explore more accurate approximation methods to accelerate GNNs.

ACKNOWLEDGMENTS

The work is supported by the NSFC for Distinguished Young Scholar (61825602) and Tsinghua-Bosch Joint ML Center.
REFERENCES


A APPENDIX

A.1 Implementation Note

A.1.1 Running environment. The experiments are conducted on a single Linux server with Intel(R) Xeon(R) CPU Gold 6240 @ 2.60GHz, 376G RAM and 10 NVIDIA GeForce RTX 3090TI-24GB. The Python version is 3.8.5.

A.1.2 Implementation details. We implement GPPush with C++, and use OpenMP to perform parallelization. We use Pytorch to implement the training process of GRAND+, and use pybind\(^2\) to create Python binding for approximation module. In GRAND+ and other baselines, we use BatchNorm \([15]\) and gradient clipping \([24]\) to stabilize the model training, and adopt Adam \([18]\) for optimization.

A.1.3 Dataset details. There are totally 7 datasets used in this paper, that is, Cora, Citeseer, Pubmed, AMiner-CS, Reddit, Amazon2M and MAG-Scholar-C. Our preprocessing scripts for Cora, Citeseer and Pubmed are implemented with reference to the codes of Planetoid\(^3\). Following the experimental setup used in \([19, 28, 30]\), we run 100 trials with random seeds for the results on Cora, Citeseer and Pubmed reported in Section 4.2. AMiner-CS is constructed by Feng et al. \([12]\) based on the AMiner citation network \([27]\). In AMiner-CS, each node represents a paper, the edges are citation relations, labels are research topics of papers. Reddit is published by Hamilton et al. \([14]\), in which each node represents a post in the Reddit community, a graph link represents the two posts have been commented by the same user. The task is to predict the category of each post. Amazon2M is published by Chiang et al. \([10]\), where each node represents a product, each edge denotes the two products are purchased together, labels represent the categories of products. MAG-Scholar-C is constructed by Bojchevski \([5]\) based on Microsoft Academic Graph (MAG) \([26]\), in which nodes refer to papers, edges represent citation relations among papers and features are bag-of-words of paper abstracts.

For AMiner-CS, Reddit, Amazon2M and MAG-Scholar-C, we use 20x#classes for training, 30x#classes nodes for validation and the remaining nodes for test. For Aminer, Reddit and MAG-Scholar-C, we randomly sample the same number of nodes for each class—20 nodes per class for training and 30 nodes per class for validation. For Amazon2M, we uniformly sample all the training and validation nodes from the whole datasets, as the node counts of some classes are less than 20. For these datasets, we report the average results of 10 trials with random splits.

A.1.4 Hyperparameter Selection. For results in Table 2, we adjust hyperparameters of GRAND+ on validation set, and use the best configuration for prediction, and the results of other baseline methods are taken from previous works \([9, 12, 28]\). For results in Table 3-5, we conduct detailed hyperparameter search for both GRAND+ and other GNN baselines (i.e., FastGCN, GraphSAINT, SGC, GBP and PPRLgo). For each search, we run 3 experiments with random seeds, and select the hyperparameter configuration which gets the best average accuracy on validation set. Then we train model with the selected configuration.

The hyperparameter selection for GRAND+ consists of two stages: We first conduct search for basic hyperparameters of neural network. Specifically, we search learning rate \(lr\) from \([10^{-2}, 10^{-3}, 10^{-4}]\), weight decay rate \(\ell\) from \([0, 10^{-5}, 10^{-3}, 10^{-2}]\), number of hidden layer \(Lm\) from \([1, 2]\) and dimension of hidden layer \(D\) from \([32, 64, 128, 256, 512, 1024]\).

In the second stage, we fix these basic hyperparameters as best configurations and search the following specific hyperparameters: Dropout rate \(\delta\), augmentation times per batch \(M\), threshold \(\ell_{max}\) in GPPush, maximum neighborhood size \(k\), propagation order \(N\), confidence threshold \(\gamma\), maximum consistency loss weight \(\ell_{max}\), size of unlabeled subset \([U']\) and consistency loss function \(D\). To reduce searching cost, we keep some hyperparameters fixed. Specifically, we fix \(\delta = 0.5\), \(M = 2\) and \(\gamma = \frac{1}{|\text{class}|}\) across all datasets. We set \([U'] = |U|\) for Cora, Pubmed and Citeseer, and set \(|U'| = 10000\) for other datasets. We also provide an analysis for the effect of \([U']\) in Appendix A.3. We adopt KL divergence as the consistency loss function for AMiner-CS, Reddit and Amazon2M, and use \(L\) distance for other datasets. This is because \(L\) distance is easily to suffer from gradient vanishing problem when dealing with datasets with a large number of classes. We then conduct hyperparameter selection for \(\ell_{max}\), \(N\) and \(\ell_{max}\). Specifically, we search \(\ell_{max}\) from \([10^{-5}, 10^{-6}, 10^{-7}]\), \(k\) from \([16, 32, 64, 128]\), \(N\) from \([2, 4, 6, 8, 10, 20]\) and \(\ell_{max}\) from \([0.5, 0.8, 1.0, 1.2, 1.5]\).

The selected best hyperparameter configurations of GRAND+ are reported in Table 4.

A.2 Theorem Proofs

To prove Theorem 1, we first introduce the following lemma:
LEMMA 1. For any reserve vector $q^{(n)}$, residue vector $r^{(n)}$ and random walk transition vector $P^o_n = (D^{-1}A)^n_0$ ($0 \leq n \leq N$), we have:

$$P^o_n = q^{(n)} + \sum_{i=1}^{n} (P^i_0)^T \cdot r^{(n-i)}$$

(15)

Proof. We prove the Lemma by induction. For brevity, we use $\mathcal{RHS}(n)$ to denote the right hand side of Equation 15. In Algorithm 1, $q^{(n)}$ and $r^{(n)}$ are initialized as $0$ for $1 \leq n \leq N$. $q^{(0)}$ and $q^{(n)}$ are initialized as $e^{(s)}$. Thus, Equation 15 holds at the algorithm beginning based on the following facts:

$$\mathcal{RHS}(0) = e^{(s)} = P^o_0$$

According to Lemma 1, we can conclude the following equations:

$$\|P_n - q^{(n)}\|_{1} = \sum_{i=1}^{n} \| (P^{i\cdot}_0)^T \cdot r^{(n-i)} \|_{1}$$

(17)

$$\leq \sum_{i=1}^{n} \| r^{(n-i)} \|_{1}$$

After algorithm termination, we have $0 \leq r^{(n)} \leq d_\cdot \cdot max$ for all $v \in V$. Hence,

$$\|P_n - q^{(n)}\|_{1} = \sum_{v \in V} \| P_v^n \cdot q^{(n)} - q^{(n)} \|_{1} \leq \sum_{v \in V} w_v \cdot n \cdot (2 |E| + |V|) \cdot r_{max}\max$$

Equation 17, we can conclude that $\|P_n - q^{(n)}\|_{1} \leq n \cdot (2 |E| + |V|) \cdot r_{max}\max$. Further, we have:

$$\|P_n - q^{(n)}\|_{1} \leq \sum_{n=0}^{N} w_v \cdot \cdot n \cdot (2 |E| + |V|) \cdot r_{max}\max$$

(19)

Table 5: Effects of unlabeled subset size ($|U'|$).

| $|U'|$ | Amazon | Reddit | AmazonM |
|-------|--------|--------|---------|
|       | Acc (%) | RT (s) | AT (ms) | Acc (%) | RT (s) | AT (ms) | Acc (%) | RT (s) | AT (ms) |
| 0     | 51.1 ± 1.4 | 10 | 149 | 92.3 ± 0.2 | 53 | 717 | 75.0 ± 0.7 | 63 | 2356 |
| 10^4  | 53.6 ± 1.6 | 9 | 153 | 92.6 ± 1.2 | 58 | 882 | 75.2 ± 0.5 | 62 | 2630 |
| 10^4  | 54.4 ± 1.2 | 13 | 121 | 920 ± 0.7 | 78 | 17670 | 76.3 ± 0.7 | 86 | 14250 |

Analysis for the size of $|U'|$. In GRAND+, a subset of unlabeled nodes $U'$ are sampled from $U$ for consistency regularization. To this end, we need to pre-compute the sparsiﬁed approximation $\tilde{Π}_v$ of row vector $Π_v$ for each node $v \in U'$. Here we empirically analyze how the size of $|U'|$ affects the classiﬁcation accuracy (Acc), running time (RT) and approximation time (AT) of GRAND+. Table 5 presents the results of GRAND+ (S) when we vary $|U'|$ from 0 to $10^4$ on AMiner-CS, Reddit and AmazonM. We have the two observations: First, as $|U'|$ changes from 0 (meaning the consistency loss degenerates to 0) to $10^4$, the classification performances are improved signiﬁcantly with little changes on running time, which indicates the consistency regularization serves as an economic way for improving GRAND+'s generalization performance under semi-supervised setting. Second, when $|U'|$ exceeds $10^4$, the increase rate of the accuracy will slow down, while the running time and approximation time increase more faster. This observation indicates the sampling procedure on unlabeled nodes is important for ensuring model’s efﬁciency, which also enables us to explicitly control the trade-off between effectiveness and efﬁciency of GRAND+ through the sampling size $|U'|$.  

A.3 Additional Experiments