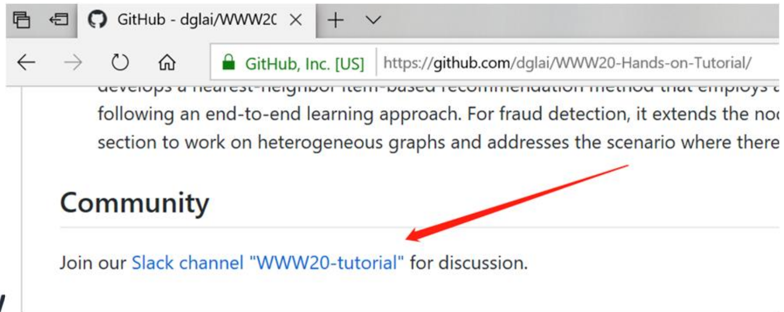
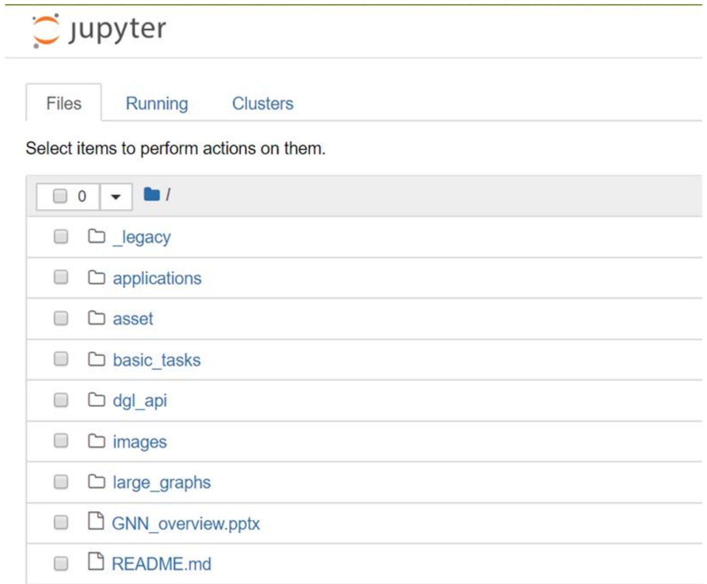
SageMaker Setup Instructions

- Send an email to dgl-www@request-nb.mxnet.io
- You will receive two emails. Click the notebook link the in the **second** one.

- Need help?
  - Join our slack channel





Deep Graph Library
an update

https:www.dgl.ai

Zheng Zhang
Director, AWS Shanghai AI Lab (zhaz@amazon.com)
zz@nyu.edu (on leave)

DGL: The history

2018

First prototype

Development started

V0.1
(NeurIPS'18)

2019

V0.2
Sampling APIs

V0.3
Fused message passing
Multi-GPU/-core

V0.3.1
NN modules
DGL-Chem

V0.4
Heterogeneous graph
DGL-KE (preview)

V0.43
TF support
DGL-KE/DGL-LifeScie
Sampling

Introducing Amazon SageMaker Support for Deep Graph Library (DGL): Build and Train Graph Neural Networks

Posted On: Dec 3, 2019

Amazon SageMaker support for the Deep Graph Library (DGL) is now available. With DGL, you can improve the prediction accuracy of recommendation, fraud detection, and drug discovery systems using Graph Neural Networks (GNNs).

Though GNNs have shown promising results in research, their use in real-world applications has been limited because of the complex infrastructure required to train large graphs and the lack of reliable domain specific models. Developing GNNs involves finding and training on very large graphs with millions of nodes, and it is time-consuming to build and maintain the computational infrastructure required to perform this training. DGL gives you the tools and infrastructure to simplify the implementation and deployment of GNNs.

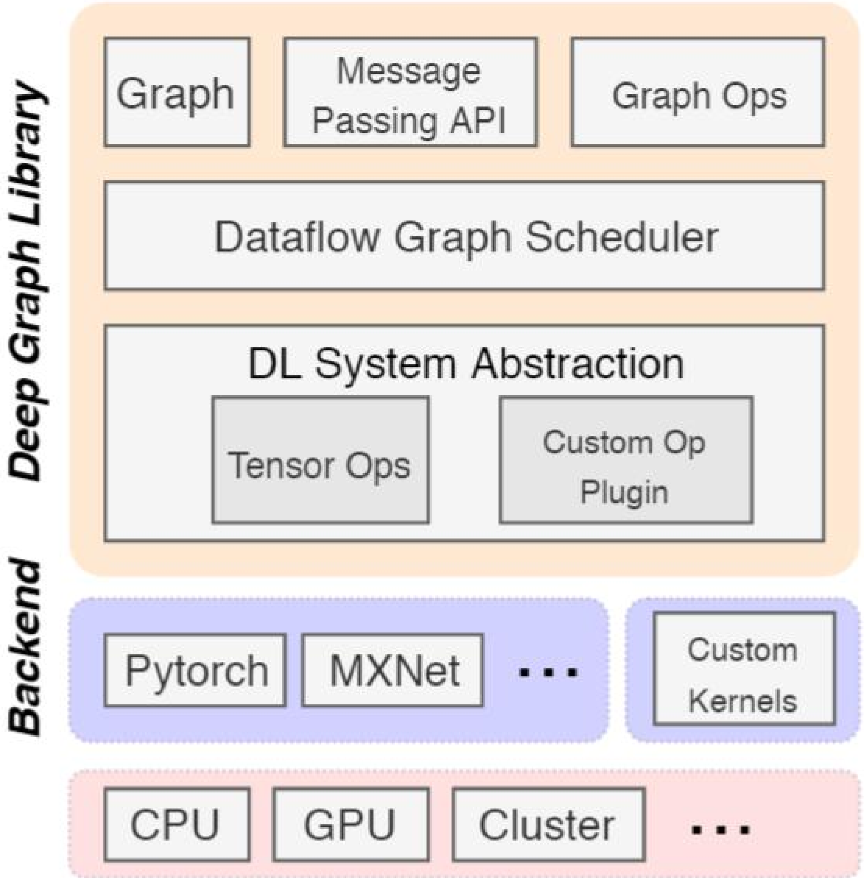
DGL support in Amazon SageMaker removes the burden of packaging software dependencies, building infrastructure, and finding validated models. As a result, you can test and implement GNNs in hours instead of weeks or months. A Deep Learning container bundles all the software dependencies and the Amazon SageMaker API automatically sets up and scales the infrastructure required to train graphs. With the bundled library of validated models, you can immediately test state-of-the-art GNN models and integrate them into applications.

To get started, please check out the DGL Get Started page, SageMaker DGL documentation and our blog.

aws

DGL meta-objectives & architecture

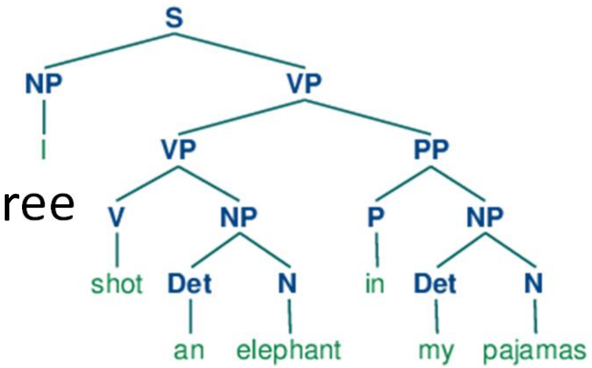
- Forward and backward compatible
  - ***Forward***: easy to develop new models
  - ***Backward***: seamless integration with existing frameworks (MXNet/Pytorch/Tensorflow)
- **Fast** and **Scalable**

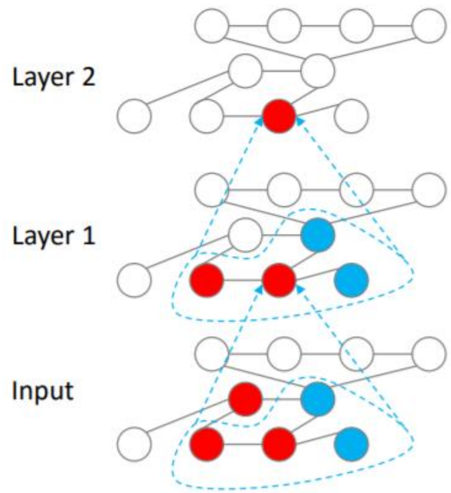


DGL: design & API

Flexible message propagation

- Full propagation ("everyone shouts to everyone near you")
- Propagation by graph traversal
  - Topological order on sentence parsing tree
  - Belief propagation order
  - Sampling
- Propagation by random walk





DGL programming interface

- Graph as the core abstraction
  - `DGLGraph`
  - `g.ndata['h']`
- Simple but versatile message passing APIs

$$\texttt{send}(\underline{\mathcal{E}}, \phi^e), \quad \texttt{recv}(\underline{\mathcal{V}}, \bigoplus, \phi^v)$$

Active set specifies which nodes/edges to trigger the computation on.

ϕ^e ϕ^v $\bigoplus$ can be user-defined functions (**UDFs**) or **built-in** symbolic functions.

Flexible message handling

Message function

$$\text{Edge-wise: } \mathbf{m}_k^{(t)} = \phi^e(\mathbf{e}_k^{(t-1)}, \mathbf{v}_{r_k}^{(t-1)}, \mathbf{v}_{s_k}^{(t-1)}),$$

$$\text{Node-wise: } \mathbf{v}_i^{(t)} = \phi^v(\mathbf{v}_i^{(t-1)}, \bigoplus_{\substack{k \\ \text{s.t. } r_k = i}} \mathbf{m}_k^{(t)}),$$

Update function

Reduce function

[Gilmer 2017, Wang 2017, Battaglia 2018]

Writing GNNs is intuitive in DGL

`update_all` is a shortcut for
`send(G.edges())` + `recv(G.nodes())`

```
# code: PyTorch + DGL
# G: DGL Graph
# H: node repr matrix (n_nodes, in_dim)
# W: weights (in_dim * 2, out_dim)
import dgl.function as fn
G.ndata['h'] = H
G.update_all(
    fn.copy_u('h', 'm'),
    fn.max('m', 'h_n'))
H_N = G.ndata['h_n']
H = torch.relu(torch.cat([H_N, H], 1) @ W)
```

$$h_v^{(t+1)} = \max_{u \in \mathcal{N}(v)} h_u^{(t)}$$

```
# code: PyTorch + DGL
# G: DGL Graph
# H: node repr matrix (n_nodes, in_dim)
# W: weights (in_dim * 2, out_dim)
import dgl.function as fn
G.ndata['h'] = H
G.update_all(
    fn.copy_u('h', 'm'),
    fn.mean('m', 'h_n'))
H_N = G.ndata['h_n']
H = torch.relu(torch.cat([H_N, H], 1) @ W)
```

$$h_v^{(t+1)} = \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} h_u^{(t)}$$

Writing GNNs is intuitive in DGL (GAT)

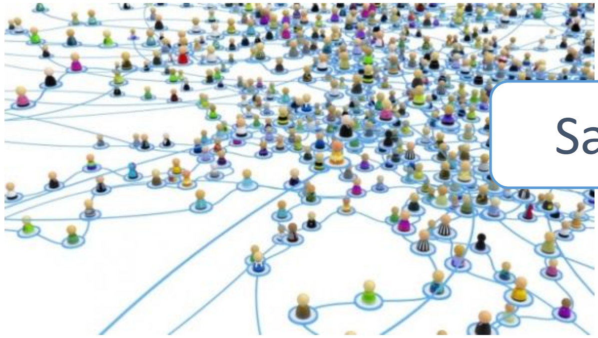
```python
def msg_func(edges):
    h_src = edges.src['h']
    h_dst = edges.dst['h']
    alpha_hat = MLP(
        torch.cat([h_dst, h_src], 1))
    return {'m': h_src, 'alpha_hat': alpha}

def reduce_func(nodes):
    # Incoming messages are batched along
    # 2nd axis.
    m = nodes.mailbox['m']
    alpha_hat = nodes.mailbox['alpha_hat']
    alpha = torch.softmax(alpha_hat, 1)
    return {'h_n':
        (m * alpha[:, None]).sum(1)}
```

```python
# code: PyTorch + DGL
# G: DGL Graph
# H: node repr matrix (n_nodes, in_dim)
# W: weights (in_dim * 2, out_dim)

import dgl.function as fn
G.ndata['h'] = H
G.update_all(msg_func, reduce_func)
H_N = G.ndata['h_n']
H = torch.relu(torch.cat([H_N, H], 1) @ W)
```

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

aws

Different scenarios require different supports

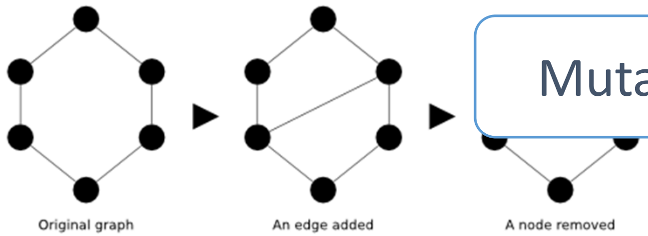


Sampling

Single giant graph

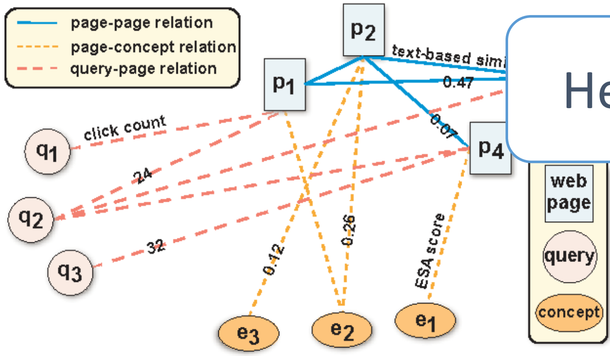
Batching graphs

Many moderate-sized graphs



Mutation

Dynamic graph



Heterogeneous

Summary of DGL Programming Interface

		DGL
Message Passing	arbitrary ϕ^e	✓
	arbitrary ϕ^v	✓
	arbitrary $\bigoplus$	✓
Propa-gation Order	full	✓
	partial	✓
	random walk	✓
	sampling	✓
Graph Type	many & small	✓
	single & giant	✓
	dynamic	✓
	heterogeneous	✓
System	multi-platform	✓

User-defined functions

Send, Recv

Active set

Batching APIs

Mutation APIs

Sampling APIs

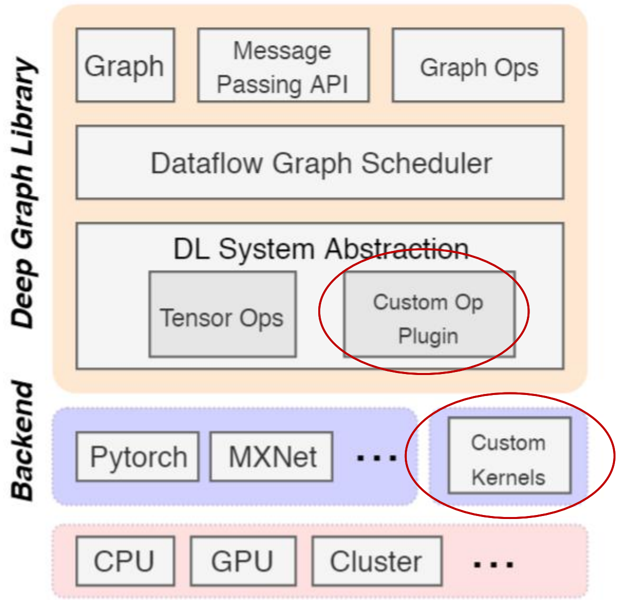
Support multi-type

Performance evaluation

Initially paid a price for multi-backend (woops!)

Table 4: Training runtime comparison.

Dataset	Method	DGL DB	DGL GS	PyG
Cora	GCN	4.19s	0.32s	**0.25s**
	GAT	6.31s	5.36s	**0.80s**
CiteSeer	GCN	3.78s	0.34s	**0.30s**
	GAT	5.61s	4.91s	**0.88s**
PubMed	GCN	12.91s	0.36s	**0.32s**
	GAT	18.69s	13.76s	**2.42s**
MUTAG	RGCN	18.81s	2.40s	**2.14s**



Fast Graph Representation Learning with PyTorch Geometric

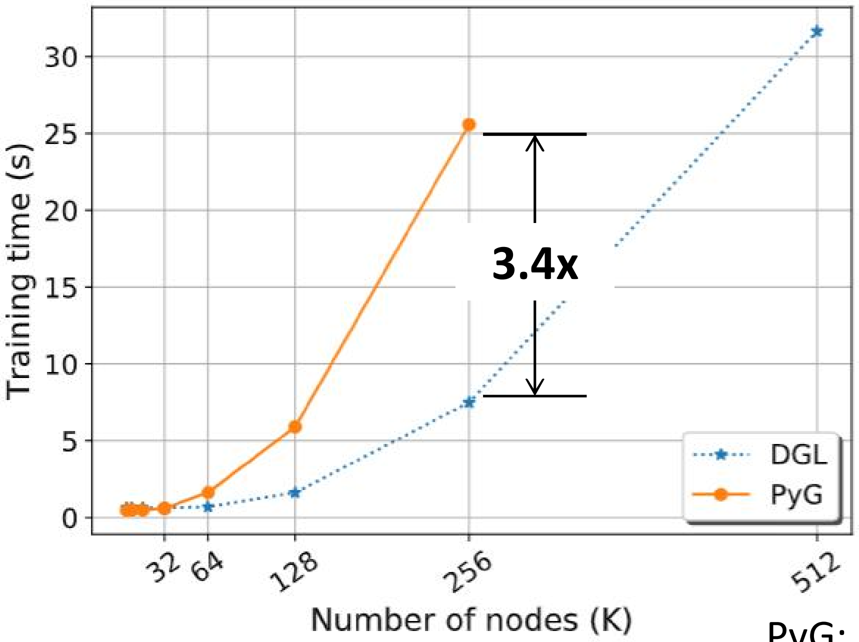
Evaluation: efficiency and memory consumption

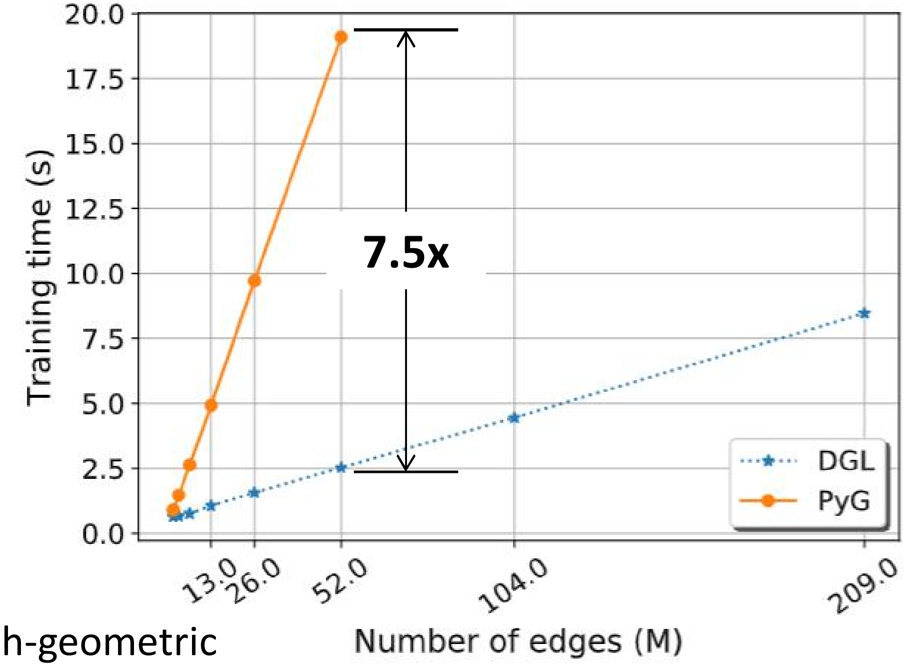
Dataset	Model	Accuracy	Time		Memory	
			PyG	DGL	PyG	DGL
Cora	GCN	81.31 ± 0.88	**0.478**	0.666	1.1	1.1
	GAT	83.98 ± 0.52	1.608	**1.399**	1.2	**1.1**
CiteSeer	GCN	70.98 ± 0.68	**0.490**	0.674	1.1	1.1
	GAT	69.96 ± 0.53	1.606	**1.399**	1.3	**1.2**
PubMed	GCN	79.00 ± 0.41	**0.491**	0.690	1.1	1.1
	GAT	77.65 ± 0.32	1.946	**1.393**	1.6	**1.2**
Reddit	GCN	93.46 ± 0.06	*OOM*	**28.6**	*OOM*	**11.7**
Reddit-S	GCN	N/A	29.12	**9.44**	15.7	**3.6**

Table 2: Training time (in seconds) for 200 epochs and memory consumption (GB).

Testbed: one V100 GPU (16GB)

Scalability: single machine, single GPU



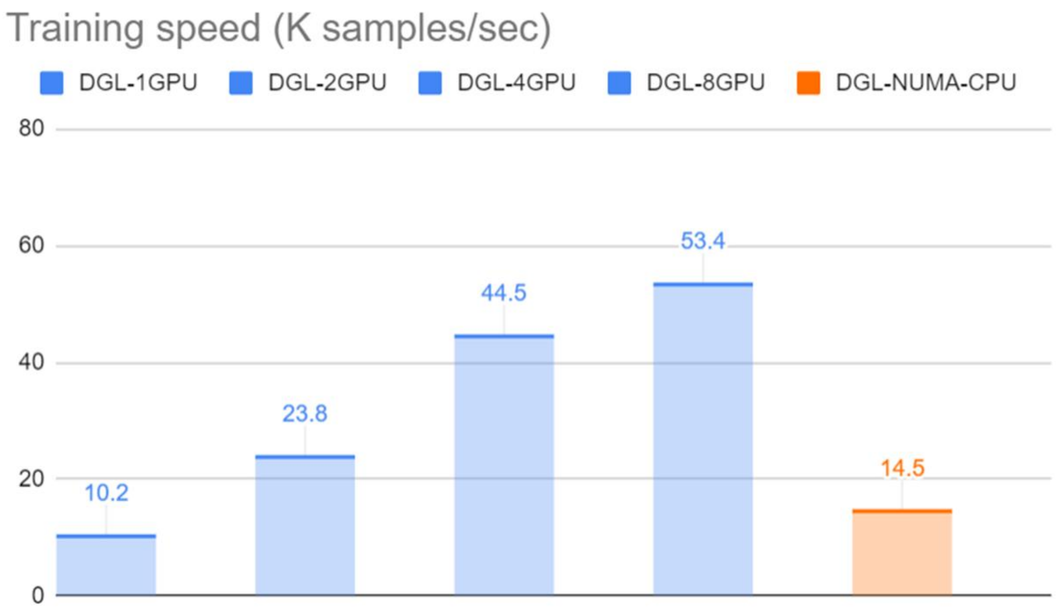


PyG: pytorch-geometric

Scalability with graph size

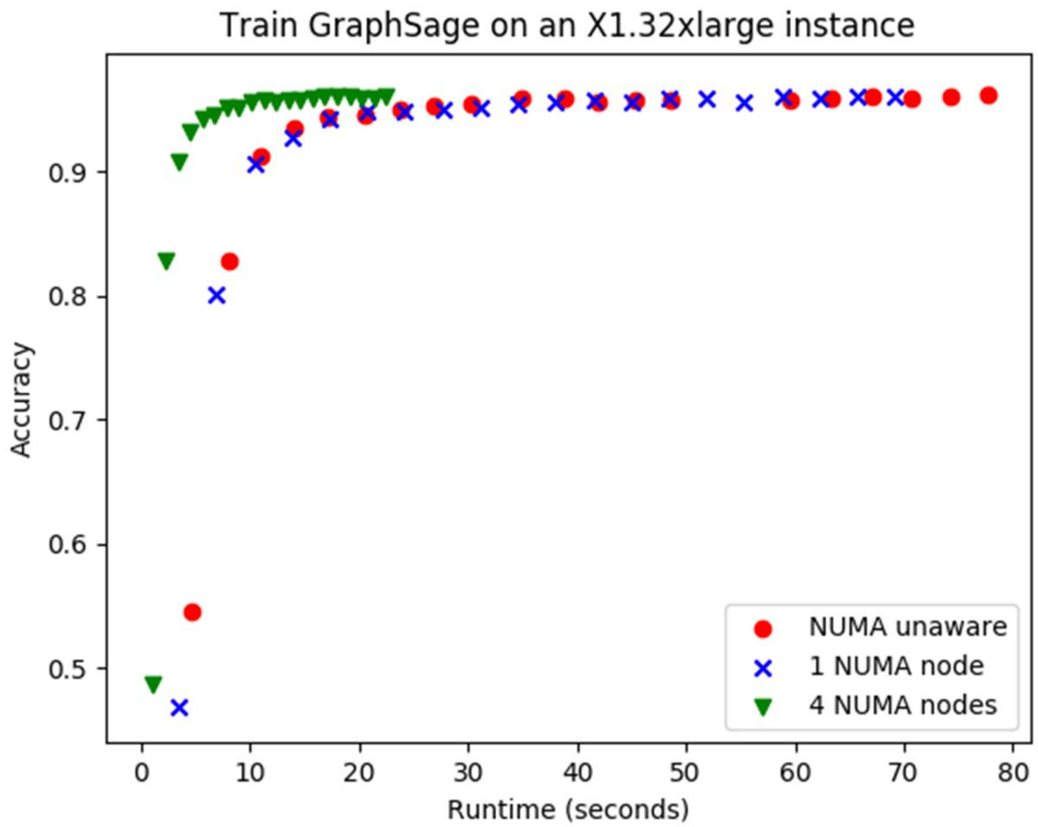
Scalability with graph density

Scalability: single machine, multi-GPU



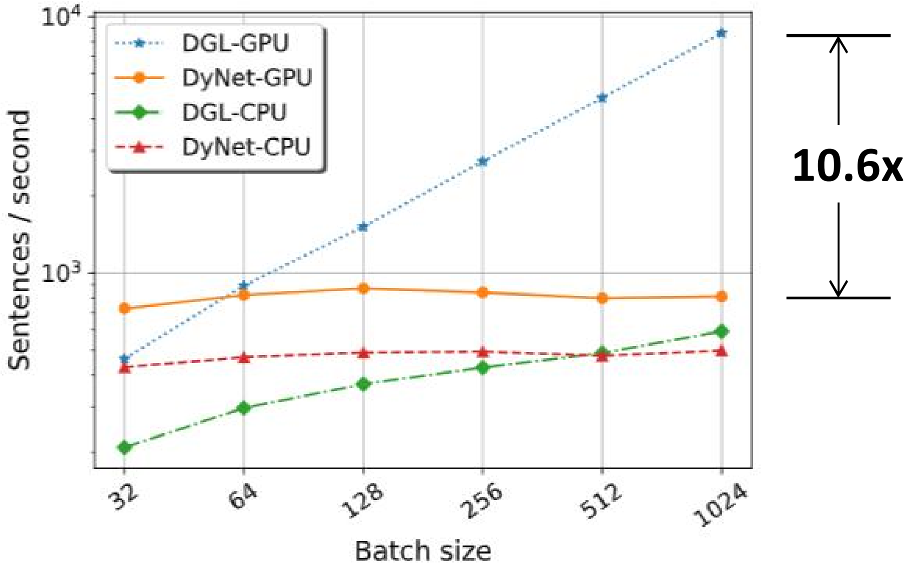
p3.16xlarge, 8 V100 GPUs, 64 vCPU
Data set: Reddit (232K nodes, 114M edges)
GraphSage neighbor sampling

Scalability: single machine, NUMA



X1, 2TB, 128 vCPU
Data set: Reddit (232K nodes, 114M edges)
Controlled-variate sampling

Evaluation: auto-batching



Compare with DyNet for training TreeLSTM

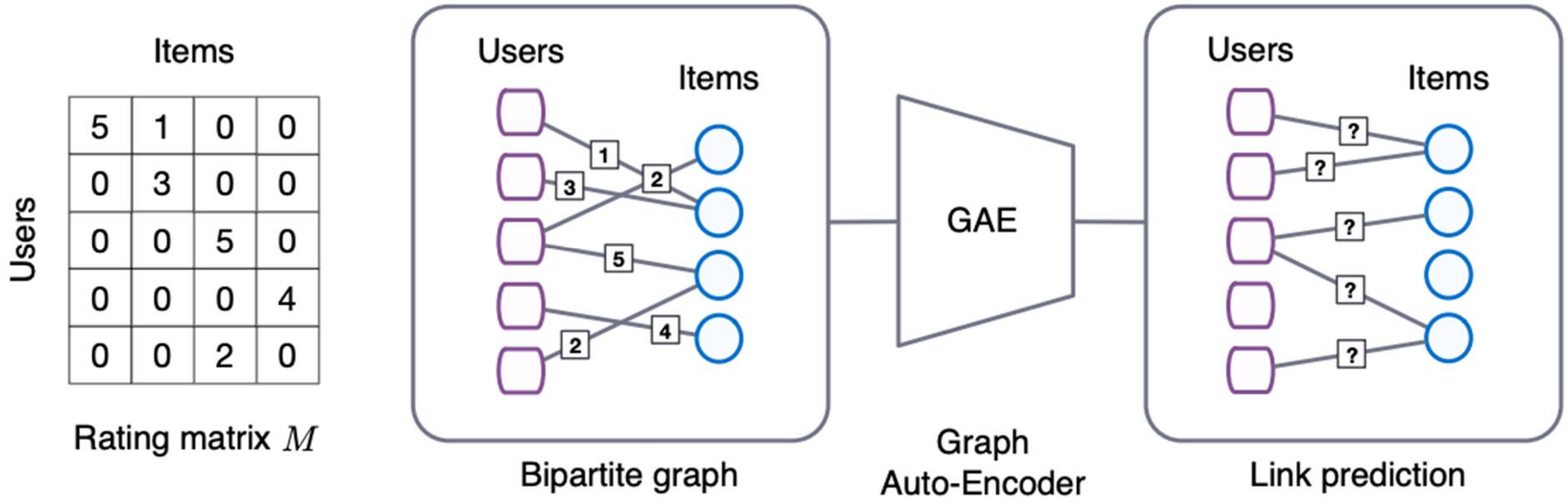
Testbed: one V100 GPU (16GB)

What’s new

Release 0.4 and 0.43

Heterogenous graph

- Example: recommendation system, GCMC



Graph Convolutional Matrix Completion

Supporting Heterogeneous Graph

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_r(i)} W_r^{(l)} h_j^{(l)}\right)$$

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import dgl.function as fn

class HeteroRGCNLayer(nn.Module):
    def __init__(self, in_size, out_size, etypes):
        super(HeteroRGCNLayer, self).__init__()
        # define parameter W_r for each relation
        self.weight = nn.ModuleDict({
                name : nn.Linear(in_size, out_size) for name in etypes
            })

    def forward(self, G, feat_dict):
        # G is a heterogeneous graph
        # feat_dict is a dictionary of features of each node type
        funcs = {}
        for srctype, etype, dsttype in G.canonical_etypes:
            # Compute W_r * h
            Wh = self.weight[etype](feat_dict[srctype])
            # Save it to graph
            G.nodes[srctype].data['Wh_%s' % etype] = Wh
            # Per-type message passing: (message_func, reduce_func)
            # All reducers write to the same field 'h', which is a hint for type-wise reducer.
            funcs[etype] = (fn.copy_u('Wh_%s' % etype, 'm'), fn.mean('m', 'h'))
        # Trigger message passing on heterograph using multi_update_all
        # Argument#1: per-type message passing functions.
        # Argument#2: type-wise reducer, could be: "sum", "max", "min", "mean", "stack"
        G.multi_update_all(funcs, 'sum')
        # Return the updated features of each node type.
        return {ntype : G.nodes[ntype].data['h'] for ntype in G.ntypes}
```

Example: graph convolutional matrix completion

Dataset	RMSE (DGL)	RMSE (Official)	Speed (DGL)	Speed (Official)	Speedup
MovieLens-100K	**0.9077**	0.910	**0.0246** s/epoch	0.1008 s/epoch	5x
MovieLens-1M	0.8377	**0.832**	**0.0695** s/epoch	1.538 s/epoch	22x
MovieLens-10M	0.7875	**0.777***	**0.6480** s/epoch	Long*	

*Official training on MovieLens-10M has to be in mini-batch, which lasts for over 24+ hours

Example: drug discovery

Training time per epoch (1 V100 GPU)

	DGL	Official	Speedup
ACNN	0.36	1.58	4.4x

Atomic Convolutional Neural Network (ACNN)

Digress: even more examples of drug discovery

Training time per epoch (1 V100 GPU)

	DGL	Official	Speedup
ACNN	0.36	1.58	4.4x
JTNN	743	1826	2.5x
GCN (Tox21)	1.9	8.4	4.4x
AttentiveFP*	1.2	6.0	5x

*Pushing the Boundaries of Molecular Representation for Drug Discovery with the Graph Attention Mechanism (GAT, deep, GRU aggregation)

DGL 0.43 Release

Minor release loaded with features:

- TF backend support
- 2 DGL domain packages
  - DGL-KGE
  - DGL-LifeSci (5 new models)
- Sampling API

0.43 – (1) TensorFlow backend support

- 15 common GNN modules (including heterography support)
- On average x3.5 faster than GraphNet, x1.9 than tf-geometric

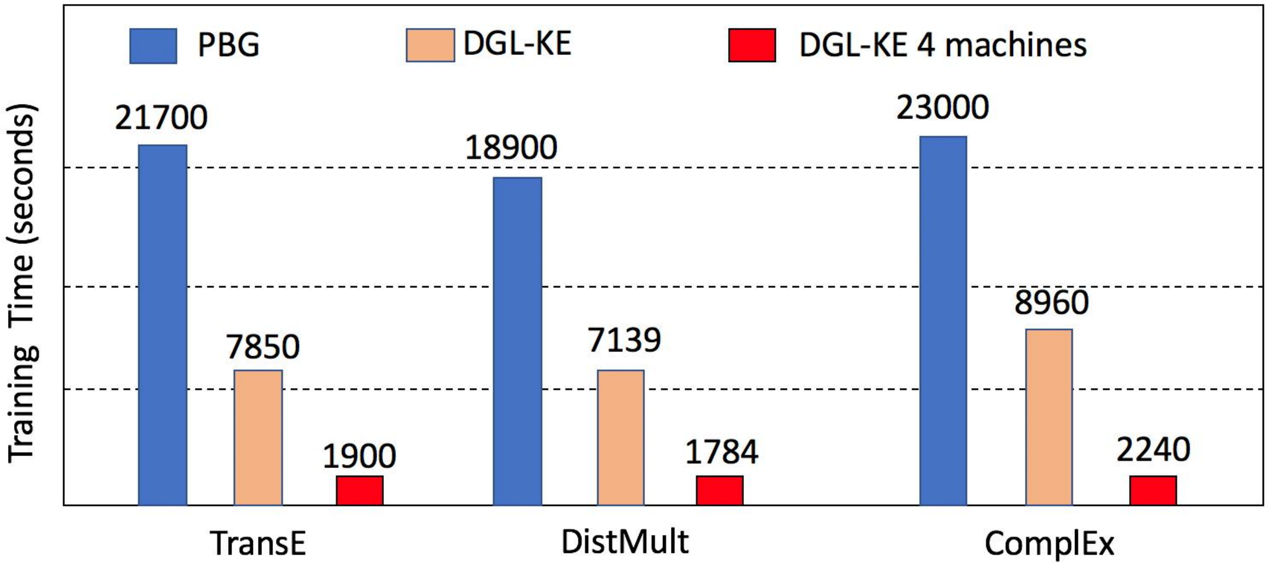
Dateset	Model	DGL	GraphNet	tf_geometric
Cora	GCN	**0.0148**	0.0152	0.0192
Reddit	GCN	**0.1095**	OOM	OOM
PubMed	GCN	**0.0156**	0.0553	0.0185
PPI	GCN	**0.09**	0.16	0.21
Cora	GAT	**0.0442**	n/a	0.058
PPI	GAT	**0.398**	n/a	0.752

aws

0.43 – (2) DGL-KE:

Light-speed for knowledge graph embedding learning, at scale

- 2~5 speedup over PyTorch Big Graph/GraphVite

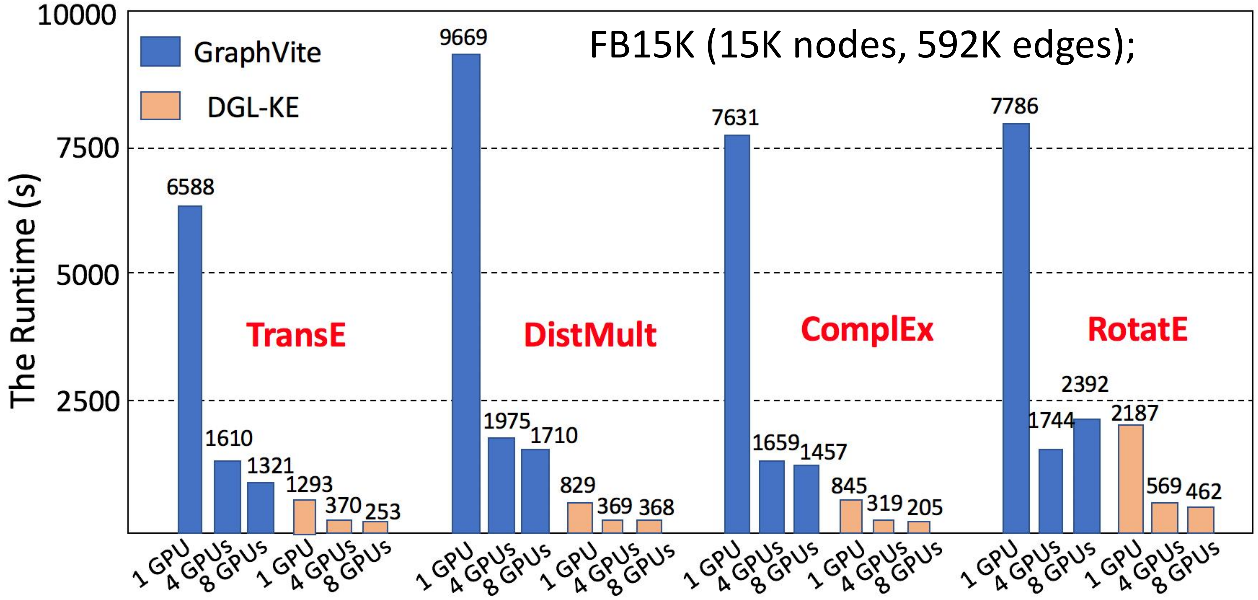


Full FreeBase
- **86M** nodes
- **348M** edges
- **30min** on 4 48core machines

0.43 – (2) DGL-KE: runs on multi-GPU, too

Light-speed for knwledge graph embedding learning, with **scale**

- 2~5 speedup over PyTorch Big Graph/GraphVite



Full FreeBase

- **86M** nodes
- **348M** edges
- **100min** on 8-GPU

aws

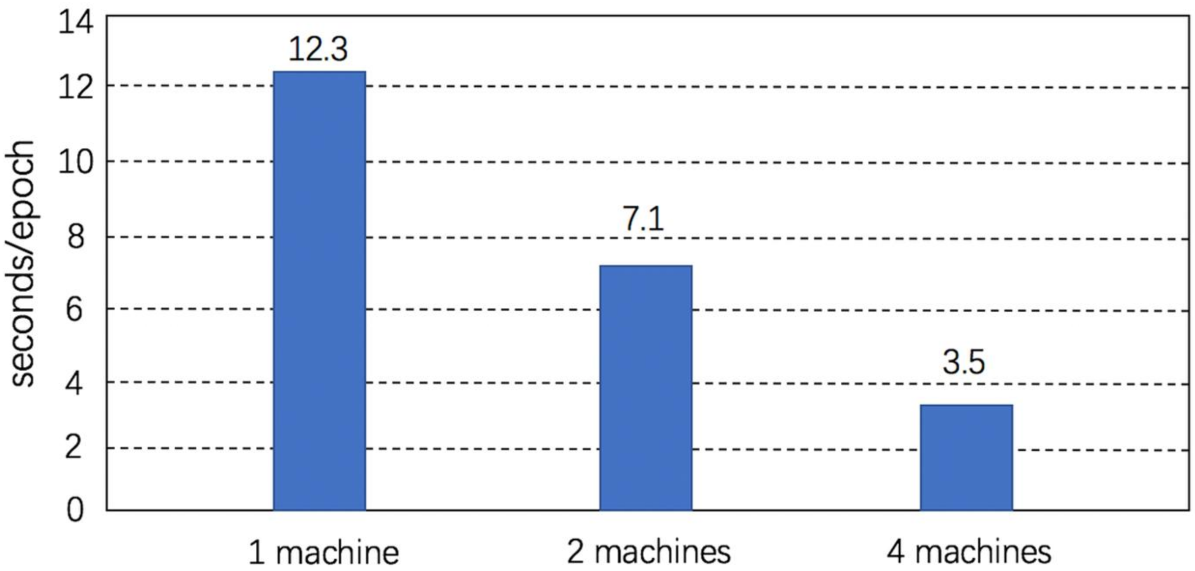
0.43 – (3) Sampling (for giant graph)

```
class NeighborSampler(object):
    def __init__(self, g, fanouts):
        self.g = g  # The full graph structure
        self.fanouts = fanouts  # fan-out of each layer

    def sample_blocks(self, seeds):
        # `seeds` are the set of nodes to build one sample from.
        blocks = []
        for fanout in self.fanouts:
            # For each seed node, sample ``fanout`` neighbors.
            frontier = dgl.sampling.sample_neighbors(g, seeds, fanout, replace=True)
            # Then we compact the frontier into a bipartite graph for message passing.
            block = dgl.to_block(frontier, seeds)
            # Obtain the seed nodes for next layer.
            seeds = block.srcdata[dgl.NID]

            blocks.insert(0, block)
        return blocks
```

Neighbor sampling, +control variate, random walk …

- Works with many GNNs (PinSAGE, GraphSAGE, GCMC, …)
- Support customization in Python.
- Support heterogeneous graphs.
- Use existing NN modules without code change.
- Multi-processing & multi-threading for maximum speed.

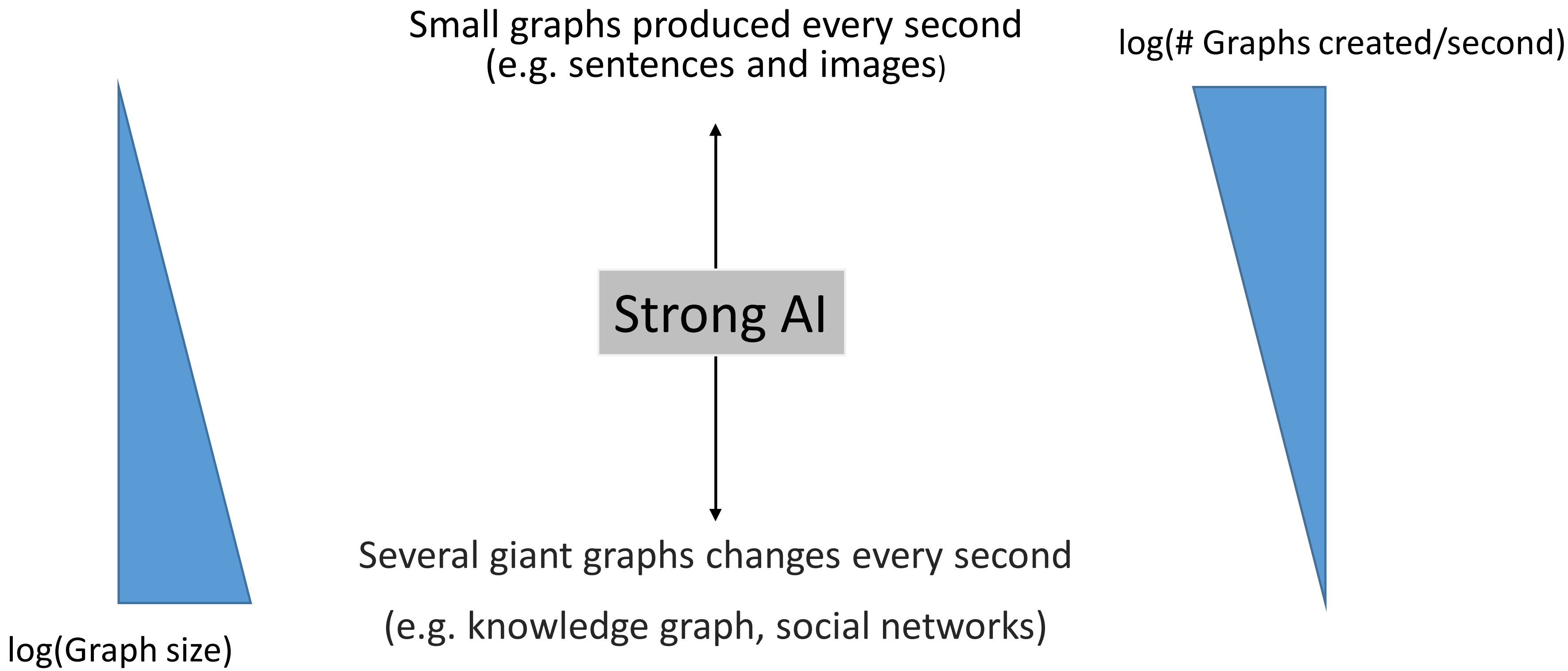
Distributed training: GCN (preliminary results)



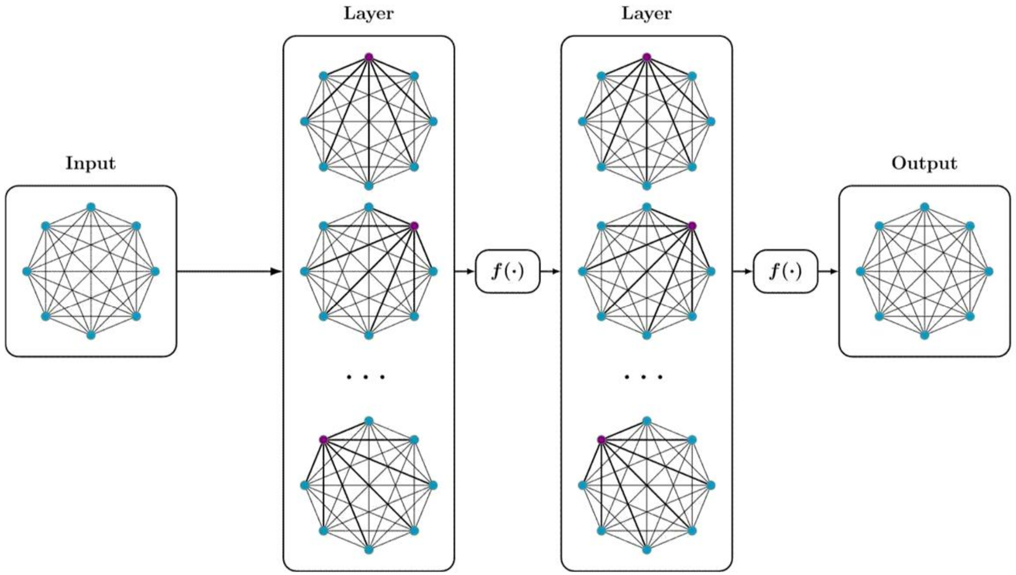
Distributed training of GCN on Reddit dataset.

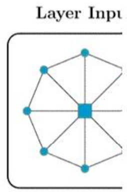
Neighbor sampling
Data set: Reddit (232K nodes, 114M edges)
Testbed: c5n.18x, 100Gb/s network, 72vCPU

Seeing the world from the lens of graphs



Transfomer is GAT (over a complete graph)





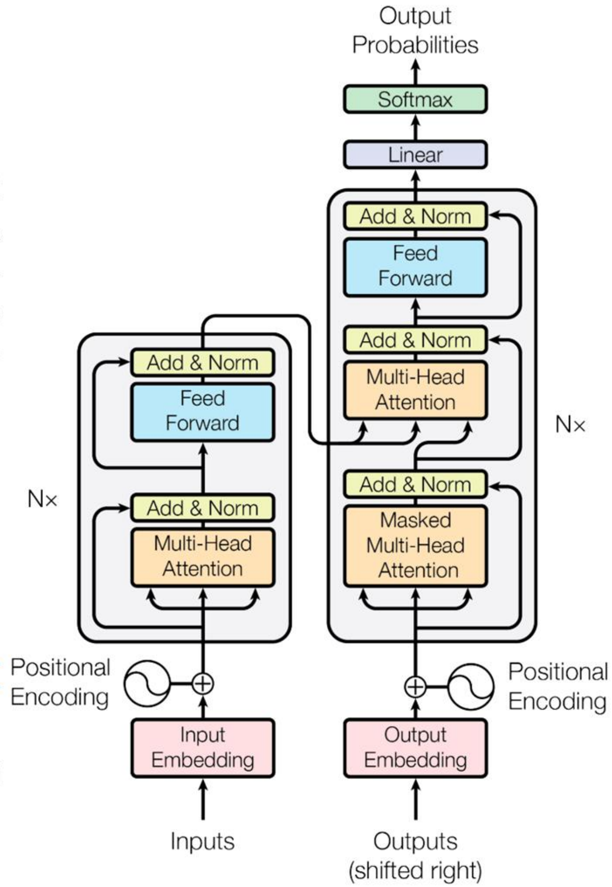
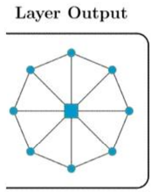


Figure 1: The Transformer - model architecture.



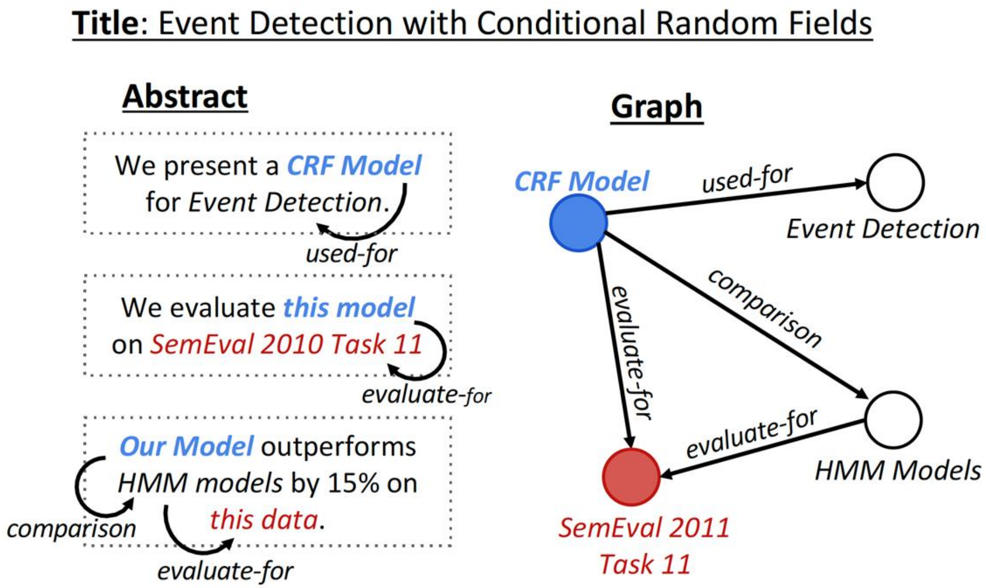
Transformer:
- $O(n^2)$: data hungry

Star-Transfor
- $O(n)$: mu
- Leverages dependency

SegTree-Transformer (in ICLR'19 RLGM)
- $O(n\log n)$: less data hungry
- A good compromise in between

Graph in Natural Language Processing

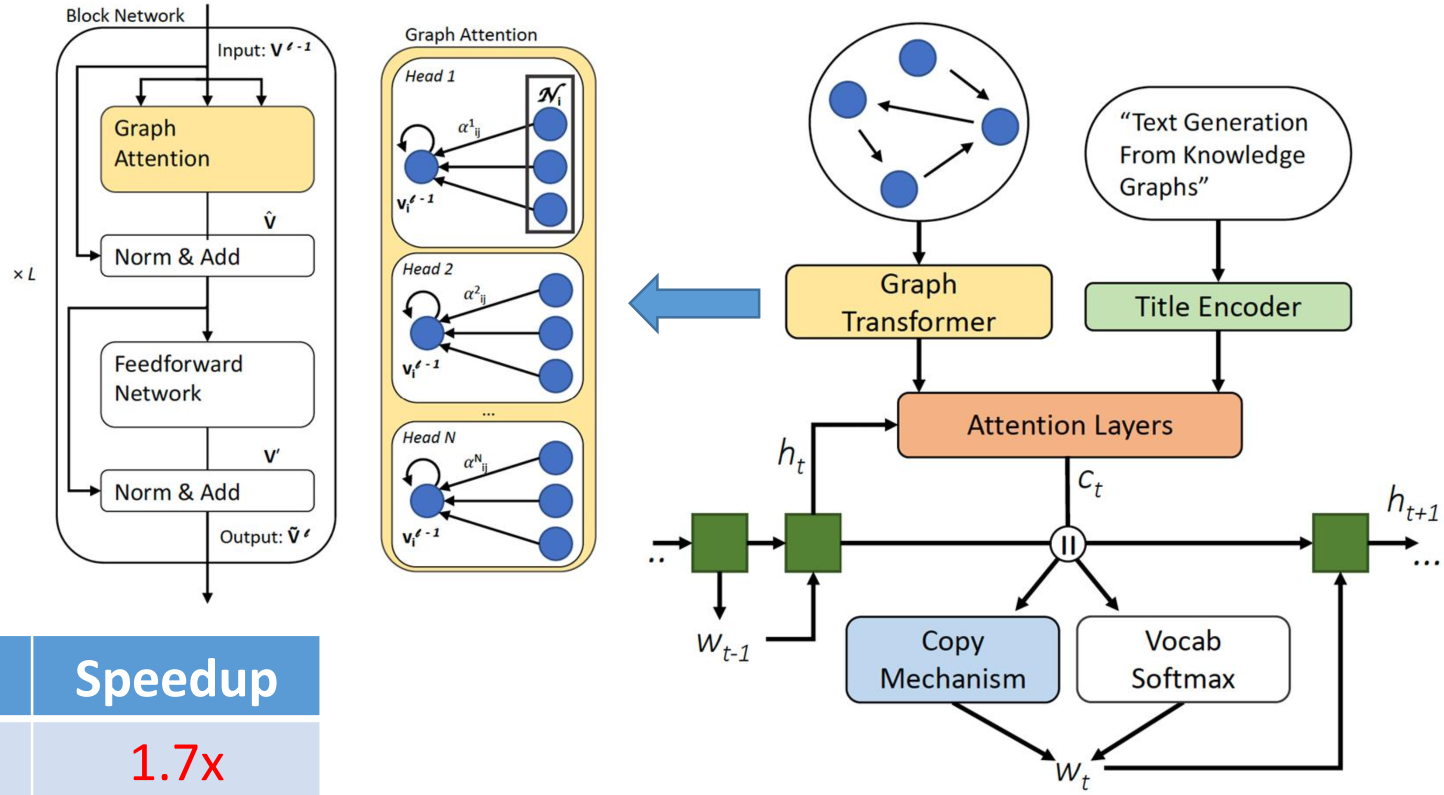
- Knowledge Graph to Sentence
  - Input: a title and knowledge graph
  - Output: abstract text



Text Generation from Knowledge Graphs with Graph Transformers

Graph in Natural Language Processing

- GraphWriter
  - Graph Attention
  - Copy or vocab

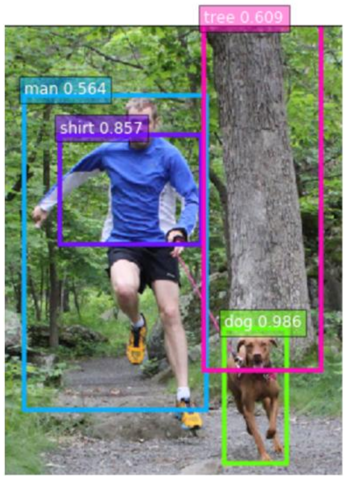


DGL	Official	Speedup
1192 (s)	1970 (s)	1.7x

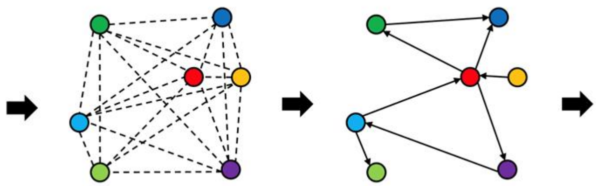
Text Generation from Knowledge Graphs with Graph Transformers

Scene graph extraction (experimental)

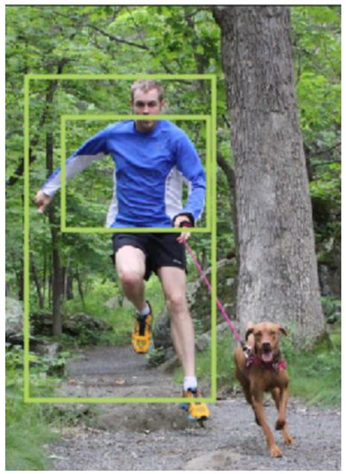
- [Graph R-CNN for Scene Graph Generation]()
  - Per epoch time 45min (vs. 2.5hr)



Object Detection



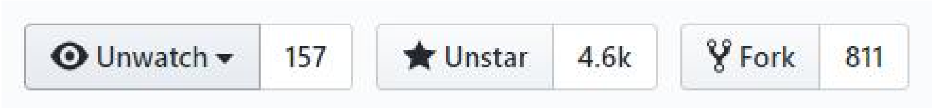
GCN

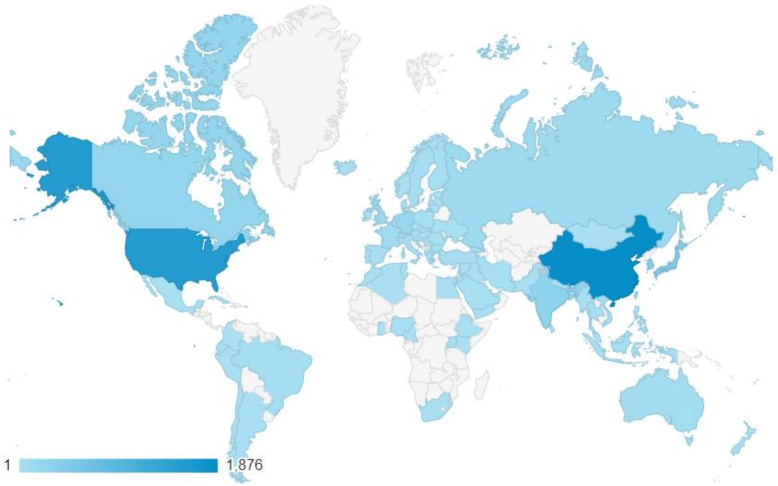


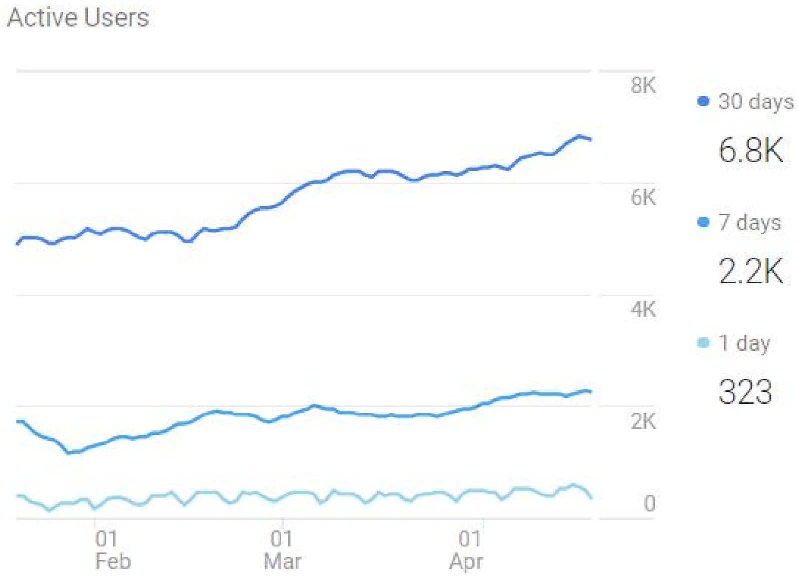
Link exists with prob 68.99%

Link type:
<man> <have> <shirt>, 43.29%
<man> <wear> <shirt>, 39.64%

Open source, the source of innovation







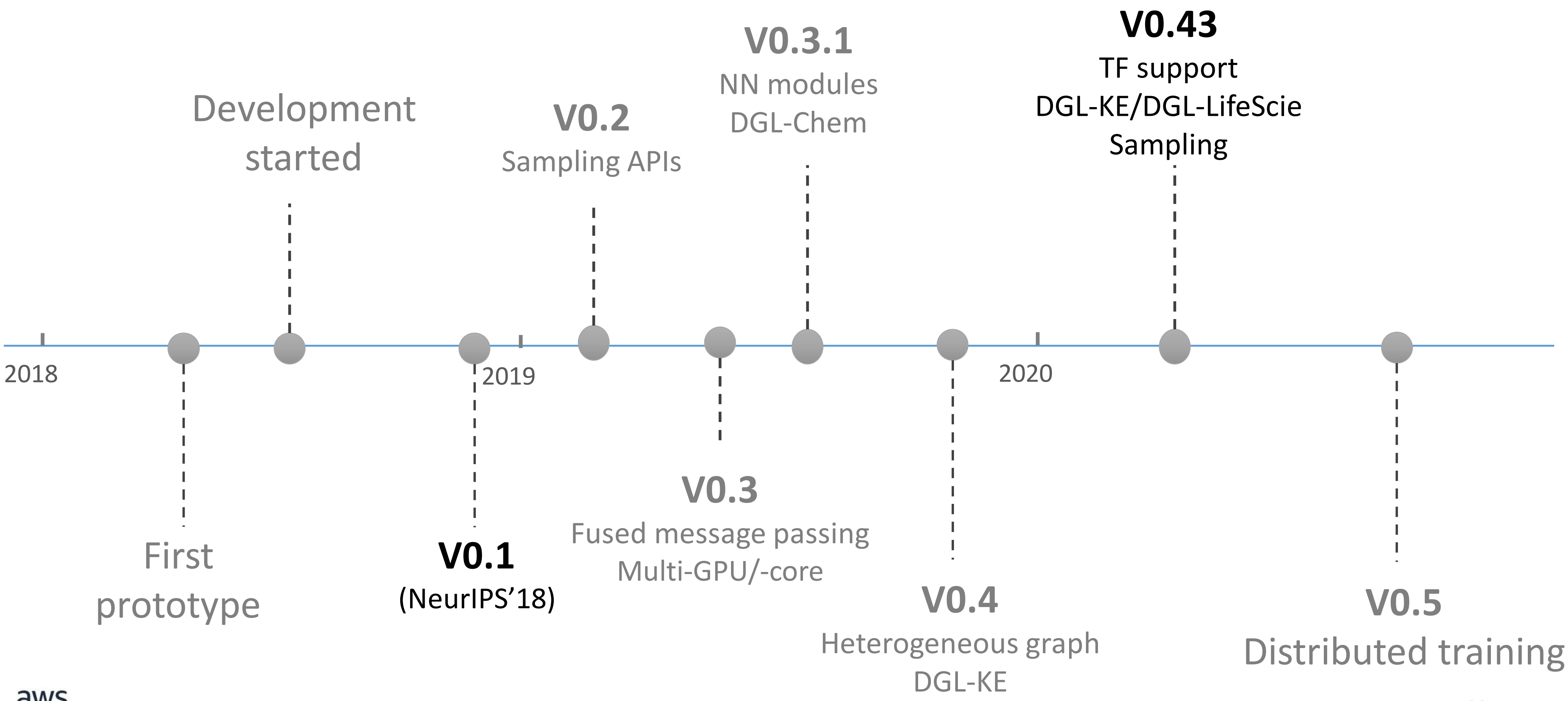
~~Neurips'19~~
WWW'20

~~3481~~ 4555 github stars
~~312k~~ 1770k downloads for all versions on Pip
~~8.8K~~ 13K downloads for all version on Conda
~~1.8K~~ 6.4K anaconda downloads of 0.4+

32 model examples, 28 NN modules (including ~~14~~ 15 GNN convolution modules)
~~6~~ 10 pretrained models for chemistry
GCN, generative, KG, RecSys…
~~45~~ 53 contributors, 10 core developers

aws

DGL: next step(s)



Welcome contributions! Please cite:

DEEP GRAPH LIBRARY: TOWARDS EFFICIENT AND SCALABLE DEEP LEARNING ON GRAPHS

Minjie Wang [1 3], Lingfan Yu [1], Da Zheng [3], Quan Gan [4], Yu Gai [2], Zihao Ye [4], Mufei Li [4], Jinjing Zhou [4], Qi Huang [2], Chao Ma [4], Ziyue Huang [5], Qipeng Guo [6], Hao Zhang [7], Haibin Lin [3], Junbo Zhao [1], Jinyang Li [1], Alexander Smola [3], Zheng Zhang [4 2]
[1] New York University, [2] NYU Shanghai, [3] Amazon Web Services, [4] AWS Shanghai AI Lab
[5] Hong Kong University of Science and Technology, [6] Fudan University,
[7] Chongqing University of Posts and Telecommunications

George Karypis (Univ Minnesota/AWS)

We are hiring!

Q&A