Scaling Graph Neural Networks with Approximate PageRank

Aleksandar Bojchevski*, Johannes Klicpera*
Bryan Perozzi, Amol Kapoor, Martin Blais,
Benedek Rózemberczki, Michal Lukasik
Stephan Günnemann

KDD 2020: Applied Data Science Track

Graph Neural Networks

Powerful approach for solving many network mining tasks

However:

Scale poorly to massive graphs with millions of nodes

Existing techniques for scaling up are still too expensive

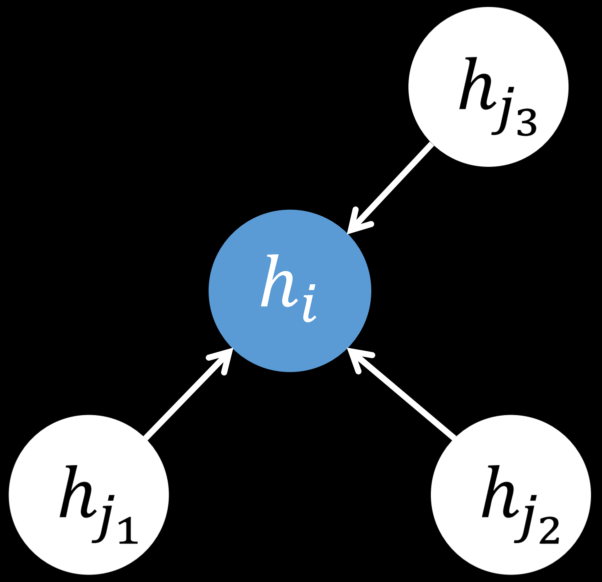
Main scalability bottleneck: Recursive message passing

Resulting in a neighborhood explosion

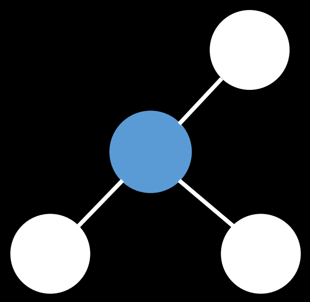
Recursive message passing

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

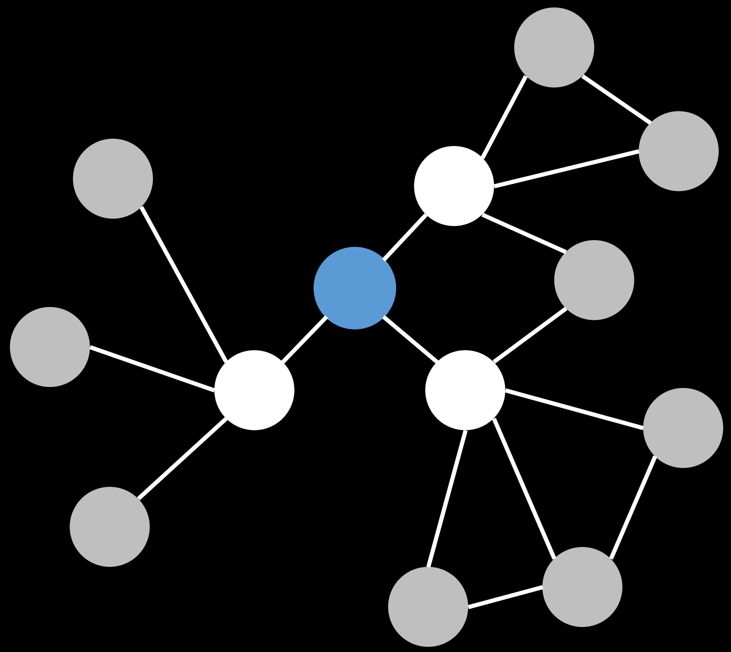
$$h_i^{(l+1)} = \sum_{j \in \mathcal{N}_i} f_\theta\left(h_j^{(l)}\right)$$



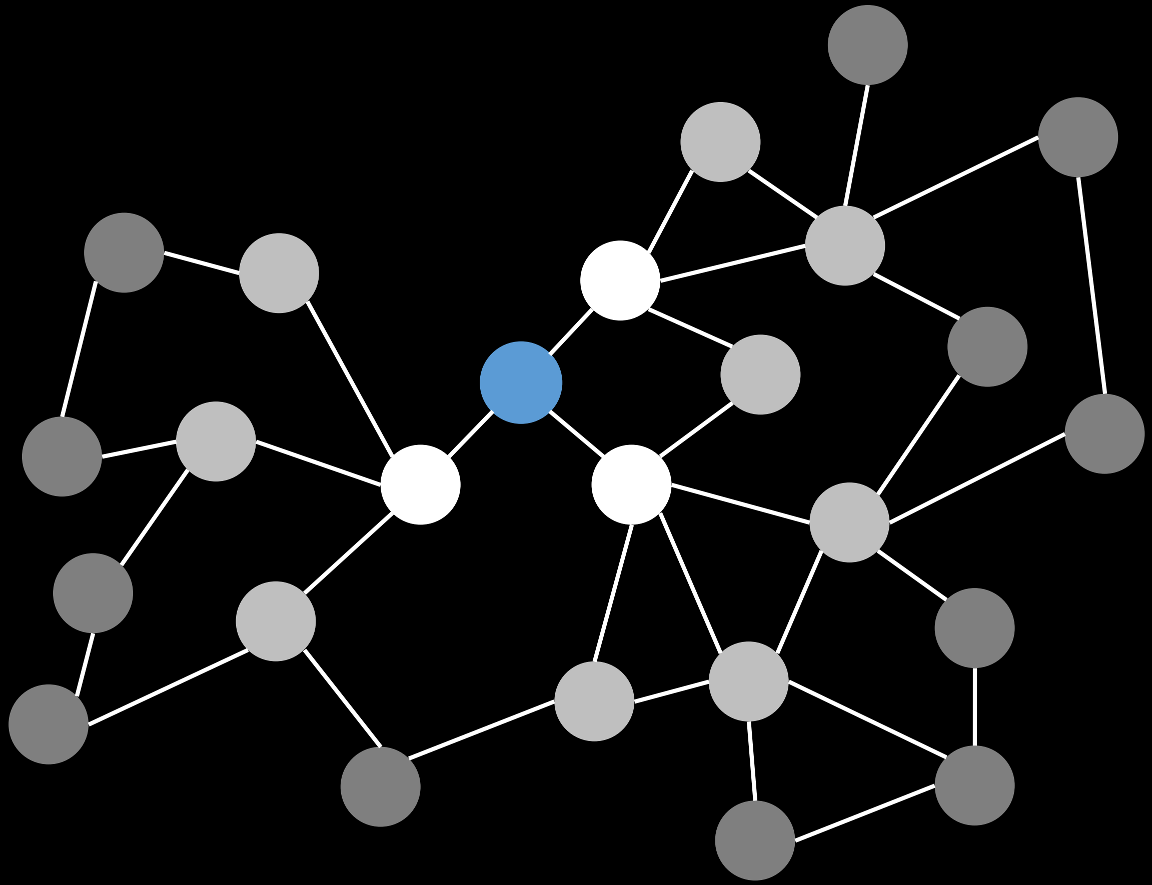
Recursive message passing



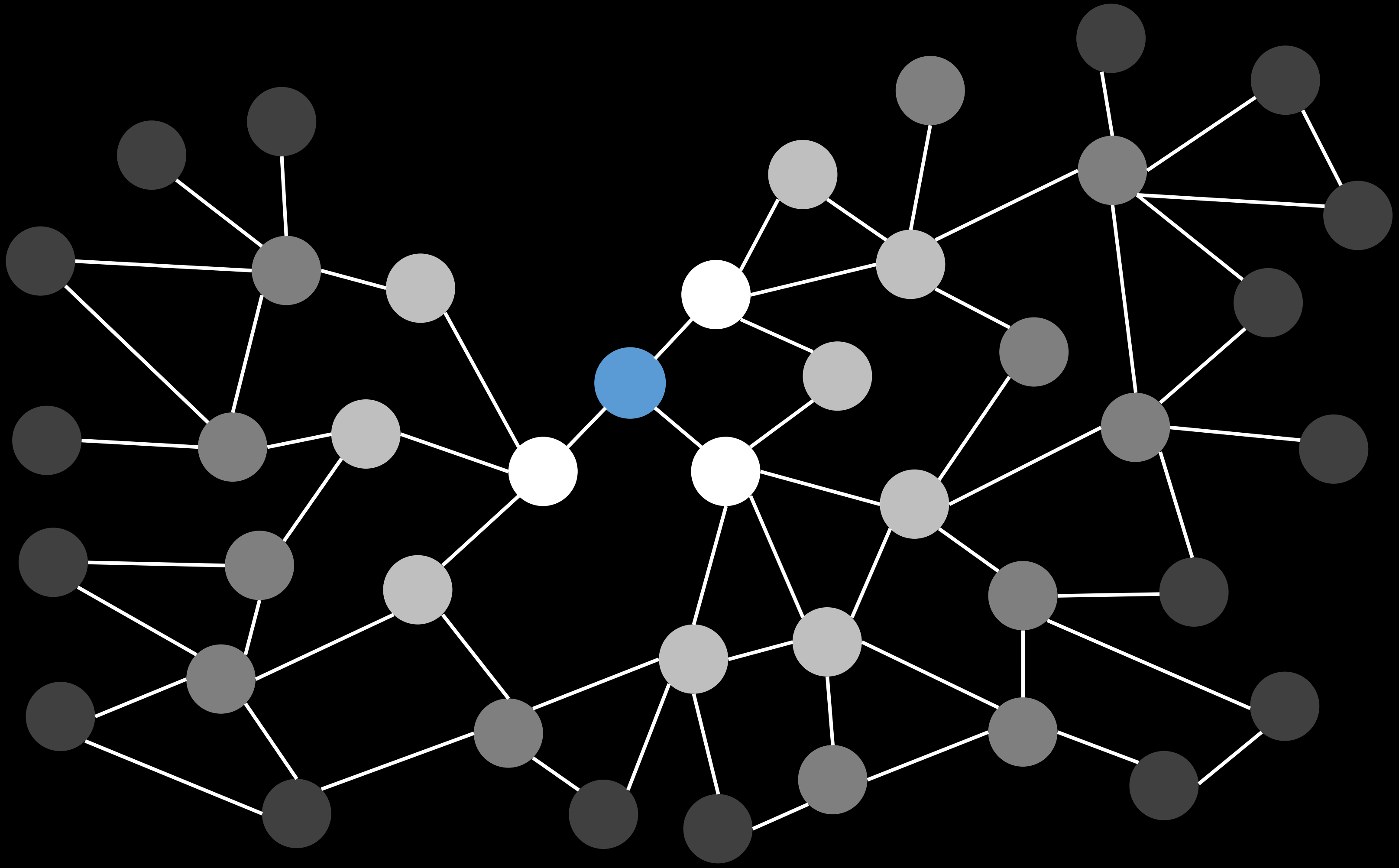
Recursive message passing



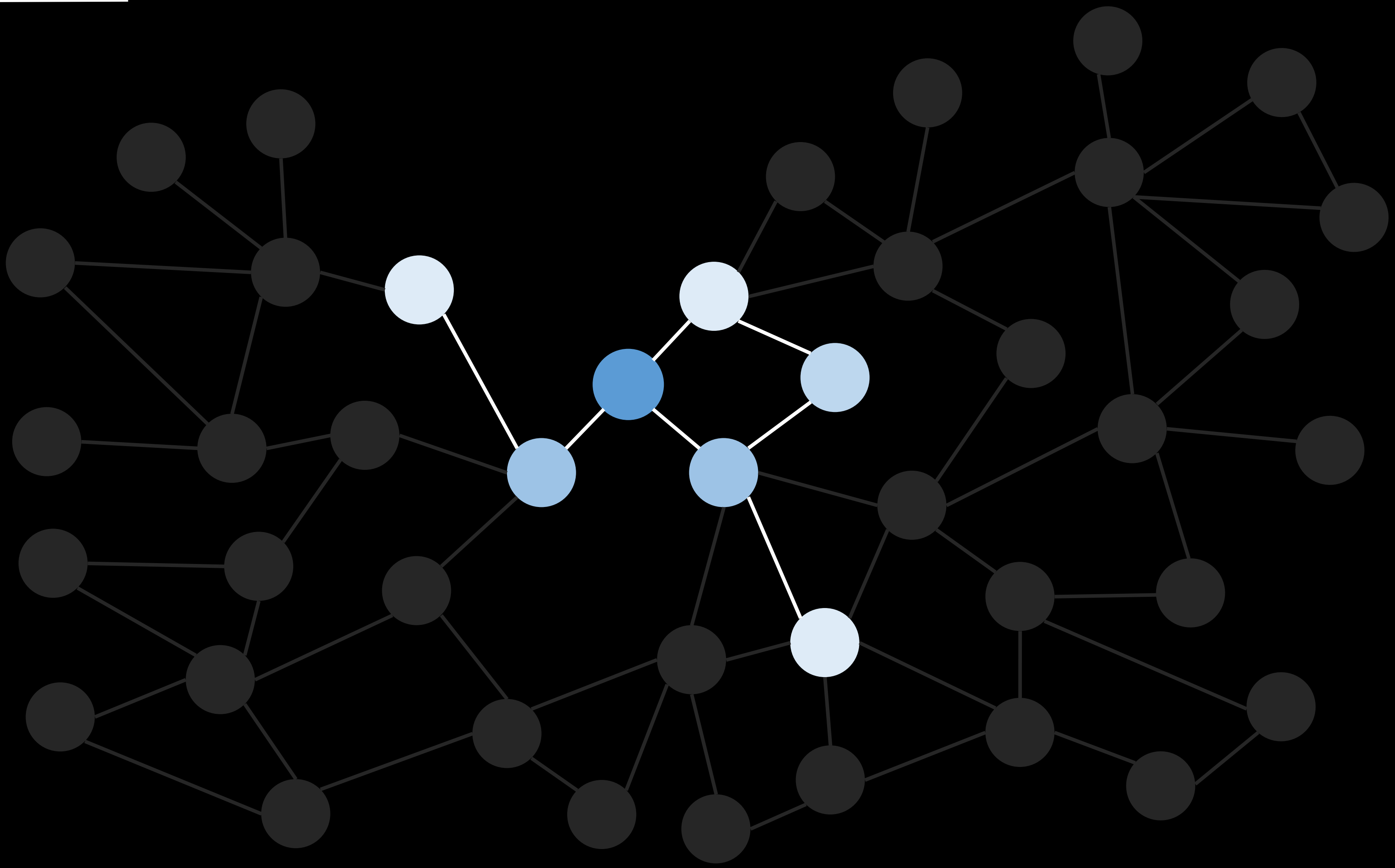
Recursive message passing



Recursive message passing



Only few nodes are important

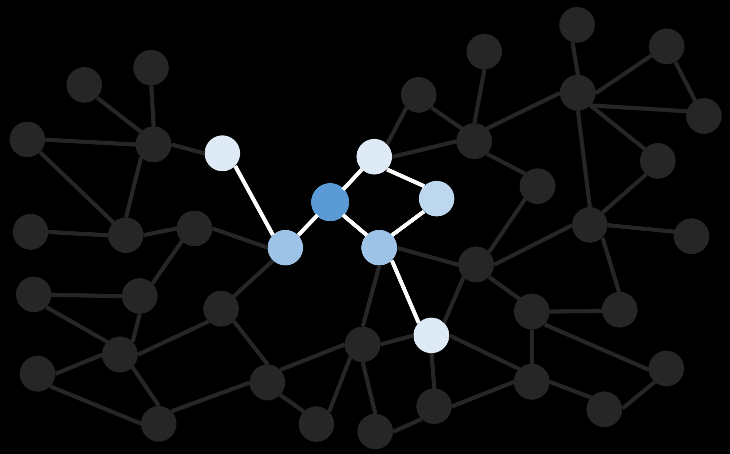


Only few nodes are important

However, we have to:

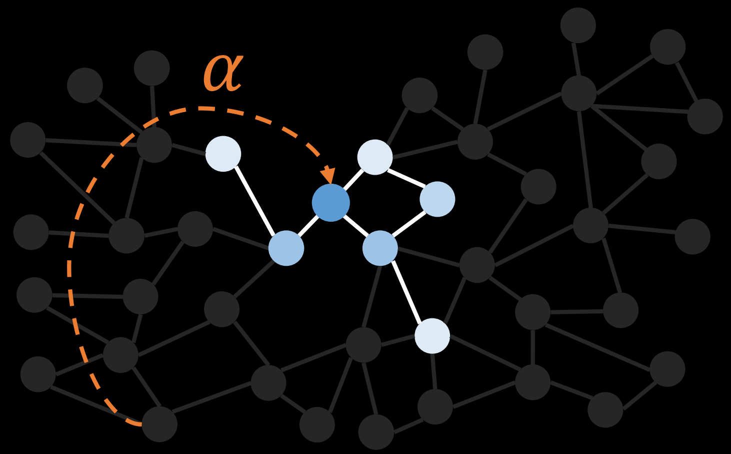
Obtain the importance a priori

Carefully weight the contributions



Personalized PageRank

Stationary distribution of a random walk with teleport

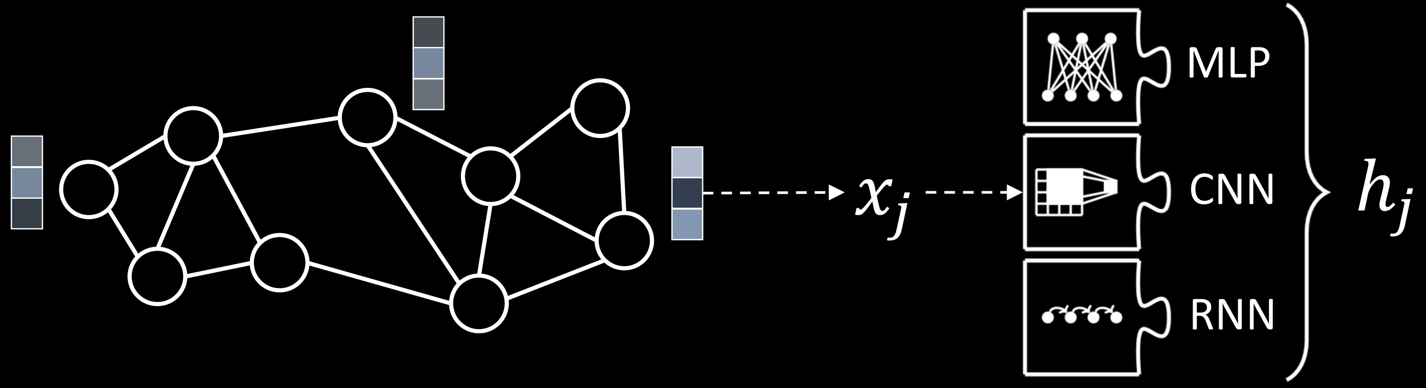


The teleport probability α controls the effective neighborhood size

From (A)PPNP to PPRGo

Predict then Propagate: Diffuse individual logits using PageRank

Neural network (depth & structure) is decoupled from propagation

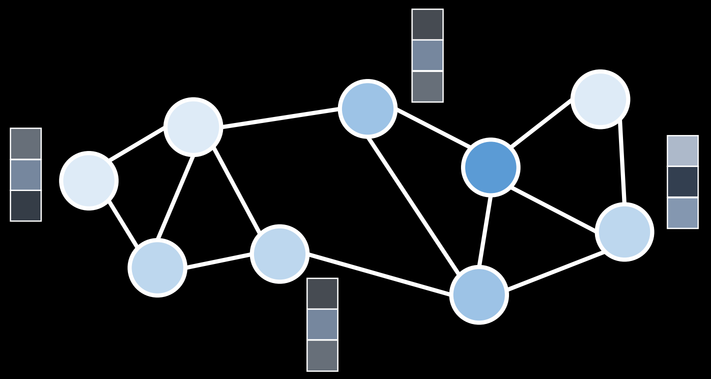


$$z_i = \pi_i^T \begin{bmatrix} - h_1 - \\ - h_2 - \\ \vdots \\ - h_n - \end{bmatrix}$$

map node features to logits

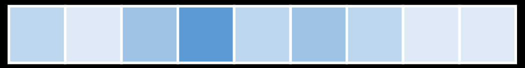
diffuse logits with PageRank

From (A)PPNP to PPRGo



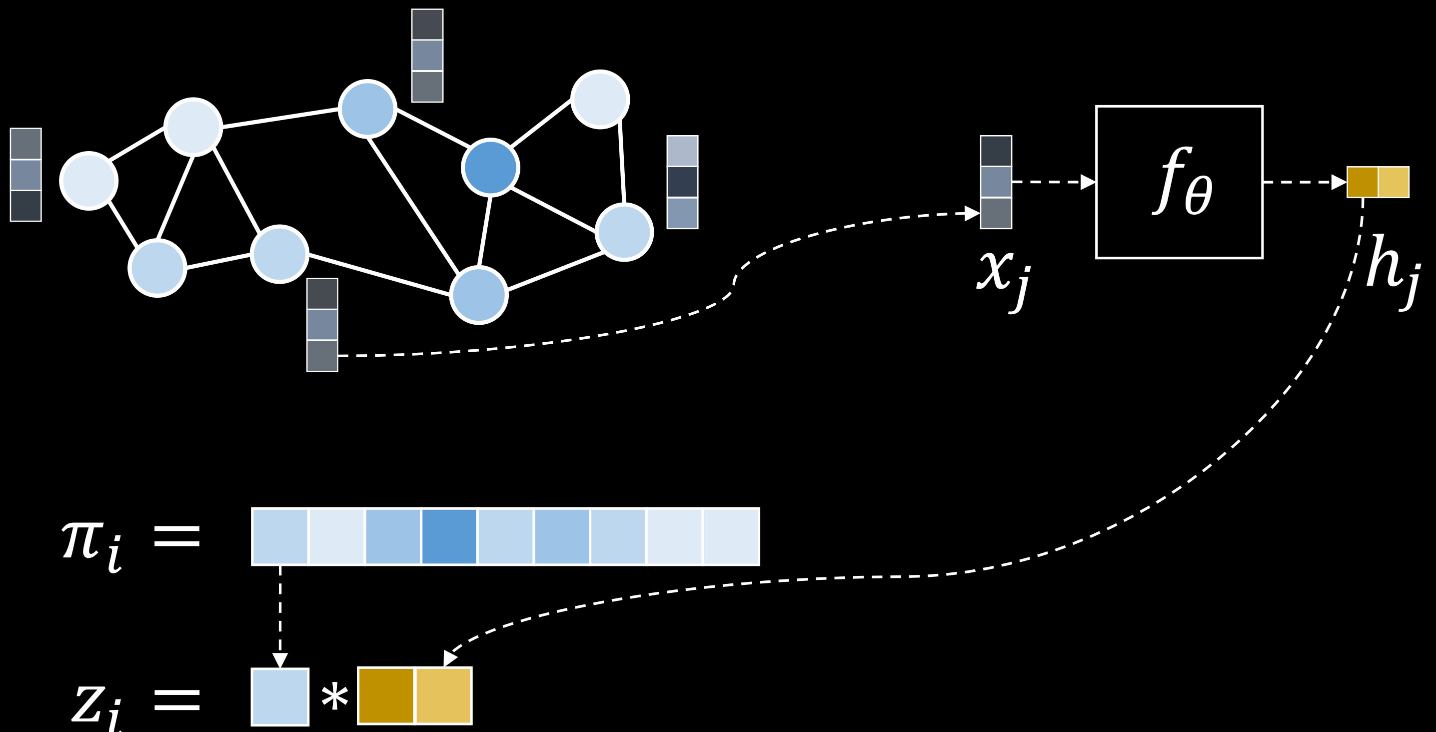
$$f_\theta$$

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

$\pi_i =$ 

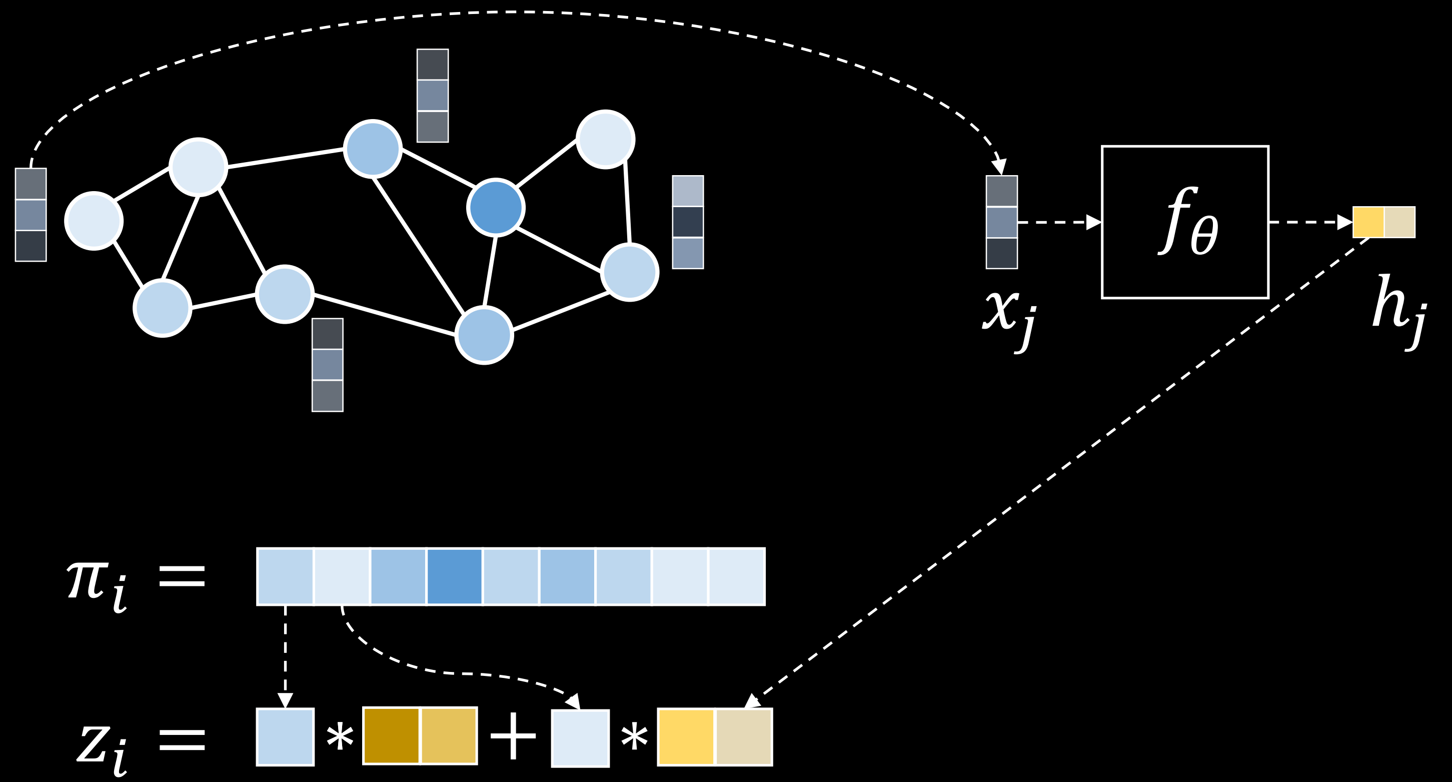
$z_i =$

From (A)PPNP to PPRGo



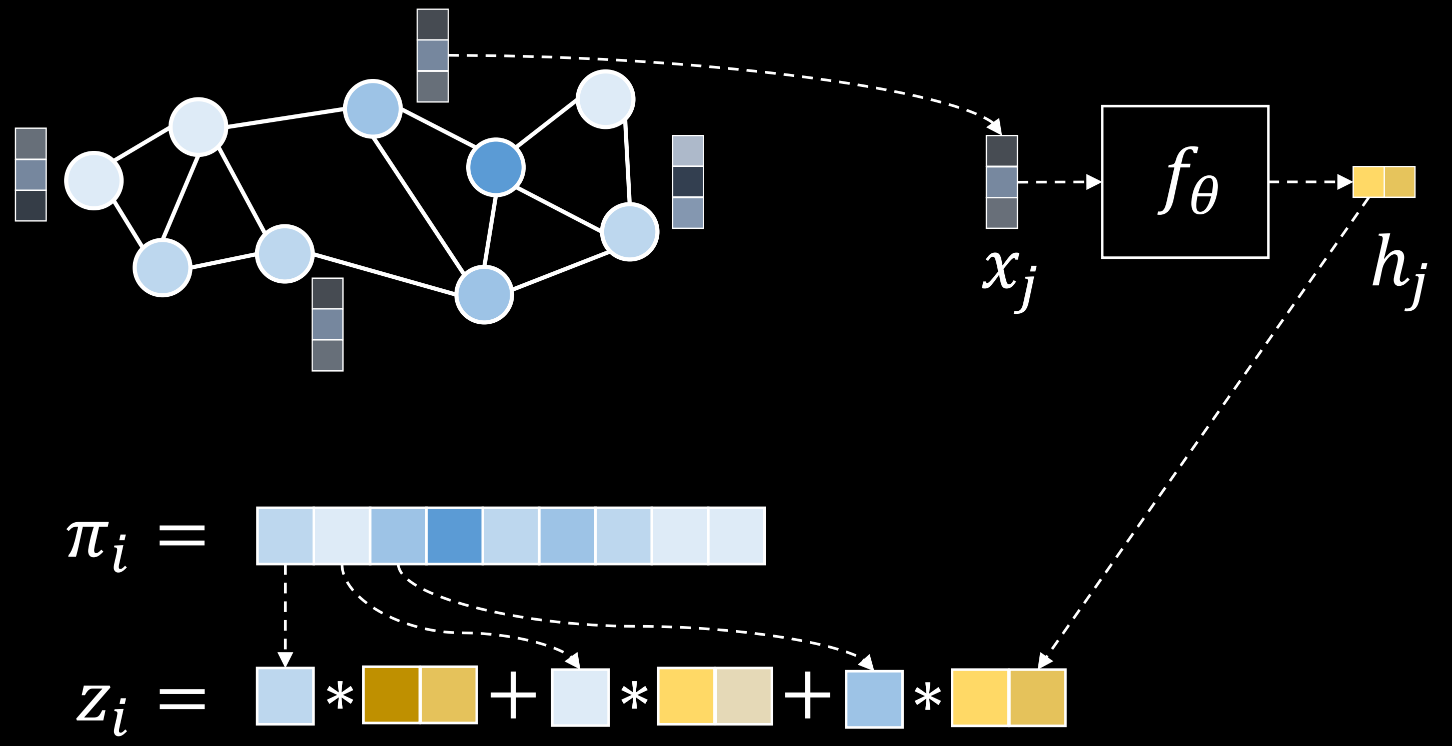
$$z_i = \pi_i^T \begin{bmatrix} - h_1 - \\ - h_2 - \\ \vdots \\ - h_n - \end{bmatrix}$$

From (A)PPNP to PPRGo



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

From (A)PPNP to PPRGo

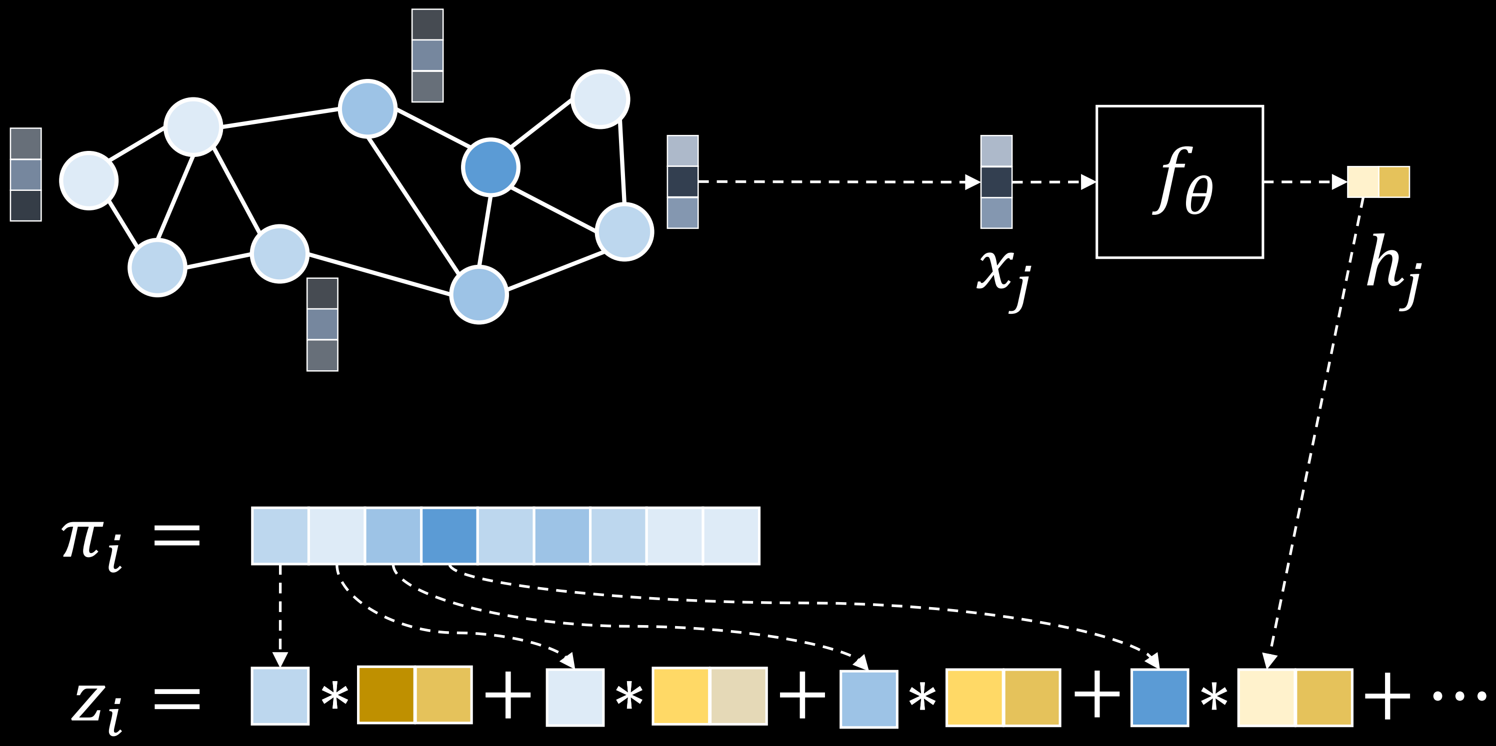


$$z_i = \pi_i^T \begin{bmatrix} - h_1 - \\ - h_2 - \\ \vdots \\ - h_n - \end{bmatrix}$$

$\pi_i =$

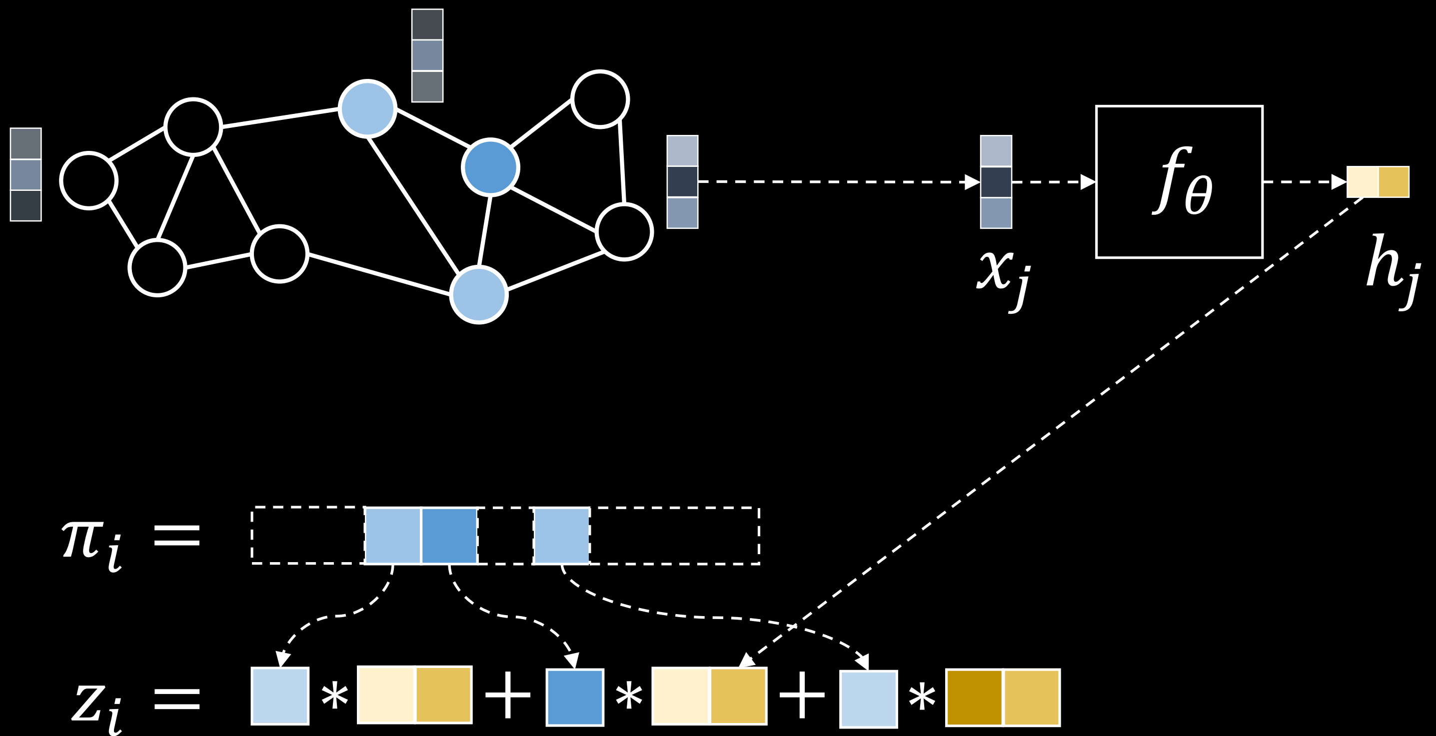
$z_i =$ * + * + *

From (A)PPNP to PPRGo



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

From (A)PPNP to PPRGo



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

Computing PageRank

Training	**Inference**
Approximate sparse PPR	1-3 Power Iterations steps
1 diffusion step	Sparse Inference

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

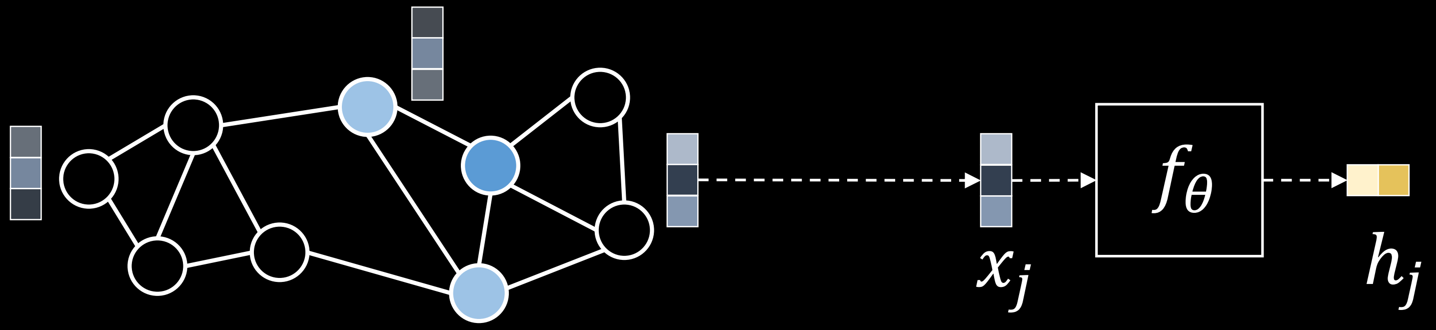
Power Iteration: $Q^{(0)} = \mathrm{H}, \qquad Q^{(p+1)} = \alpha\mathrm{H} + (1-\alpha)\mathrm{D}^{-1}\mathrm{A}Q^{(p)}$

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_i^ϵ for training nodes
2. Train the mapping $f_\theta(x_j)$ using SGD
3. Run Power Iteration during inference



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

All components can be implemented in a large-scale distributed setup

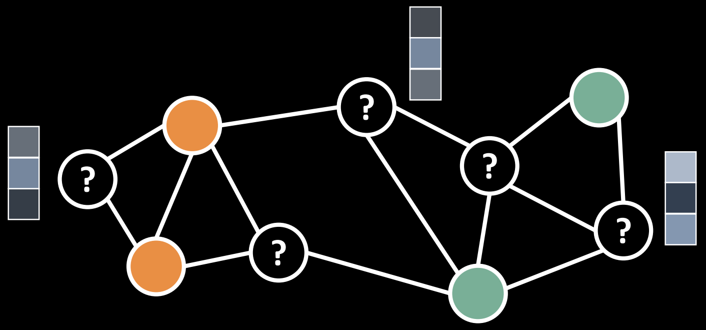
Experimental setup

Semi-supervised node classification

Sparsely labeled scenario

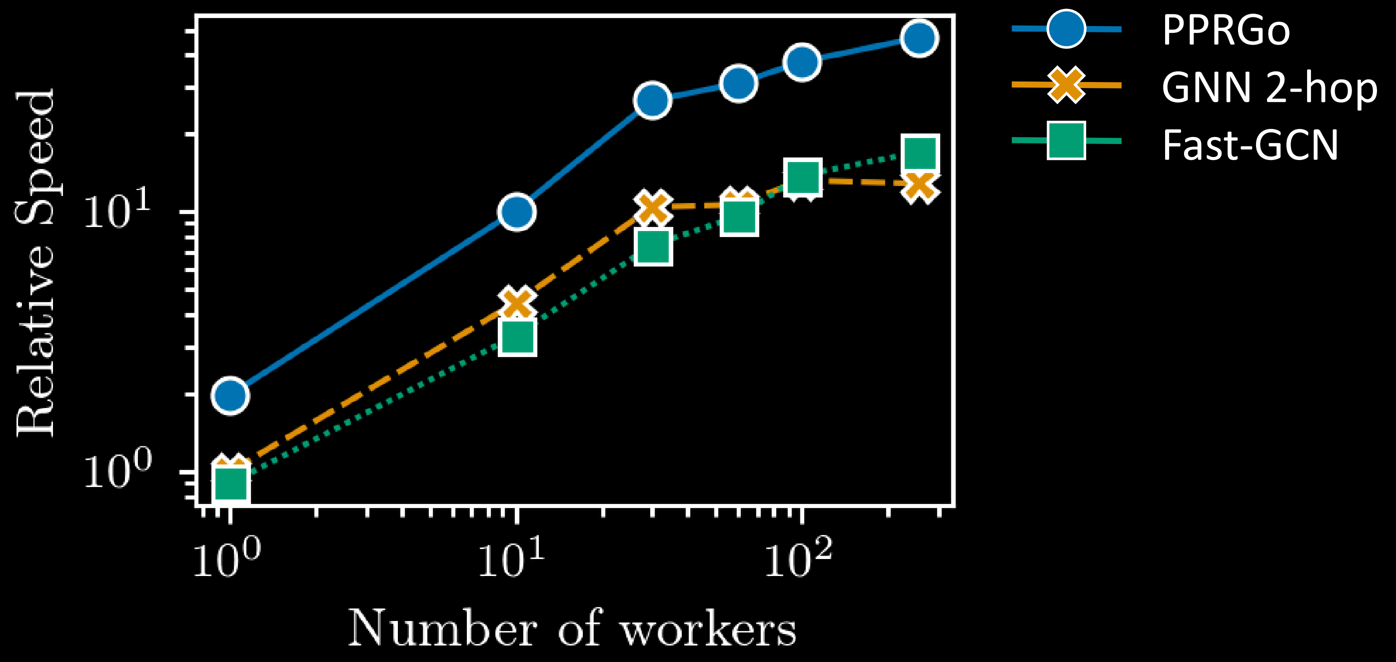
We introduce the MAG-Scholar dataset

- 12.4M nodes, 173M edges, and 2.8M features

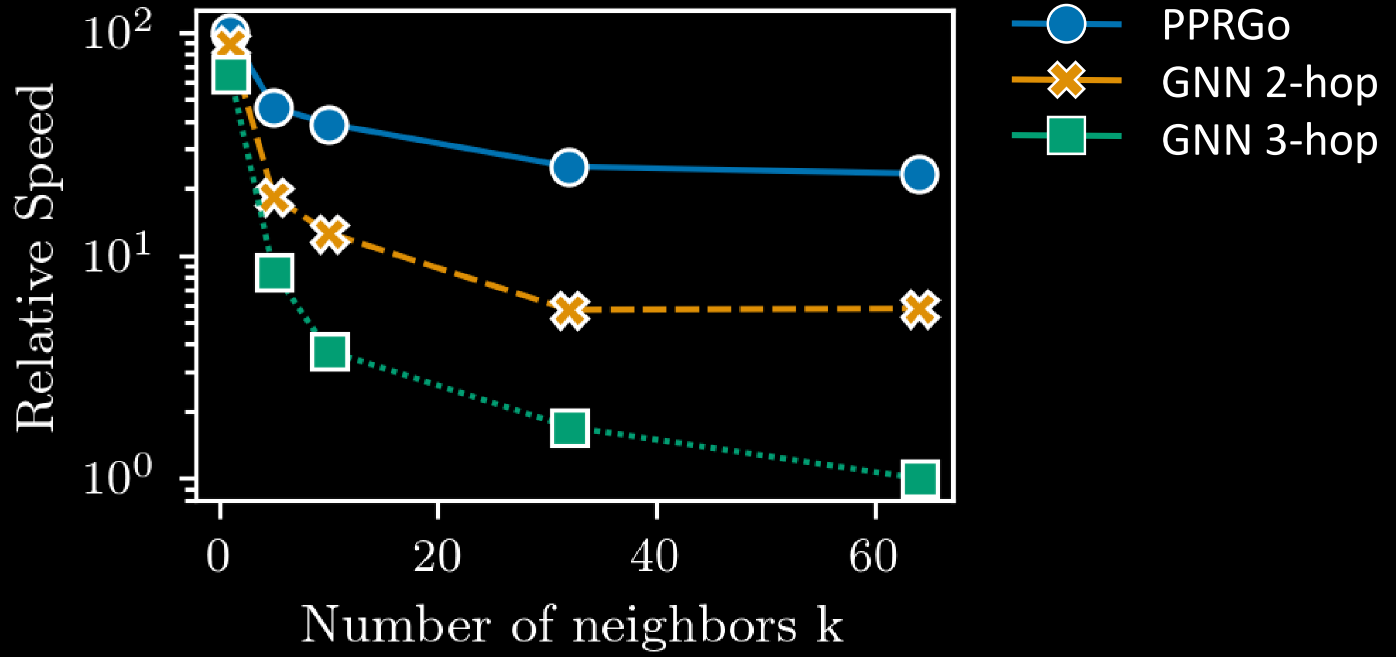


Measure: preprocessing + training + inference time and memory.

PPRGo utilizes additional workers best



PPRGo is most efficient



Performance breakdown on Reddit

	Runtime (s)				Memory (GB)		Accuracy
	Pre-proc.	Training	Inference	Total	RAM	GPU	
Cluster-GCN	1175	953	186	2310			
SGC	313	0.53	7470	7780			
PPRGo (1 PI step)	2.26	4.67	6.19	13.10			
PPRGo (2 PI steps)	2.22	4.1	10.5	16.8			

Performance breakdown on Reddit

	Runtime (s)				Memory (GB)		Accuracy
	Pre-proc.	Training	Inference	Total	RAM	GPU	
Cluster-GCN	1175	953	186	2310	20.97	0.071	17.1
SGC	313	0.53	7470	7780	10.12	0.027	12.1
PPRGo (1 PI step)	2.26	4.67	6.19	13.10	5.56	0.073	26.5
PPRGo (2 PI steps)	2.22	4.1	10.5	16.8	5.42	0.073	26.6

Performance on different datasets

	PubMed			Reddit			Mag-Scholar-C		
	Time (s)	Mem.	Acc.	Time (s)	Mem.	Acc.	Time (s)	Mem.	Acc.
Cluster-GCN	54.3	1.90	74.5	2310	21.04	17.1	>24h	-	-
SGC	5.3	2.17	75.7	7780	10.15	12.1	>24h	-	-
PPRGo ($\epsilon = 10^{-4}, k = 32$)	3.8	1.63	75.2	16.8	5.49	26.5	98.9	24.51	69.3
PPRGo ($\epsilon = 10^{-2}, k = 32$)	2.9	1.62	73.7	16.3	5.61	26.6	89.0	24.59	63.4

Performance on different datasets

	PubMed			Reddit			Mag-Scholar-C		
	Time (s)	Mem.	Acc.	Time (s)	Mem.	Acc.	Time (s)	Mem.	Acc.
Cluster-GCN	54.3	1.90	74.5	2310	21.04	17.1	>24h	-	-
SGC	5.3	2.17	75.7	7780	10.15	12.1	>24h	-	-
PPRGo ($\epsilon = 10^{-4}, k = 32$)	3.8	1.63	75.2	16.8	5.49	26.5	98.9	24.51	69.3
PPRGo ($\epsilon = 10^{-2}, k = 32$)	2.9	1.62	73.7	16.3	5.61	26.6	89.0	24.59	63.4

Trade speed for accuracy

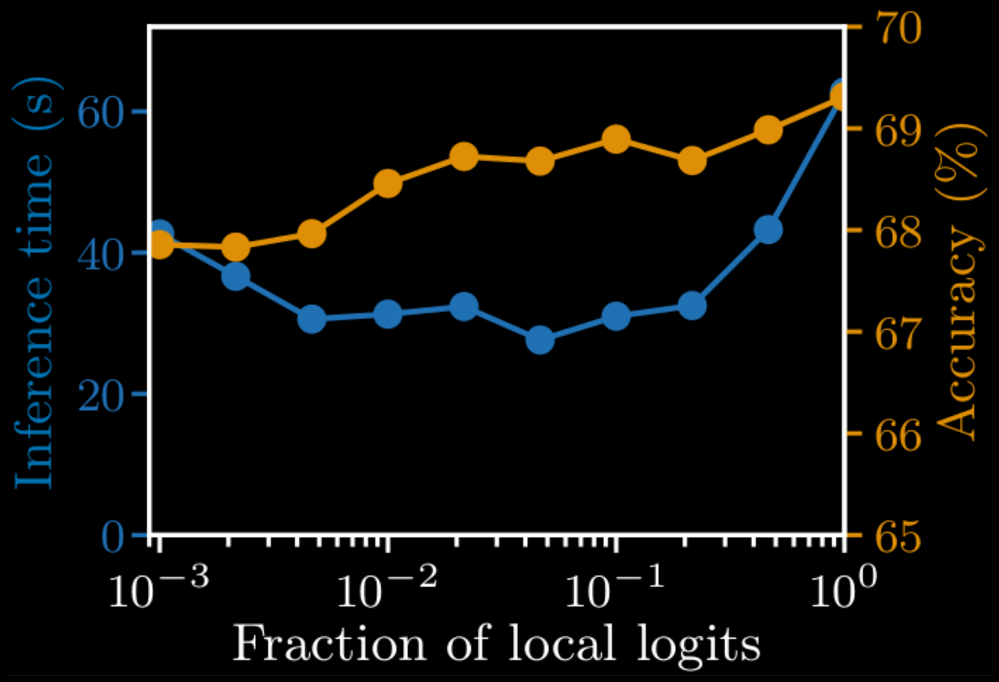
Accuracy (%)

ϵ \ top-k neighbors	2	4	8	16	32	64	128	256
10^{-4}	61.3	63.5	64.0	66.6	69.3	69.5	70.6	69.8
10^{-3}	61.4	63.2	65.5	66.0	66.7	66.9	66.9	66.9
10^{-2}	61.2	61.7	63.4	63.4	63.4	63.4	63.4	63.4
10^{-1}	59.6	59.2	59.2	59.2	59.2	59.2	59.2	59.2

Runtime (s)

ϵ \ top-k neighbors	2	4	8	16	32	64	128	256
10^{-4}	16.6	17.6	17.7	19.4	22.6	26.3	31.9	32.6
10^{-3}	15.5	16.9	16.5	17.8	17.7	18.5	18.3	17.7
10^{-2}	16.5	15.7	16.2	15.7	16.8	16.3	15.9	16.1
10^{-1}	16.7	16.0	15.9	15.3	16.5	15.9	15.8	15.2

Efficient inference

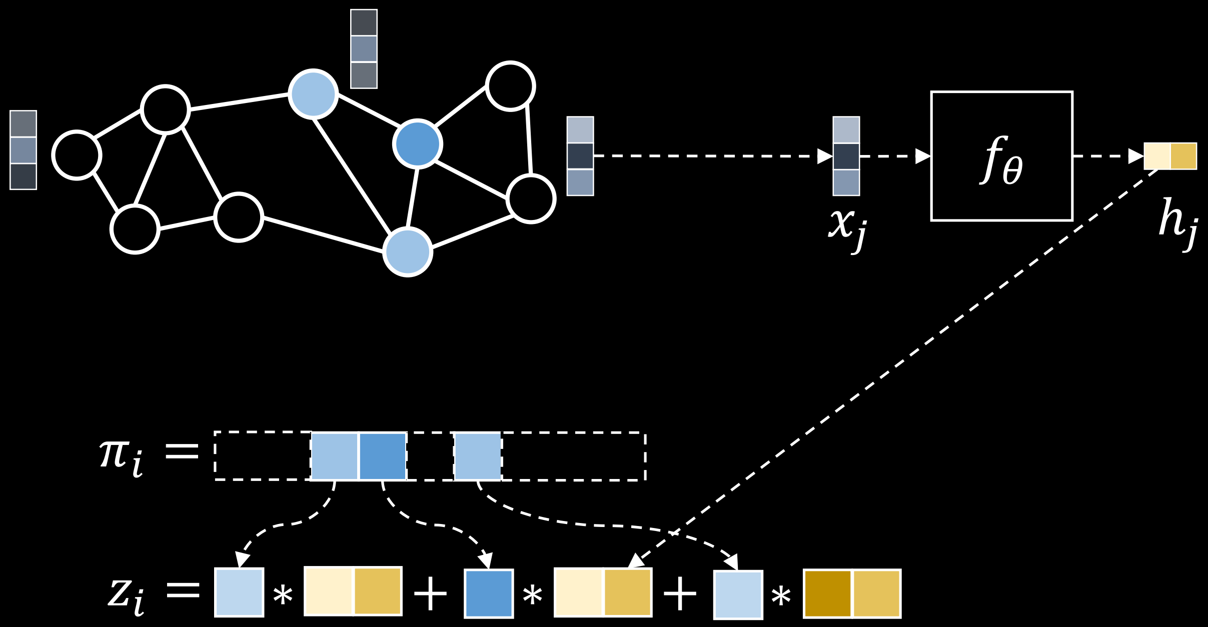


PPRGo

Utilizes distributed training significantly better

< 2 minutes runtime on a single machine for 12M nodes

Speed-up at no cost to accuracy



www.daml.in.tum.de/pprgo

@abojchevski, @klicperajo