Scaling Graph Neural Networks With approximate PageRank

KDD 2020 : Applied Data Science Track

INTRODUCTION

- Graph Neural Networks (GNNs) excel on a wide variety of network mining tasks from semi-supervised node classification and link prediction to community detection and graph classification.

- Unfortunately, there are few large graph baseline datasets available, 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

Graph Neural Networks

Powerful approach for solving many network mining task

However,

- Scale poorly to massive graphs with millions of node
- Existing techniques for scaling up are still too expensive

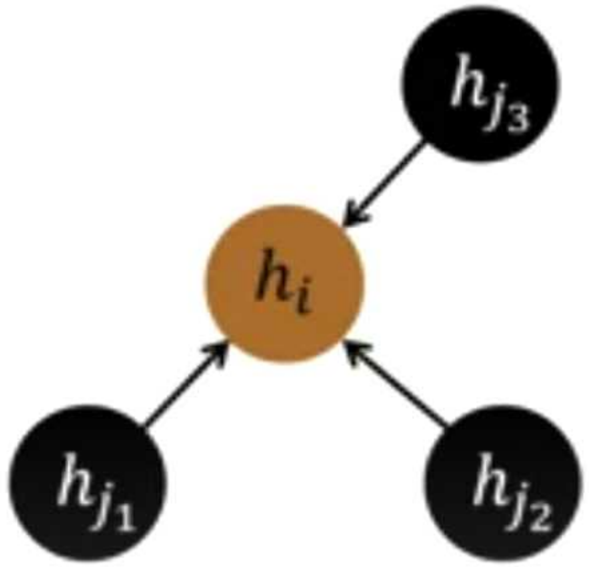
BACKGROUND

- GNNs and Message-Passing
- Personalized PageRank and Localization
- Related Work

Recursive message passing

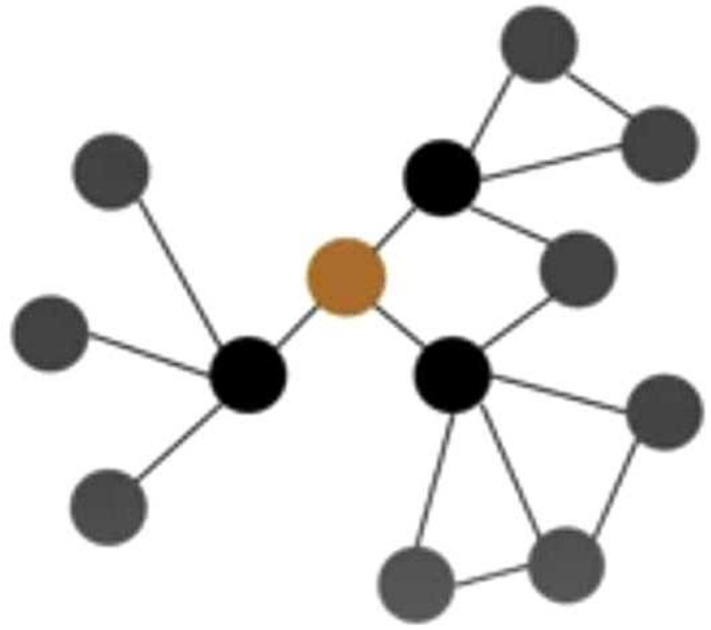
The hidden representataion for node i is a sum of messages from its neighbors

$$h_i^{(l+1)} = \sum_{j \in N_i} f_\theta(h_j^{(l)})$$

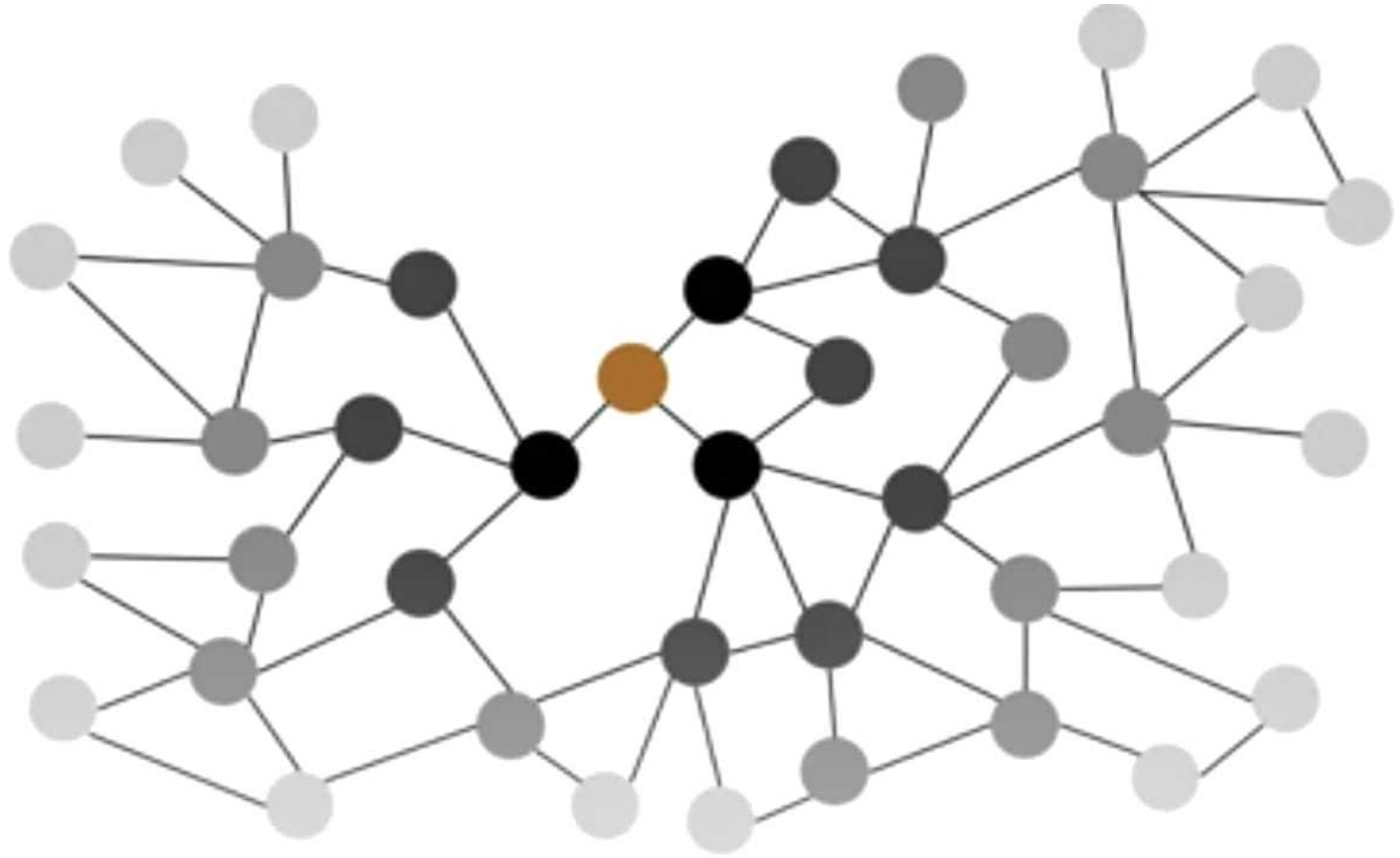


Recursive message passing

Recursive message passing



<u>Recursive message passing</u>



Problem

the recursive neighborhood expansion implies an exponential increase in the overall number of nodes we need to aggregate to produce the output at the final layer

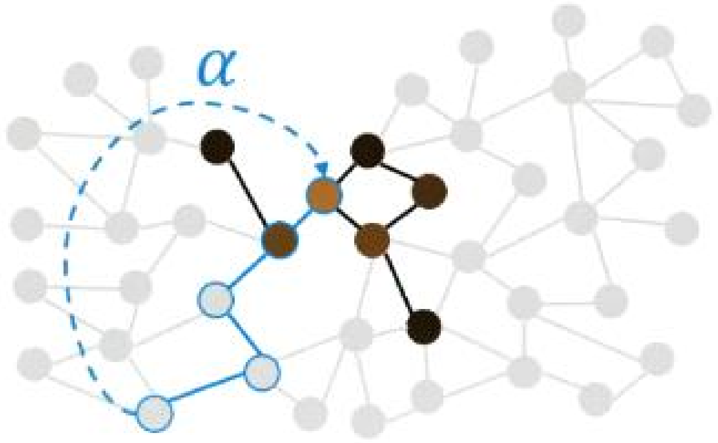
stacking multiple layers may suffer from over-smoothing that can reduce predictive performance.

↓

the feature transformation from the propagation

Recursive message passing

Stationary distribution of a random walk with teleport

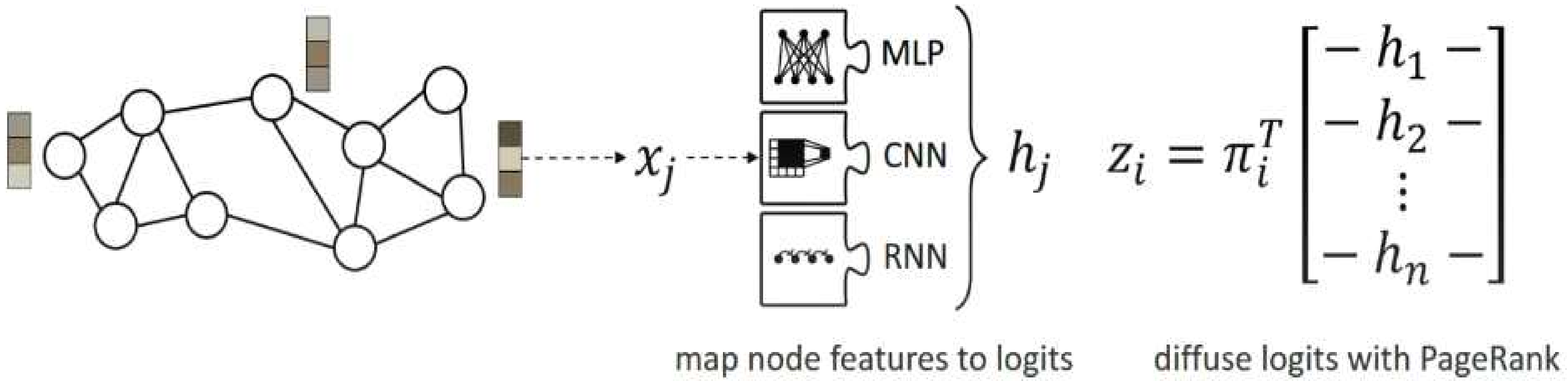


The teleport probability α controls the effective neighborhood size

PPNP

Predict then Propagate : Diffuse individual logits using PageRank

Neural network (depth & structure) is decoupled from propagation



x_j
MLP
CNN
RNN
h_j
$z_i = \pi_i^T \begin{bmatrix} - h_1 - \\ - h_2 - \\ \vdots \\ - h_n - \end{bmatrix}$
map node features to logits
diffuse logits with PageRank

PPNP

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.

$$Z = \mathrm{softmax}\left(\Pi^{\mathrm{sym}} H\right), \qquad H_{i,:} = f_\theta(x_i) \qquad (1)$$

$\Pi^{\mathrm{sym}} = \alpha(I_n - (1-\alpha)\tilde{A})^{-1}$ is a symmetric propagation matrix

$\tilde{A} = D^{-1/2} A D^{-1/2}$ is the normalized adjacency matrix with added self-loops,

α is a teleport (restart) probability,

PPNP

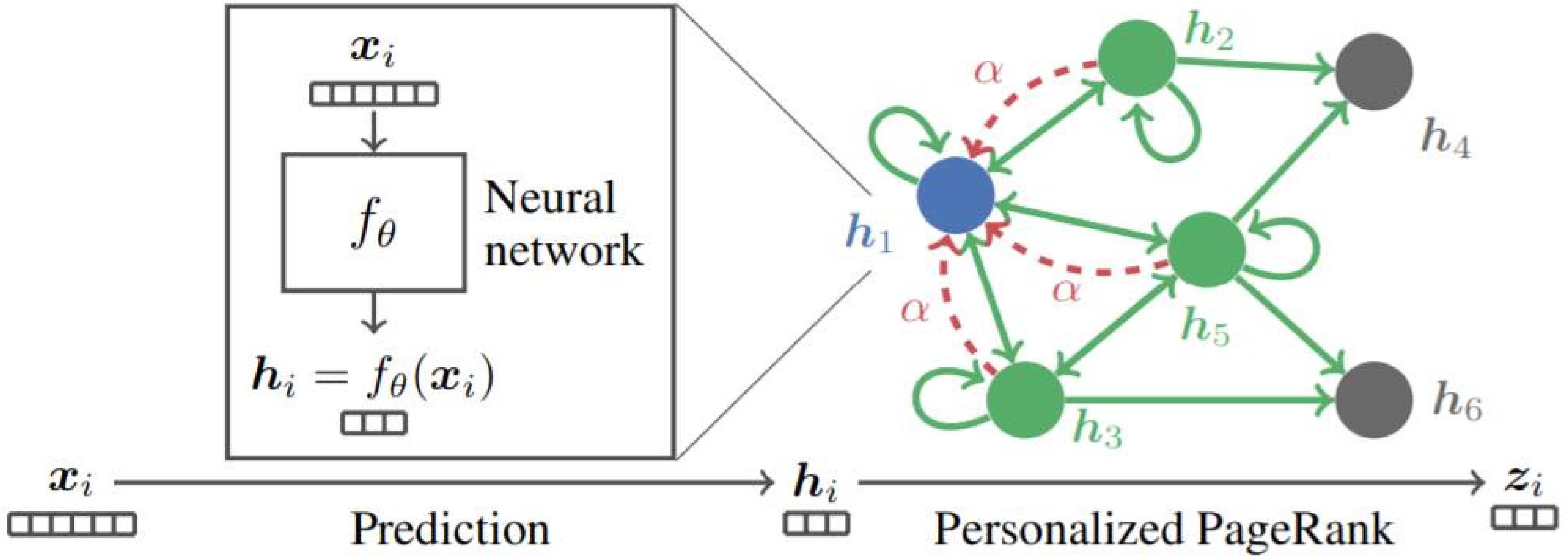
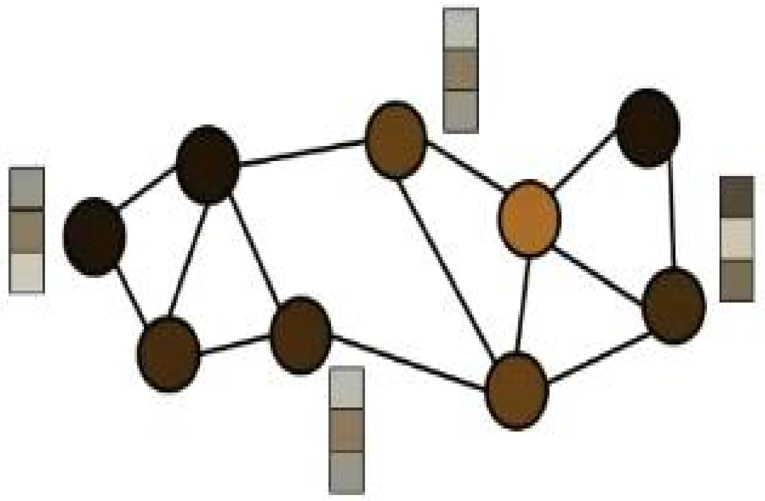


Figure 1: Illustration of (approximate) personalized propagation of neural predictions (PPNP, APPNP). Predictions are first generated from each node's own features by a neural network and then propagated using an adaptation of personalized PageRank. The model is trained end-to-end.

PPNP



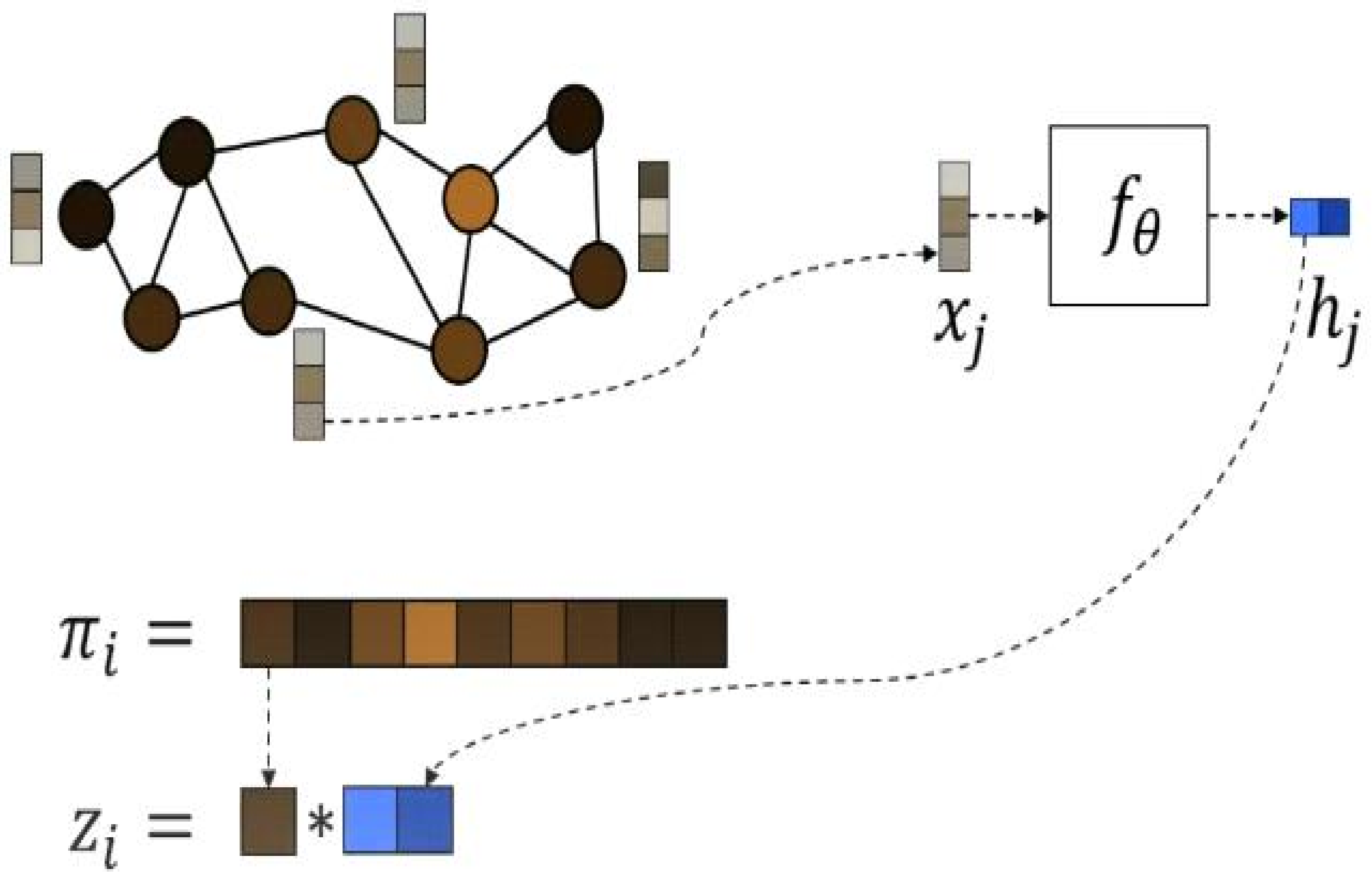
$$\boxed{f_\theta}$$

$$z_i = \pi_i^T \begin{bmatrix} - h_1 - \\ - h_2 - \\ \vdots \\ - h_n - \end{bmatrix}$$

$\pi_i =$

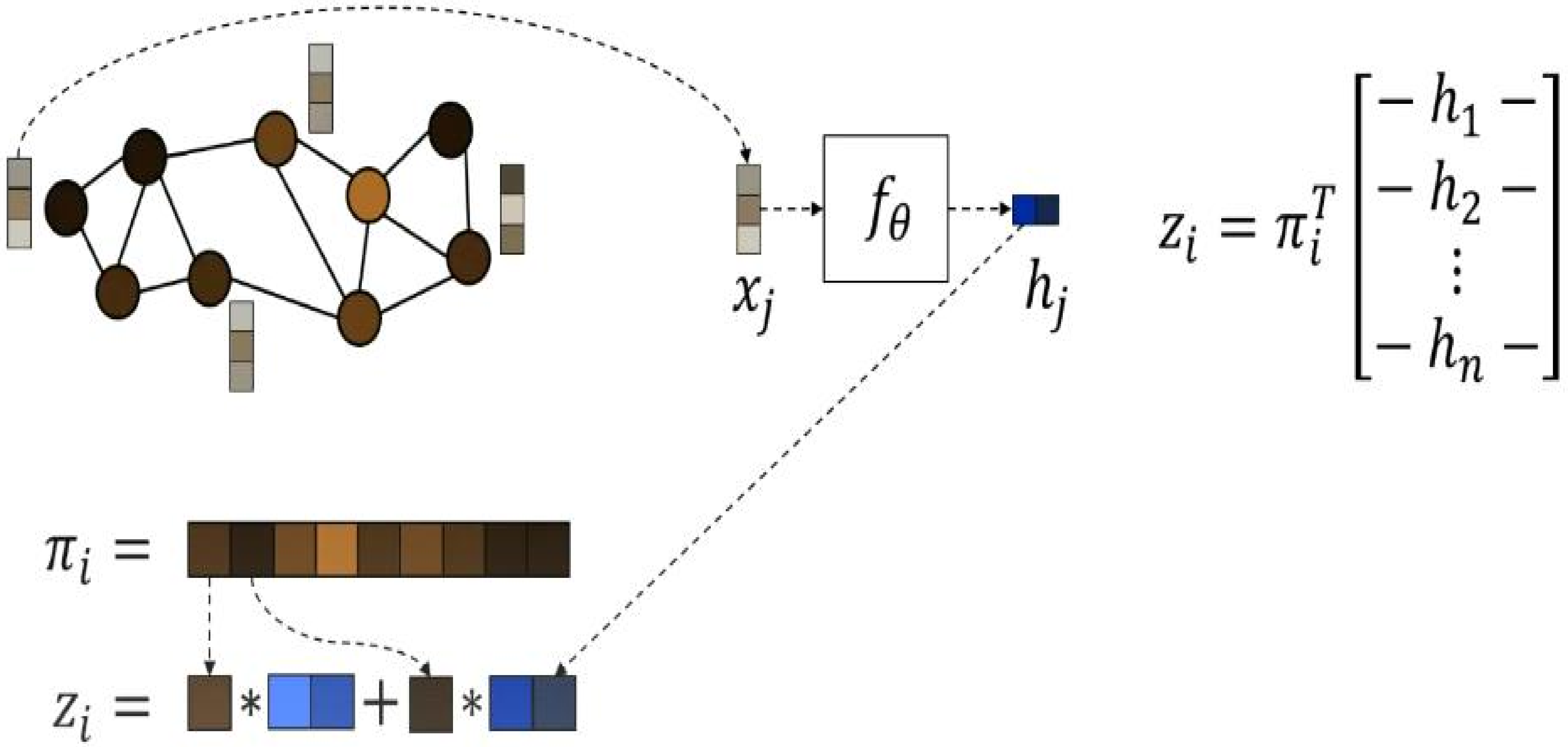
$z_i =$

PPNP



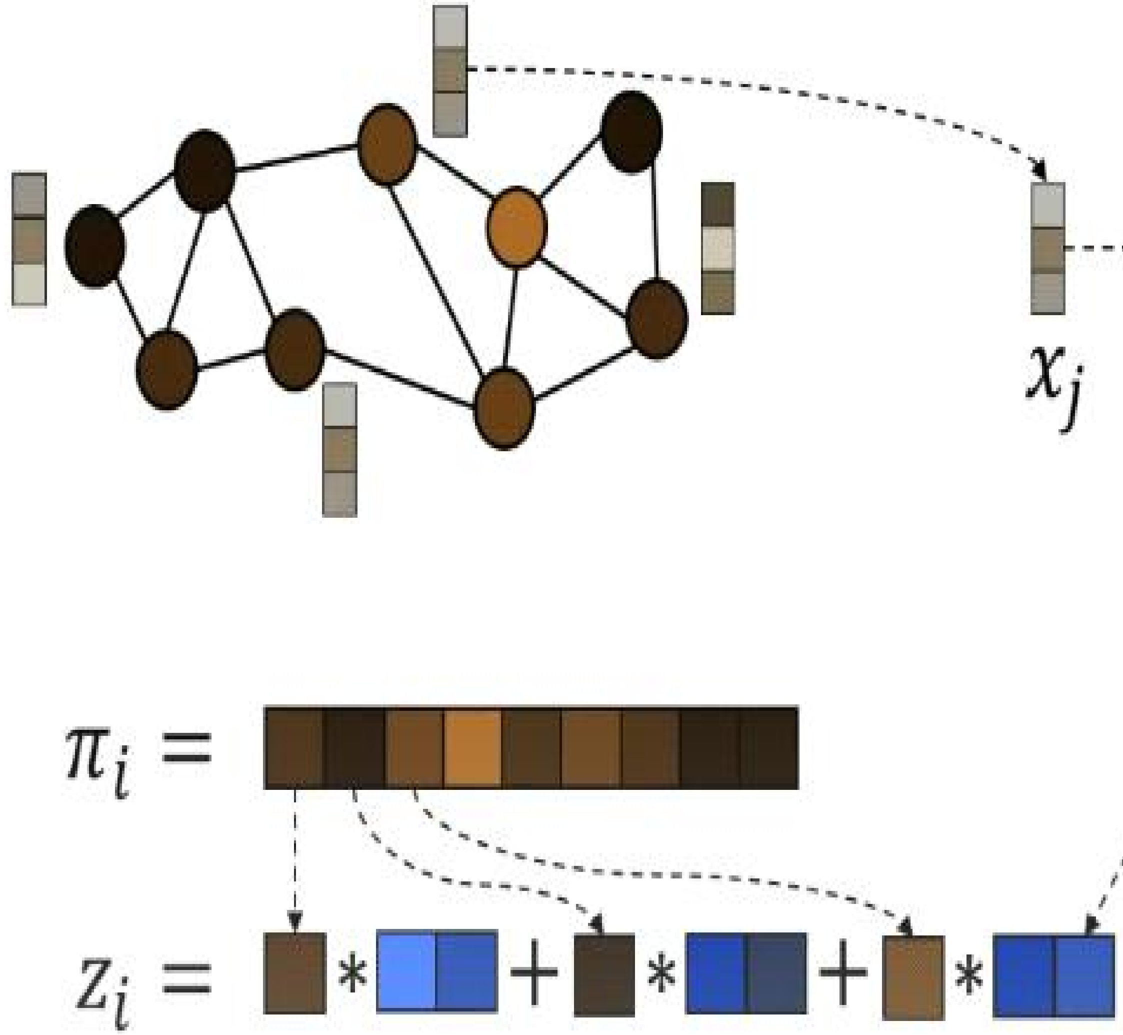
$$z_i = \pi_i^T \begin{bmatrix} - h_1 - \\ - h_2 - \\ \vdots \\ - h_n - \end{bmatrix}$$

PPNP



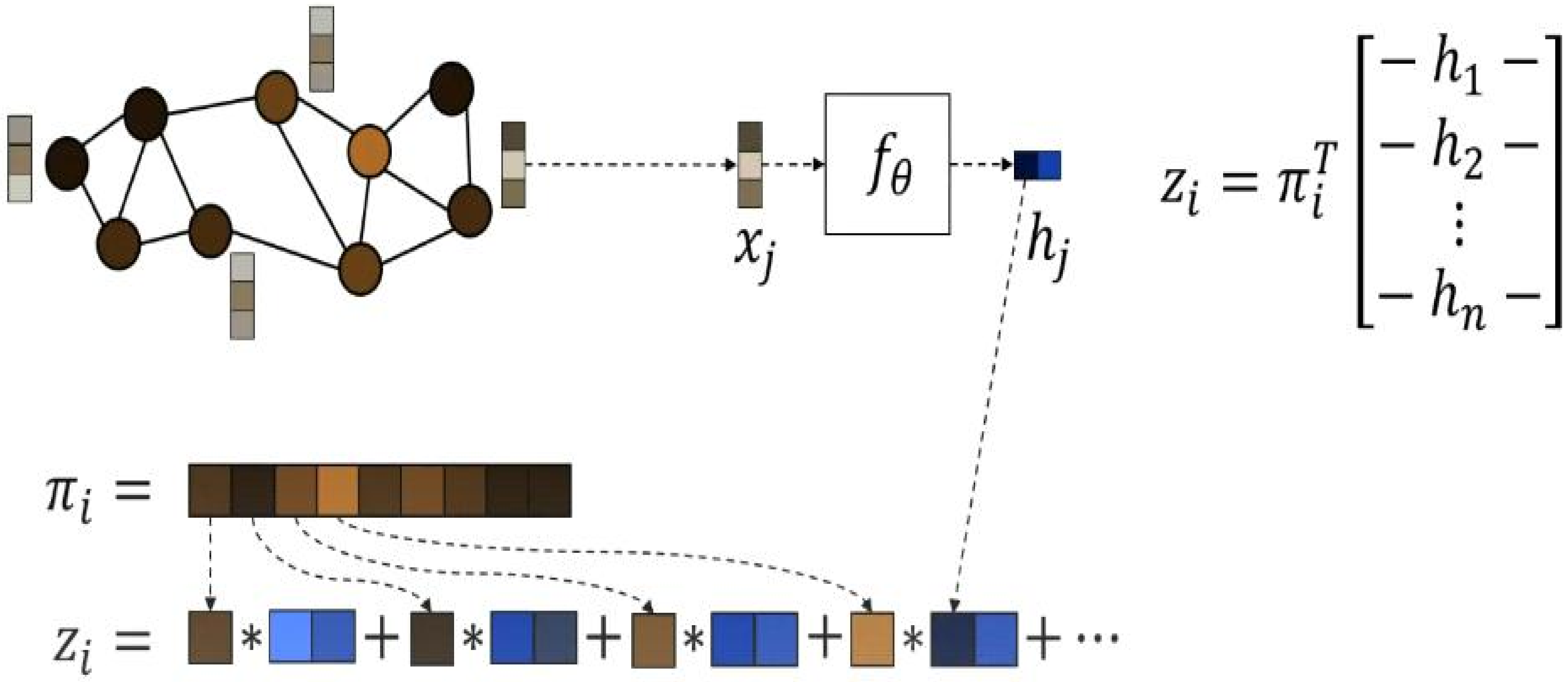
x_j
f_θ
h_j
$z_i = \pi_i^T \begin{bmatrix} - h_1 - \\ - h_2 - \\ \vdots \\ - h_n - \end{bmatrix}$
$\pi_i =$
$z_i =$
$*$
$+$
$*$

PPNP



$$z_i = \pi_i^T \begin{bmatrix} - h_1 - \\ - h_2 - \\ \vdots \\ - h_n - \end{bmatrix}$$

PPNP



x_j
f_θ
h_j
$z_i = \pi_i^T \begin{bmatrix} - h_1 - \\ - h_2 - \\ \vdots \\ - h_n - \end{bmatrix}$
$\pi_i =$
$z_i =$ * + * + * + * + ⋯

PPNP

Because, calculating the dense propagation matrix Π^{sym} in Eq. 1 is inefficient

Unfortunately, even a moderate number of power iteration evaluations (e.g. Klicpera et al. [32] used K = 10 to achieve a good approximation) is prohibitively expensive for large graphs

Moreover, despite the fact that $A^{\sim}$ is sparse, graphs beyond a certain size cannot be stored in memory

Personalized PageRank and Localization

parsonalized PageRank matrix $\Pi^{\text{ppr}} = \alpha(\boldsymbol{I}_n - (1-\alpha)\boldsymbol{D}^{-1}\boldsymbol{A})^{-1}$
Each row is equal to the personalized (seeded) PageRank vector of node i .

Random walk sampling is one such approximation technique. While simple to implement, in order to guarantee at most ϵ absolute error with probability of $1 - 1/n$ we need $O(\log n/\varepsilon^2)$ random walks.

Personalized PageRank and Localization

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

Related work

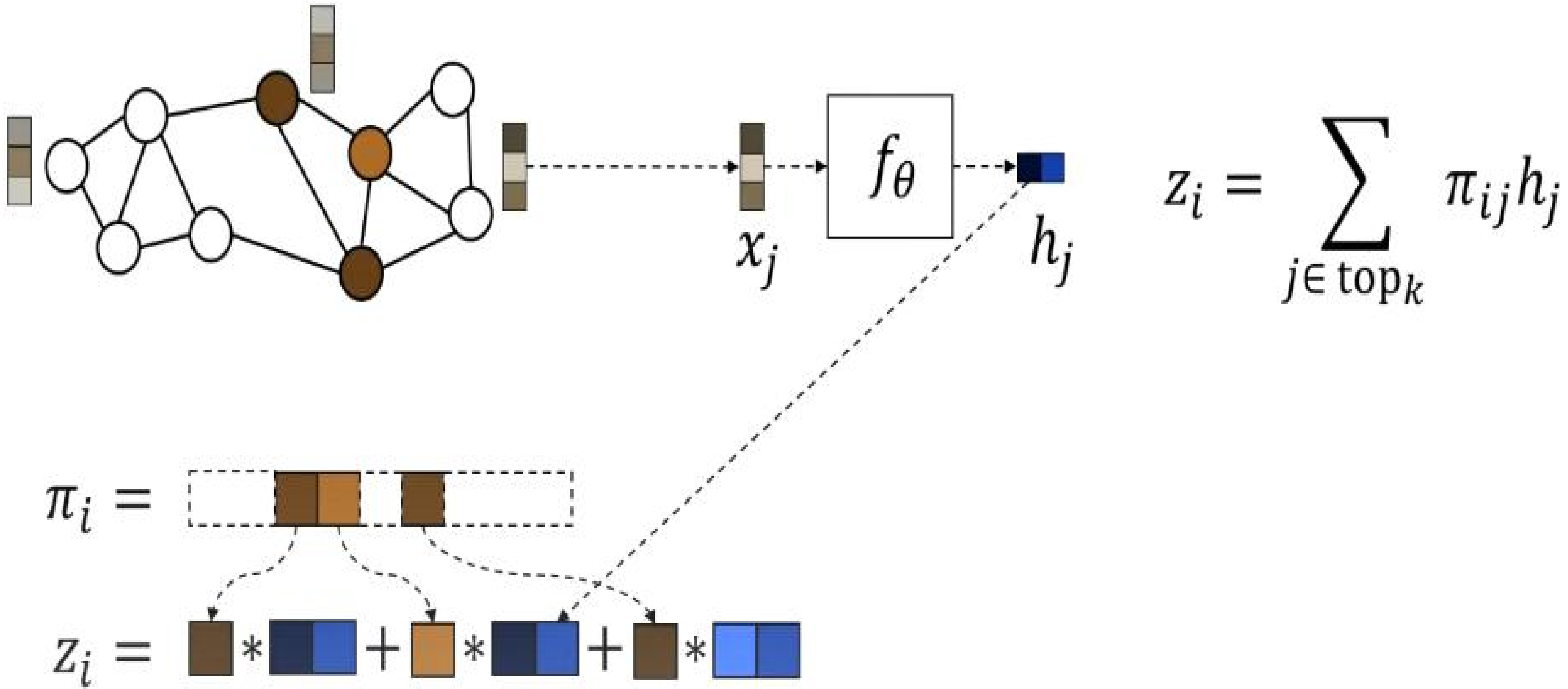
Scalability

- 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

Approximating PageRank

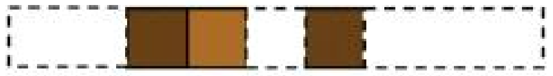
- TopPPR algorithm combining the strengths of random walks and forward/backward search simultaneously. They can compute the top k entries of a personalized PageRank vector up to a given precision using a filterand-refine paradigm

PPRGo



Approximate PageRank for training nodes

Approximate the PageRank vector with ACL's algorithm

$$\pi_i^{(\epsilon)} =$$ 

The algorithm is local (needs only neighbors) and highly parallelizable

Result : Sparse vector with PageRank scores of only the relevant nodes

Power iteration and sparse inference

Computing predictions: $\underbrace{\alpha\left(\mathrm{I}_n - (1-\alpha)\mathrm{D}^{-1}\mathrm{A}\right)^{-1}}_{\text{each row is a PPR vector for one node}} \cdot \mathrm{H}$

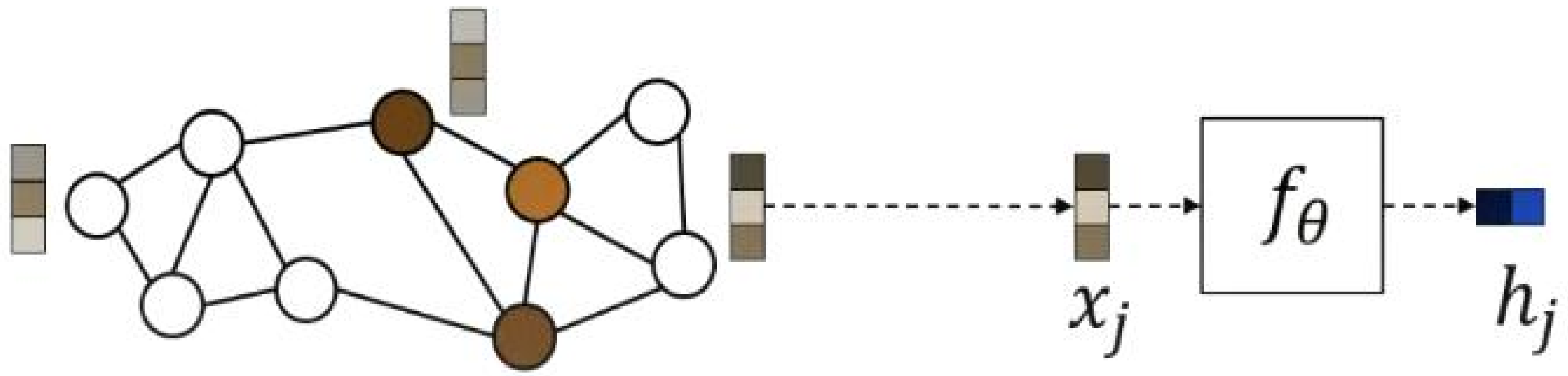
Power Iteration: $Q^{(0)} = \mathrm{H}, \qquad Q^{(p+1)} = \alpha\mathrm{H} + (1-\alpha)\mathrm{D}^{-1}\mathrm{A}Q^{(p)}$

Sparse Inference: forward pass only for a fraction of nodes

$$H = \begin{bmatrix} - h_1 - \\ - h_2 - \\ \vdots \\ - h_n - \end{bmatrix} \approx \begin{bmatrix} - \ 0 \ - \\ - h_j - \\ \vdots \\ - \ 0 \ - \end{bmatrix}$$

PPRGo

1. Precompute approximate sparse PPR vectors π for training nodes
2. Train the mapping f using SGD
3. Run Power iteraion during inference



$$z_i = \sum_{j \in \mathrm{top}_k} \pi_{ij}^{\epsilon} h_j$$

PPRGo

Algorithm 1 Approximate personalized PageRank (G, α, t, ϵ) [4]

```
Inputs: Graph $G$, teleport prob. $\alpha$, target node $t$, max. residual $\epsilon$
 1: Initialize the (sparse) estimate-vector $\boldsymbol{\pi}^{(\epsilon)} = \mathbf{0}$ and the (sparse)
    residual-vector $\boldsymbol{r} = \alpha \cdot \boldsymbol{e}_t$ (i.e. $\boldsymbol{e}_t = 1, \boldsymbol{e}_v = 0, v \neq t$)
 2: while $\exists v \; s.t. \; \boldsymbol{r}_v > \alpha \cdot \epsilon \cdot \boldsymbol{d}_v$ do        # $\boldsymbol{d}_v$ is the out-degree
 3:     $\boldsymbol{\pi}_v^{(\epsilon)} \mathrel{+}= \boldsymbol{r}_v$
 4:     $\boldsymbol{r}_v = 0$
 5:     $m = (1 - \alpha) \cdot \boldsymbol{r}_v / \boldsymbol{d}_v$
 6:     for $u \in \mathcal{N}_G^{\text{out}}(v)$ do        # $v$'s outgoing neighbors
 7:         $\boldsymbol{r}_u \mathrel{+}= m$
 8:     end for
 9: end while
10: return $\boldsymbol{\pi}^{(\epsilon)}$
```

Node Classification in the Real World

Many web graphs have interesting node classification problems that can be addressed via semi-supervised learning.

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 realworld settings

Distributed Training

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.

Distributed Training

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 [22, 4] which in turn means we can reliably estimate the runtime beforehand.

Efficient Inference

during inference we still need to compute the PPR vector for every test node

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

Efficient Inference

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

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

Efficient Inference

Hence we only need very few sparse matrix-matrix multiplications for inference, which can be implemented very efficiently

Computing PageRank

Training
- Approximate sparse PPR 1 diffusion step

Inference
- 1~3 Power iterations steps Sparse Inference

Experimental setup

Semi-supervised node classification
Sparsely labeled scenario

MAG-Scholar dataset
- 12.4M nodes, 173M edges, and 2.8M features

Measure : preprocessing + training + inference
time and memory

Experimental setup

- 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)

Experimental setup

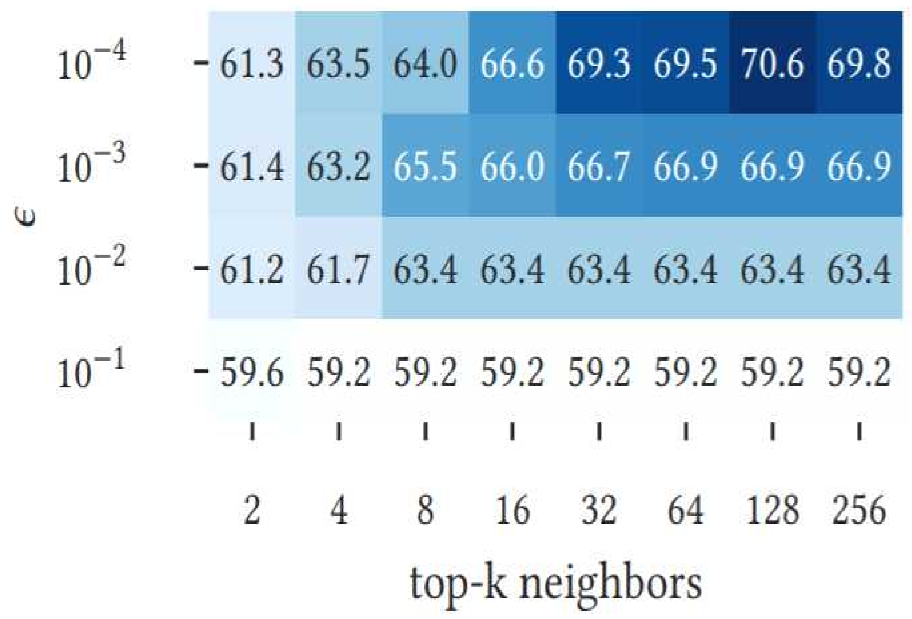
- How much resources (memory, compute) does PPRGo need compared to other scalable GNNs? (§ 5.4)
- How efficient is the proposed sparse inference scheme? (§ 5.5)

Scalability vs. Accuracy Trade-of

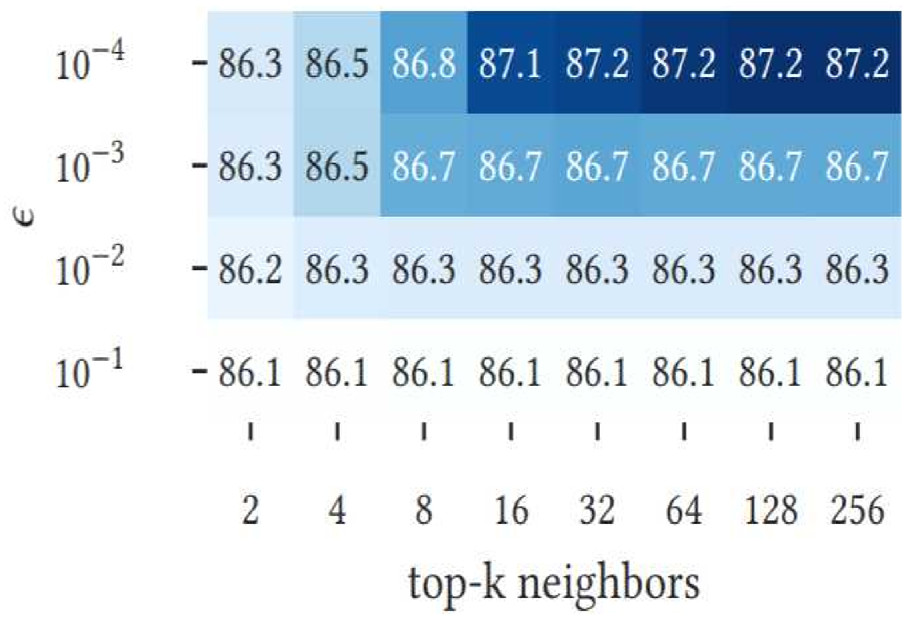
The approximation parameter ϵ and the number of top-k nodes are important hyper-parameters that modulate scalability and accuracy

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)

Scalability vs. Accuracy Trade-of



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



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

Scalability vs. Accuracy Trade-of

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 ϵ) 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 ϵ are computationally more expensive.

Scalability vs. Accuracy Trade-of

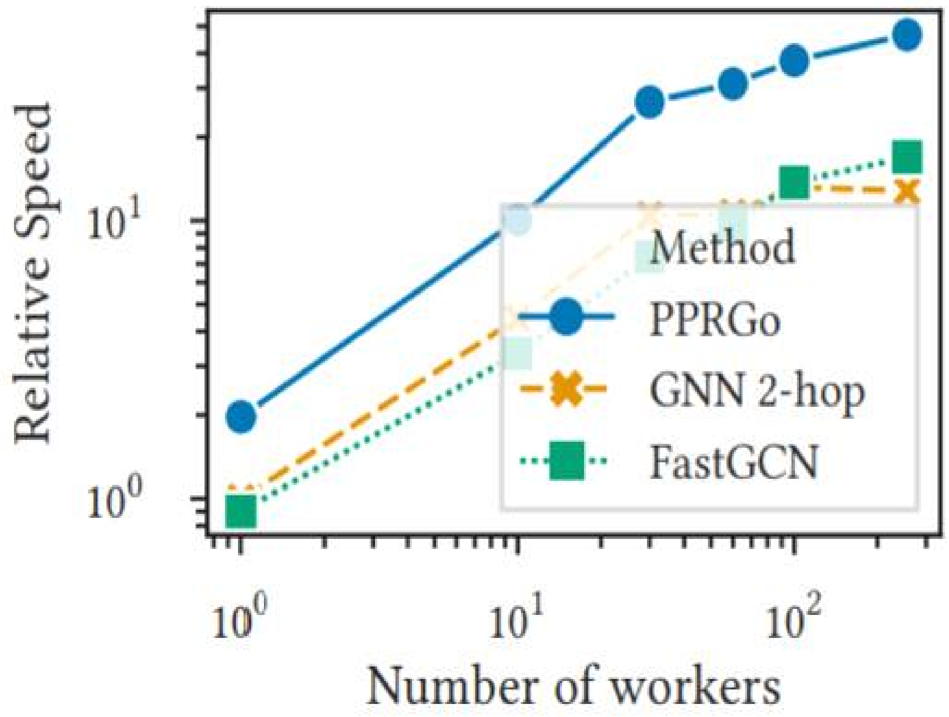
For example, in the academic setting (Fig. 2b) a model with ϵ = 0.1, k = 2 had an overall (preprocessing + training + inference) runtime of 6 minutes, while a model with ϵ = 0.001, k = 256 had an overall runtime of 12 minutes. Since many nodes are labeled (1 %) the difference between the highest accuracy (top right corner) and lowest accuracy (bottom left corner) is under 2 % and the model is not sensitive to the hyperparameters.

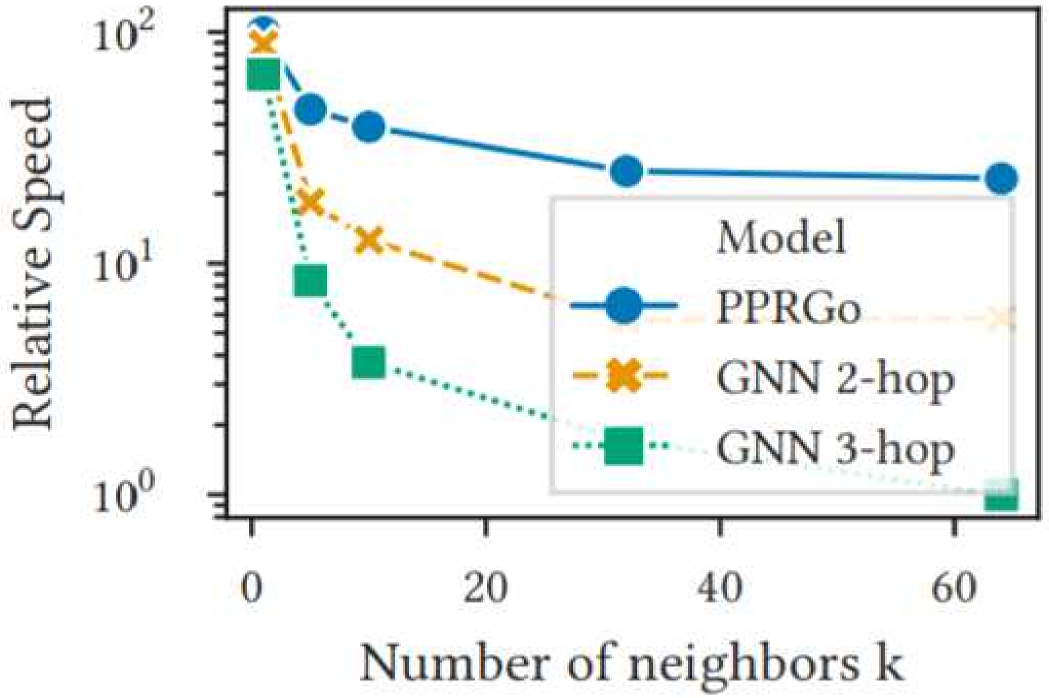
Distributed Training

we aim to compare the performance of one-hop propagation using personalized PageRank and traditional multihop message passing propagation in a real distributed environment at Google

we implement simple 2-hop and 3-hop GNN models [31], which are also trained in a distributed manner using the same infrastructure as PPRGo

Experimental result





Runtime and Memory on a Single Machine

To highlight the benefits of PPRGo we compare the runtime, memory, and predictive performance with SGC [49] and ClusterGCN [15]

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.

Runtime and Memory on a Single Machine

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

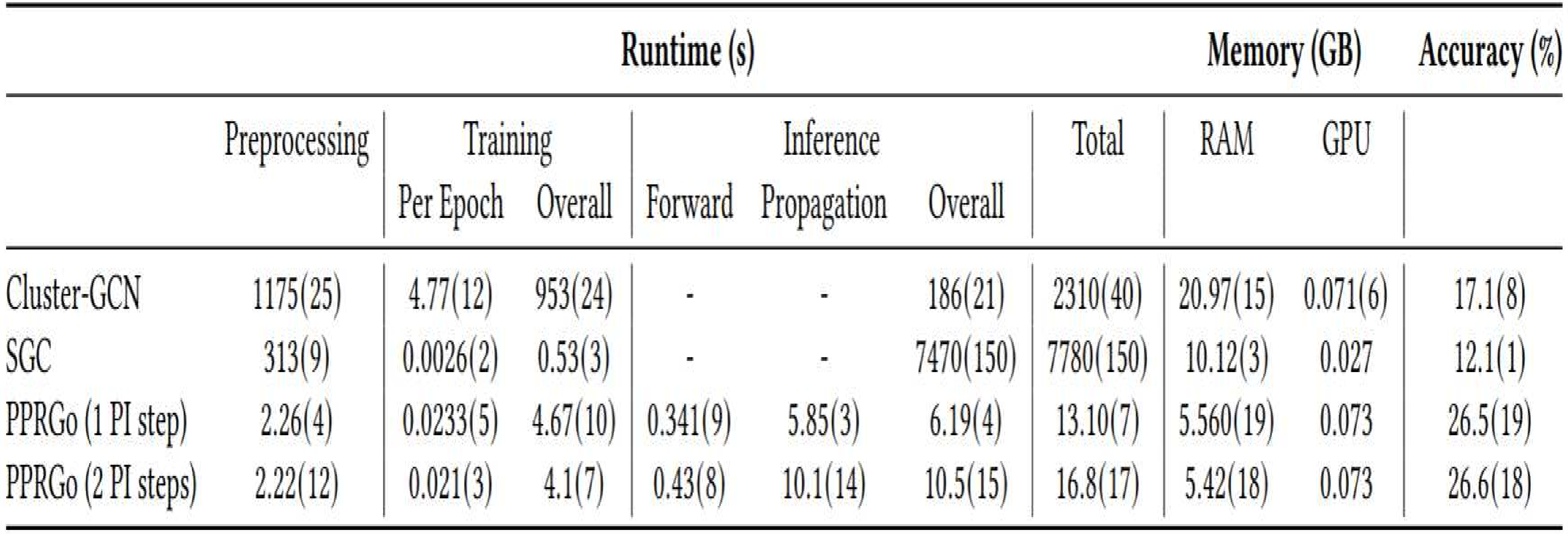
Moreover, we see that the amount of memory used by PPRGo is 4 times smaller compared to Cluster-GCN and 2 times smaller compared to SGC

Runtime and Memory on a Single Machine

the preprocessing step involves computing the approximate personalized PageRank vectors using Algorithm 1 and selecting the top k neighbors

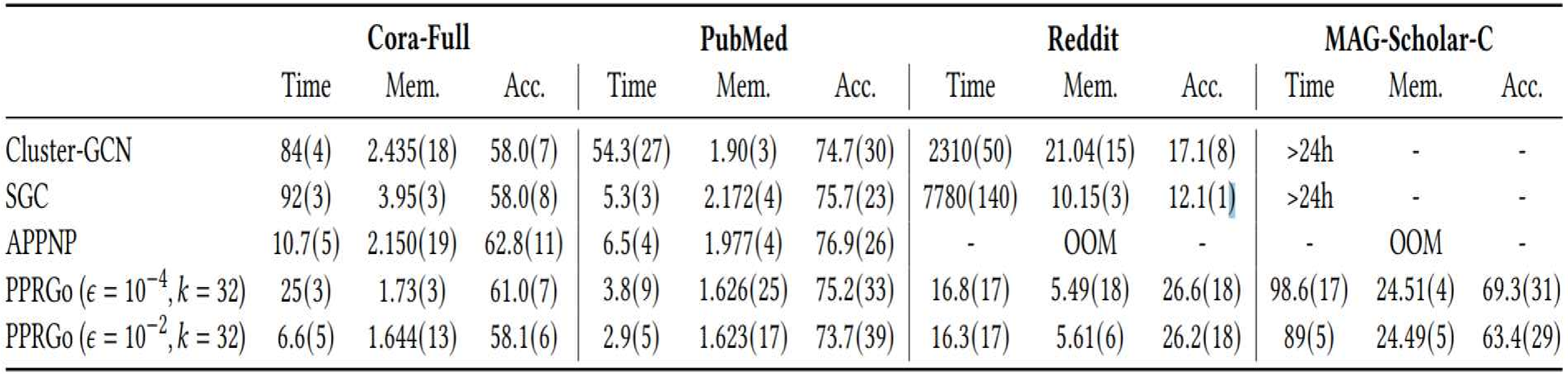
The results when training a model on the Reddit dataset (233K nodes, 11.6M edges, 602 node features) are summarized in Table 1

Experimental result



	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)

Experimental result



	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)

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

Experimental result

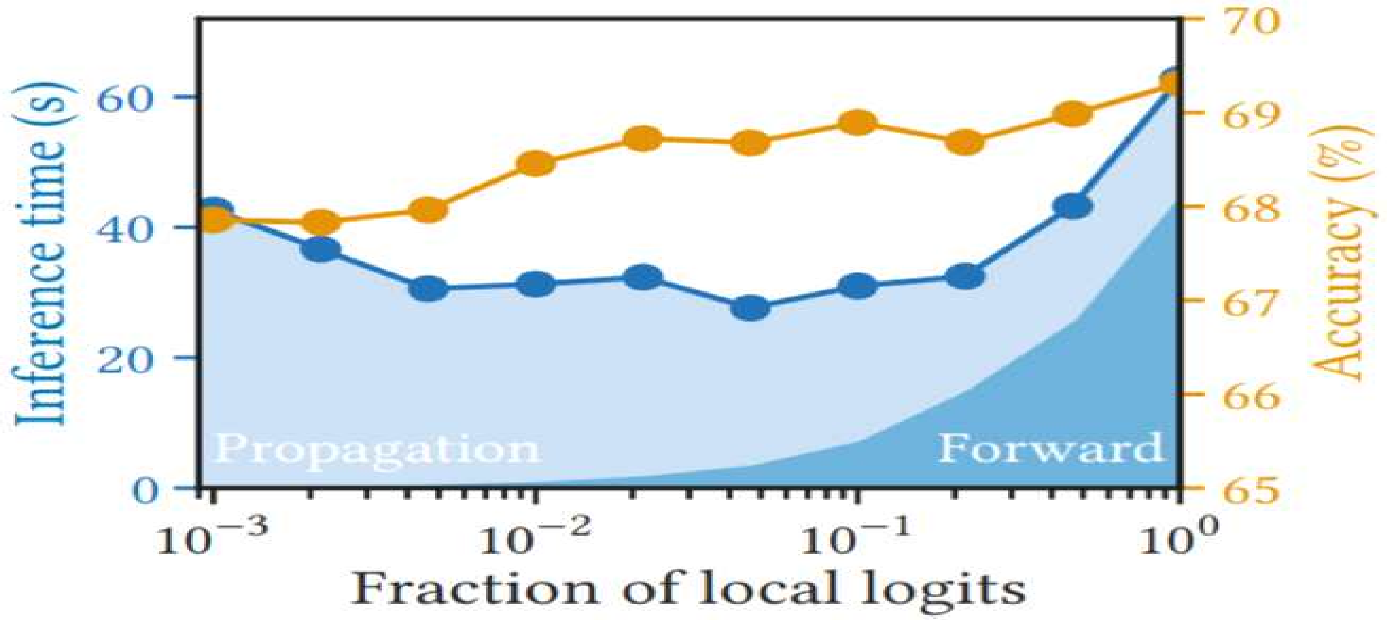


Figure 6: Accuracy and corresponding inference time (NN inference (dark blue) + propagation (light blue)) on MAG-Scholar-C w.r.t. the fraction of nodes for which local logits *H* are inferred by the NN. PPRGo performs very well even if the NN is evaluated on very few nodes. We need more power iteration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

Experimental result

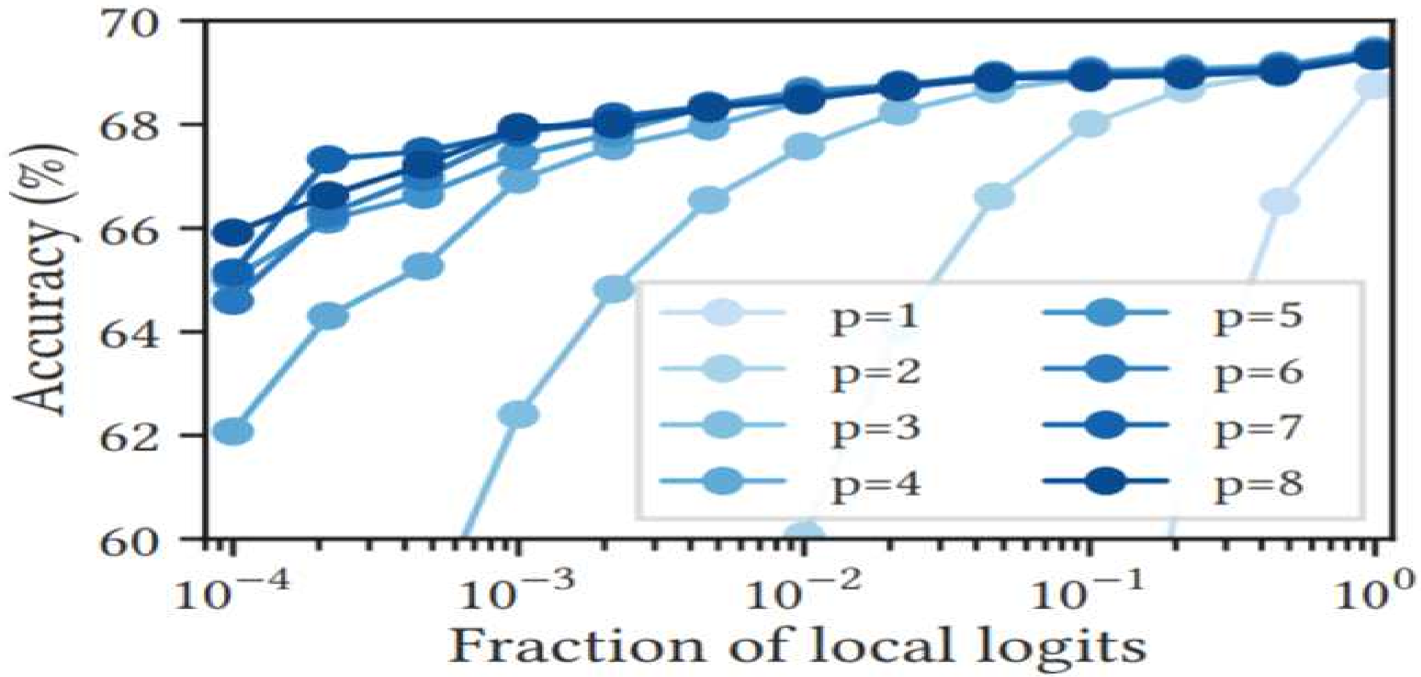


Figure 7: Accuracy on MAG-Scholar-C w.r.t. the fraction of nodes for which local logits H are inferred and number of power iteration steps p. The fewer logits we calculate, the more power iteration steps we need for stabilizing the prediction. Note the logarithmic scale.

CONCLUSION

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.