Deep Graph Library

Overview, Updates, and Future Directions

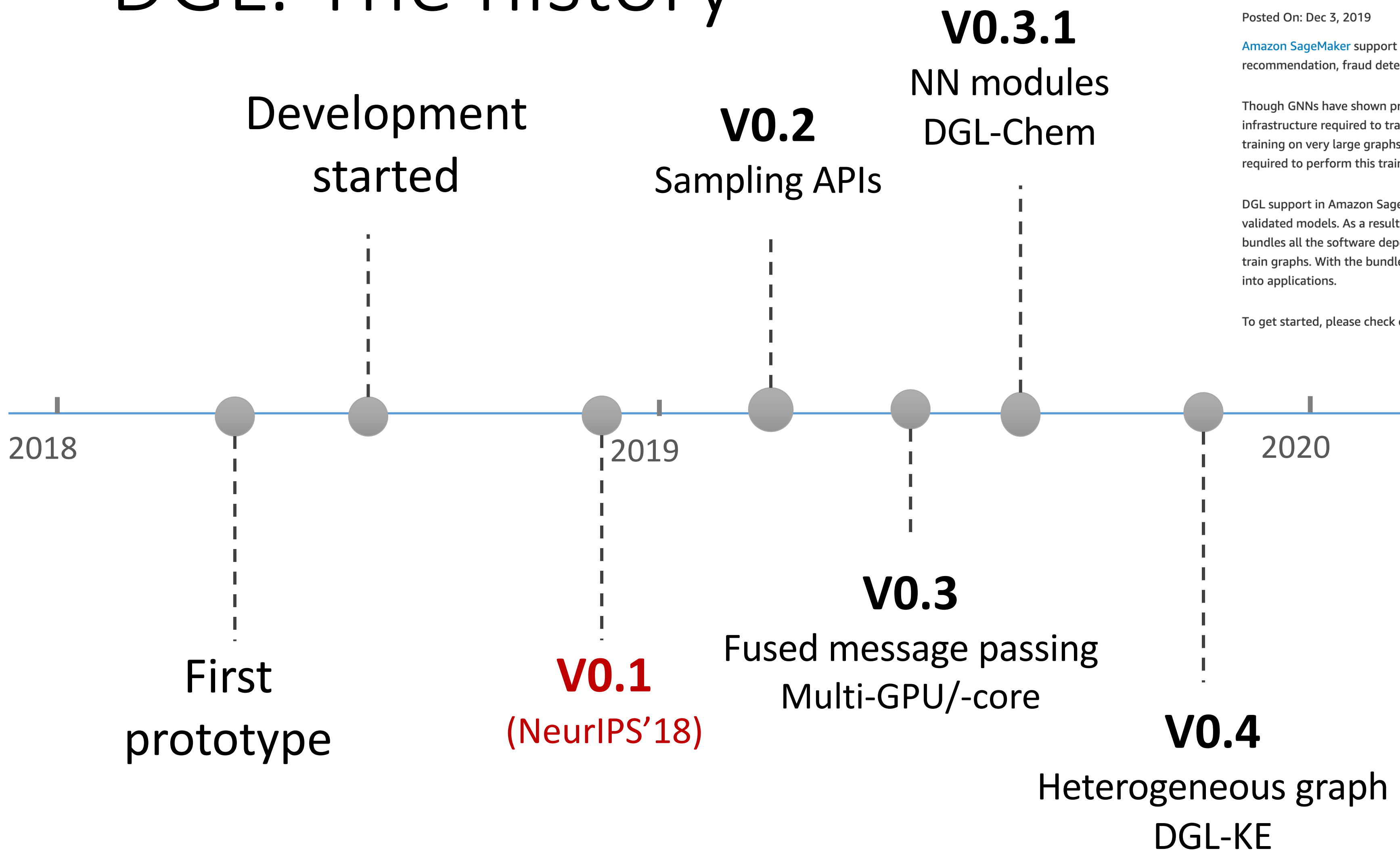
www.dgl.ai

George Karypis

University of Minnesota (karypis@umn.edu)

AWS Deep Learning Science (gkarypis@amazon.com) (on leave)

DGL: The history



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.

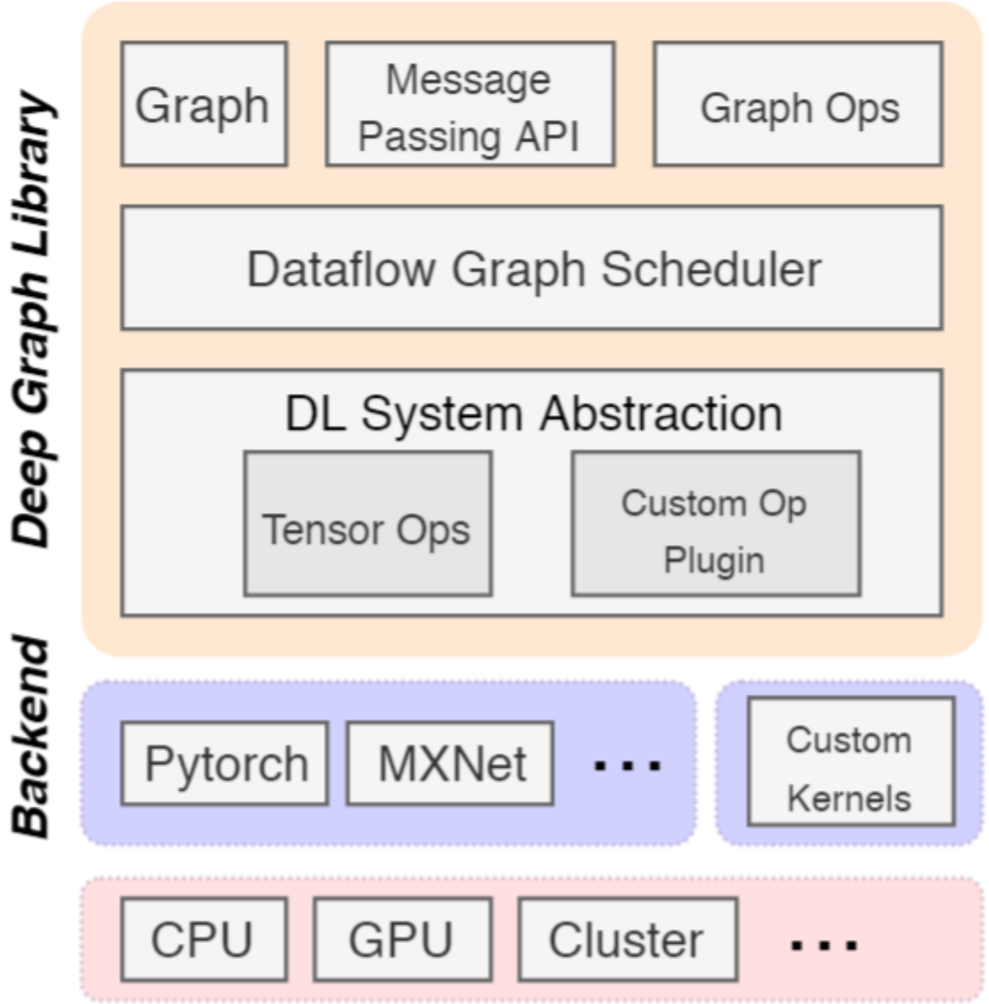
To get started, please check out the DGL Get Started page, SageMaker DGL documentation and our blog.

DGL: Design & API

DGL meta-objective & architecture

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

- **Fast** and **Scalable**



Flexible message handling

Message function

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

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

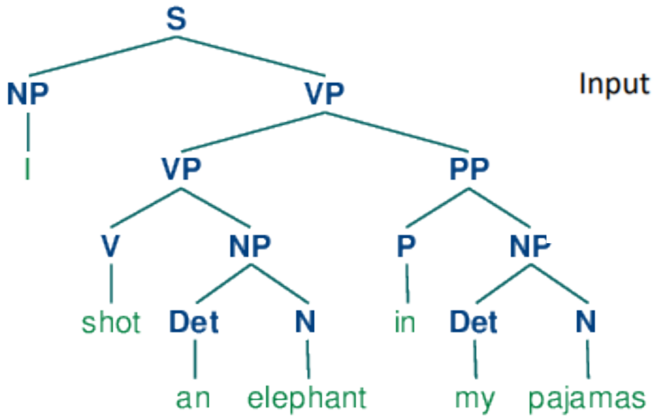
Update function

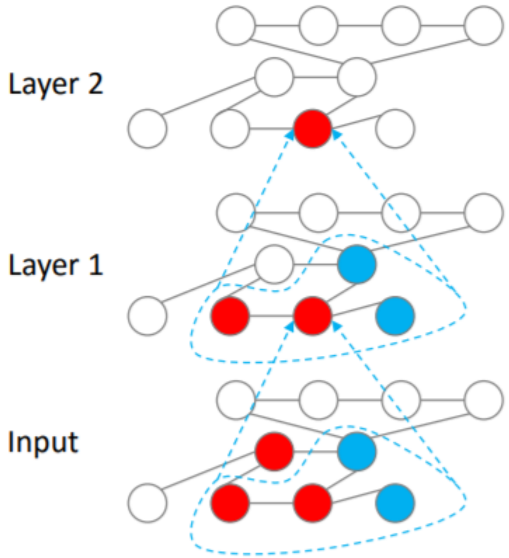
Reduce function

[Gilmer 2017, Wang 2017, Battaglia 2018]

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

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

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.

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)
```

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

aws

Writing GNNs is intuitive in DGL (GAT)

```
# 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)
```

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

```
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)}
```

aws

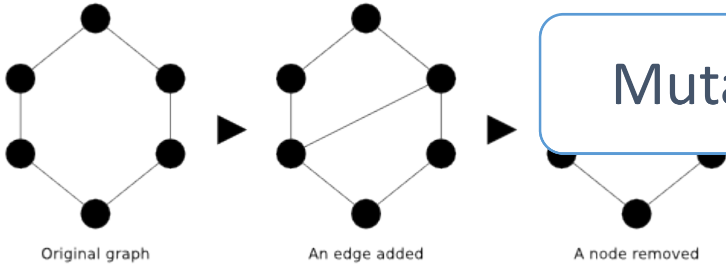
Different scenarios require different supports

Sampling

Single giant graph

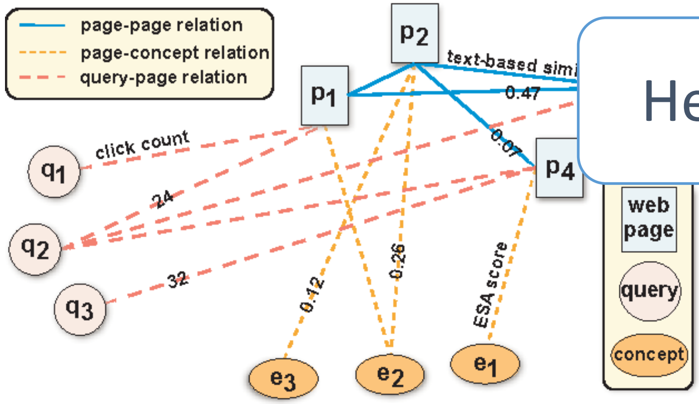
Batching graphs

Many moderate-sized graphs



Mutation

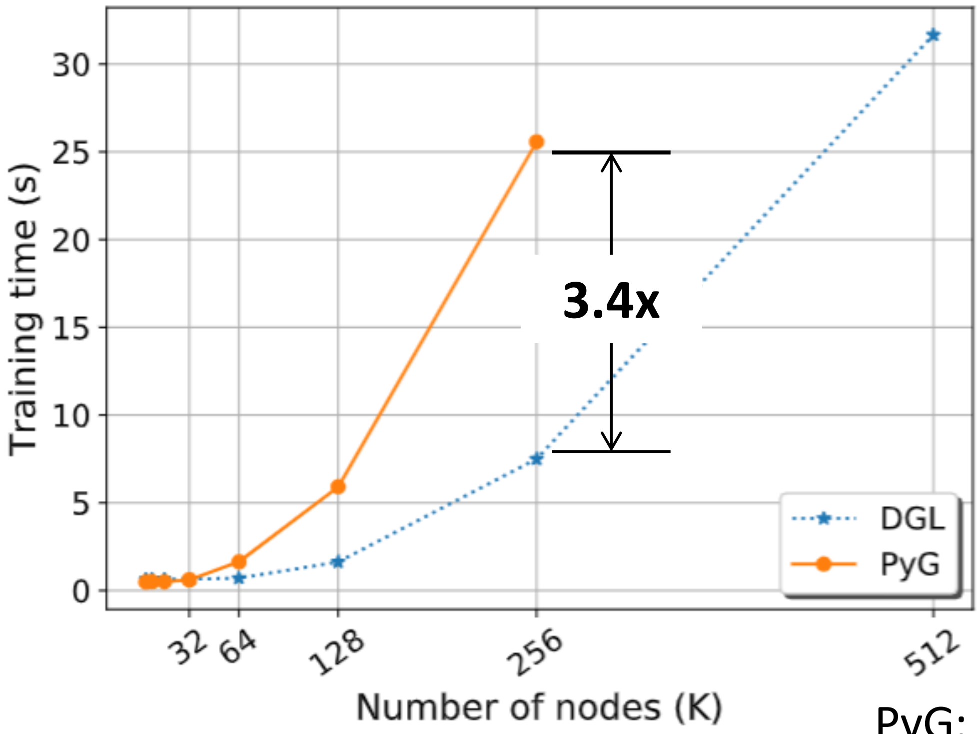
Dynamic graph

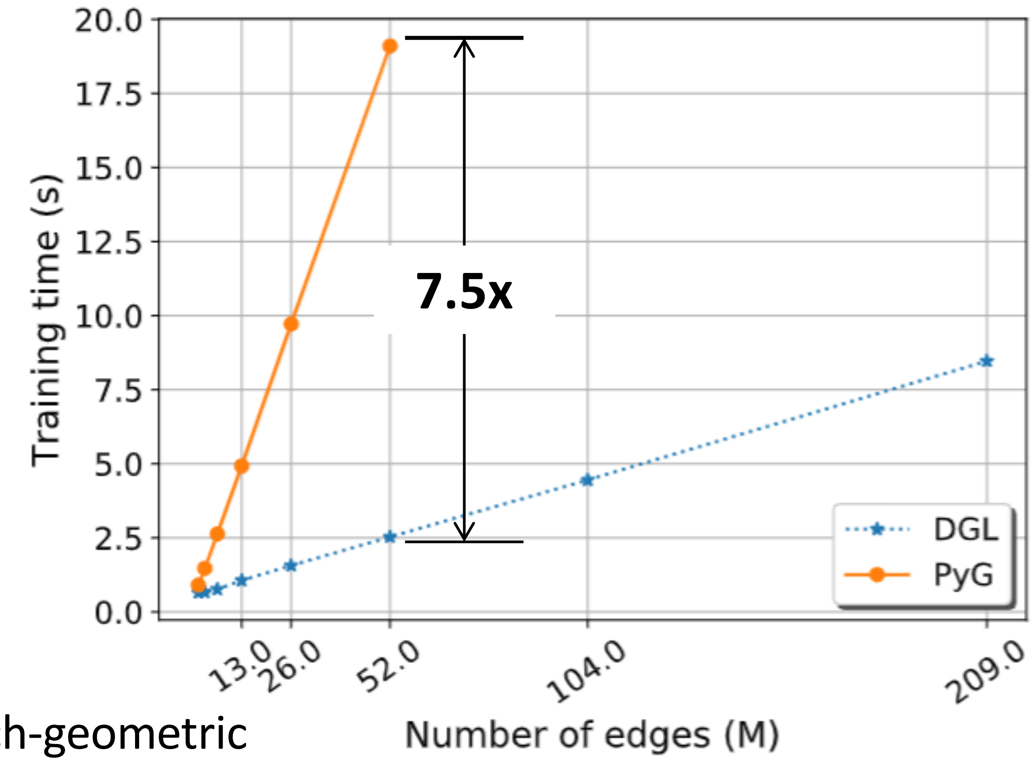


Heterogeneous

Performance

Scalability: single machine, single GPU



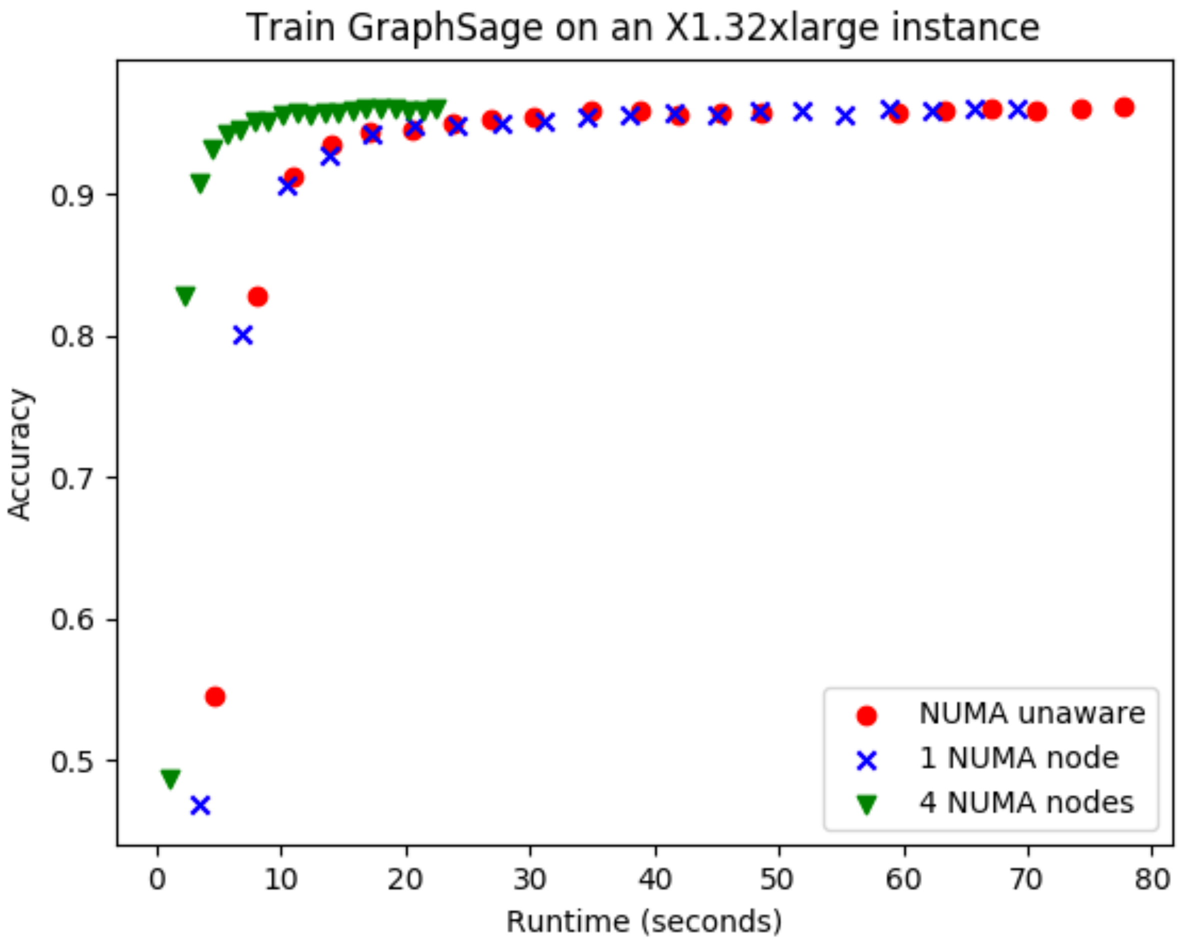


PyG: pytorch-geometric

Scalability with graph size

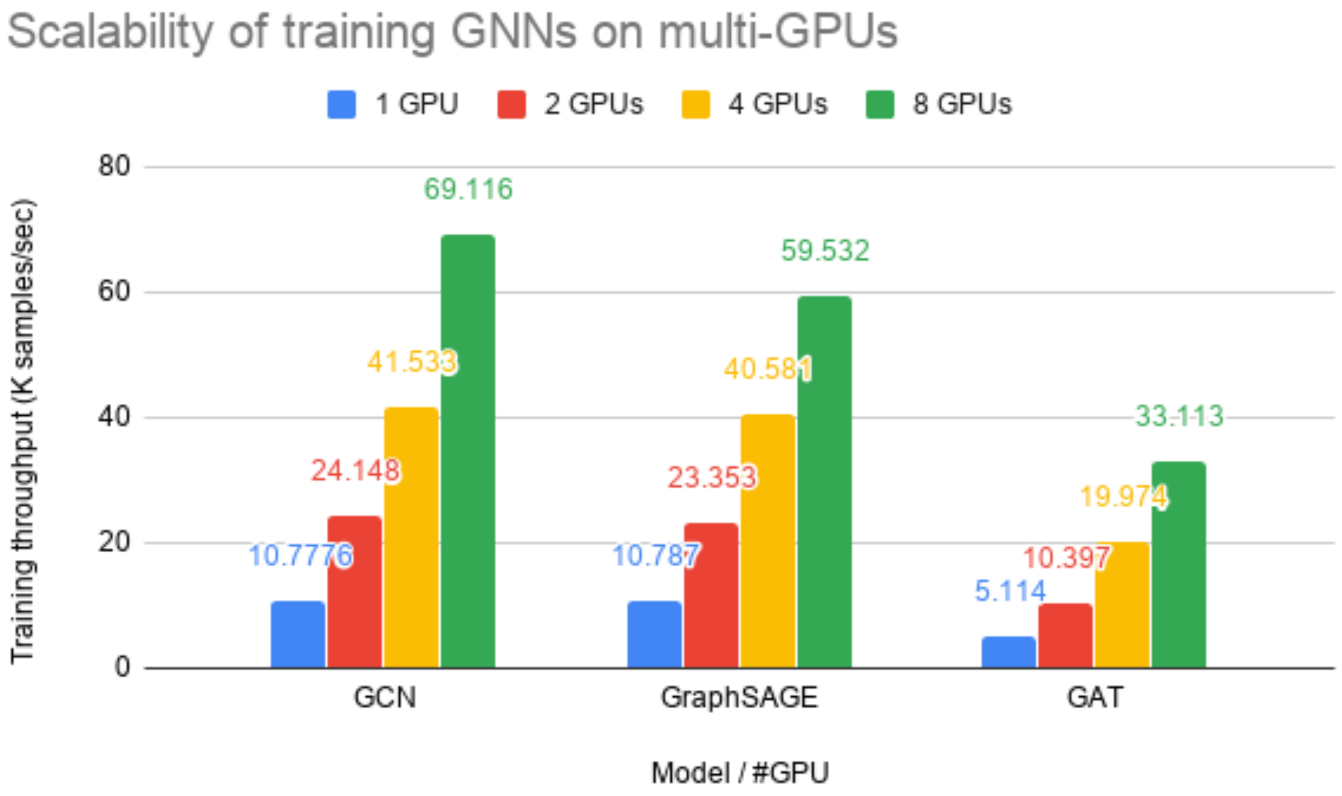
Scalability with graph density

Scalability: single machine, NUMA



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

Scalability: single machine, multi-GPU

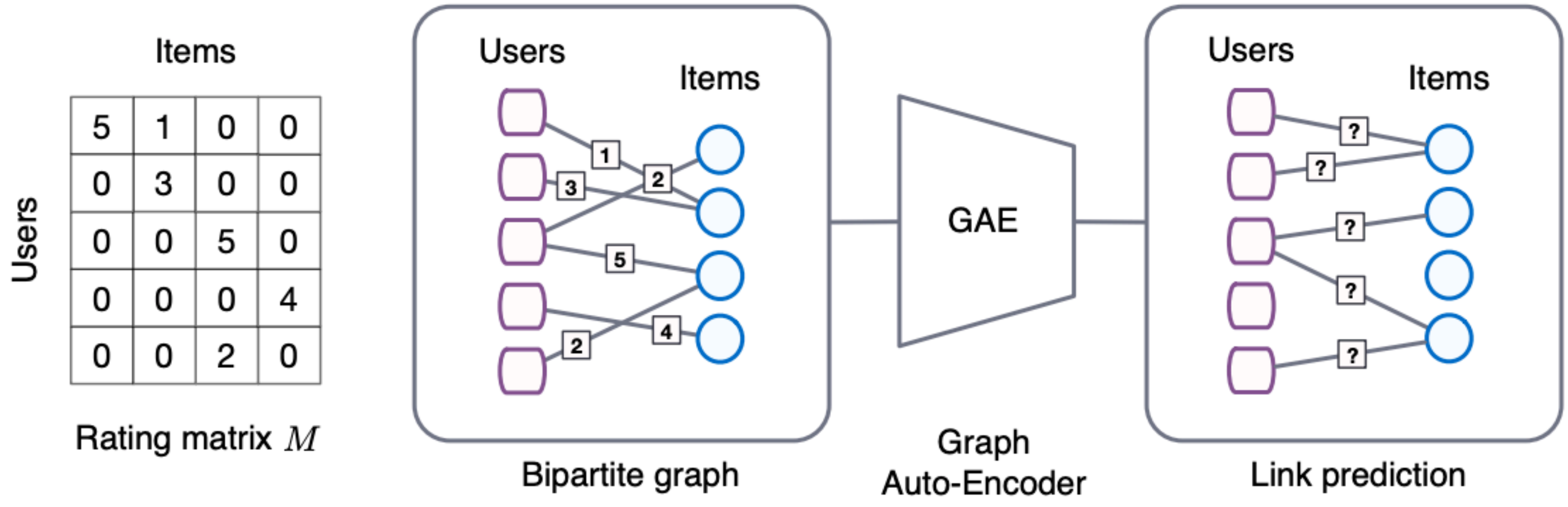


p3.16xlarge, 8 V100 GPUs, 64 vCPU
Data set: Reddit (232K nodes, 114M edges)
Trained with neighbor sampling

What’s new and what’s in the pipeline?

Heterogenous graph

Example: recommendation system, GCMC



Graph Convolutional Matrix Completion

Supporting Heterogeneous Graph

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

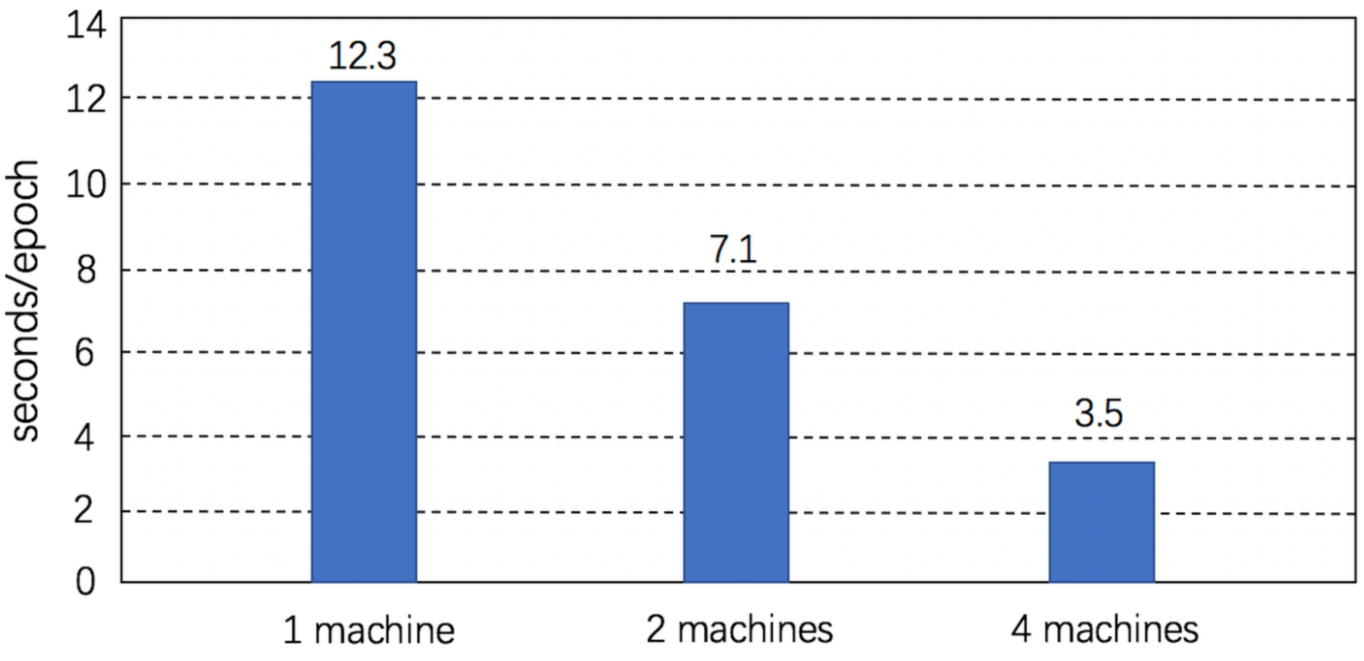
```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import dgl.function as fn

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	**0.025** s/epoch	0.101 s/epoch	5x
MovieLens-1M	0.8377	**0.832**	**0.070** s/epoch	1.538 s/epoch	22x
MovieLens-10M	0.7875	**0.777***	**0.648** s/epoch	Long*	

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

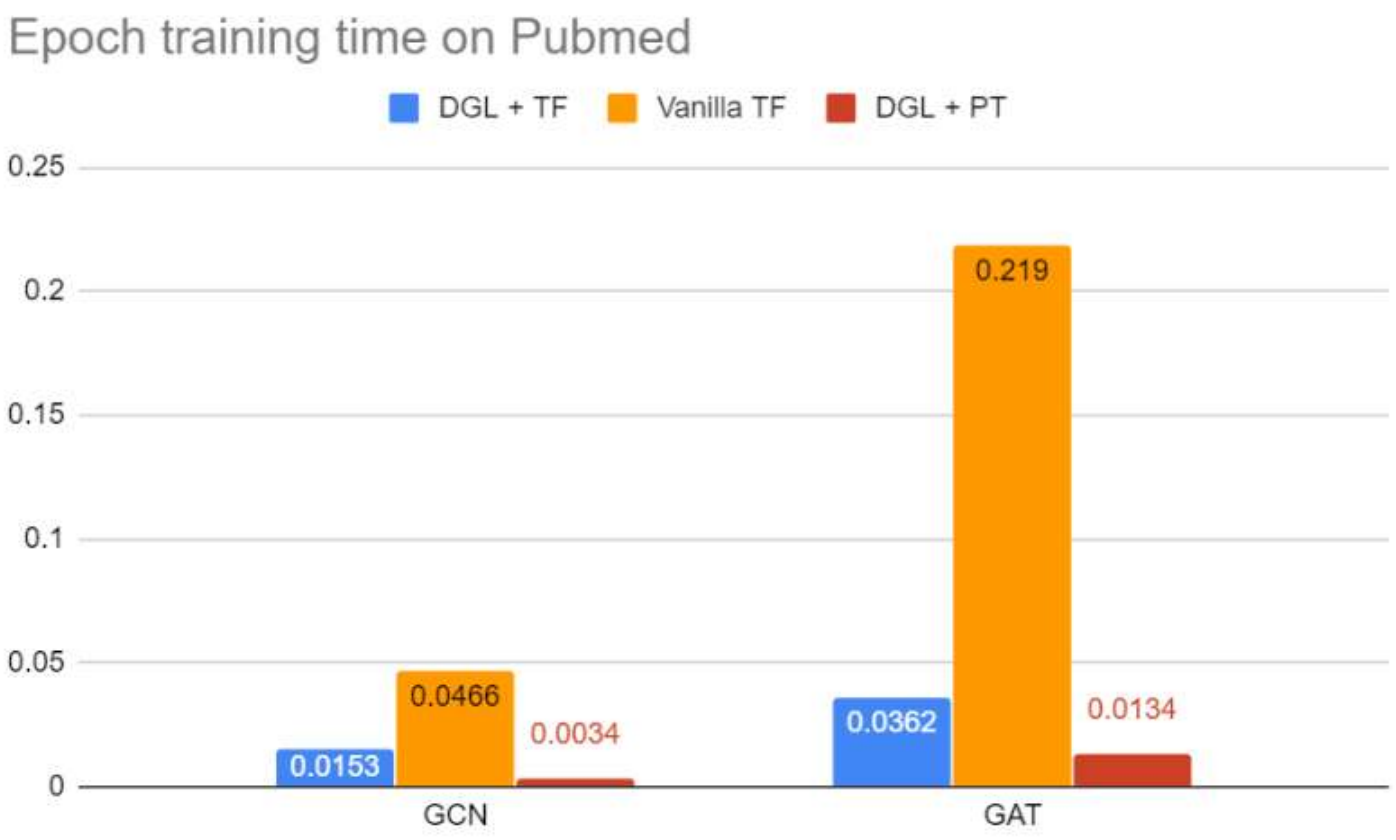
Distributed training: GCN (preliminary)



Distributed training of GCN on Reddit dataset.

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

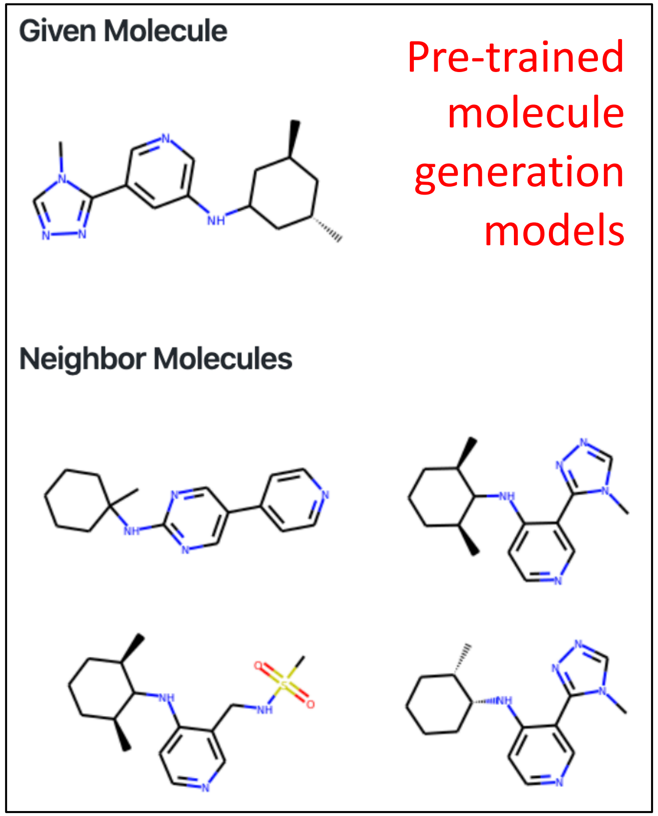
TF backend (preliminary)



Vanilla TF (TF 1.0)
DGL + TF (TF 2.0)

DGL Package: DGL-LifeSci

- Utilities for data processing
- Models for molecular property prediction and molecule generation
  - Graph Conv, GAT, MPNN, AttentiveFP, SchNet, MGCN, ACNN, DGMG, JTNN
- Efficient implementations
- Training scripts
- Pre-trained models

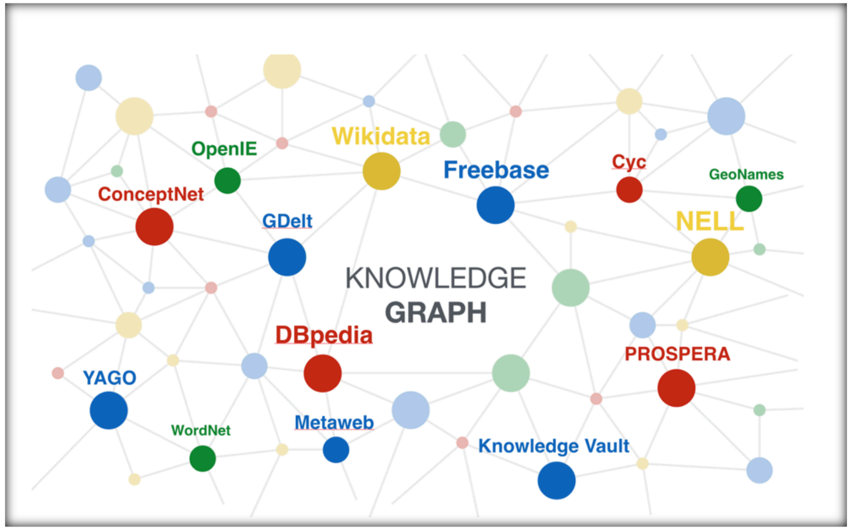


Efficient implementations

	DGL	Official	Speedup
Graph Conv	1.9s	8.4s (DeepChem)	4.4x
AttentiveFP	1.2s	6.0s	5.0x
JTNN	743s	1826s	2.5x

DGL Package: DGL-KE

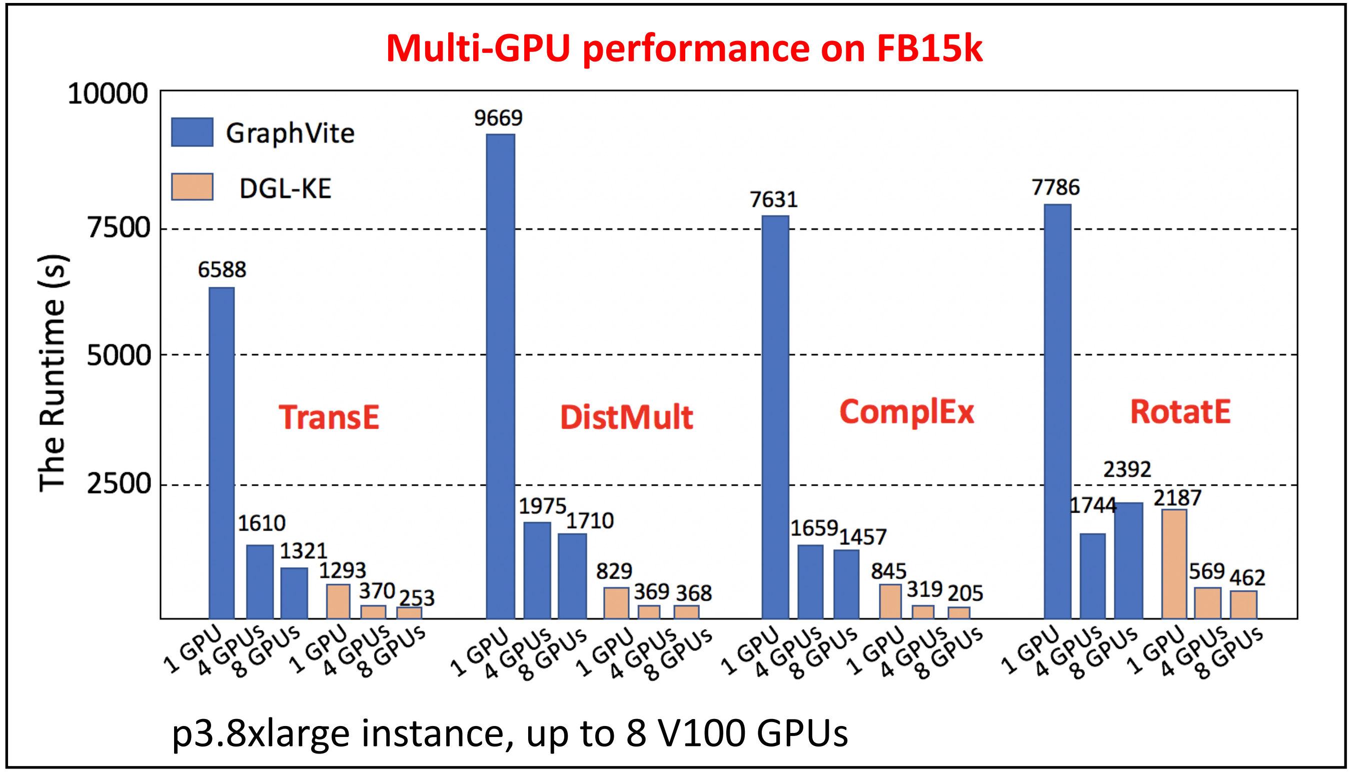
- An open-source package to efficiently compute knowledge graph embedding in various hardware:
  - Many-core CPU machine
  - Multi-GPU machine
  - A cluster of machines
- DGL-KE support popular KGE models:
  - TransE, TransR
  - DistMult, ComplEx, RESCAL
  - RotatE
- Applications: search, recommendation, question & answering

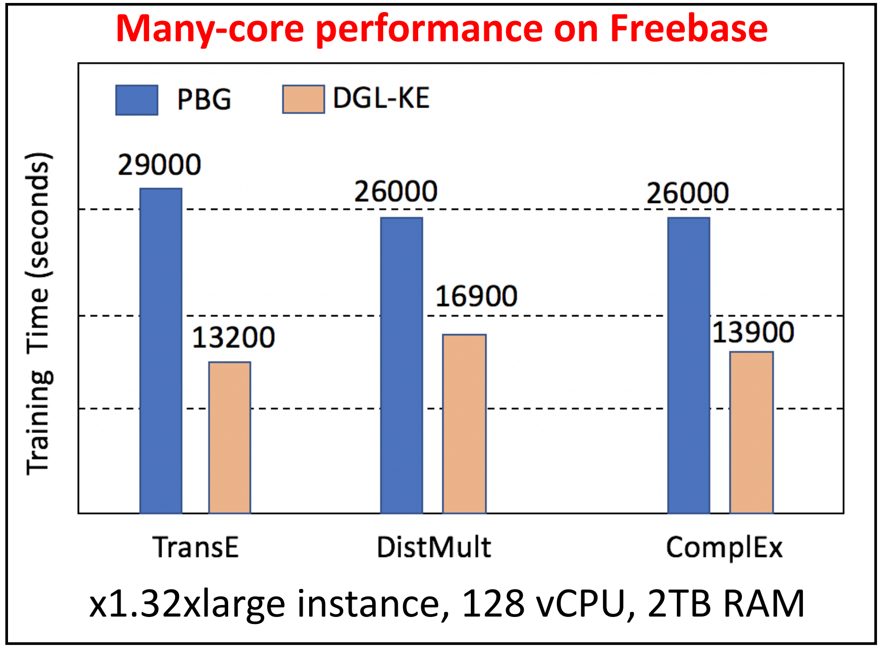


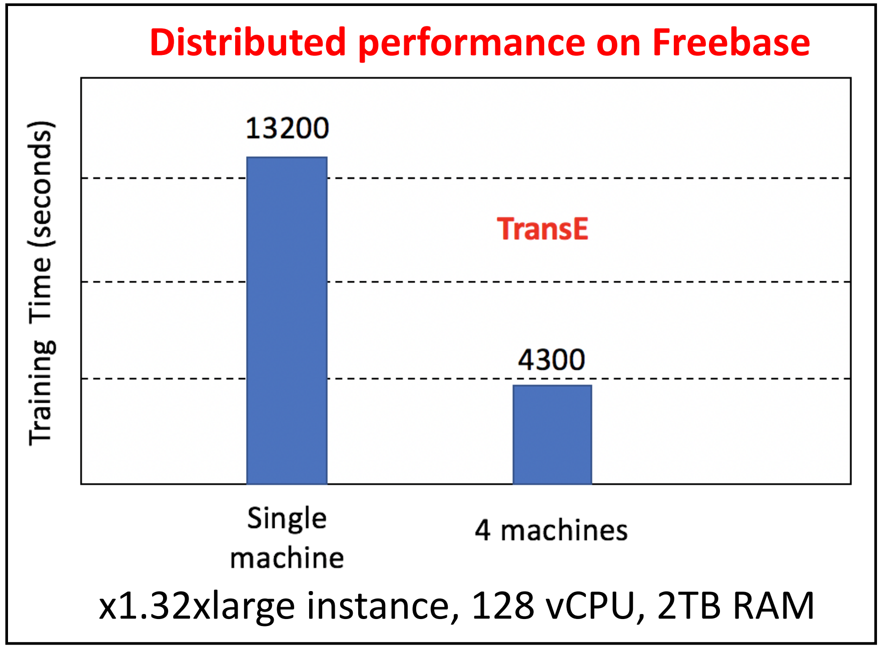
DGL-KE – Focus on high performance

- Maximize locality:
  - Metis graph partitioning to reduce network communication in distributed training.
  - Relation partitioning to avoid communication for relations in multi-GPU training.
- Increase computation-to-memory intensity:
  - Joint negative sampling to reduce the number of entities in a mini-batch
- Reduce the demands on memory bandwidth:
  - Sparse relation embeddings to reduce computation and data access in a batch.
- Hide data access latency:
  - Overlap gradient update with batch computation.

DGL-KE: Performance

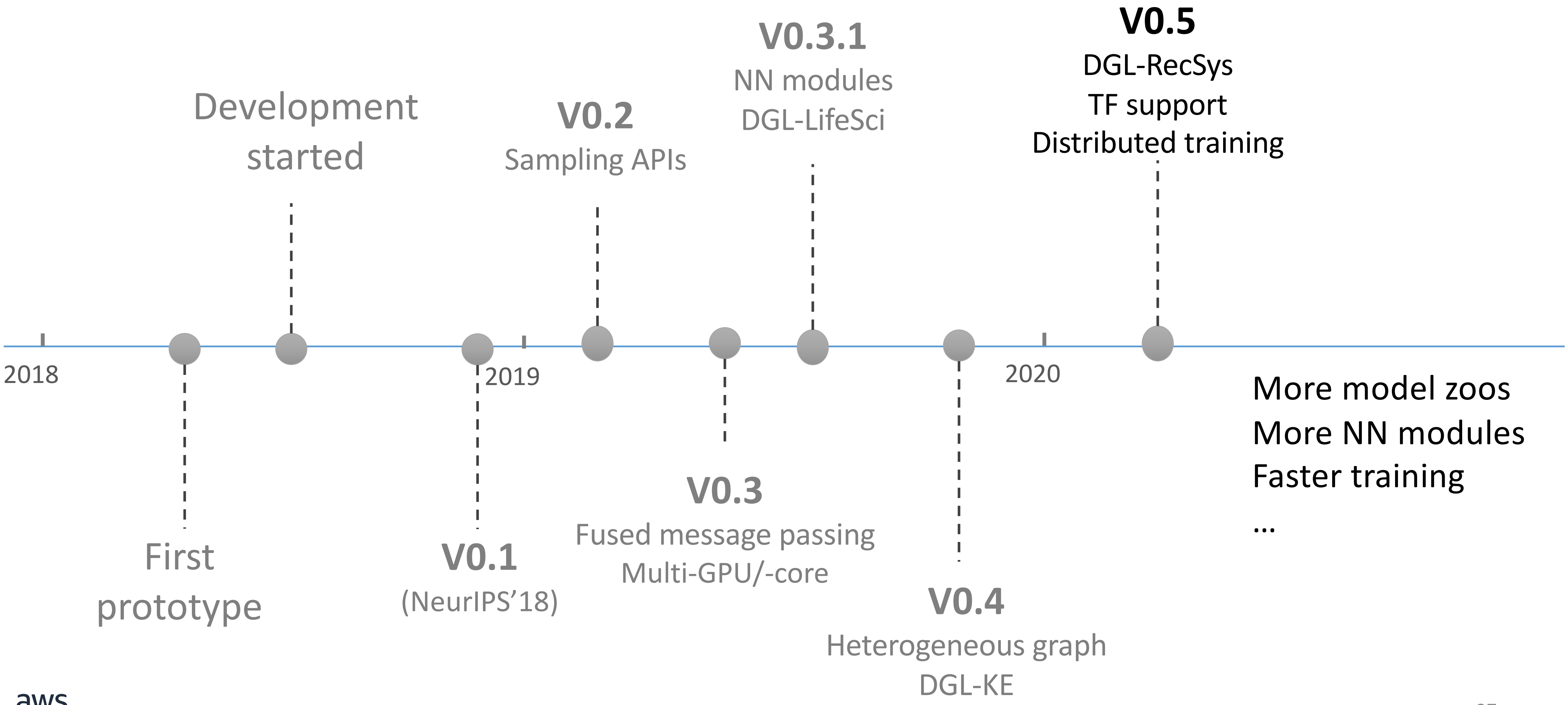






Datasets: FB15K (15K nodes, 592K edges); Freebase (86M nodes, 338M edges)

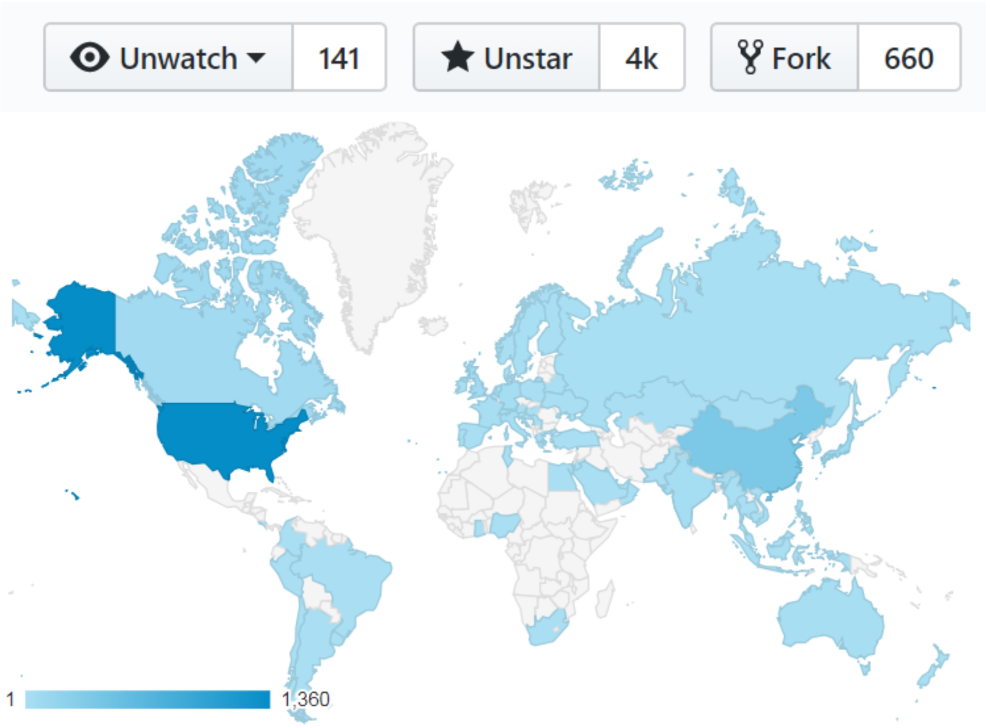
DGL: next step(s)

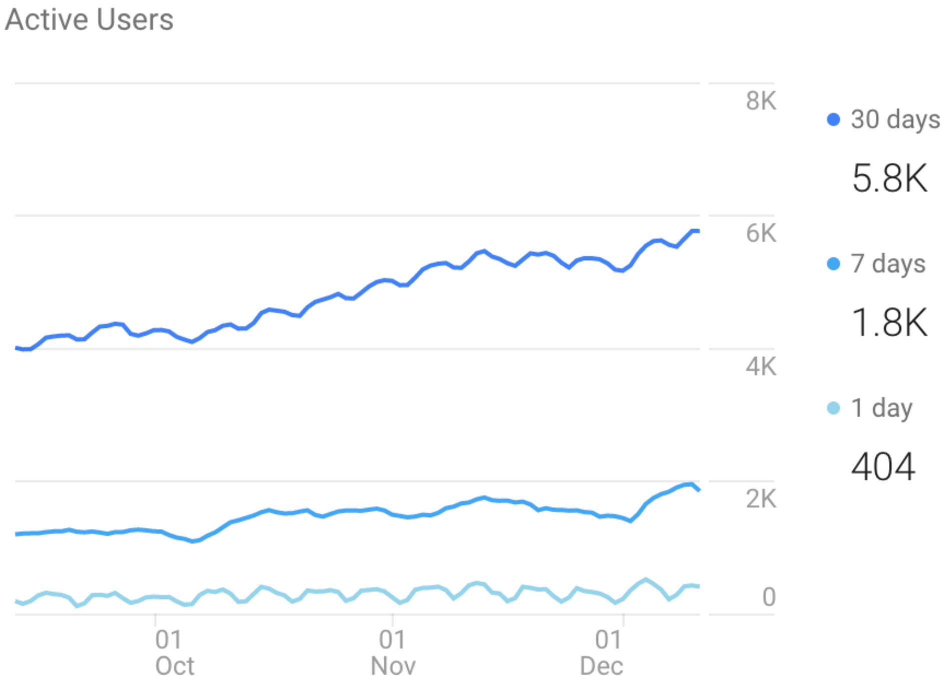


More model zoos
More NN modules
Faster training
…

Community

Open source, the source of innovation



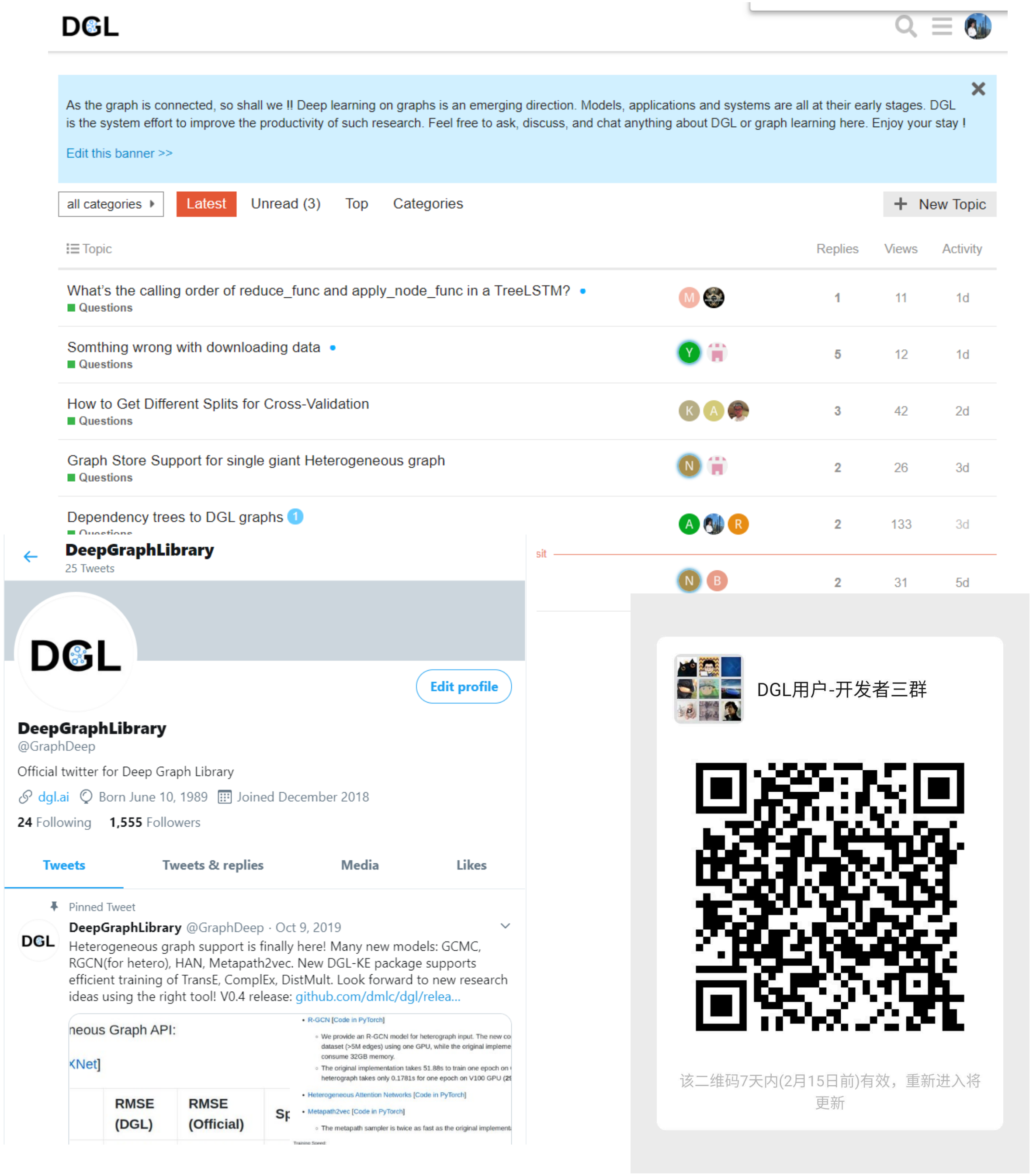


3975 github stars
312k downloads for all versions on Pip
8.8K downloads for all version on Conda
1.8K anaconda downloads of 0.4.1

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

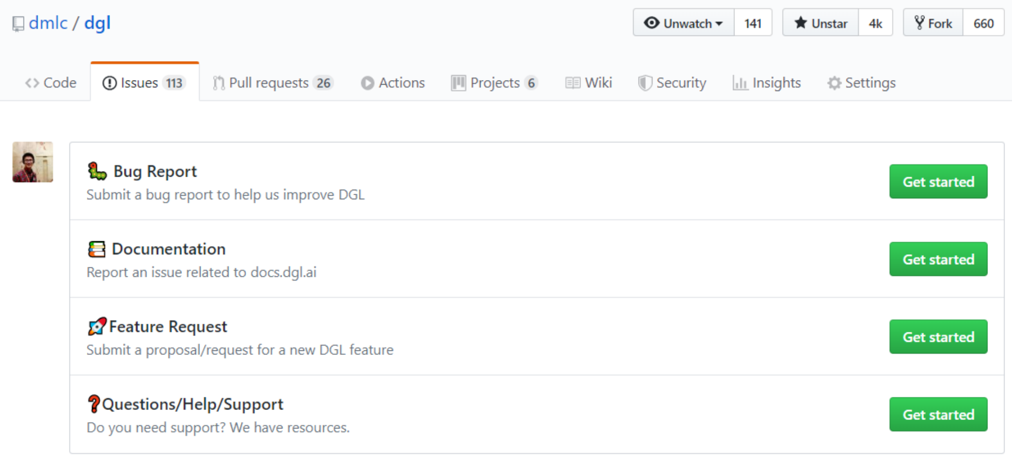
Channels

- Discuss forum https://discuss.dgl.ai
  - Any questions about DGL
  - Average response time: <1 day
- Github Issues https://github.com/dmlc/dgl/issues
  - Bug report and feature request.
- Twitter @GraphDeep
  - Latest news and releases
- Wechat group
  - 24/7 on-call ☺



Do you want to contribute?

- Data scientist? Researcher? or just ML lover?
  - Develop new models & applications.
- Tech writer? Native speaker?
  - Revise documents.
- System hacker?
  - More algorithms and operators on graphs.
- Share your work and experience from using DGL: https://github.com/dglai/awesome-dgl



DeepGraphLibrary

About — About DGL | Get Started — Install | Tutorials — Tutorials | Blog — Blog Posts | Doc — Documents | Forum — Discuss Forum | GitHub — Source Codes

https://www.dgl.ai

DEEP GRAPH LIBRARY

Easy Deep Learning on Graphs

Latest Updates

Get Started

Q&A

We are hiring!