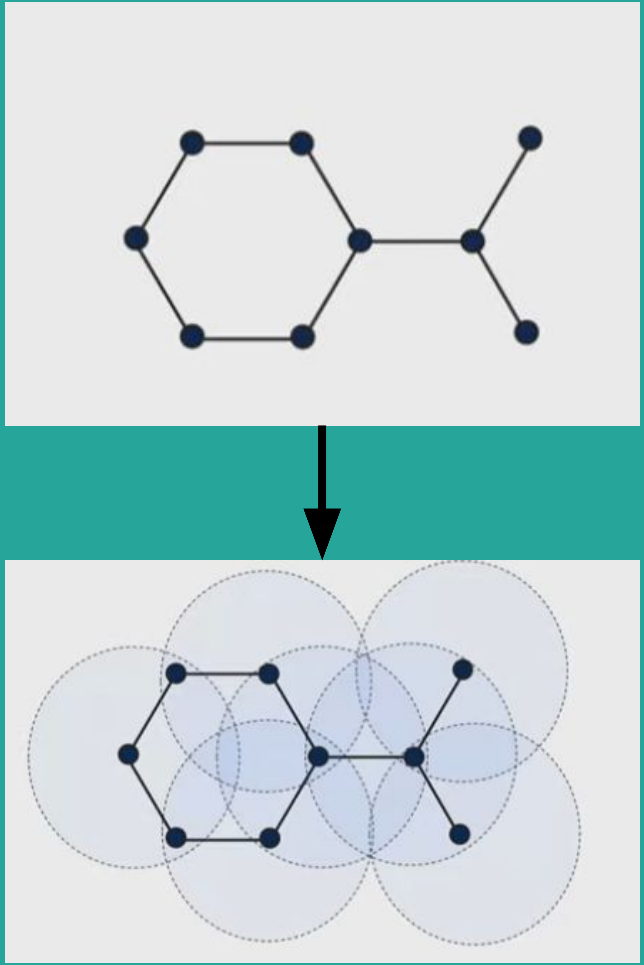
GNNs

Collin Giguere, Zanhua Huang

What are GNNs?

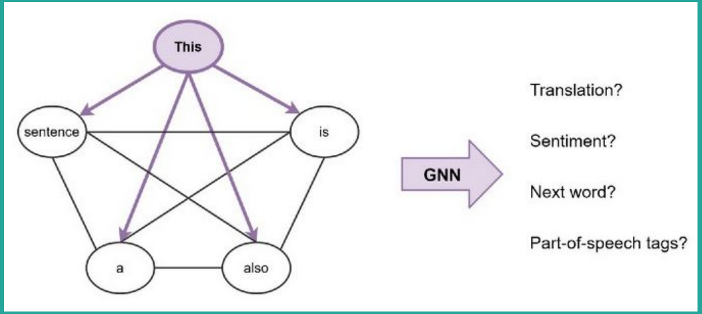
- Capture dependencies via message passing between nodes
- Each node retains information about its neighborhood with arbitrary depth
- Most common uses include node labeling (member classification), edge predictions (recommenders), and graph classification

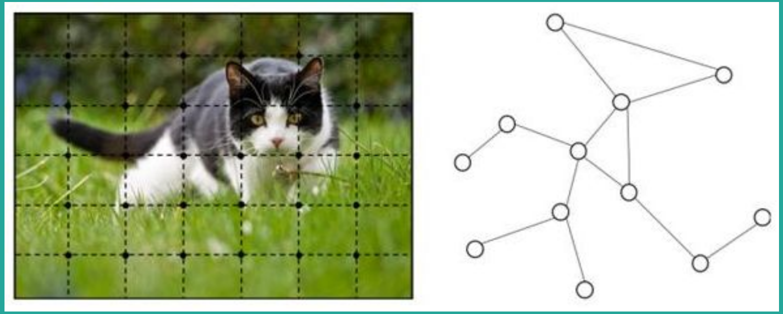


Why GNNs?

Many problems easily modeled by graphs are highly conducive to deep learning, so GNN was developed to model such problems like chemical reactions, molecular fingerprints, social networks, drug interactions, etc.

Other problems that are not traditionally modeled with graphs can also gain a lot from a GNN model (dependency graph of sentences, scene graph of an image)

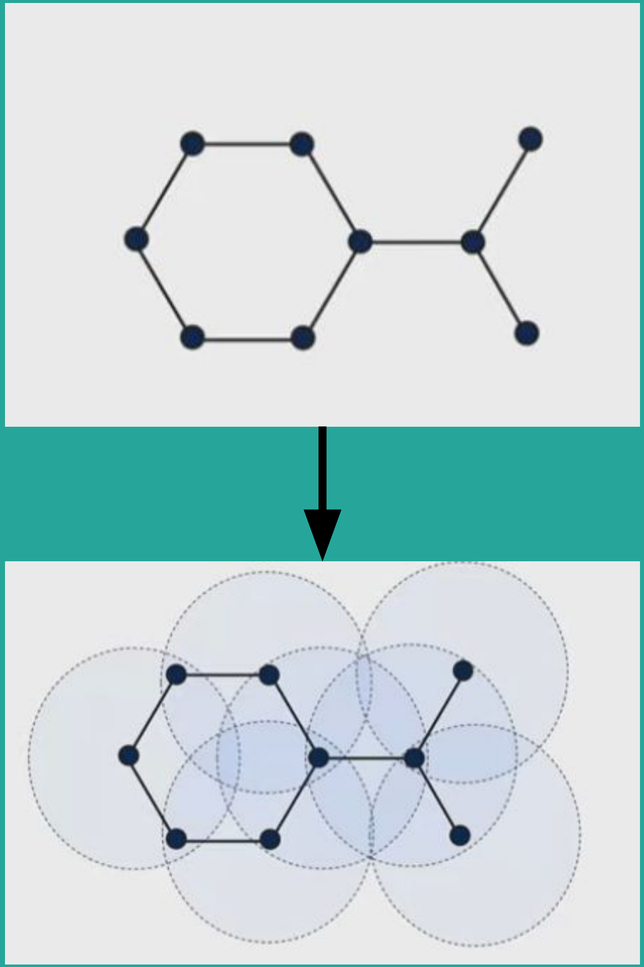




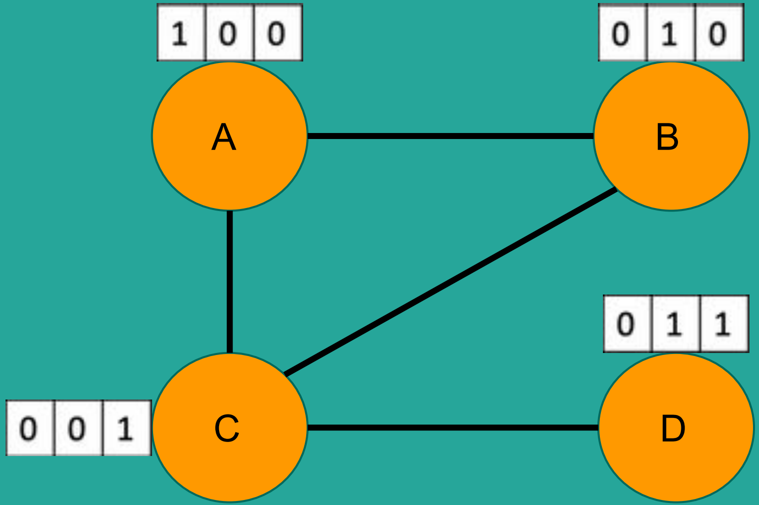
Message Passing

At each time step:

1. Nodes prepare their messages
2. Pass this message along applicable edges to neighbors
3. Each node summarizes their messages along with their own data
4. Each node processes their neighborhoods data in some model-specific way



Message Passing



$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 1 & 1 & 1 \\ 1 & 2 & 2 \\ 0 & 1 & 2 \end{bmatrix}$$

Examples of GNNs

- Pinterest recommending posts
- Spotify recommending music
- Amazon recommending items
- Pharmaceutical interactions

Deep Graph Library (DGL)

MinjieWang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, & Zheng Zhang

Amazon Web Services, AWS Shanghai AI Lab, New York University, NYU Shanghai

The current tools have not yet caught up to advancing research in deep graph learning

- Tensor computation on graphs is lacking
- Gaps between current deep learning models and graphs
    - Hardware optimized for dense tensor operations
    - Traditional memory access patterns

$$S = \begin{bmatrix} 0 & 0 & 1 & 2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

DGL aims to be the new tool to facilitate deep learning on graphs.

- Define easy to use primitives to coordinate the forward inference and backward gradient computing paths
- Abstracts away the need to manipulate graph data
- Object-Oriented approach to graphs
- Framework independent

SpMM

(Sparse-dense matrix multiplication)
Used to aggregate nodes' inbound edges

$$Y = AX$$

Sparse adjacency matrix

Dense feature matrix

$$\text{g-SpMM}_{\mathcal{G},\phi_z,\rho} : \mathbb{R}^{|\mathcal{V}|\times d_1}, \mathbb{R}^{|\mathcal{V}|\times d_2}, \mathbb{R}^{|\mathcal{E}|\times d_3} \mapsto \mathbb{R}^{|\mathcal{V}|\times d_4}$$

$$\mathbf{z}_v = \rho\left(\left\{\phi_z\left(\mathbf{x}_u, \mathbf{y}_v, \mathbf{w}_e\right) : (u,e,v) \in \mathcal{E}\right\}\right), \quad \forall v \in \mathcal{V}.$$

SDDMM

(Sampled dense-dense matrix multiplication)
Used to aggregate edges' incident nodes

$$W = A \odot (XX^T)$$

Sparse adjacency matrix

Dense feature matrix

$$\text{g-SDDMM}_{\mathcal{G},\phi_m} : \mathbb{R}^{|\mathcal{V}|\times d_1}, \mathbb{R}^{|\mathcal{V}|\times d_2}, \mathbb{R}^{|\mathcal{E}|\times d_3} \mapsto \mathbb{R}^{|\mathcal{E}|\times d_4}$$

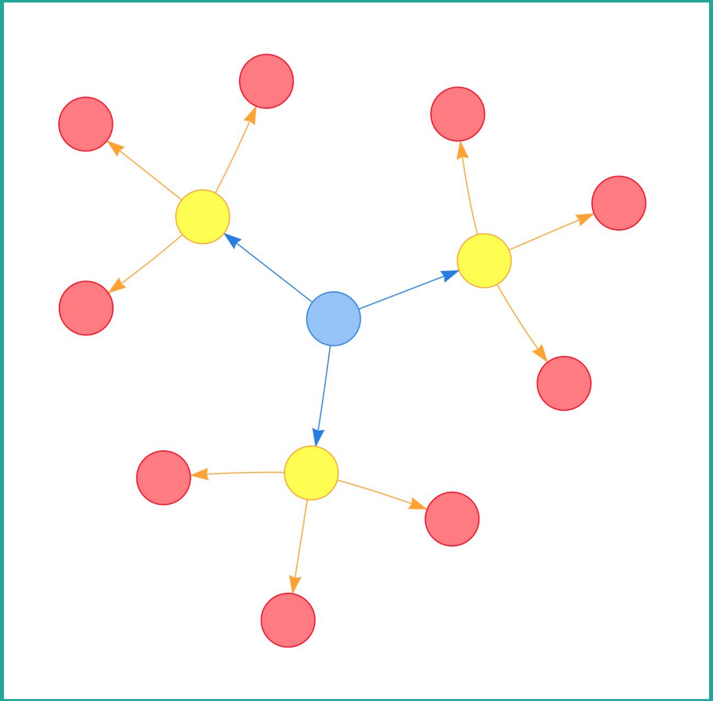
$$\mathbf{m}_e = \phi_m(\mathbf{x}_u, \mathbf{y}_v, \mathbf{w}_e), \quad \forall (u,e,v) \in \mathcal{E}.$$

What does this gain us?

- Forward inference path is essentially a series of SpMM to derive a stack of node representations
- The gradient w.r.t. SpMM and SDDMM inputs can be expressed as SpMM and SDDMM
- Lays foundation for optimizations
- SpMM naturally avoids generating intermediate storage for messages
- SDDMM avoids copying node representations to edges

Graph objects

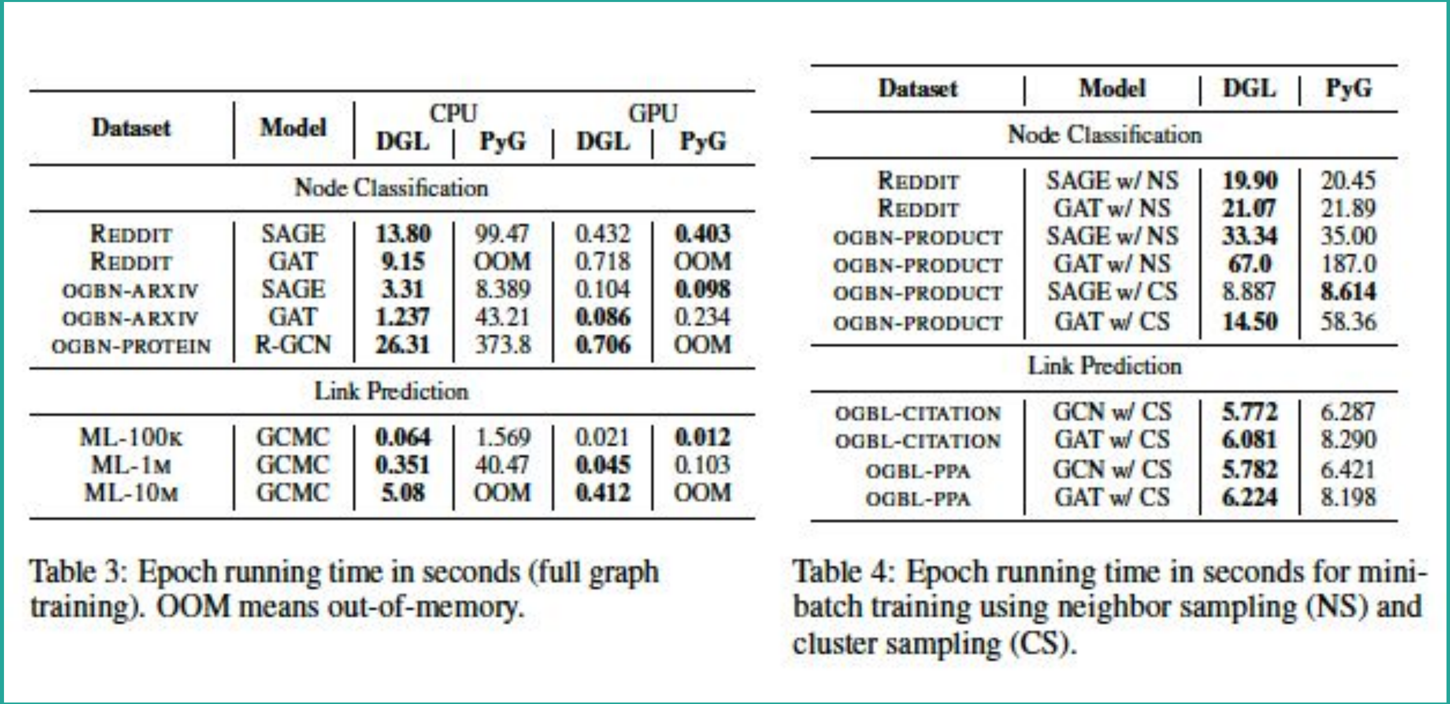
- DGL fully embraces the object-oriented paradigm at the graph level
- Adopts a familiar structure
- Exposes low-level structures to allow users to innovate beyond the API
- Seamless integration with DL frameworks



Framework-less

- DGL can be used on top of any of the most popular frameworks (PyTorch, TensorFlow, MXNet).
- Not framework-agnostic
- OO approach helps abstract some changes away
- Implements sparse tensor operations natively
- Delegates the rest

Evaluation



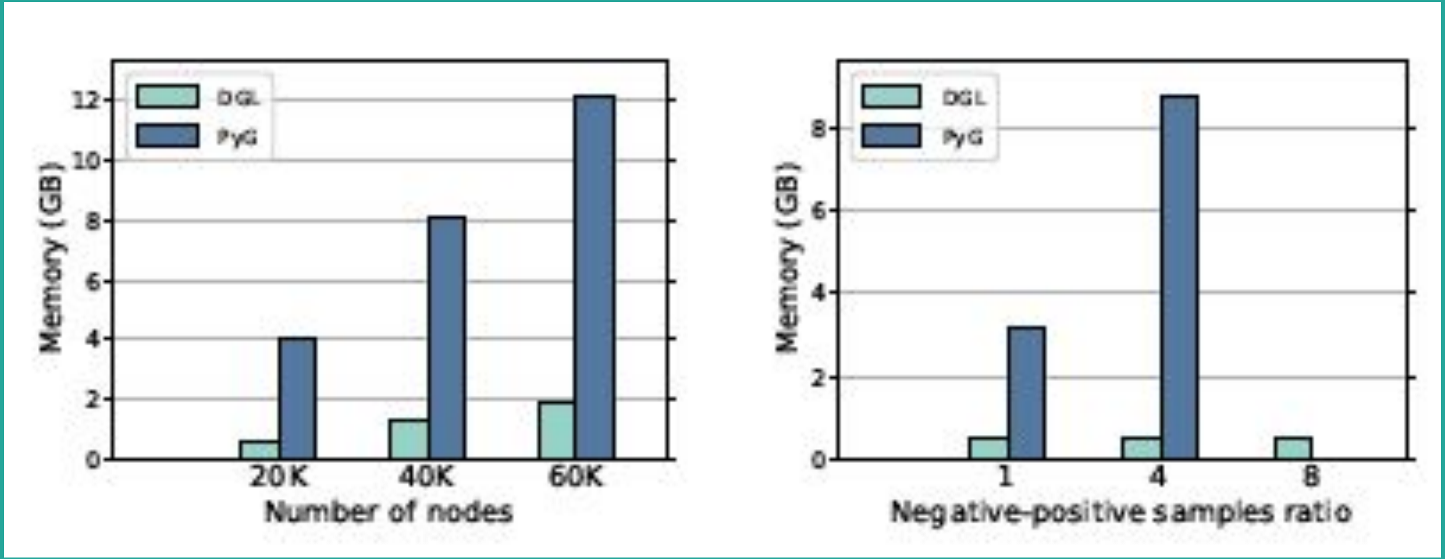
Dataset	Model	CPU DGL	CPU PyG	GPU DGL	GPU PyG
Node Classification					
Reddit	SAGE	**13.80**	99.47	0.432	**0.403**
Reddit	GAT	**9.15**	OOM	0.718	OOM
ogbn-arxiv	SAGE	**3.31**	8.389	0.104	**0.098**
ogbn-arxiv	GAT	**1.237**	43.21	**0.086**	0.234
ogbn-protein	R-GCN	**26.31**	373.8	**0.706**	OOM
Link Prediction					
ML-100k	GCMC	**0.064**	1.569	0.021	**0.012**
ML-1m	GCMC	**0.351**	40.47	**0.045**	0.103
ML-10m	GCMC	**5.08**	OOM	**0.412**	OOM

Table 3: Epoch running time in seconds (full graph training). OOM means out-of-memory.

Dataset	Model	DGL	PyG
Node Classification			
Reddit	SAGE w/ NS	**19.90**	20.45
Reddit	GAT w/ NS	**21.07**	21.89
ogbn-product	SAGE w/ NS	**33.34**	35.00
ogbn-product	GAT w/ NS	**67.0**	187.0
ogbn-product	SAGE w/ CS	8.887	**8.614**
ogbn-product	GAT w/ CS	**14.50**	58.36
Link Prediction			
ogbl-citation	GCN w/ CS	**5.772**	6.287
ogbl-citation	GAT w/ CS	**6.081**	8.290
ogbl-ppa	GCN w/ CS	**5.782**	6.421
ogbl-ppa	GAT w/ CS	**6.224**	8.198

Table 4: Epoch running time in seconds for mini-batch training using neighbor sampling (NS) and cluster sampling (CS).

Memory Management



Page deliberately
left blank

Aligraph: A Comprehensive Graph Neural Network Platform

Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, Jingren Zhou.
Alibaba Group

Current GNN challenges:

Graph data related to real-world commercial scenarios exhibits four properties

Large-scale:
- Current GNN: optimized for grid-structure data (images)
- How to scale on real-world graphs with exceedingly large size.
- Time and Space efficiency

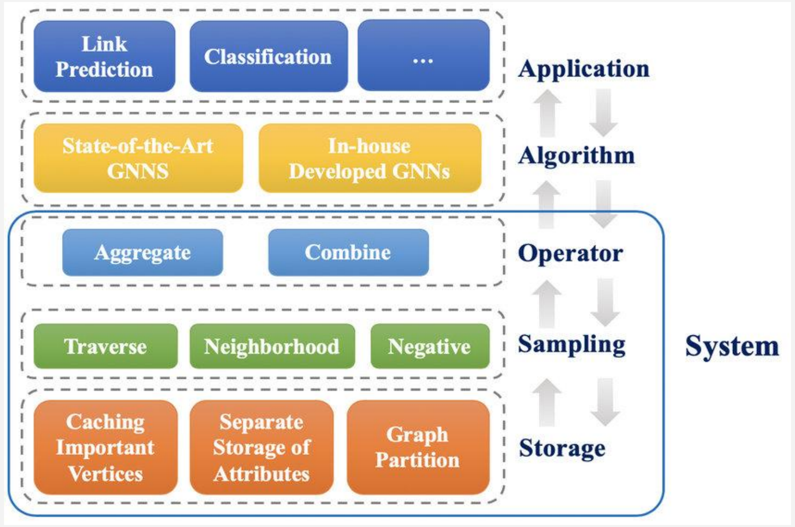
Heterogeneous:
- Nodes and edges: different types and attributes
- How to embedding to one result (vector)

Attributed:
- Topological structure information vs Unstructured attribute information
- How to unify them

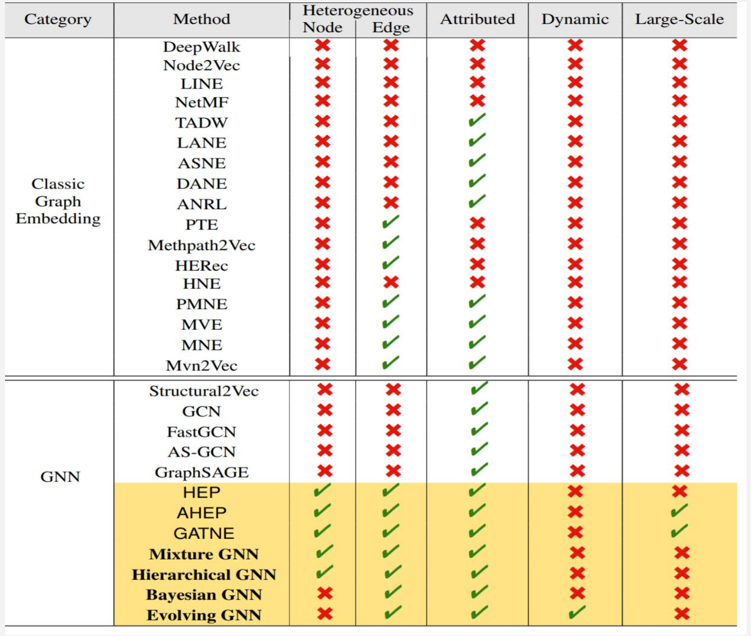
Dynamic:
- How to handle updates to the graph (nodes, edges, attributes,etc)
- Do not want to retrain from scratch

Contributions of Aligraph:

1. Designed a comprehensive graph neural network system.
2. Designed several GNN algorithms



Aligraph architecture



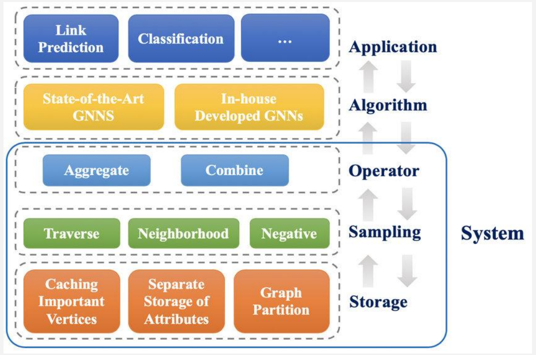
Category	Method	Heterogeneous Node	Heterogeneous Edge	Attributed	Dynamic	Large-Scale
Classic Graph Embedding	DeepWalk	✗	✗	✗	✗	✗
	Node2Vec	✗	✗	✗	✗	✗
	LINE	✗	✗	✗	✗	✗
	NetMF	✗	✗	✗	✗	✗
	TADW	✗	✗	✓	✗	✗
	LANE	✗	✗	✓	✗	✗
	ASNE	✗	✗	✓	✗	✗
	DANE	✗	✗	✓	✗	✗
	ANRL	✗	✗	✓	✗	✗
	PTE	✗	✓	✗	✗	✗
	Methpath2Vec	✗	✓	✗	✗	✗
	HERec	✗	✓	✗	✗	✗
	HNE	✗	✗	✗	✗	✗
	PMNE	✗	✓	✓	✗	✗
	MVE	✗	✓	✓	✗	✗
	MNE	✗	✓	✓	✗	✗
	Mvn2Vec	✗	✓	✓	✗	✗
GNN	Structural2Vec	✗	✗	✓	✗	✗
	GCN	✗	✗	✓	✗	✗
	FastGCN	✗	✗	✓	✗	✗
	AS-GCN	✗	✗	✓	✗	✗
	GraphSAGE	✗	✗	✓	✗	✗
	HEP	✓	✓	✓	✗	✗
	AHEP	✓	✓	✓	✗	✓
	GATNE	✓	✓	✓	✗	✓
	Mixture GNN	✓	✓	✓	✗	✗
	Hierarchical GNN	✓	✓	✓	✗	✗
	Bayesian GNN	✗	✓	✓	✗	✗
	Evolving GNN	✗	✓	✓	✓	✗

New GNN algorithms shaded in yellow

Graph Partition:

Goal:

1. Save the entire graph in distributed environment
2. Minimize the number of crossing edges between different workers.



Aligraph architecture

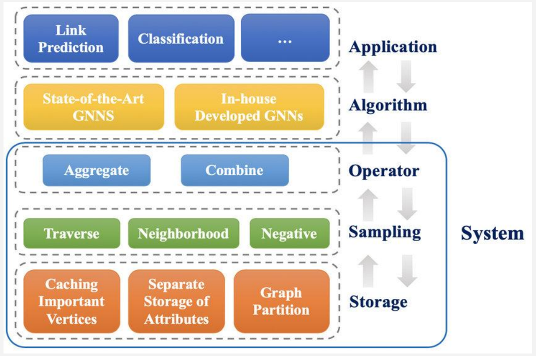
Methods:

1. METIS	sparse graphs
2. Vertex cut and edge cut partitions	dense graphs
3. 2-D partition	used when number of workers is fixed
4. Streaming-style partition strategy	graphs with frequently edge updates

Separate Storage of Attributes:

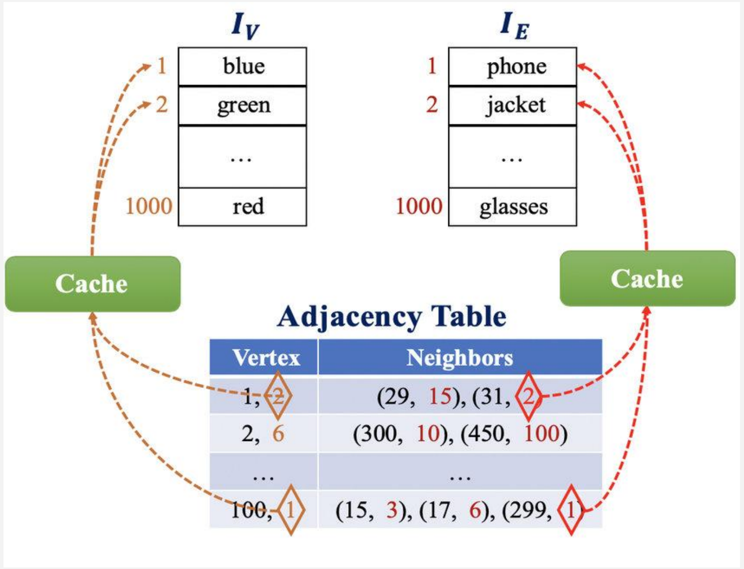
Overview:

1. Attributes are too large to store in AdjList
2. Many vertices/edges share same attributes
3. Separately store attributes and index them
4. Use Cache



Aligraph architecture

Methods:



1	blue
2	green
	…
1000	red

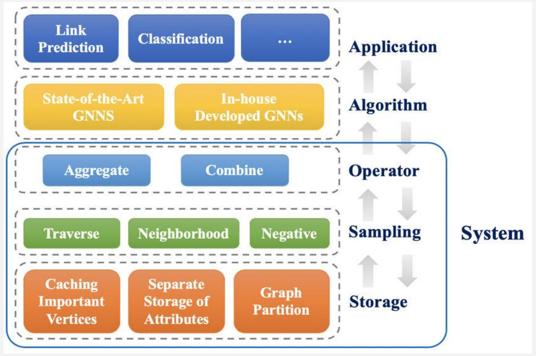
1	phone
2	jacket
	…
1000	glasses

Vertex	Neighbors
1, 2	(29, 15), (31, 2)
2, 6	(300, 10), (450, 100)
…	…
100, 1	(15, 3), (17, 6), (299, 1)

Caching Neighbors of Important Vertices:

Overview:

1. If a vertex v is frequently accessed by other vertices, we can store v's out-neighbors.
2. Don't want to cache too many things if the number of neighbors of v is large.



Aligraph architecture

Methods:

1. Define the importance of each node
2. Cache out-neighbors of a vertex if its importance value > a threshold

$$Imp^{(k)}(v) = \frac{D_i^{(k)}(v)}{D_o^{(k)}(v)}.$$

Sampling:

Overview:

Sample a subset of neighbors with aligned sizes so that:

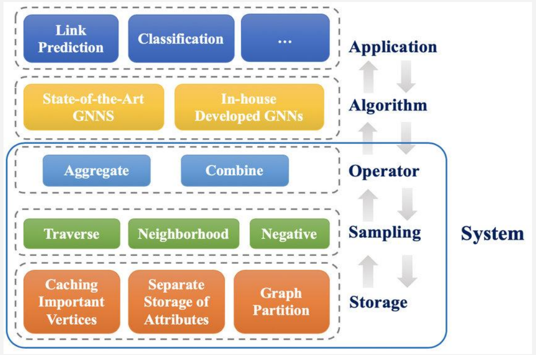
1. Size is smaller
2. Size is aligned (easier to do convolution)

Provided Samplers:

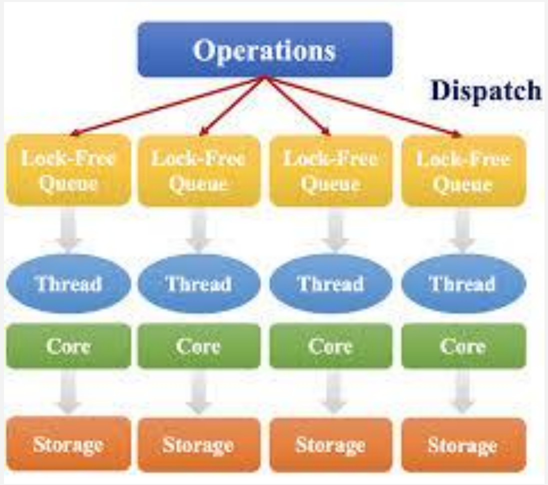
1. TRAVERSE: sample over vertices and edges
2. NEIGHBORHOOD: get contexts of (sampled) neighbors
3. NEGATIVE: generate negative samples to accelerate the convergence of the training process

Improvements:

split the vertices on a graph server into groups



Aligraph architecture



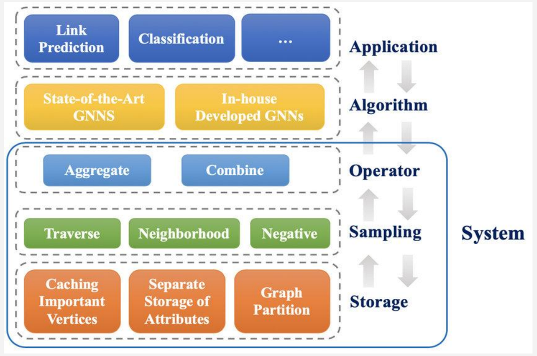
Operator:

Overview:

- Aggregate: collects the information of each vertex's neighbors to produce a unified result.
- Combine: combine the result of Aggregate with current feature vector.

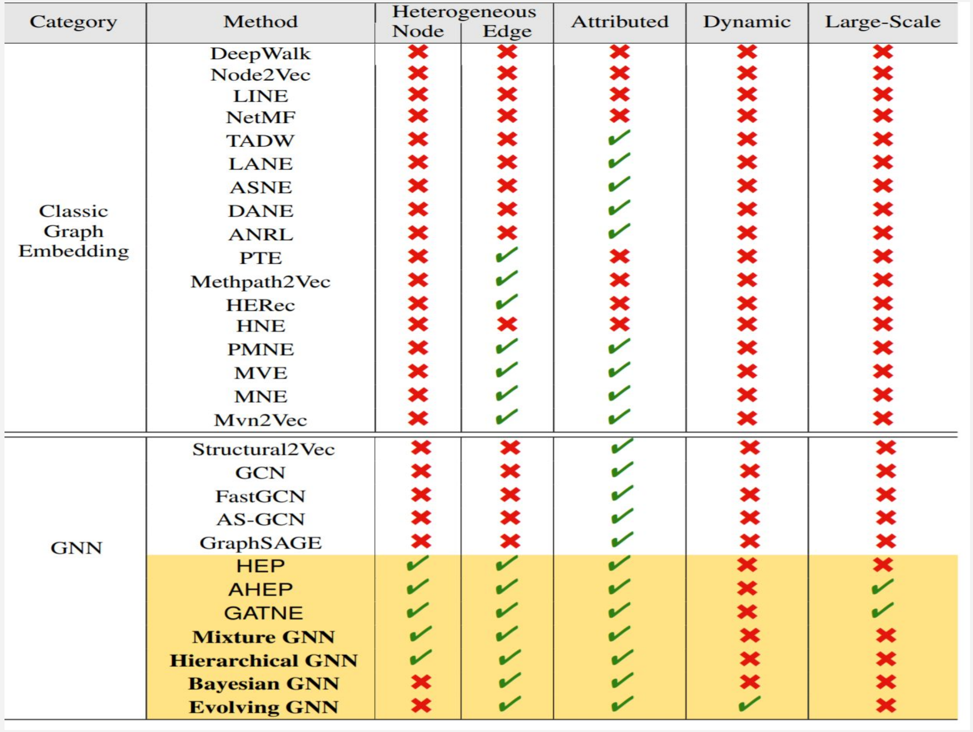
Acceleration:

- share the set of sampled neighbors for all vertices in the mini-batch.
- share the intermediate feature vectors h_v among vertices in the same mini-batch



Aligraph architecture

Algorithms:

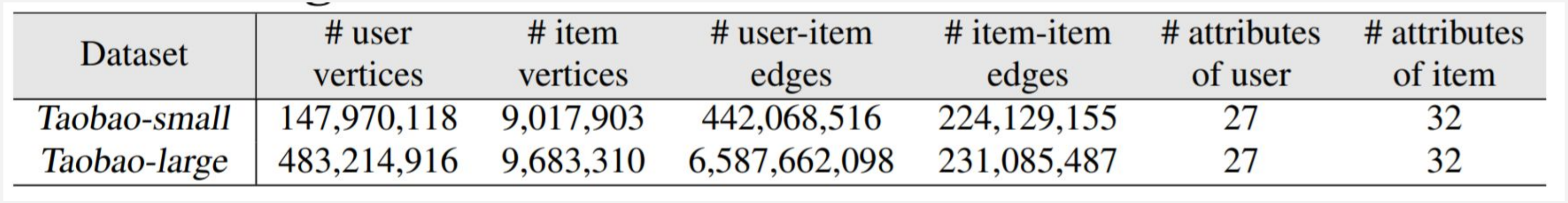


Category	Method	Heterogeneous		Attributed	Dynamic	Large-Scale
		Node	Edge			
Classic Graph Embedding	DeepWalk	✗	✗	✗	✗	✗
	Node2Vec	✗	✗	✗	✗	✗
	LINE	✗	✗	✗	✗	✗
	NetMF	✗	✗	✗	✗	✗
	TADW	✗	✗	✓	✗	✗
	LANE	✗	✗	✓	✗	✗
	ASNE	✗	✗	✓	✗	✗
	DANE	✗	✗	✓	✗	✗
	ANRL	✗	✗	✓	✗	✗
	PTE	✗	✓	✗	✗	✗
	Methpath2Vec	✗	✓	✗	✗	✗
	HERec	✗	✓	✗	✗	✗
	HNE	✗	✗	✗	✗	✗
	PMNE	✗	✓	✓	✗	✗
	MVE	✗	✓	✓	✗	✗
	MNE	✗	✓	✓	✗	✗
	Mvn2Vec	✗	✓	✓	✗	✗
GNN	Structural2Vec	✗	✗	✓	✗	✗
	GCN	✗	✗	✓	✗	✗
	FastGCN	✗	✗	✓	✗	✗
	AS-GCN	✗	✗	✓	✗	✗
	GraphSAGE	✗	✗	✓	✗	✗
	HEP	✓	✓	✓	✗	✗
	AHEP	✓	✓	✓	✗	✓
	GATNE	✓	✓	✓	✗	✓
	Mixture GNN	✓	✓	✓	✗	✗
	Hierarchical GNN	✓	✓	✓	✗	✗
	Bayesian GNN	✗	✓	✓	✗	✗
	Evolving GNN	✗	✓	✓	✓	✗

New GNN algorithms shaded in yellow

Evaluation (system):

Dataset



Dataset	# user vertices	# item vertices	# user-item edges	# item-item edges	# attributes of user	# attributes of item
Taobao-small	147,970,118	9,017,903	442,068,516	224,129,155	27	32
Taobao-large	483,214,916	9,683,310	6,587,662,098	231,085,487	27	32

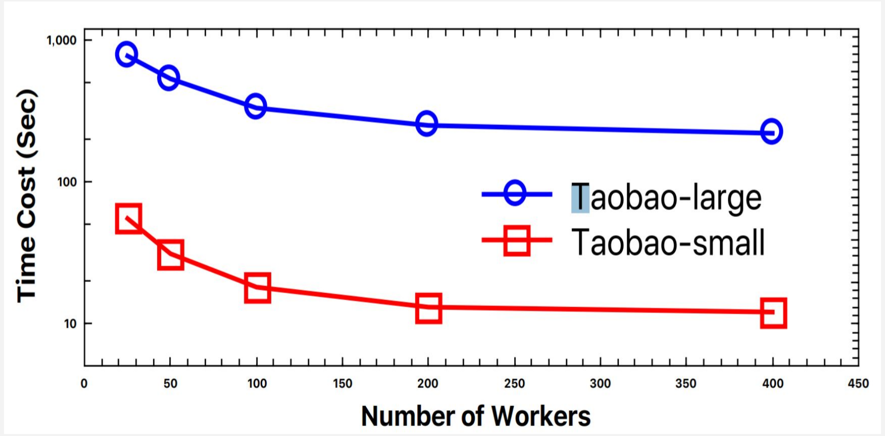
Evaluation (system):

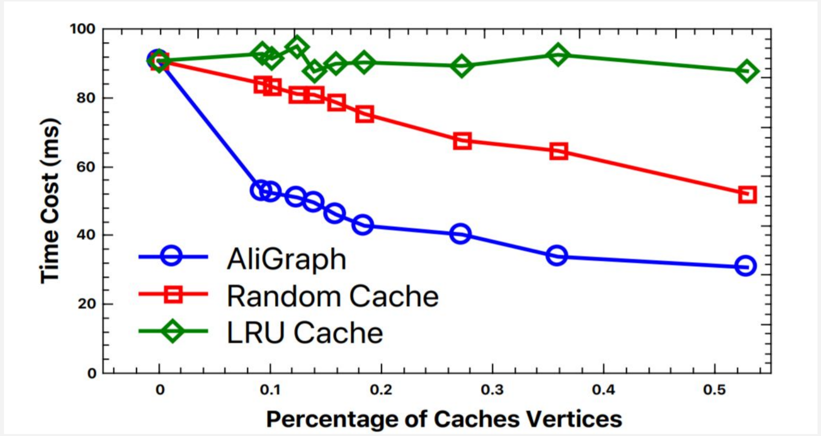
Graph Building:

1. time explicitly decreases w.r.t. the number of workers
2. 5 minutes for Taobao-large vs. several hours using PowerGraph

Effects of Caching Neighbors:

1. All 1-hop neighbours are cached
2. Vary the threshold to control 2-hop cached neighbours
3. Performs better than Random and LRU.

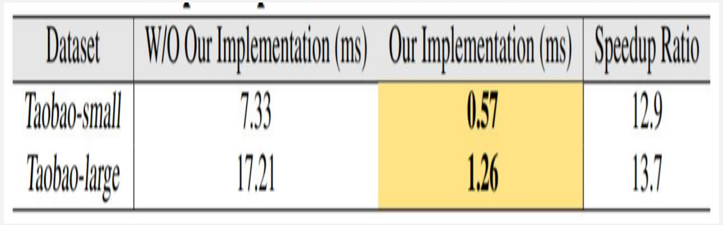




Evaluation (system):

Effects of Operators:

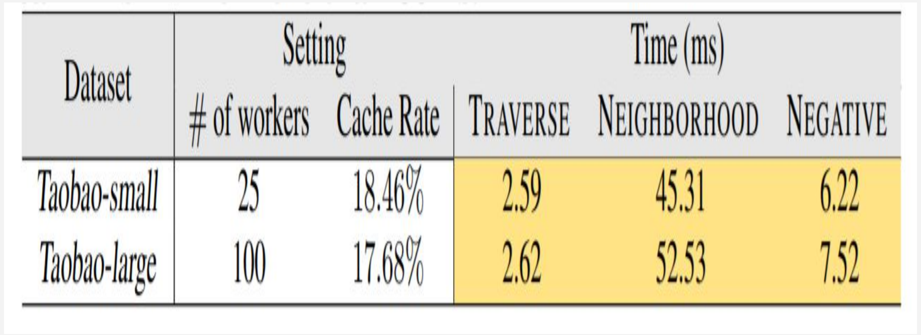
1. Less time because of caching



Dataset	W/O Our Implementation (ms)	Our Implementation (ms)	Speedup Ratio
Taobao-small	7.33	**0.57**	12.9
Taobao-large	17.21	**1.26**	13.7

Effects of Sampling:

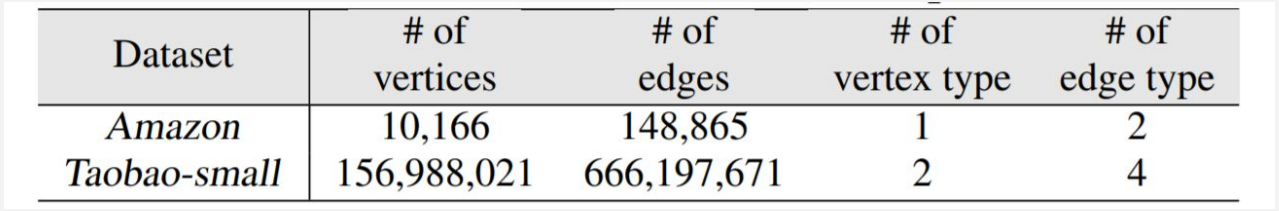
1. sample with the batch size of 512 and cache rate 20%
2. very efficient: finish in 60ms
3. Scalable: sampling time grows slowly w.r.t. the graph size



Dataset	Setting		Time (ms)		
	# of workers	Cache Rate	TRAVERSE	NEIGHBORHOOD	NEGATIVE
Taobao-small	25	18.46%	2.59	45.31	6.22
Taobao-large	100	17.68%	2.62	52.53	7.52

Evaluation (algorithms):

Dataset



Dataset	# of vertices	# of edges	# of vertex type	# of edge type
Amazon	10,166	148,865	1	2
Taobao-small	156,988,021	666,197,671	2	4

Taobao-small is chosen due to the reason of the scalability of several competitors

Competitors:
DeepWalk, LINE, Node2Vec, ANRL, Methpath2Vec, PMNE, MVE, MNE, Methpath2Vec, Structural2Vec, GCN, Fast-GCN, AS-GCN, GraphSAGE and HEP.

Metrics:

1. execution time, ROC-AUC, PR-AUC, F1-score, hit recall rate
2. algorithm applied on the widely adopted link prediction task

Evaluation (algorithms):

AHEP

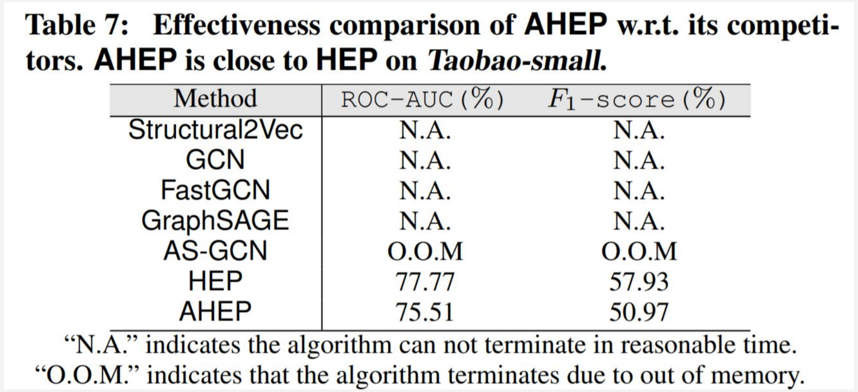


Table 7: Effectiveness comparison of AHEP w.r.t. its competitors. AHEP is close to HEP on *Taobao-small*.

Method	ROC-AUC (%)	F_1-score (%)
Structural2Vec	N.A.	N.A.
GCN	N.A.	N.A.
FastGCN	N.A.	N.A.
GraphSAGE	N.A.	N.A.
AS-GCN	O.O.M	O.O.M
HEP	77.77	57.93
AHEP	75.51	50.97

"N.A." indicates the algorithm can not terminate in reasonable time.
"O.O.M." indicates that the algorithm terminates due to out of memory.

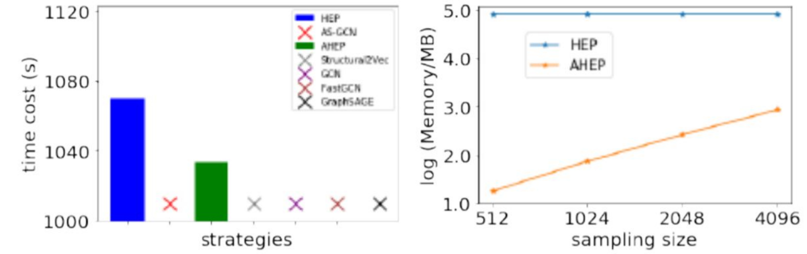


Figure 10: Average memory cost and running time of per batch. × indicates the algorithm can not terminate in reasonable time. AHEP is 2–3 faster than HEP and uses much less memory on *Taobao-small*.

Evaluation (algorithms):

GATNE:

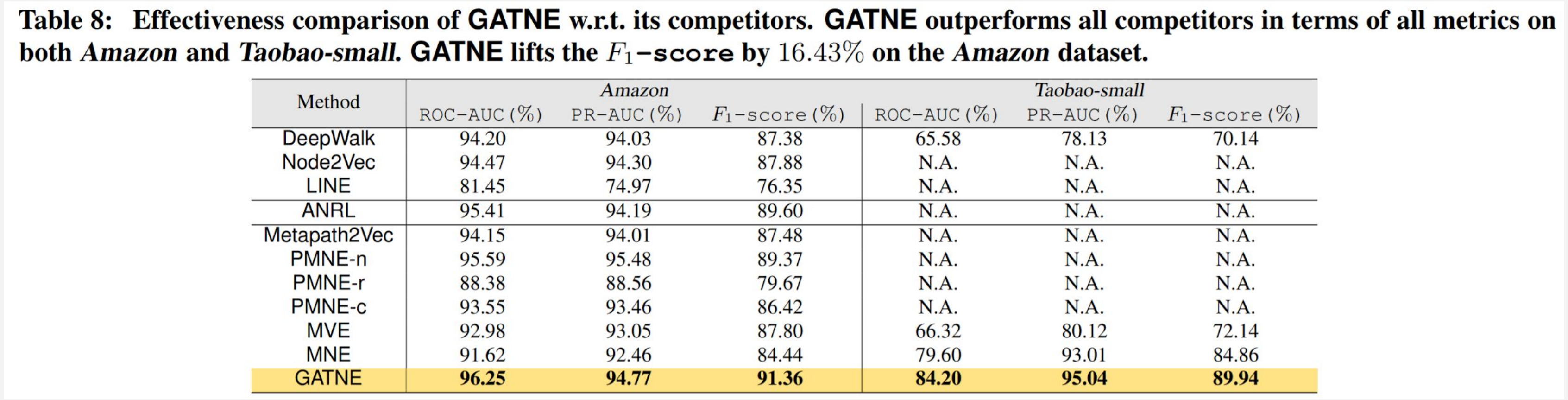


Table 8: Effectiveness comparison of GATNE w.r.t. its competitors. GATNE outperforms all competitors in terms of all metrics on both *Amazon* and *Taobao-small*. GATNE lifts the F_1-score by 16.43% on the *Amazon* dataset.

Method	*Amazon*			*Taobao-small*		
	ROC-AUC (%)	PR-AUC (%)	F_1-score (%)	ROC-AUC (%)	PR-AUC (%)	F_1-score (%)
DeepWalk	94.20	94.03	87.38	65.58	78.13	70.14
Node2Vec	94.47	94.30	87.88	N.A.	N.A.	N.A.
LINE	81.45	74.97	76.35	N.A.	N.A.	N.A.
ANRL	95.41	94.19	89.60	N.A.	N.A.	N.A.
Metapath2Vec	94.15	94.01	87.48	N.A.	N.A.	N.A.
PMNE-n	95.59	95.48	89.37	N.A.	N.A.	N.A.
PMNE-r	88.38	88.56	79.67	N.A.	N.A.	N.A.
PMNE-c	93.55	93.46	86.42	N.A.	N.A.	N.A.
MVE	92.98	93.05	87.80	66.32	80.12	72.14
MNE	91.62	92.46	84.44	79.60	93.01	84.86
GATNE	**96.25**	**94.77**	**91.36**	**84.20**	**95.04**	**89.94**

Evolving GNN:

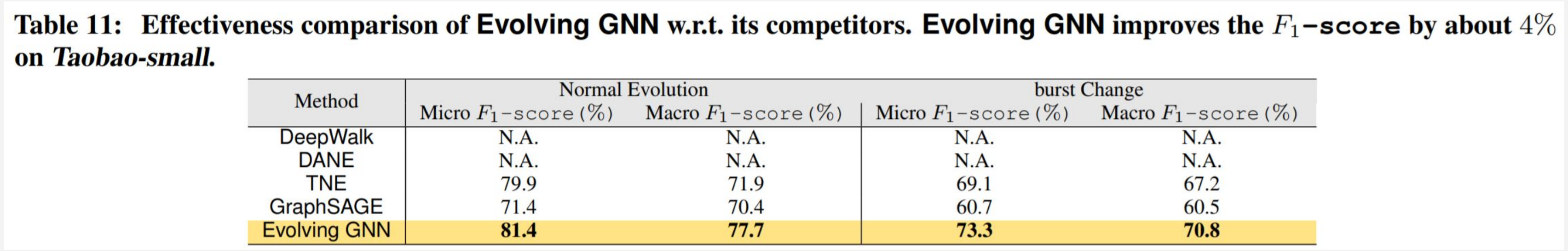


Table 11: Effectiveness comparison of Evolving GNN w.r.t. its competitors. Evolving GNN improves the F_1-score by about 4% on *Taobao-small*.

Method	Normal Evolution		burst Change	
	Micro F_1-score (%)	Macro F_1-score (%)	Micro F_1-score (%)	Macro F_1-score (%)
DeepWalk	N.A.	N.A.	N.A.	N.A.
DANE	N.A.	N.A.	N.A.	N.A.
TNE	79.9	71.9	69.1	67.2
GraphSAGE	71.4	70.4	60.7	60.5
Evolving GNN	**81.4**	**77.7**	**73.3**	**70.8**

Evaluation (algorithms):

Mixture GNN:

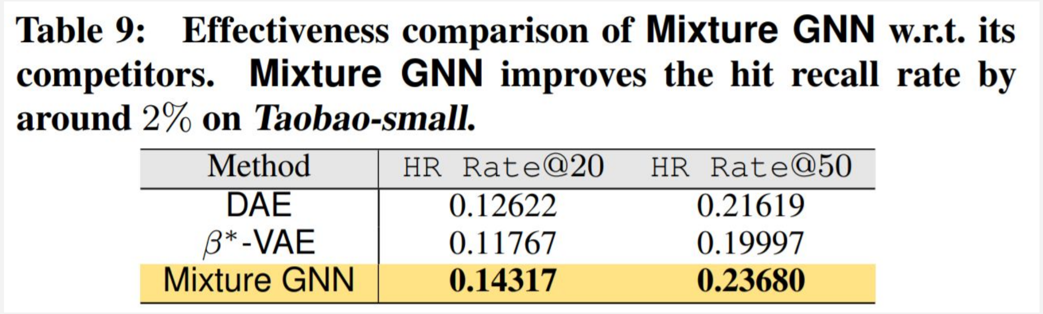


Table 9: Effectiveness comparison of Mixture GNN w.r.t. its competitors. Mixture GNN improves the hit recall rate by around 2% on *Taobao-small*.

Method	HR Rate@20	HR Rate@50
DAE	0.12622	0.21619
β^*-VAE	0.11767	0.19997
Mixture GNN	**0.14317**	**0.23680**

Hierarchical GNN:

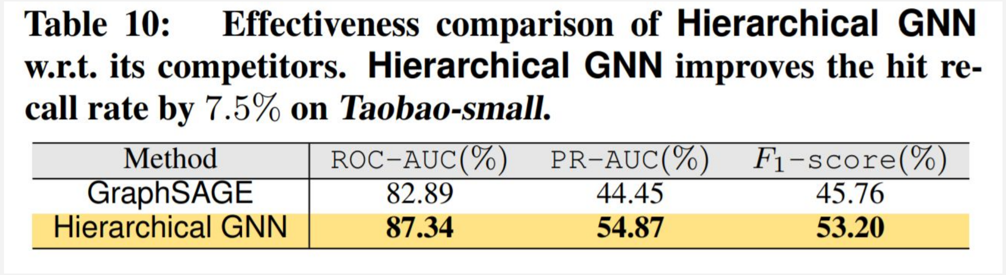


Table 10: Effectiveness comparison of Hierarchical GNN w.r.t. its competitors. Hierarchical GNN improves the hit recall rate by 7.5% on *Taobao-small*.

Method	ROC-AUC(%)	PR-AUC(%)	F_1-score(%)
GraphSAGE	82.89	44.45	45.76
Hierarchical GNN	**87.34**	**54.87**	**53.20**

Evaluation (algorithms):

Bayesian GNN:

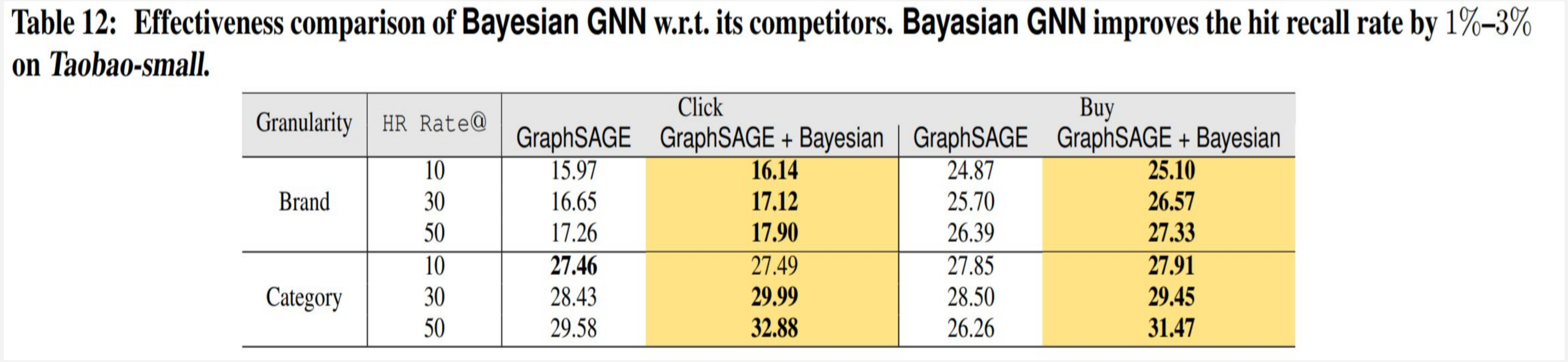


Table 12: Effectiveness comparison of Bayesian GNN w.r.t. its competitors. Bayasian GNN improves the hit recall rate by 1%–3% on *Taobao-small*.

Granularity	HR Rate@	Click		Buy	
		GraphSAGE	GraphSAGE + Bayesian	GraphSAGE	GraphSAGE + Bayesian
Brand	10	15.97	**16.14**	24.87	**25.10**
	30	16.65	**17.12**	25.70	**26.57**
	50	17.26	**17.90**	26.39	**27.33**
Category	10	**27.46**	27.49	27.85	**27.91**
	30	28.43	**29.99**	28.50	**29.45**
	50	29.58	**32.88**	26.26	**31.47**

GraphSAINT

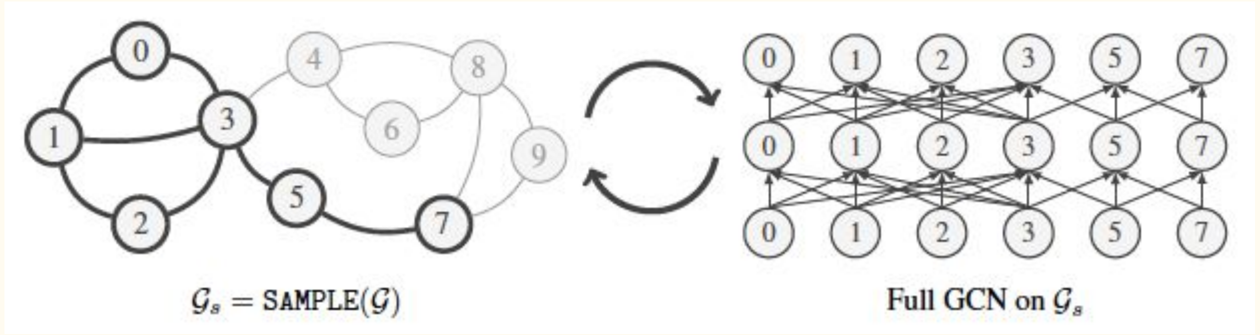
Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava,
Rajgopal Kannan, Viktor Prasanna

Neighbor Explosion

A node can have a relatively small number of one-hop neighbors while having an exponentially higher number of two and three-hop neighbors. Current methods use layer sampling to help alleviate this issue, but GraphSAINT uses a graph sampling based inductive learning method to construct mini-batches by sampling the training graph, rather than the nodes or edges across GCN layers.

General Idea

1. Perform sampling on the training graph G. (Related nodes appear together =>bias)
2. Construct GCN on the sampled graph.
3. Do the forward and backward propagation with normalization
4. Start next minibatch


$\mathcal{G}_s = \text{SAMPLE}(\mathcal{G})$
Full GCN on $\mathcal{G}_s$

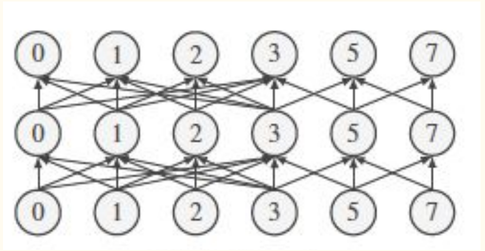
Goal:

1. Give an algorithm of sampling.
2. Give an algorithm of normalization.

Normalization

Given the feature vectors X(l)s at layer l

$$\zeta_v^{(\ell+1)} = \sum_{u \in \mathcal{V}} \frac{\widetilde{\boldsymbol{A}}_{v,u}}{\alpha_{u,v}} \left(\boldsymbol{W}^{(\ell)}\right)^{\mathsf{T}} \boldsymbol{x}_u^{(\ell)} \mathbb{1}_{u|v} = \sum_{u \in \mathcal{V}} \frac{\widetilde{\boldsymbol{A}}_{v,u}}{\alpha_{u,v}} \tilde{\boldsymbol{x}}_u^{(\ell)} \mathbb{1}_{u|v},$$



is an unbiased estimator for the feature vector of v at layer l+1 if

1. Assuming that each layer independently learns an embedding
2. $$\alpha_{u,v} = \frac{p_{u,v}}{p_v}.$$
3. Point 2 shows that the unbiased estimator depends on the sampling probability

Variance (and edge samplers)

1. variance can be calculated for the unbiased estimator.
2. Find a sampling probability that minimize the TOTAL variance

$$\zeta = \sum_{\ell} \sum_{v \in \mathcal{G}_s} \frac{\zeta_v^{(\ell)}}{p_v}$$

is the sum of estimators.
We want the variance of this estimator to be small.
Sample each edge following:

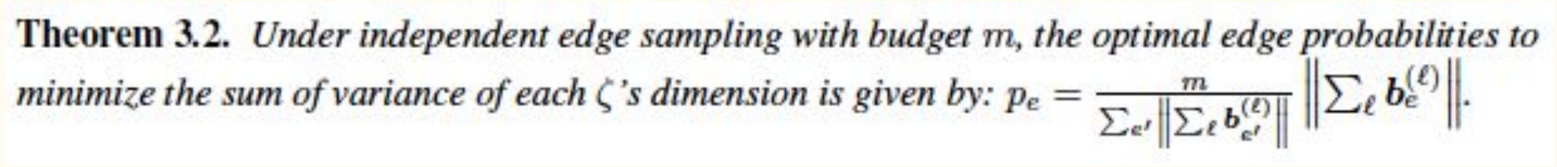
$$p_e = \frac{m}{\sum_{e'} \left\| \sum_{\ell} \boldsymbol{b}_{e'}^{(\ell)} \right\|} \left\| \sum_{\ell} \boldsymbol{b}_e^{(\ell)} \right\|$$

Samplers

1. Random node sampler

$$P(u) \propto \left|\left|\tilde{A}_{:,u}\right|\right|^2$$

2. Random edge sampler



Theorem 3.2. *Under independent edge sampling with budget m, the optimal edge probabilities to minimize the sum of variance of each ζ's dimension is given by:* $p_e = \frac{m}{\sum_{e'} \left\| \sum_\ell \mathbf{b}_{e'}^{(\ell)} \right\|} \left\| \sum_\ell \mathbf{b}_e^{(\ell)} \right\|$.

3. Random walk samper
    - R root nodes selected uniformly at random
    - Each walker goes h hops

Evaluation

Dataset:

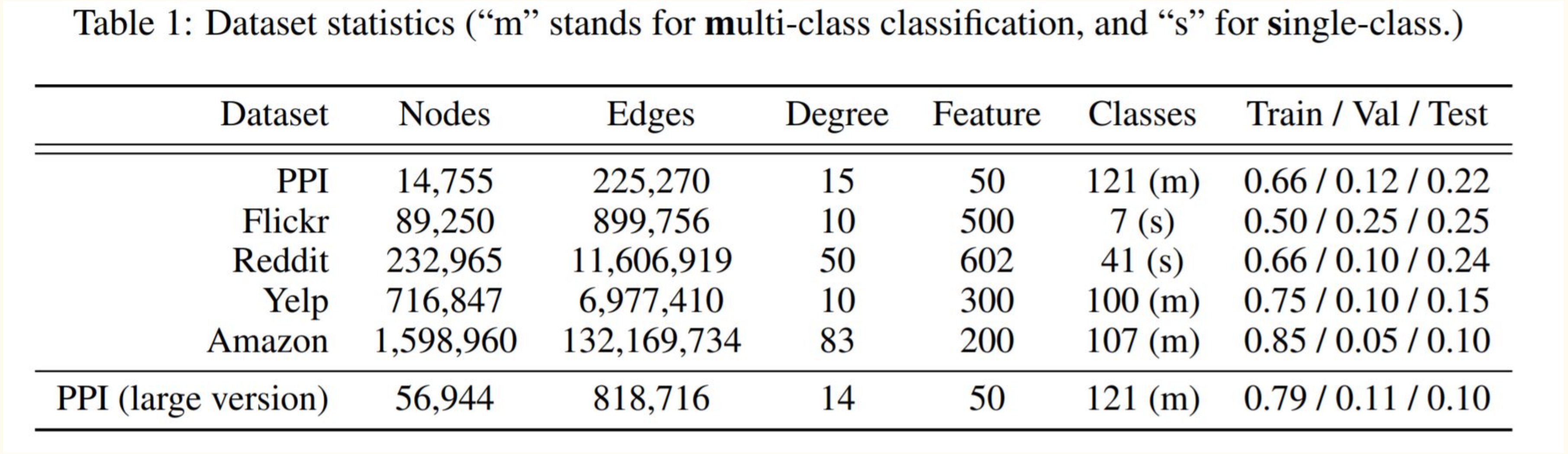


Table 1: Dataset statistics ("m" stands for **m**ulti-class classification, and "s" for **s**ingle-class.)

Dataset	Nodes	Edges	Degree	Feature	Classes	Train / Val / Test
PPI	14,755	225,270	15	50	121 (m)	0.66 / 0.12 / 0.22
Flickr	89,250	899,756	10	500	7 (s)	0.50 / 0.25 / 0.25
Reddit	232,965	11,606,919	50	602	41 (s)	0.66 / 0.10 / 0.24
Yelp	716,847	6,977,410	10	300	100 (m)	0.75 / 0.10 / 0.15
Amazon	1,598,960	132,169,734	83	200	107 (m)	0.85 / 0.05 / 0.10
PPI (large version)	56,944	818,716	14	50	121 (m)	0.79 / 0.11 / 0.10

Evaluation

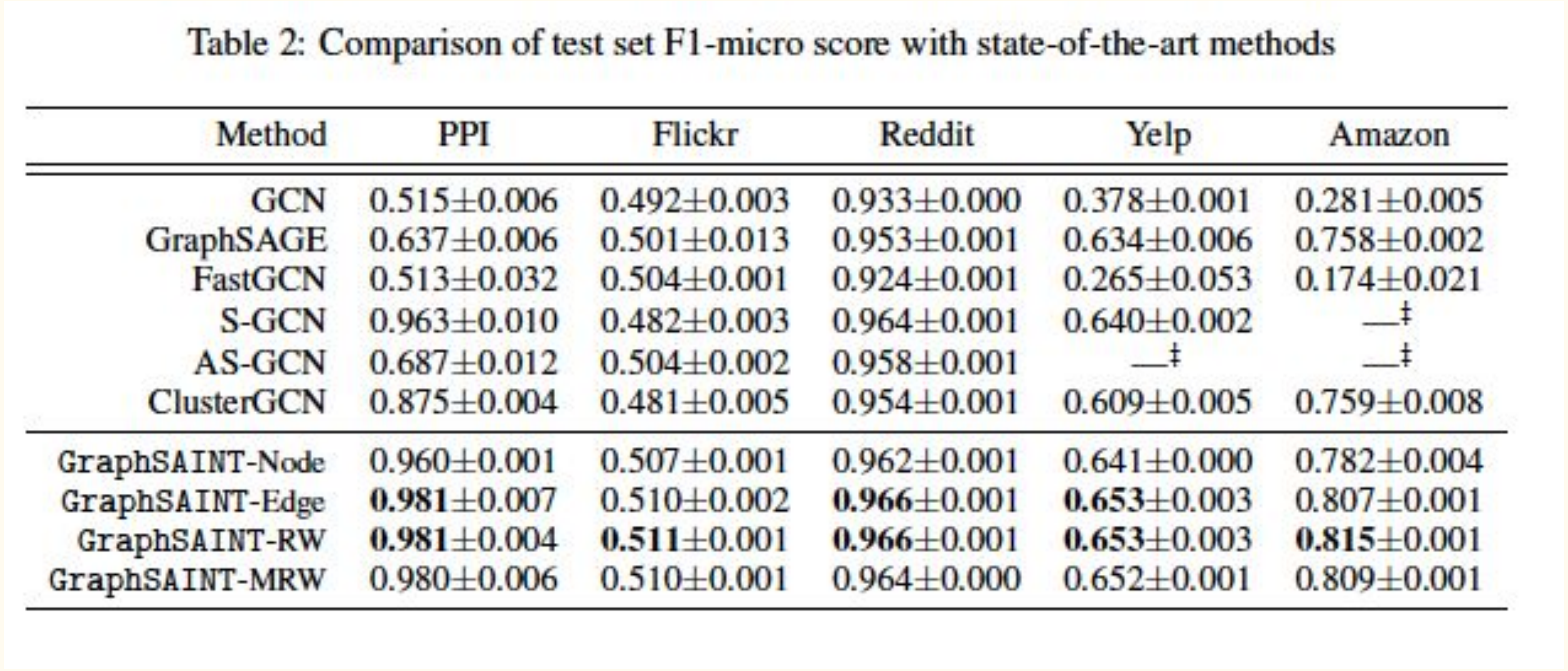


Table 2: Comparison of test set F1-micro score with state-of-the-art methods

Method	PPI	Flickr	Reddit	Yelp	Amazon
GCN	0.515±0.006	0.492±0.003	0.933±0.000	0.378±0.001	0.281±0.005
GraphSAGE	0.637±0.006	0.501±0.013	0.953±0.001	0.634±0.006	0.758±0.002
FastGCN	0.513±0.032	0.504±0.001	0.924±0.001	0.265±0.053	0.174±0.021
S-GCN	0.963±0.010	0.482±0.003	0.964±0.001	0.640±0.002	—‡
AS-GCN	0.687±0.012	0.504±0.002	0.958±0.001	—‡	—‡
ClusterGCN	0.875±0.004	0.481±0.005	0.954±0.001	0.609±0.005	0.759±0.008
`GraphSAINT-Node`	0.960±0.001	0.507±0.001	0.962±0.001	0.641±0.000	0.782±0.004
`GraphSAINT-Edge`	**0.981**±0.007	0.510±0.002	**0.966**±0.001	**0.653**±0.003	0.807±0.001
`GraphSAINT-RW`	**0.981**±0.004	**0.511**±0.001	**0.966**±0.001	**0.653**±0.003	**0.815**±0.001
`GraphSAINT-MRW`	0.980±0.006	0.510±0.001	0.964±0.000	0.652±0.001	0.809±0.001

Training Time

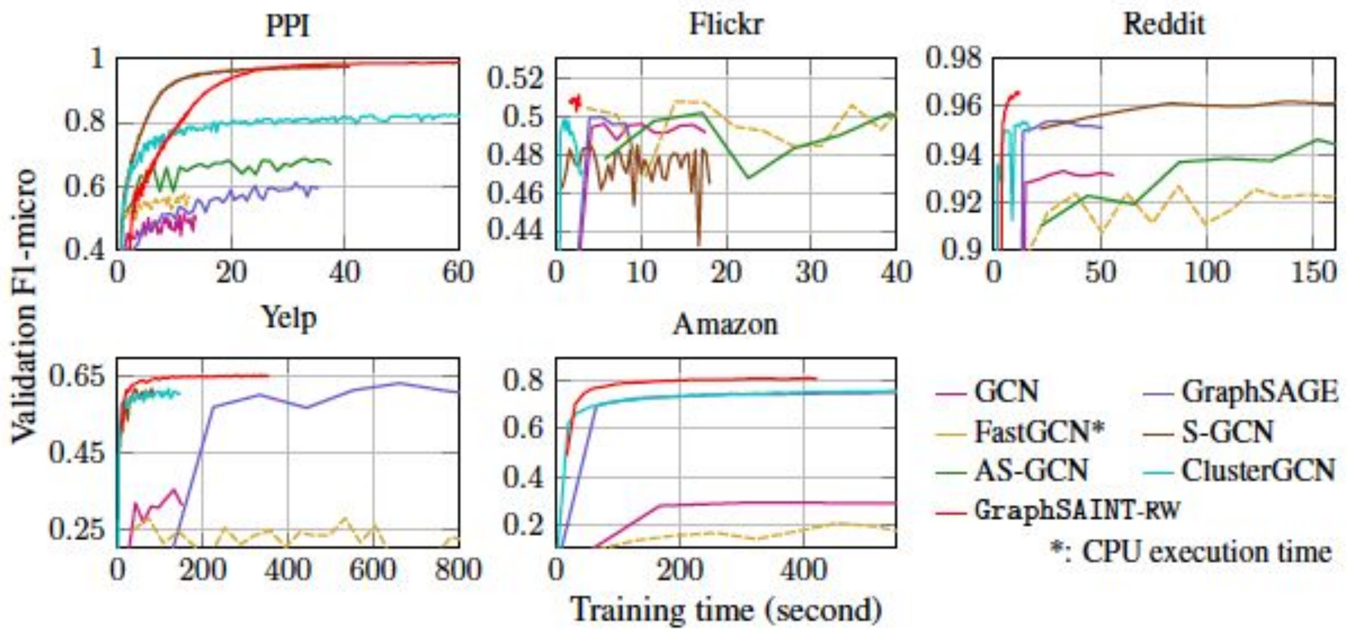


Figure 2: Convergence curves of 2-layer models on GraphSAINT and baselines