# Scaling Graph Neural Networks with Approximate PageRank

Aleksandar Bojchevski*
Johannes Gasteiger*
a.bojchevski@in.tum.de
j.gasteiger@in.tum.de
Technical University of Munich

Bryan Perozzi
Amol Kapoor
Martin Blais
Benedek Rózemberczki
Michal Lukasik
bperozzi@acm.org
Google Research

Stephan Günnemann
guennemann@in.tum.de
Technical University of Munich



## ABSTRACT

Graph neural networks (GNNs) have emerged as a powerful approach for solving many network mining tasks. However, learning on large graphs remains a challenge – many recently proposed scalable GNN approaches rely on an expensive message-passing procedure to propagate information through the graph. We present the PPRGo model which utilizes an efficient approximation of information diffusion in GNNs resulting in significant speed gains while maintaining state-of-the-art prediction performance. In addition to being faster, PPRGo is inherently scalable, and can be trivially parallelized for large datasets like those found in industry settings.

We demonstrate that PPRGo outperforms baselines in both distributed and single-machine training environments on a number of commonly used academic graphs. To better analyze the scalability of large-scale graph learning methods, we introduce a novel benchmark graph with 12.4 million nodes, 173 million edges, and 2.8 million node features. We show that training PPRGo from scratch and predicting labels for all nodes in this graph takes under 2 minutes on a single machine, far outpacing other baselines on the same graph. We discuss the practical application of PPRGo to solve large-scale node classification problems at Google.[1]

## CCS CONCEPTS

• **Computing methodologies** → **Machine learning**.

## KEYWORDS

graph neural networks; personalized pagerank; scalability

**ACM Reference Format:**
Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling Graph Neural Networks with Approximate PageRank. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3394486.3403296

*Both authors contributed equally to this research.
[1]You can find the code and data online: https://www.daml.in.tum.de/pprgo


*KDD '20, August 23–27, 2020, Virtual Event, CA, USA*

ACM ISBN 978-1-4503-7998-4/20/08.
https://doi.org/10.1145/3394486.3403296

## 1 INTRODUCTION

Graph Neural Networks (GNNs) excel on a wide variety of network mining tasks from semi-supervised node classification and link prediction [32, 25, 44, 55] to community detection and graph classification [22, 36, 14, 40]. The success of GNNs on academic datasets has generated significant interest in scaling these methods to larger graphs for use in real-world problems [13, 12, 25, 20, 54, 27, 41, 15]. Unfortunately, there are few large graph baseline datasets available; apart from a handful of exceptions [54, 15], the scalability of most GNN methods has been demonstrated on graphs with fewer than 250K nodes. Moreover, the majority of existing work focuses on improving scalability on a single machine. Many interesting network mining problems involve graphs with billions of nodes and edges that require distributed computation across many machines. As a result, we believe most of the current literature does not accurately reflect the major challenges of large scale GNN computing.

The main scalability bottleneck of most GNNs stems from the recursive message-passing procedure that propagates information through the graph. Computing the hidden representation for a given node requires joining information from its neighbors, and the neighbors in turn have to consider *their own* neighbors, and so on. This process leads to an expensive neighborhood expansion, growing exponentially with each additional layer.

In many proposed GNN pipelines, the exponential growth of neighborhood size corresponds to an exponential IO overhead. A common strategy for scaling GNNs is to sample the graph structure during training, e.g. sample a fixed number of nodes from the $k$-hop neighborhood of a given node to generate its prediction [25, 54]. The key differences between many scalable techniques lies in the design of the sampling scheme. For example, Chen et al. [13] directly sample the receptive field for each layer using importance sampling, while Chen et al. [12] use the historical activations of the nodes as a control variate. Huang et al. [27] propose an adaptive sampling strategy with a trainable sampler per layer, and Chiang et al. [15] sample a block of nodes corresponding to a dense subgraph identified by the clustering algorithm METIS [30]. Because these approaches still rely on a multi-hop message passing procedure, there is an extremely steep trade-off between runtime and accuracy. Unfortunately, for many of the proposed methods sampling does not directly reduce the number of nodes that need to be retrieved, since e.g. we have first have to compute the importance scores [13].

Recent work shows that personalized PageRank [28] can be used to directly incorporate multi-hop neighborhood information of a node without explicit message-passing [21]. Intuitively, propagation based on personalized PageRank corresponds to infinitely many

neighborhood aggregation layers where the node influence decays exponentially with each layer. However, as proposed, Gasteiger et al. [21]'s approach does not easily scale to large graphs since it performs an expensive variant of power iteration during training.

In this work, we present PPRGo, a GNN model that scales to large graphs in both single and multi-machine (distributed) environments by using an adapted propagation scheme based on *approximate* personalized PageRank. Our approach removes the need for performing expensive power iteration during each training step by utilizing the (strong) localization properties [23, 35] of personalized PageRank vectors for real-world graphs. These vectors can be readily approximated with sparse vectors and efficiently pre-computed in a distributed manner [4]. Using the sparse pre-computed approximations we can maintain the influence of relevant nodes located multiple hops away without prohibitive message-passing or power iteration costs. We make the following contributions:

- We introduce the PPRGo model based on approximate personalized PageRank. On a graph of over 12 million nodes, PPRGo runs in under 2 minutes on a single machine, including pre-processing, training and inference time.
- We show that PPRGo scales better than message-passing GNNs, especially with distributed training in a real-world setting.
- We introduce the *MAG-Scholar* dataset (12.4M nodes, 173M edges, 2.8M node features), a version of the Microsoft Academic Graph that we augment with "ground-truth" node labels. The dataset is orders of magnitude larger than many commonly used benchmark graphs.
- Most previous work exclusively focuses on training time. We also show a significantly reduced *inference* time and furthermore propose sparse inference to achieve an additional 2x speed-up.

## 2 BACKGROUND

### 2.1 GNNs and Message-Passing

Many proposed GNN models can be analyzed using the message-passing framework proposed by Gilmer et al. [22] or other similar frameworks [5, 50, 11]. Typically, the computation is carried out in two phases: (i) messages are propagated along the neighbors; and (ii) the messages are aggregated to obtain the updated representations. At each layer, transformation of the input (e.g. linear projection plus a non-linearity) is coupled with aggregation/propagation among the neighbors (e.g. averaging). Increasing the number of layers is desirable since: (i) it allows the model to incorporate information from more distant neighbors; and (ii) it enables hierarchical feature extraction and thus the learning of richer node representations.

However, this has both computational and modelling consequences. First, the recursive neighborhood expansion at each layer implies an exponential increase in the overall number of nodes we need to aggregate to produce the output at the final layer which is computationally prohibitive for large graphs.[2] Second, it has been shown [33, 52] that naively stacking multiple layers may suffer from over-smoothing that can reduce predictive performance.

To tackle both of these challenges Gasteiger et al. [21] suggest decoupling the feature transformation from the propagation. In their PPNP model, predictions are first generated (e.g. with a neural network) for each node utilizing only that node's own features, and then propagated using an adaptation of personalized PageRank. Specifically, PPNP is defined as:

$$\boldsymbol{Z} = \operatorname{softmax}\left(\boldsymbol{\Pi}^{\text{sym}}\boldsymbol{H}\right), \qquad \boldsymbol{H}_{i,:} = f_\theta(\boldsymbol{x}_i) \tag{1}$$

where $\boldsymbol{\Pi}^{\text{sym}} = \alpha(\boldsymbol{I}_n - (1-\alpha)\tilde{\boldsymbol{A}})^{-1}$ is a symmetric propagation matrix, $\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ is the normalized adjacency matrix with added self-loops, $\alpha$ is a teleport (restart) probability, $\boldsymbol{H}$ is a matrix where each row is a vector representation for a specific node, and $\boldsymbol{Z}$ is a matrix where each row is a prediction vector for each node, after propagation. The local per-node representations $\boldsymbol{H}_{i,:}$ are generated by a neural network $f_\theta$ that processes the features $\boldsymbol{x}_i$ of every node $i$ independently. The responsibility for learning good representations is delegated to $f_\theta$, while $\boldsymbol{\Pi}^{\text{sym}}$ ensures that the representations are smoothly changing w.r.t. the graph.

Because directly calculating the dense propagation matrix $\boldsymbol{\Pi}^{\text{sym}}$ in Eq. 1 is inefficient, the authors propose a variant of power iteration to compute the final predictions instead. Unfortunately, even a moderate number of power iteration evaluations (e.g. Gasteiger et al. [21] used $K = 10$ to achieve a good approximation) is prohibitively expensive for large graphs since they need to be computed during each gradient-update step. Moreover, despite the fact that $\tilde{\boldsymbol{A}}$ is sparse, graphs beyond a certain size cannot be stored in memory.

### 2.2 Personalized PageRank and Localization

Since it is more amenable to efficient approximation we analyze the personalized PageRank matrix $\boldsymbol{\Pi}^{\text{ppr}} = \alpha(\boldsymbol{I}_n - (1-\alpha)\boldsymbol{D}^{-1}\boldsymbol{A})^{-1}$. Each row $\boldsymbol{\pi}(i) := \boldsymbol{\Pi}^{\text{ppr}}_{i,:}$ is equal to the personalized (seeded) PageRank vector of node $i$. PageRank and its many variants [37, 28, 47] have been extensively studied in the literature. Here we are interested in efficient and scalable algorithms for computing (an approximation) of personalized PageRank. Luckily, given the broad applicability of PageRank, many such algorithms have been developed [18, 4, 3, 23, 34, 45, 46, 48, 19].

Random walk sampling [18] is one such approximation technique. While simple to implement, in order to guarantee at most $\epsilon$ absolute error with probability of $1 - 1/n$ we need $O(\frac{\log n}{\epsilon^2})$ random walks. Forward search [4, 23] and backward search [3] can be viewed as deterministic variants of the random walk sampling method. Given a starting configuration, the PageRank scores are updated by traversing the out-links (respect., in-links) of the nodes.

For this work we adapt the approach by Andersen et al. [4] since it offers a good balance of scalability, approximation guarantees, and ease of distributed implementation. They show that $\boldsymbol{\pi}(i)$ can be weakly approximated with a low number of non-zero entries using a scalable algorithm that applies a series of push operations which can be executed in a distributed manner.

When the graph is strongly connected $\boldsymbol{\pi}(i)$ is non-zero for all nodes. Nevertheless, we can obtain a good approximation by truncating small elements to zero since most of the probability mass in the personalized PageRank vectors $\boldsymbol{\pi}(i)$ is *localized* on a small number of nodes [35, 23, 4]. Thus, we can approximate $\boldsymbol{\pi}(i)$ with a sparse vector and in turn approximate $\boldsymbol{\Pi}^{\text{ppr}}$ with a sparse matrix.

Once we obtain an approximation $\boldsymbol{\Pi}^{(\epsilon)}$ of $\boldsymbol{\Pi}^{\text{ppr}}$ we can either use it directly to propagate information, or we can renormalize it via $\boldsymbol{D}^{1/2}\boldsymbol{\Pi}^{(\epsilon)}\boldsymbol{D}^{-1/2}$ to obtain an approximation of the matrix $\boldsymbol{\Pi}^{\text{sym}}$.

[2] For large graphs on distributed storage, just gathering the required neighborhood data requires many expensive remote procedure calls that greatly increase run time.

## 2.3 Related work

**Scalability**. GNNs were first proposed in Gori et al. [24] and in Scarselli et al. [42] and have since emerged as a powerful approach for solving many network mining tasks [8, 16, 32, 22, 44, 42, 1, 2]. Most GNNs do not scale to large graphs since they typically need to perform a recursive neighborhood expansion to compute the hidden representations of a given node. While several approaches have been proposed to improve the efficiency of graph neural networks [13, 12, 25, 20, 54, 27, 41, 49, 15], the scalability of GNNs to massive (web-scale) graphs is still under-studied. As we discussed in § 1 the most prevalent approach to scalability is to sample a subset of the graph, e.g. based on different importance scores for the nodes [25, 54, 20, 41, 15].[3] Beyond sampling, Gao et al. [20] collect the representations from a node's neighborhood into a matrix, sort independently along each column/feature, and use the $k$ largest entries as input to a 1-dimensional CNN. These techniques all focus on single-machine environments with limited (GPU) memory.

Buchnik et al. [9] propose feature propagation which can be viewed as a simplified linearized GNN. They perform graph-based smoothing as a preprocessing step (before learning) to obtain diffused node features which are then used to train a logistic regression classifier to predict the node labels. Wu et al. [49] propose an equivalent simple graph convolution (SGC) model and diffuse the features by multiplication with the k-th power of the normalized adjacency matrix. However, node features are often high dimensional, which can make the preprocessing step computationally expensive. More importantly, while node features are typically sparse, the obtained diffused features become denser, which significantly reduces the efficiency of the subsequent learning step. Both of these approaches are a special case of the PPNP model [21] which experimentally shows higher classification performance [21, 17].

**Approximating PageRank**. Recent approaches combine basic techniques to create algorithms with enhanced guarantees [34, 45, 46]. For example Wei et al. [48] propose the TopPPR algorithm combining the strengths of random walks and forward/backward search simultaneously. They can compute the top $k$ entries of a personalized PageRank vector up to a given precision using a filter-and-refine paradigm. Another family of approaches [19] are based on the idea of maintaining upper and lower bounds on the PageRank scores which are then used for early termination with certain guarantees. For our purpose the basic techniques are sufficient.

## 3 THE PPRGO MODEL

The design of our model is motivated by: (i) the insights from § 2.1, namely that we can decouple the feature transformation from the information propagation, and (ii) the insights from § 2.2, namely that we can approximate $\mathbf{\Pi}^{\text{ppr}}$ with a sparse matrix. Analogous to Eq. 1 we define the final predictions of our model (see Fig. 1):

$$\boldsymbol{Z} = \text{softmax}\big(\mathbf{\Pi}^{(\epsilon)}\boldsymbol{H}\big), \qquad \boldsymbol{H}_{i,:} = f_\theta(\boldsymbol{x}_i) \tag{2}$$

where $\mathbf{\Pi}^{(\epsilon)}$ is a sparse approximation of $\mathbf{\Pi}^{\text{ppr}}$. To obtain each row of $\mathbf{\Pi}^{(\epsilon)}$ we adapt the push-flow algorithm described in Andersen et al. [4]. We additionally truncate $\mathbf{\Pi}^{(\epsilon)}$ to contain only the top $k$ largest entries for each row. That is, for each node $i$ we only consider the set of nodes with top $k$ largest scores according to $\boldsymbol{\pi}(i)$. Combined, the predictions for a given node $i$ are:

$$\boldsymbol{z}_i = \text{softmax}\Bigg(\sum_{j \in \mathcal{N}^k(i)} \boldsymbol{\pi}^{(\epsilon)}(i)_j \boldsymbol{H}_j\Bigg) \tag{3}$$

where $\mathcal{N}^k(i)$ enumerates the indices of the top $k$ largest non-zero entries in $\boldsymbol{\pi}^{(\epsilon)}(i)$. Eq. 3 highlights that we only have to consider a small number of other nodes to compute the final prediction for a given node. Furthermore, this definition allows us to explicitly trade-off scalability and performance by increasing/decreasing the number of neighbors $k$ we take into account. We can achieve a similar trade-off by changing the threshold $\epsilon$ which effectively controls the norm of the residual. We show the pseudo-code for computing $\boldsymbol{\pi}^{(\epsilon)}$ in Algorithm 1. For further details see § A.4.

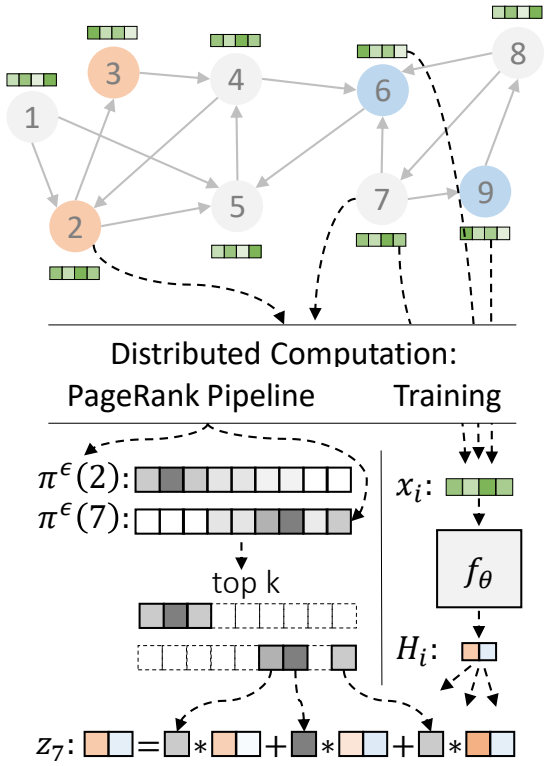



**Figure 1: An illustration of PPRGo. For each node $i$ we pre-compute an approximation of its personalized PageRank vector $\boldsymbol{\pi}^{(\epsilon)}(i)$. The approximation is computed efficiently and in parallel using a distributed batch data processing pipeline. The final prediction $\boldsymbol{z}_i$ is then generated as a weighted average of the local (per-node) representations $\boldsymbol{H}_{j,:} = f_\theta(\boldsymbol{x}_j)$ for the top $k$ nodes ordered by largest personalized PageRank score $\boldsymbol{\pi}(i)_j$. To train the model $f_\theta(\cdot)$ that maps node attributes $\boldsymbol{x}_i$ to local representations $\boldsymbol{H}_i$, we only need the personalized PageRank vectors of the training nodes and attributes of the their respective top $k$ neighbors. The model is trained in a distributed manner on multiple batches of data in parallel.**

[3] The importance sampling score by Ying et al. [54] can be seen as an approximation of the non-personalized PageRank, however the number of random walks required to achieve a good approximation is relatively high [18] making it a suboptimal choice.

**Algorithm 1** Approximate personalized PageRank $(G, \alpha, t, \epsilon)$ [4]

**Inputs:** Graph $G$, teleport prob. $\alpha$, target node $t$, max. residual $\epsilon$

Initialize the (sparse) estimate-vector $\boldsymbol{\pi}^{(\epsilon)} = \mathbf{0}$ and the (sparse) residual-vector $\boldsymbol{r} = \alpha \cdot \boldsymbol{e}_t$ (i.e. $\boldsymbol{e}_t = 1$, $\boldsymbol{e}_v = 0, v \neq t$)
**while** $\exists v \, s.t. \, \boldsymbol{r}_v > \alpha \cdot \epsilon \cdot \boldsymbol{d}_v$ **do** # $\boldsymbol{d}_v$ is the out-degree
  $\boldsymbol{\pi}_v^{(\epsilon)}$ += $\boldsymbol{r}_v$
  $\boldsymbol{r}_v = 0$
  $m = (1 - \alpha) \cdot \boldsymbol{r}_v / \boldsymbol{d}_v$
  **for** $u \in \mathcal{N}_G^{\text{out}}(v)$ **do** # $v$'s outgoing neighbors
    $\boldsymbol{r}_u$ += $m$
  **end for**
**end while**
**return** $\boldsymbol{\pi}^{(\epsilon)}$

In contrast to the PPNP model, a big advantage of PPRGo is that we can pre-compute the sparse matrix $\mathbf{\Pi}^{(\epsilon)}$ before we start training. Pre-computation allows PPRGo to calculate the training and inference predictions in $O(k)$ time, where $k \ll N$, and $N$ is number of nodes. Better still, for training we only require the rows of $\mathbf{\Pi}^{(\epsilon)}$ corresponding to the training nodes and the representations $f_\theta(\boldsymbol{x}_i)$ of their top-k neighbors. Furthermore, our model lends itself nicely to batched computation. For example, for a batch of nodes of size $b$ we have to load in memory the features of at most $b \cdot k$ nodes. In practice, this number is smaller than $b \cdot k$ since the nodes that appear in $\mathcal{N}^k(i)$ often overlap for the different nodes in the batch. We discuss the applicability and limitations of PPRGo in § A.5.

### 3.1 Effective Neighborhood, $\alpha$ and $k$

From the definition of personalized PageRank we can conclude that the hyper-parameter $\alpha$ controls the amount of information we are incorporating from the neighborhood of a node. Namely, for values of $\alpha$ close to 1 the random walks return (teleport) to the node $i$ more often and we are therefore placing more importance on the immediate neighborhood of the node. As the value of $\alpha$ decreases to 0 we instead give more and more importance to the extended (multi-hop) neighborhood of the node. Intuitively, the importance of the $k$-hop neighborhood is proportional to $(1 - \alpha)^k$. Note that the importance that each node assigns to itself (i.e. the value of $\boldsymbol{\pi}(i)_i$) is typically higher than the importance it assigns to the rest of the nodes. In conjunction with $\alpha$, we can modify the number of $k$ largest entries we consider to increase or decrease the size of the effective neighborhood. This stands in stark contrast to message-passing frameworks, where incorporating information from the extended neighborhood requires additional layers, thereby significantly increasing the computational complexity.

## 4 SCALABILITY

Here we discuss the properties of PPRGo which make it suitable for large-scale classification problems occurring in industry.

### 4.1 Node Classification in the Real World

The web is an incredibly rich data source and many different large graphs (potentially with *hundreds of billions* of nodes and edges) can be derived from it. Many web graphs have interesting node classification problems that can be addressed via semi-supervised learning. Their applications occur across all media types and power many different Google products [29, 39, 38]. In web-scale datasets, the node sets are large, the graphs commonly have power-law degrees, the datasets change frequently, and labels can quickly become stale. Therefore, having a model that trains as fast as possible is desirable to reduce the latency. Arguably even more important is having a model for which inference is as fast as possible, since inference is typically performed much more frequently than training in real-world settings. A low enough inference time may even open the door to using the model for online tasks, an impactful domain of problems where these models have limited penetration. Our proposed model, PPRGo, ameliorates many of the difficulties associated with scaling these learning systems. We have successfully tested it on internal graphs with billions of nodes and edges.

### 4.2 Distributed Training

In contrast to most previously proposed methods [54, 25, 49] we utilize distributed computing techniques which significantly reduce the overall runtime of our method. Our model is trained in two stages. First, we pre-compute the approximated personalized PageRank vectors using the distributed version of Algorithm 1 (see § A.4). Second, we train the model parameters with stochastic gradient descent. Both stages are implemented in a distributed fashion.

For the first stage we use an efficient batch data processing pipeline [10] similar to MapReduce. Since we can compute the PageRank vectors for every node in parallel our implementation easily scales to graphs with billions of nodes. Moreover, we can *a priori* determine the number of iterations we need for achieving a desired approximation accuracy [23, 4] which in turn means we can reliably estimate the runtime beforehand.

We implement PPRGo in Tensorflow and optimize the parameters with *asynchronous* distributed stochastic gradient descent. We store the model parameters on a parameter server (or several parameter servers depending on the model size) and multiple workers process the data in parallel. We use asynchronous training to avoid the communication overhead between many workers. Each worker fetches the most up-to-date parameters and computes the gradients for a mini-batch of data independently of the other workers.

### 4.3 Efficient Inference

As discussed in § 3 we only need to compute the approximate personalized PageRank vectors for the nodes in the training/validation set in order to train the model. In the semi-supervised classification setting these typically comprise only a small subset of all nodes (a few 100s or 1000s). However, during inference we still need to compute the PPR vector for every test node (see Eq. 3). Specifically, to predict the class label for $m < n$ test nodes we have to compute $Z = \text{softmax}(\mathbf{\Pi} \boldsymbol{H})$ where $\mathbf{\Pi}$ is a $m \times n$ matrix such that each row contains the personalized PageRank vector for a given test node, and $\boldsymbol{H}$ is a $n \times c$ matrix of logits. Even though the computation of each of these $m$ PPR vectors can be trivially parallelized, when $m$ is extremely large the overall runtime can still be considerable. However, during inference we only use the PPR vectors a single time. In this case it is more efficient to circumvent this calculation and fall back to power iteration, i.e.

$$\boldsymbol{Q}^{(0)} = \boldsymbol{H}, \qquad \boldsymbol{Q}^{(p+1)} = (1 - \alpha) \boldsymbol{D}^{-1} \boldsymbol{A} \boldsymbol{Q}^{(p)} + \alpha \boldsymbol{H}. \quad (4)$$

We furthermore found that, as opposed to training, during inference only very few (i.e. 1-3) steps of power iteration are necessary until accuracy improvements level off (see § 5.5). Hence we only need very few sparse matrix-matrix multiplications for inference, which can be implemented very efficiently.

Since this truncated power iteration is very fast to compute, the neural network $f_\theta$ quickly becomes the limiting factor for inference time, especially if it is computationally expensive (e.g. a deep ResNet architecture [26] or recurrent neural network (RNN)). With PPRGo, we can leverage the graph's homophily to reduce the number of nodes that need to be analyzed. Since nearby nodes are likely to be similar we only need to calculate predictions $\boldsymbol{H}$ for a small, randomly chosen fraction of nodes. Setting the remaining entries to zero we can smooth out these sparse labels over the rest via Eq. 4.

In the very sparse case, using homophily to limit the number of needed predictions can be viewed as a label propagation problem with labels given by logits $\boldsymbol{H}$. In the context of label propagation, the power iteration in Eq. 4 is a common algorithm known as "label propagation with return probability". This algorithm is known to perform well; we find that we can almost match the performance of full prediction with only a small fraction (e.g. 10 % or 1 %) of logits (see § 5.5). Overall, this approach allows us to reduce the runtime even below a model that ignores the graph and instead considers each node independently, without sacrificing accuracy.

# 5 EXPERIMENTS

**Setup.** We focus on semi-supervised node classification on attributed graphs and demonstrate the strengths and scalability of PPRGo in both distributed and single-machine environments. To best align with real use cases we only use 20 · number of classes uniformly sampled (non-stratified) training nodes. We fix the value of the teleport parameter to a common $\alpha = 0.25$ for all experiments except the unusually dense Reddit dataset, where $\alpha = 0.5$. For details regarding training, hyperparameters, and metrics see § A.3 in the appendix. We answer the following research questions:

- What kind of trade-offs between scalability and accuracy can we achieve with PPRGo? (§ 5.2)
- How effectively can we leverage distributed training? (§ 5.3)
- How much resources (memory, compute) does PPRGo need compared to other scalable GNNs? (§ 5.4)
- How efficient is the proposed sparse inference scheme? (§ 5.5)

## 5.1 Large-Scale Datasets

The majority of previous approaches are evaluated on a small set of publicly available benchmark datasets [13, 12, 25, 20, 27, 41, 49, 2]. The size of these datasets is relatively small, with the Reddit graph (233K nodes, 11.6M edges, 602 node features) [25] typically being the largest graph used for evaluation.[4] Chiang et al. [15] recently introduced the Amazon2M graph (2.5M nodes, 61M edges, 100 node features) which is large in terms of number of nodes, but tiny in terms of node feature size.[5]

[4]The Twitter geo-location datasets used in previous work [49] have limited usefulness for evaluating GNNs since they have no meaningful graph structure, e.g. 70% of the nodes in the Twitter-World dataset only have a self-loop and no other edges.

[5]While larger benchmark graphs can be found in the literature, they either do not have node features or they do not have "ground-truth" node labels.

**MAG-Scholar.** To facilitate the development of scalable GNNs we create a new benchmark dataset based on the Microsoft Academic Graph (MAG) [43]. Nodes represent papers, edges denote citations, and node features correspond to a bag-of-words representation of paper abstracts. We augmented the graph with "ground-truth" node labels corresponding to the papers' field of study.

We extract the node labels semi-automatically by mapping the publishing venues (conferences and journals) to a field of study using metadata on the top venues from Google Scholar. We create two sets of labels for the same graph. Coarse-grained labels correspond to the following 8 coarse-grained fields of study: biology, engineering, humanities, medicine, physics, sociology, business, and other. We refer to this graph as MAG-Scholar-C. Fine-grained labels correspond to 253 fine-grained fields of study such as: architecture, epidemiology, geology, ethics, anthropology, linguistics, etc. The fine-grained labels make the classification problem more difficult. We refer to this graph as MAG-Scholar-F.

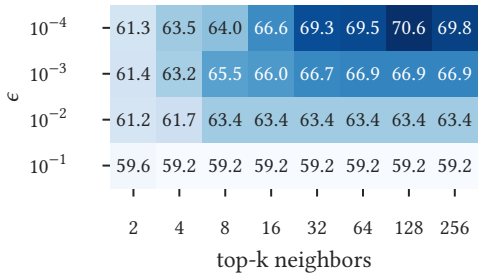
The resulting *MAG-Scholar* graph is a few orders of magnitude larger then the commonly used benchmark graphs (12.4M nodes, 173M edges, 2.8M node features). The graphs and the code to generate them will be made publicly available. See § A.2 for a detailed description of the graph construction and node labelling process.

## 5.2 Scalability vs. Accuracy Trade-off

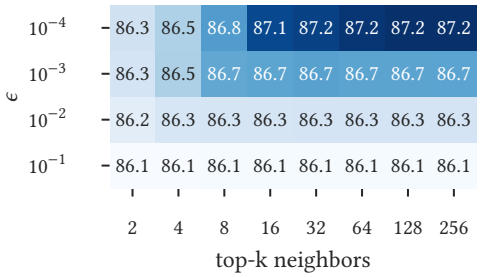
The approximation parameter $\epsilon$ and the number of top-$k$ nodes are important hyper-parameters that modulate scalability and accuracy (see Eq. 3). We note that $\alpha$ and $k$ play similar roles, so we choose to analyze $k$ for a fixed $\alpha$. To examine their effect on the performance of PPRGo we train our model on the MAG-Scholar-C graph for different values of $k$ and $\epsilon$. We repeat the experiment five times and report the mean performance. We investigate two cases: a sparsely labeled scenario similar to industry settings (160 nodes), and an "academic" setting with many more labeled nodes (105415 nodes).

As expected, we can see in Fig. 2 that the performance consistently increases if we either use a more accurate approximation of the PageRank vectors (smaller $\epsilon$) or a larger number of top-k neighbors. This also shows that we can smoothly trade-off performance for scalability since models with higher value of $k$ and lower value of $\epsilon$ are computationally more expensive. For example, in the academic setting (Fig. 2b) a model with $\epsilon = 0.1, k = 2$ had an overall (preprocessing + training + inference) runtime of 6 minutes, while a model with $\epsilon = 0.001, k = 256$ had an overall runtime of 12 minutes. Since many nodes are labeled (1 %) the difference between the highest accuracy (top right corner) and lowest accuracy (bottom left corner) is under 2 % and the model is not sensitive to the hyperparameters. In the sparsely labeled setting (Fig. 2a) the choice of hyperparameters is more important and depends on the desired trade-off level (slowest overall runtime was <2 minutes).

Interestingly, we can see on Fig. 2 that for any value of $\epsilon$ the performance starts to plateau at around top-$k = 32$. The reason for this behavior becomes more clear by examining Fig. 3. Here, for each node $i$ we calculate the sum of the top-$k$ largest scores in $\boldsymbol{\pi}^{(\epsilon)}(i)$ and we plot the average across all nodes. We see that by looking at a very few nodes – e.g. 32 out of 12.4 million – we are able to capture the majority of the PageRank scores on average (recall that $\sum_j \boldsymbol{\pi}^{(\epsilon)}(i)_j \leq 1$). Therefore, the curves in both Fig. 2 and




**(a) Sparsely labeled setting (160 nodes, 0.0015 %)**




**(b) Setting with a large number of labeled nodes (105415 nodes, 1 %)**

**Figure 2: Mean accuracy (%) over 5 runs on MAG-Scholar-C as we vary the number of neighbors and the approx. parameter $\epsilon$.**

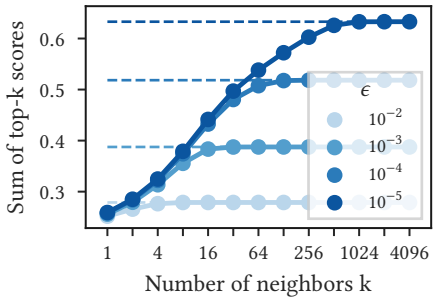



**Figure 3: For each node in MAG-Scholar-C we calculate the sum of the top-$k$ largest scores in $\pi^{(\epsilon)}(i)$ and we plot the average across all nodes for different values of $\epsilon$. The dashed line indicates $k = n$, i.e. the entire sum of $\pi^{(\epsilon)}(i)$ averaged across nodes. The 95% confidence intervals around the mean (estimated with bootstrapping) are too small to be visible.**

Fig. 3 plateau around the same value of $k$. These figures validate our approach of approximating the dense (but localized) personalized PageRank vectors with their respective sparse top-$k$ versions.

## 5.3 Distributed Training

In this section we aim to compare the performance of one-hop propagation using personalized PageRank and traditional multi-hop message passing propagation in a real distributed environment at Google. To make sure that the differences we observe are only due to the used model and not other factors, we implement simple 2-hop and 3-hop GNN models [32], which are also trained in a distributed manner using the same infrastructure as PPRGo. Specifically, we make sure that both the multi-hop models and PPRGo consider the same number of neighbors, e.g. if PPRGo uses $k = 64$ then the 2-hop model uses information from $8 \times 8 = 64$ nodes from its first and second hop respectively. To select these neighborhoods we use a weighted sampling scheme similar to previous work [54]. Additionally, we implement a distributed version of FastGCN [13] to evaluate the effect of different sampling schemes.

Our first observation is that there is no significant difference in terms of predictive performance between the different models (around 61% accuracy). However, there is a significant difference in terms of runtime. On Fig. 4 we show the speedup in terms of number of gradient-update steps per second on the MAG-Scholar-F graph as we increase the number of worker machines used for distributed training. Specifically, we show the relative speedup compared to the baseline method – 2-hop GCN on a single worker. We see that PPRGo is considerably faster than the baseline (note that both axes are on a log-scale). PPRGo also requires fewer steps in total to converge. Moreover, the speedup gap between the 2 hop model and PPRGo increases with the number of workers. Crucially, since we have to fetch all neighbors to calculate their importance scores and since the runtime in the distributed setting is dominated by IO we see that FastGCN does not offer any significant scalability advantage over the baseline GNN 2-hop model. In § A.1 we additionally analyze parallel efficiency, i.e. how well the different models utilize the additional workers.

PPRGo is able to process all top-$k$ neighbors at once, compared to the multi-hop models which have to recursively update the hidden representations. Therefore, while increasing the number of top-$k$ neighbors makes all model computationally more expensive, we expect the runtime of PPRGo to increase the least. To validate this claim, we analyze the relative speed (number of gradient updates

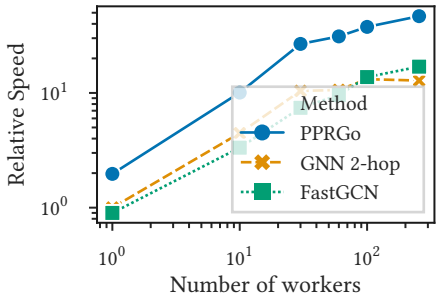



**Figure 4: Relative speed in terms of number of gradient-update steps per second on the MAG-Scholar-F graph compared to the baseline method (GNN 2-hop, single worker). Both axes are on a log scale. PPRGo is consistently the fastest method and can best utilize additional workers.**

**Table 1: Breakdown of the runtime, memory, and predictive performance on a single machine for different models on the Reddit dataset. We use 820 (20 · #classes) nodes for training. We see that PPRGo has a total runtime of less than 20 s and is two orders of magnitude faster than SGC and Cluster-GCN. PPRGo also requires less memory overall.**

| | Runtime (s) | | | | | | | Memory (GB) | | Accuracy (%) |
|---|---|---|---|---|---|---|---|---|---|---|
| | Preprocessing | Training | | Inference | | | Total | RAM | GPU | |
| | | Per Epoch | Overall | Forward | Propagation | Overall | | | | |
| Cluster-GCN | 1175(25) | 4.77(12) | 953(24) | - | - | 186(21) | 2310(40) | 20.97(15) | 0.071(6) | 17.1(8) |
| SGC | 313(9) | 0.0026(2) | 0.53(3) | - | - | 7470(150) | 7780(150) | 10.12(3) | 0.027 | 12.1(1) |
| PPRGo (1 PI step) | 2.26(4) | 0.0233(5) | 4.67(10) | 0.341(9) | 5.85(3) | 6.19(4) | 13.10(7) | 5.560(19) | 0.073 | 26.5(19) |
| PPRGo (2 PI steps) | 2.22(12) | 0.021(3) | 4.1(7) | 0.43(8) | 10.1(14) | 10.5(15) | 16.8(17) | 5.42(18) | 0.073 | 26.6(18) |

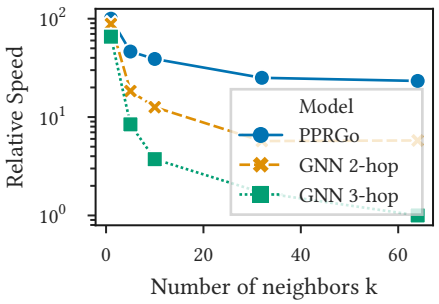



**Figure 5: Relative speed comparison (num. gradient updates per second) between PPRGo and multi-hop models for different values of $k$ on MAG-Scholar-F. Distributed training.**

per second) compared to the slowest method at different values of $k$. The results in Fig. 5 exactly match our intuition, and indeed the curve of PPRGo has the smallest slope as we increase $k$, while the relative speed of the 2- and 3-hop GNNs deteriorate faster. FastGCN again matches GNN 2-hop, like it does in Fig. 4 (not shown here).

## 5.4 Runtime and Memory on a Single Machine

**Setup.** To highlight the benefits of PPRGo we compare the runtime, memory, and predictive performance with SGC [49] and Cluster-GCN [15], two strong baselines that represent the current state-of-the-art scalable GNNs. Since SGC and Cluster-GCN report significant speedup over FastGCN [13] and VRGCN [12] we omit these models from our comparison.

We run the experiments on Nvidia 1080Ti GPUs and on Intel CPUs (5 cores), using CUDA and TensorFlow. We run each experiment on five different random splits and report mean values and standard deviation. For SGC we use the second power of the graph Laplacian as suggested by the authors (i.e. we effectively have a 2-hop model). For PPRGo we set $\epsilon = 10^{-4}$ and $k = 32$ following the discussion in § 5.2. We compute the overall runtime including the preprocessing time, the time to train the models, and the time to perform inference for all test nodes. This is in contrast to previous work which rarely report preprocessing time and almost never report inference time. For training, we report both the overall training time, as well as the time per epoch.

**Preprocessing time.** For each model, during preprocessing we perform the computation only for the training nodes. For SGC, the preprocessing step involves computing the diffused features using the second power of the graph Laplacian. We significantly optimized preprocessing for Cluster-GCN, resulting in node cluster computation with METIS [30] becoming its main bottleneck. For PPRGo, the preprocessing step involves computing the approximate personalized PageRank vectors using Algorithm 1 and selecting the top $k$ neighbors.

**Inference time.** For SGC, during inference we have to compute the diffused features for the test nodes (again using the second power of the graph Laplacian). Following the implementation by the original authors of Cluster-GCN, we do not cluster the test nodes, but rather perform "standard" message-passing inference on the full graph. For PPRGo, as discussed in § 4.3, we run power iteration rather than computing the approximate PPR vectors for the test nodes. Two iteration steps were already sufficient to obtain good accuracy. We analyze the inference step in more detail in § 5.5.

The results when training a model on the Reddit dataset (233K nodes, 11.6M edges, 602 node features) are summarized in Table 1. Both SGC and Cluster-GCN are several orders of magnitude slower than PPRGo . Interestingly, SGC is significantly slower w.r.t. inference time (since we have to compute the diffused features for all test nodes) while Cluster-GCN is significantly slower w.r.t. preprocessing and training time. The overall runtime of Cluster-GCN (2310 s) and SGC (7470 s) is in stark contrast to our proposed approach: under 20 s. Moreover, we see that the amount of memory used by PPRGo is 4 times smaller compared to Cluster-GCN and 2 times smaller compared to SGC. Given that Cluster-GCN and SGC achieve significantly worse accuracy, the benefits of our proposed approach in terms of scalability are apparent.

We extend the above analysis to several other datasets. We chose two comparatively small academic graphs that are commonly used as benchmark datasets – Cora-Full [6] (18.7K nodes, 62.4K edges, 8.7K node features) and PubMed [53] (19.7K nodes, 44.3K edges, 0.5K node features) – as well as our newly introduced MAG-Scholar-C dataset (10.5M nodes, 133M edges, 2.8M node features). In addition to the two scalable baselines, we also evaluate how PPRGo compares to the APPNP model [21] which we build upon. The results are summarized in Table 2. We can see that the performance of most models is comparable in terms of accuracy. In most cases our proposed model PPRGo has the smallest overall runtime and it always uses the least amount of memory. PPRGo's comparatively long runtime on Cora-Full can be explained by its training set size: The training set is so large that PPRGo accesses more neighbors per batch than there are nodes in this graph, not leveraging the

**Table 2: Single machine runtime (s), memory (GB), and accuracy (%) for different models and datasets using $20 \cdot$ #classes training nodes. PPRGo shows comparable accuracy and scales much better to large datasets than its competitors.**

| | Cora-Full | | | PubMed | | | Reddit | | | MAG-Scholar-C | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | Mem. | Acc. | Time | Mem. | Acc. | Time | Mem. | Acc. | Time | Mem. | Acc. |
| Cluster-GCN | 84(4) | 2.435(18) | 58.0(7) | 54.3(27) | 1.90(3) | 74.7(30) | 2310(50) | 21.04(15) | 17.1(8) | >24h | - | - |
| SGC | 92(3) | 3.95(3) | 58.0(8) | 5.3(3) | 2.172(4) | 75.7(23) | 7780(140) | 10.15(3) | 12.1(1) | >24h | - | - |
| APPNP | 10.7(5) | 2.150(19) | 62.8(11) | 6.5(4) | 1.977(4) | 76.9(26) | - | OOM | - | - | OOM | - |
| PPRGo ($\epsilon = 10^{-4}, k = 32$) | 25(3) | 1.73(3) | 61.0(7) | 3.8(9) | 1.626(25) | 75.2(33) | 16.8(17) | 5.49(18) | 26.6(18) | 98.6(17) | 24.51(4) | 69.3(31) |
| PPRGo ($\epsilon = 10^{-2}, k = 32$) | 6.6(5) | 1.644(13) | 58.1(6) | 2.9(5) | 1.623(17) | 73.7(39) | 16.3(17) | 5.61(6) | 26.2(18) | 89(5) | 24.49(5) | 63.4(29) |

duplicate information. This can only happen with small graphs, for which runtime is not an issue. We see that the APPNP model runs out of memory for even the moderately sized Reddit graph, highlighting the necessity of our approach.

More importantly, on the largest graph MAG-Scholar-C, we successfully trained PPRGo from scratch and obtained the predictions for all test nodes in under 2 minutes, while Cluster-GCN and SGC were not able to finish in over 24 hours, with Cluster-GCN still stuck in preprocessing.

## 5.5 Efficient Inference

Inference time is crucial for real-world applications since a machine learning model needs to be trained only once, while inference is run continuously when the model is put into production. We found that PPRGo can achieve an accuracy of 68.7 % with a *single* power iteration step, i.e. without even calculating the PPR vectors. At this point, the neural network $f_\theta$ and not the propagation becomes the limiting factor. However, as described in § 4.3, we can reduce the neural network cost by only computing logits for a small, random subset of nodes. Fig. 6 shows that the accuracy only reduces by around 0.6 percentage points when reducing the number of inferred nodes by a *factor of 10*. We can therefore trade in a small amount of accuracy to significantly reduce inference time, in this case by 50 %. With this approximation, PPRGo has a *shorter* inference time than the forward pass of a simple neural network executed on each node independently. Furthermore, note that we use a rather simple feed-forward neural network in our experiments. This reduction will become even more dramatic in cases that leverage more computationally expensive neural networks for $f_\theta$. Fig. 7 shows that when reducing the fraction of inferred nodes, the corresponding accuracy drops off earlier if we perform fewer power iteration steps $p$. Therefore, we need to increase the number of power iteration steps when we calculate fewer logits. This furthermore shows that subsampling logits would not be possible with methods that use locally sampled subgraphs (e.g. FastGCN). Note that we do not use this additional improvement in Table 1 and 2.

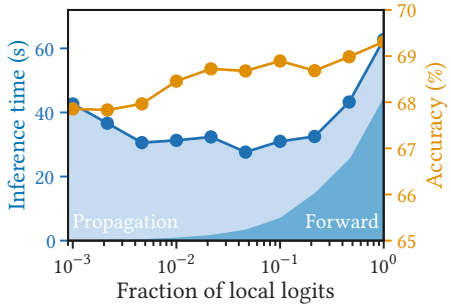



**Figure 6: Accuracy and corresponding inference time (NN inference (dark blue) + propagation (light blue)) on MAG-Scholar-C w.r.t. the fraction of nodes for which local logits $H$ are inferred by the NN. PPRGo performs very well even if the NN is evaluated on very few nodes. We need more power iteration steps $p$ if we do fewer forward passes (see Fig. 7), increasing the propagation time. Note the logarithmic scale.**

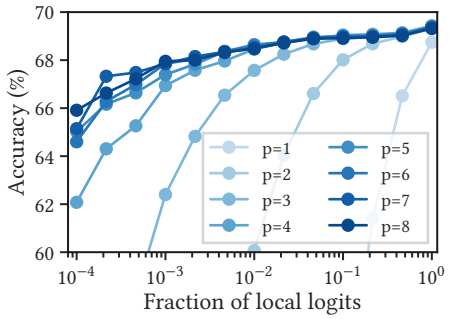



**Figure 7: Accuracy on MAG-Scholar-C w.r.t. the fraction of nodes for which local logits $H$ are inferred and number of power iteration steps $p$. The fewer logits we calculate, the more power iteration steps we need for stabilizing the prediction. Note the logarithmic scale.**

## 6 CONCLUSION

We propose a GNN for semi-supervised node classification that scales easily to graphs with millions of nodes. In comparison to previous work our model does not rely on expensive message-passing, making it well suited for use in large-scale distributed environments. We can trade scalability and performance via a few intuitive hyperparameters. To stimulate the development of scalable GNNs we present MAG-Scholar – a new large-scale graph (12.4M nodes, 173M edges, and 2.8M node features) with coarse/fine-grained "ground-truth" node labels. On this web-scale dataset PPRGo achieves high performance in under 2 minutes (preprocessing + training + inference time) on a single machine. Beyond the single machine scenario, we demonstrate the scalability of PPRGo in a distributed setting and show that it is more efficient compared to multi-hop models.

## 7 ACKNOWLEDGEMENTS

We would like to thank Chandan Yeshwanth for his assistance with conducting the experiments. This research was supported by the Deutsche Forschungsgemeinschaft (DFG) through the Emmy Noether grant GU 1409/2-1 and the TUM International Graduate School of Science and Engineering (IGSSE), GSC 81.

## REFERENCES

[1] S. Abu-El-Haija, A. Kapoor, B. Perozzi, and J. Lee. 2018. N-gcn: multi-scale graph convolution for semi-supervised node classification. *arXiv preprint arXiv:1802.08888*.

[2] S. Abu-El-Haija et al. 2019. MixHop: higher-order graph convolutional architectures via sparsified neighborhood mixing. *ICML*, 21–29.

[3] R. Andersen, C. Borgs, J. T. Chayes, J. E. Hopcroft, V. S. Mirrokni, and S.-H. Teng. 2008. Local computation of pagerank contributions. *Internet Mathematics*, 5, 23–45.

[4] R. Andersen, F. Chung, and K. Lang. 2006. Local graph partitioning using pagerank vectors. *FOCS*, 475–486.

[5] P. W. Battaglia et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

[6] A. Bojchevski and S. Günnemann. 2018. Deep gaussian embedding of graphs: unsupervised inductive learning via ranking.

[7] P. Boldi, V. Lonati, M. Santini, and S. Vigna. 2006. Graph fibrations, graph isomorphism, and pagerank. *RAIRO Theor. Informatics Appl.*, 40, 2, 227–253.

[8] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.

[9] E. Buchnik and E. Cohen. 2018. Bootstrapped graph diffusions: exposing the power of nonlinearity. *SIGMETRICS*, 8–10.

[10] C. Chambers, A. Raniwala, F. Perry, S. Adams, R. R. Henry, R. Bradshaw, and N. Weizenbaum. 2010. Flumejava: easy, efficient data-parallel pipelines. *ACM Sigplan Notices*, 45, 6, 363–375.

[11] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy. 2020. Machine learning on graphs: a model and comprehensive taxonomy. *arXiv preprint arXiv:2005.03675*.

[12] J. Chen, J. Zhu, and L. Song. 2018. Stochastic training of graph convolutional networks with variance reduction. *ICML*, 941–949.

[13] J. Chen, T. Ma, and C. Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*.

[14] Z. Chen, L. Li, and J. Bruna. 2018. Supervised community detection with line graph neural networks.

[15] W. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C. Hsieh. 2019. Cluster-gcn: an efficient algorithm for training deep and large graph convolutional networks. *KDD*. ACM, 257–266.

[16] M. Defferrard, X. Bresson, and P. Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems*, 3844–3852.

[17] M. Fey and J. E. Lenssen. 2019. Fast graph representation learning with pytorch geometric. *CoRR*, abs/1903.02428. arXiv: 1903.02428.

[18] D. Fogaras and B. Rácz. 2004. Towards scaling fully personalized pagerank. *WAW*.

[19] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka. 2013. Fast and exact top-k algorithm for pagerank. *AAAI*.

[20] H. Gao, Z. Wang, and S. Ji. 2018. Large-scale learnable graph convolutional networks. *KDD*, 1416–1424.

[21] J. Gasteiger, A. Bojchevski, and S. Günnemann. 2019. Predict then propagate: graph neural networks meet personalized pagerank. *ICLR*.

[22] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*.

[23] D. F. Gleich, K. Kloster, and H. Nassar. 2015. Localization in seeded pagerank. *arXiv preprint arXiv:1509.00016*.

[24] M. Gori, G. Monfardini, and F. Scarselli. 2005. A new model for learning in graph domains. *IJCNN*, 729–734.

[25] W. Hamilton, Z. Ying, and J. Leskovec. 2017. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 1024–1034.

[26] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. *CVPR*, 770–778.

[27] W. Huang, T. Zhang, Y. Rong, and J. Huang. 2018. Adaptive sampling towards fast graph representation learning. *Advances in Neural Information Processing Systems*, 4563–4572.

[28] G. Jeh and J. Widom. 2003. Scaling personalized web search. *WWW*.

[29] A. Kannan et al. 2016. Smart reply: automated response suggestion for email. *KDD*, 955–964.

[30] G. Karypis and V. Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20, 1.

[31] D. P. Kingma and J. Ba. 2014. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[32] T. N. Kipf and M. Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

[33] Q. Li, Z. Han, and X.-M. Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. *arXiv preprint arXiv:1801.07606*.

[34] P. Lofgren, S. Banerjee, and A. Goel. 2016. Personalized pagerank estimation and search: a bidirectional approach. *WSDM*.

[35] H. Nassar, K. Kloster, and D. F. Gleich. 2015. Strong localization in personalized pagerank vectors. *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 190–202.

[36] M. Niepert, M. Ahmed, and K. Kutzkov. 2016. Learning convolutional neural networks for graphs. *ICML*, 2014–2023.

[37] L. Page, S. Brin, R. Motwani, and T. Winograd. 1998. The pagerank citation ranking: bringing order to the web.

[38] B. Perozzi, M. Schueppert, J. Saalweachter, and M. Thakur. 2016. When recommendation goes wrong: anomalous link discovery in recommendation networks. *KDD*, 569–578.

[39] S. Ravi. 2016. Graph-powered machine learning at google. https://ai.googleblog.com/2016/10/graph-powered-machine-learning-at-google.html. (2016).

[40] R. Al-Rfou, D. Zelle, and B. Perozzi. 2019. Ddgk: learning graph representations for deep divergence graph kernels. *WWW*, 37–48.

[41] R. Sato, M. Yamada, and H. Kashima. 2019. Constant time graph neural networks. *arXiv preprint arXiv:1901.07868*.

[42] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks*, 20, 1.

[43] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J. P. Hsu, and K. Wang. 2015. An overview of microsoft academic service (mas) and applications. *WWW 2015*.

[44] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 1, 2.

[45] S. Wang, Y. Tang, X. Xiao, Y. Yang, and Z. Li. 2016. Hubppr: effective indexing for approximate personalized pagerank. *PVLDB*, 10, 205–216.

[46] S. Wang, R. Yang, X. Xiao, Z. Wei, and Y. Yang. 2017. Fora: simple and effective approximate single-source personalized pagerank. *KDD*.

[47] X. Wang, A. Shakery, and T. Tao. 2005. Dirichlet pagerank. *SIGIR*.

[48] Z. Wei, X. He, X. Xiao, S. Wang, S. Shang, and J.-R. Wen. 2018. Topppr: top-k personalized pagerank queries with precision guarantees on large graphs. *SIGMOD*.

[49] F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger. 2019. Simplifying graph convolutional networks. *arXiv preprint arXiv:1902.07153*.

[50] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

[51] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

[52] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*.

[53] J. Yang and J. Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42, 1, 181–213.

[54] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. *arXiv preprint arXiv:1806.01973*.

[55] M. Zhang and Y. Chen. 2018. Link prediction based on graph neural networks. *arXiv preprint arXiv:1802.09691*.

# A APPENDIX

## A.1 Parallel Efficiency

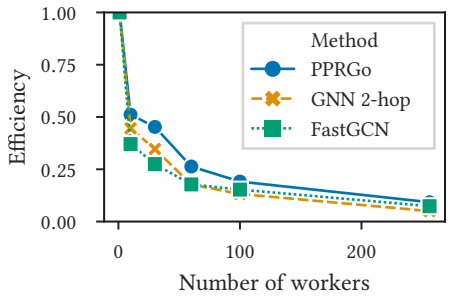



**Figure 8: Parallel efficiency w.r.t. the number of distributed workers on MAG-Scholar-F for different models.**

To further investigate the performance of different models in the distributed training setting we also evaluate parallel efficiency. Intuitively, this efficiency measures how well we can utilize additional workers. Let $m_t$ be the number of steps per second using $t$ workers, then the parallel efficiency of a model is defined as $\frac{m_t}{m_1 \cdot t}$. In Fig. 8 we see that PPRGo achieves the best parallel efficiency.

## A.2 MAG-Scholar Graph Construction

First, we obtain the "raw" data from the Microsoft Academic Graph (MAG) [43] repository, specifically we downloaded a snapshot of the data on 01.25.2019. We construct a graph where each node is a paper and the edges indicate citations/references between the papers. The node features are a bag-of-words representation of the paper's abstract. We preprocess the feature matrix by keeping only those words that appear in at least 5 abstracts. We preprocess the graph by keeping only the nodes that belong to the largest connected component. The resulting MAG-Scholar-F graph has 12.40393 million nodes, 2.78424 million features, and 173.050172 million edges. The MAG-Scholar-C graph has 10.54156 million nodes, 2.78424 million features, and 132.817644 million edges. To obtain the fields of study for each paper, we first create a mapping between a venue (i.e. conference or journal) and its respective field of study. Specifically, we consider the top-20 venues in each field of study according to Google Scholar[6]. We manually match the same venues that have different tittles (e.g. because of abbreviations) in the MAG data compared to the Google Scholar data. These venues are categorized in 8 different coarse-grained categories (e.g. engeneering[7]) and 253 different fine-grained categories (e.g. biophysics[8]) and we use them to define coarse/fine-grained "ground-truth" labels for the nodes.

## A.3 Experimental Details

We keep all PPRGo hyperparameters constant across all datasets, except the value of the teleport parameter $\alpha = 0.25$, which we set to $\alpha = 0.5$ for reddit. The feed-forward neural network has two layers, i.e. a single hidden layer of size 32. We use a dropout of 0.1 and set the weight decay to $10^{-4}$. We train for 200 epochs using a learning rate of 0.005 and the Adam optimizer [31] with a batch size of 512. To achieve a consistent setup across all models and datasets we always use the same number of epochs, use no early stopping and only evaluate validation accuracy after training. For the validation set we randomly sample 10 times the number of training nodes.

We standardize the graphs as a preprocessing step, i.e. we choose only the subset of nodes belonging to the largest connected component and make the graph undirected and unweighted. We do not include dataset loading time in the overall runtime since it is the same for all models.

## A.4 Further Implementational Details

The pseudo code in Algorithm 1 shows how we compute the approximate personalized PageRank based on [4]. For single-machine experiments we implement the algorithm as described in Python using Numba for acceleration (not parallelized). In the distributed setting instead of carrying out push-flow iterations until convergence, we perform a fixed number of iterations (i.e. we replace the while with a for loop), and drop nodes whose residual score is below a specified threshold in each iteration. Additionally, we truncate nodes with a very large degree ($\geq$ 10000) by randomly sampling their neighbors. The above modifications proved to be just as effective as Andersen et al. [4]'s method while being significantly faster in terms of runtime.

## A.5 Applicability and Limitations

When using PPRGo for your own purposes you should first be aware that this model assumes a homophilic graph, which is mostly, but not always the case. Furthermore, it cannot perform arbitrary message passing schemes like GNNs do, since we essentially compress the message passing into a single step. It therefore has less theoretical expressiveness than GNNs [7, 51], even if it practically shows the same or better accuracy. However, note that PPRGo allows arbitrary realizations of $f_\theta$ and can therefore be used with more complex data and models such as images and CNNs, audio and LSTMs, or text and Transformers.

[6] https://scholar.google.com/citations?view_op=top_venues&hl=en
[7] https://scholar.google.com/citations?view_op=top_venues&hl=en&vq=eng
[8] https://scholar.google.com/citations?view_op=top_venues&hl=en&vq=phy_biophysics