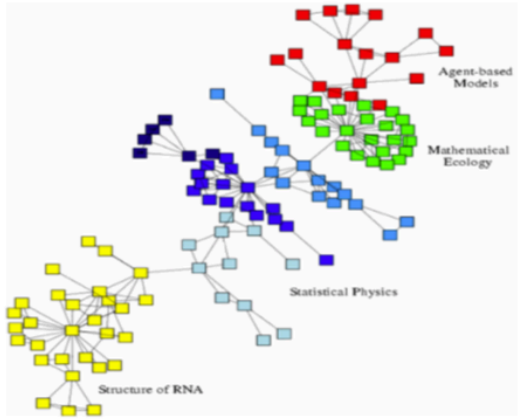
CogDL: 图深度学习工具包

清华大学KEG/NICS实验室
阿里巴巴
智谱.AI
北京智源人工智能研究院

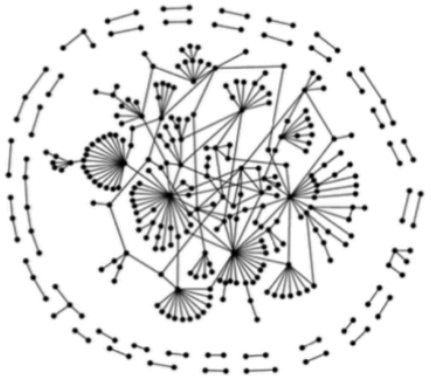
https://github.com/THUDM/cogdl

很多数据都是以网络的形式存在

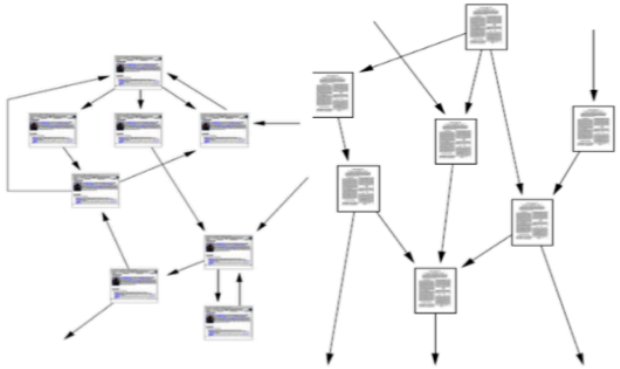
Social networks



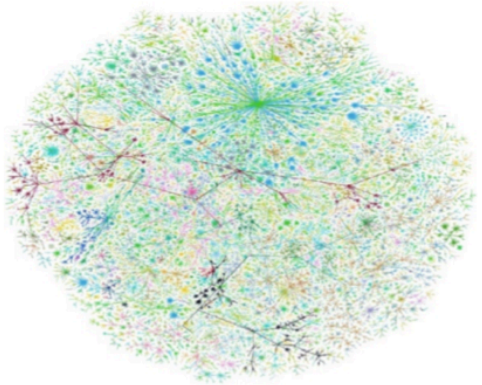
Economic networks



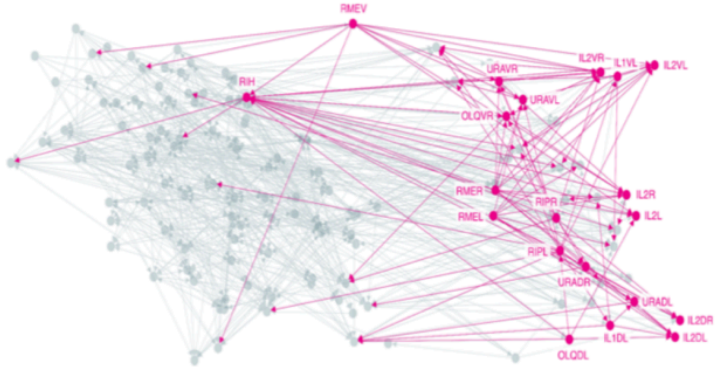
Biomedical networks



Information networks:
Web & citations



Internet



Networks of neurons

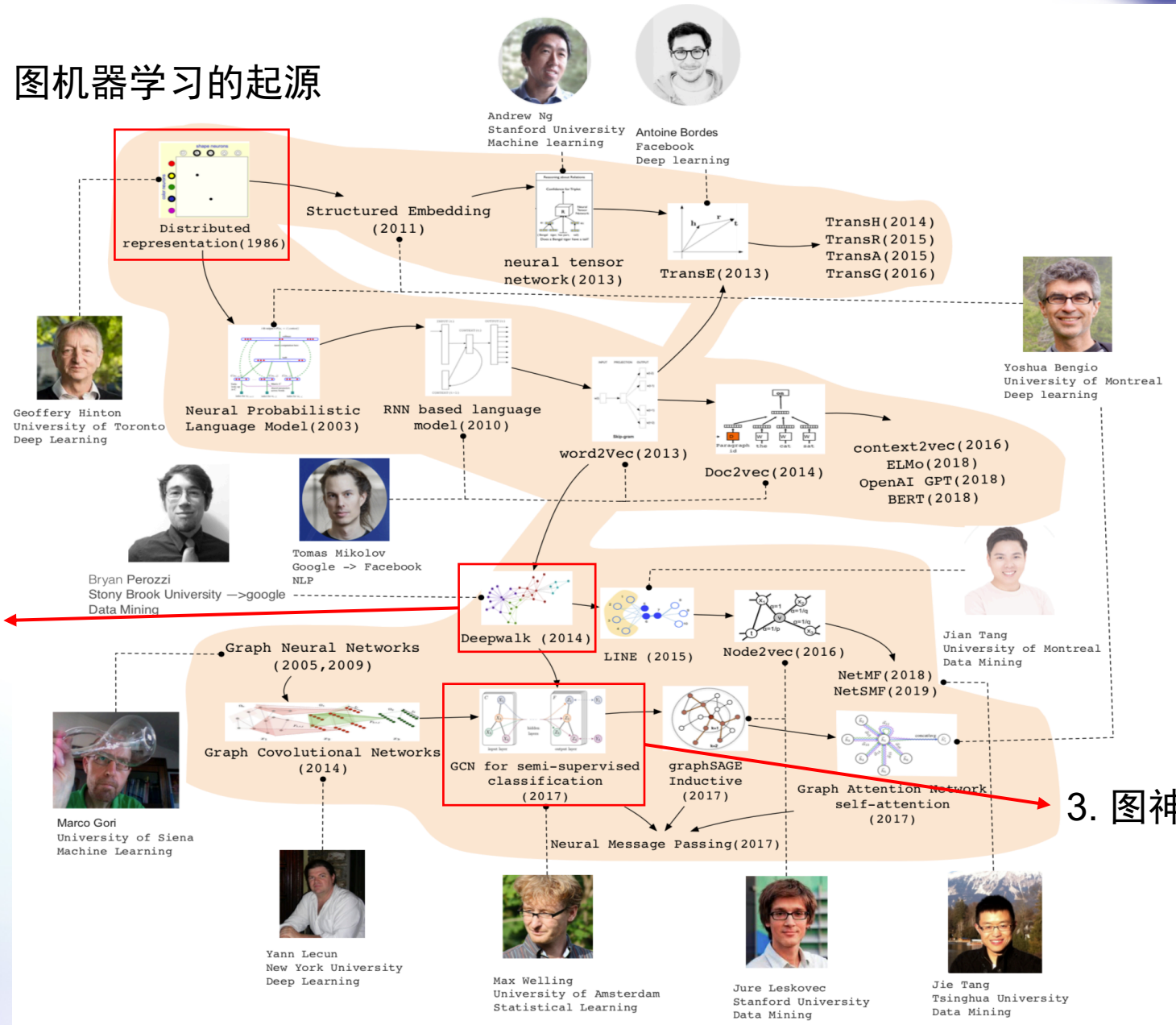
from Prof. Jure's slides

网络数据中的机器学习任务

- 节点分类：
  - 预测网络中节点的属性
- 链接预测：
  - 预测两个节点是否会产生联系
- 图分类：
  - 预测整个图的某种性质
- 社区检测：
  - 找出网络中关系紧密的聚类簇
- 网络相似度：
  - 计算两个网络之间的相似度

图机器学习的发展

1. 图机器学习的起源



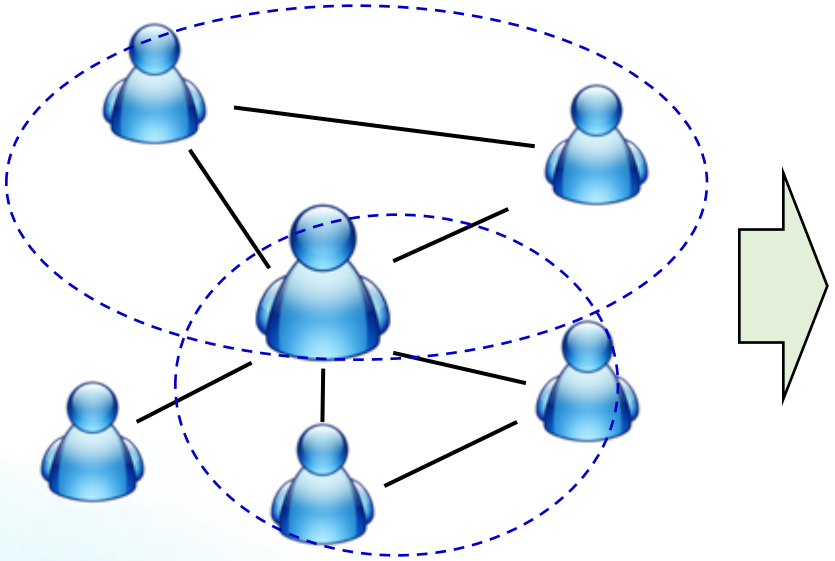
2. 图表示学习

3. 图神经网络

from Prof. Jie Tang's slides

回顾图表示学习

图表示学习

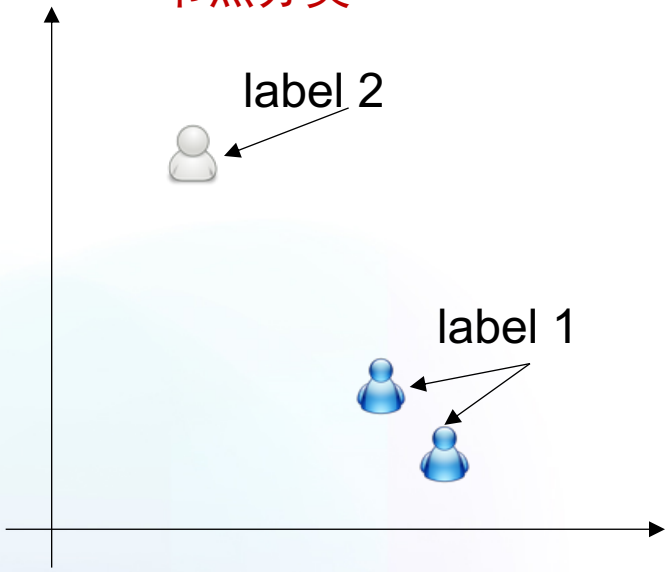


*d*维向量, $d<<|V|$

0.8	0.2	0.3	…	0.0	0.0

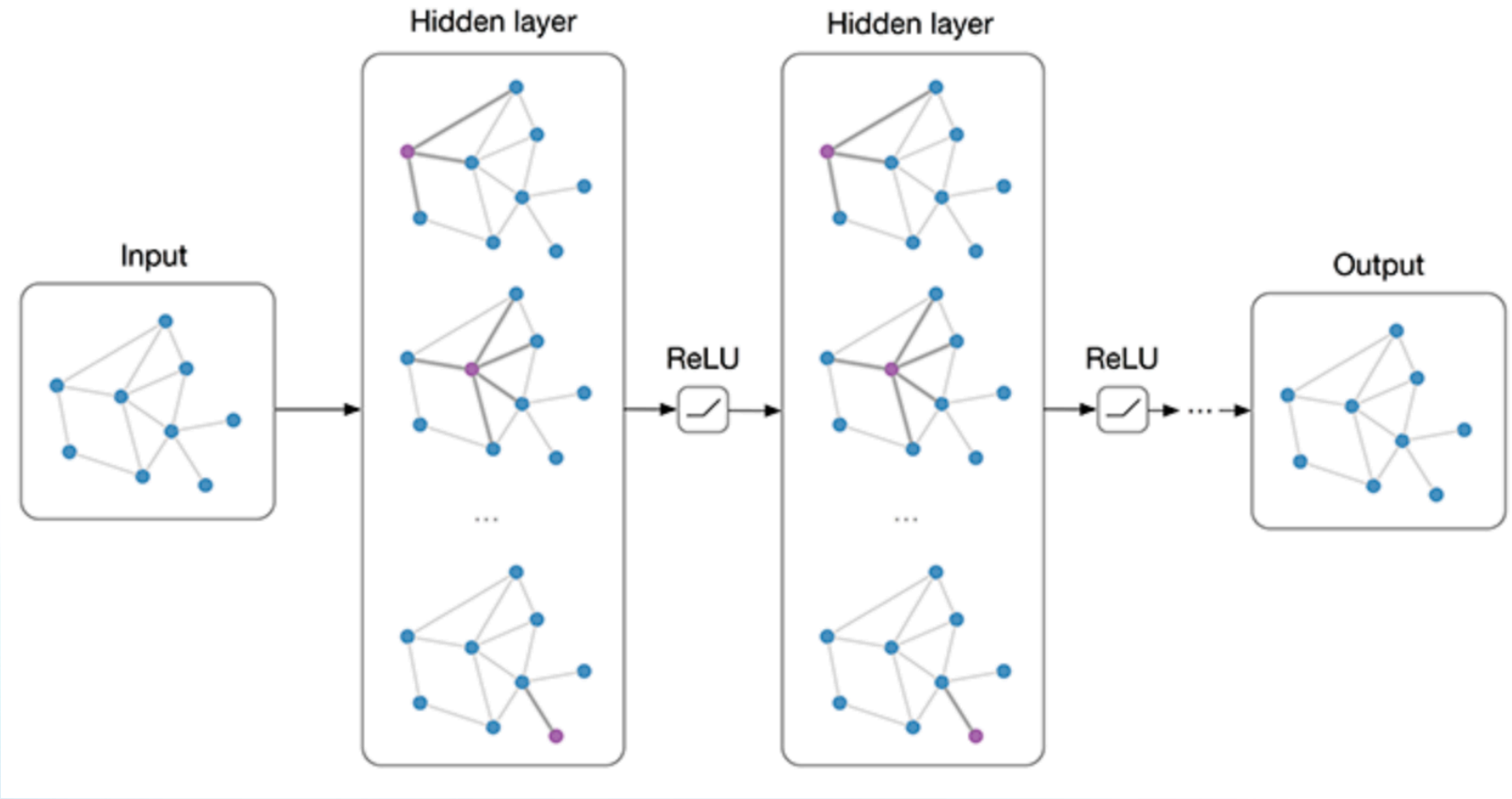
拥有相同标签的用户在d维的向量空间中的距离更接近。

节点分类



回顾图神经网络

- 对特征进行逐层传播和变换 $f(H^{(l)}, A) = \sigma\left(AH^{(l)}W^{(l)}\right)$



Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In ICLR ’17.

现有的问题

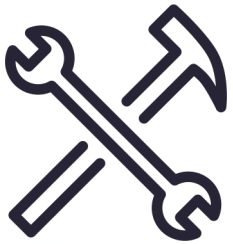
- **模型评估方面**
  - 没有建立起完整的评测数据集及评估方式
  - 相关论文报告的结果无法直接比较
- **模型实现方面**
  - 不同模型可能使用了不同的实现框架
  - 运行不同的模型需要搭建不同的运行环境
- **底层性能方面**
  - 很多框架对稀疏矩阵的计算支持不足
  - 图神经网络的实现和性能受到了限制

CogDL简介

愿景

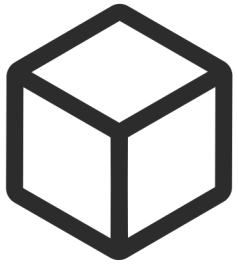
在大多数图机器学习相关的下游任务和应用中，为广大的研发人员实时维护完整可复现的排行榜。

理念



易用接口

可复现性



高效计算

Yukuo Cen, Zhenyu Hou, Yan Wang, Qibin Chen, Yizhen Luo, Xingcheng Yao, Aohan Zeng, Shiguang Guo, Peng Zhang, Guohao Dai, Yu Wang, Chang Zhou, Hongxia Yang, and Jie Tang. CogDL: An Extensive Toolkit for Deep Learning on Graphs. arXiv preprint 2021.

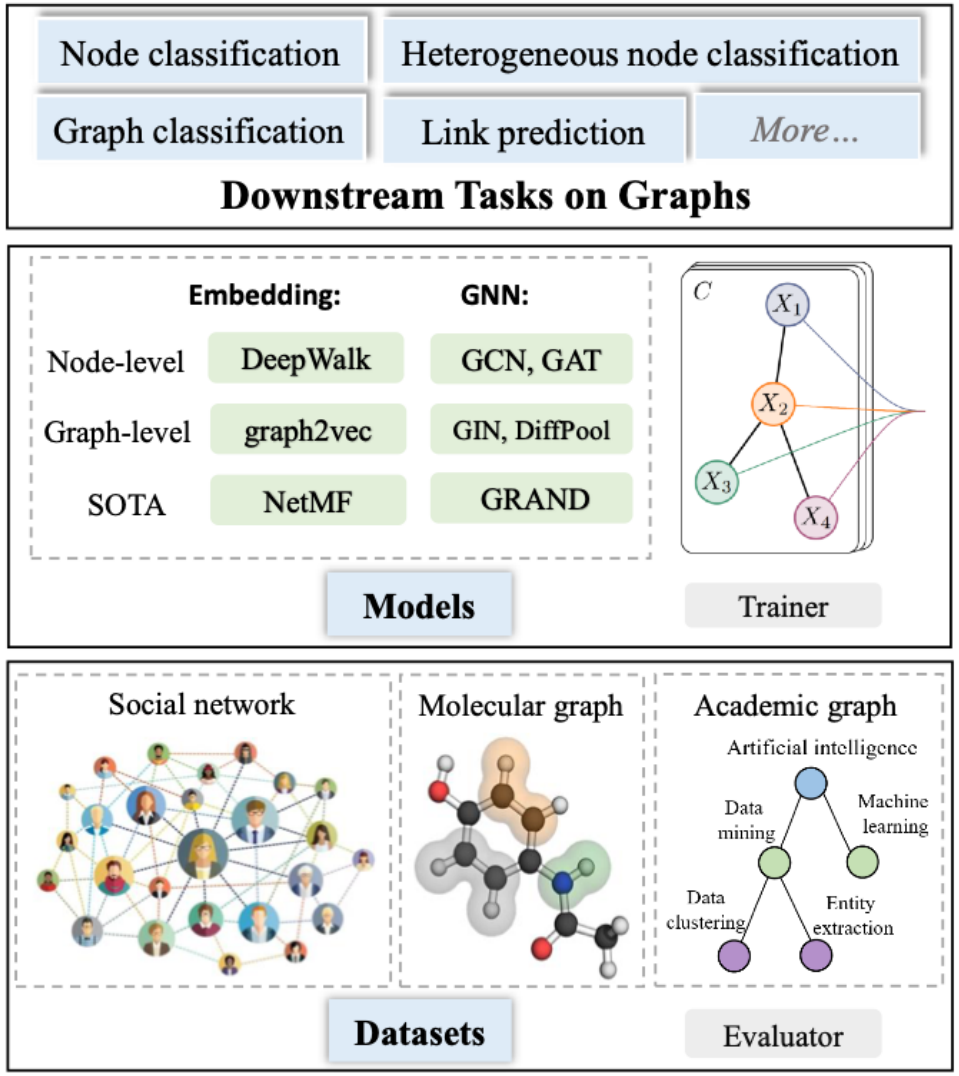
CogDL开发进展

Star 671 Fork 172

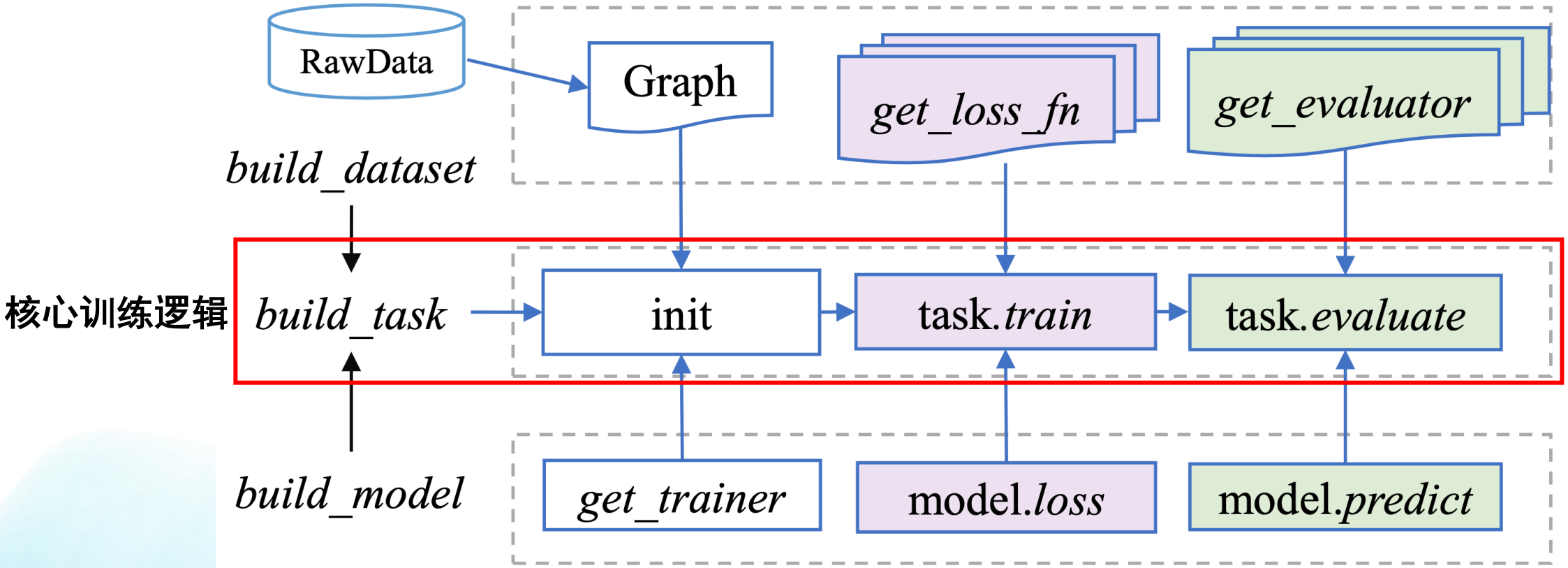
https://github.com/THUDM/cogdl

CogDL中集成的任务、模型、数据集

- 集成10+类下游任务用于评测
  - 节点分类
  - 图分类
- 集成60+个不同的数据集
  - 社交网络
  - 学术网络
  - 分子图
- 集成70+种不同的算法/模型
  - 表示学习算法
  - 图神经网络模型



易用接口 – 核心逻辑


RawData
Graph
get_loss_fn
get_evaluator
build_dataset
核心训练逻辑
build_task
init
task.train
task.evaluate
build_model
get_trainer
model.loss
model.predict

易用接口 - “一行即训练”

- experiment API：传入任务、数据集、模型、（超参）、（超参搜索范围）

```python
from cogdl import experiment

# basic usage
experiment(task="node_classification", dataset="cora", model="gcn")

# set other hyper-parameters
experiment(task="node_classification", dataset="cora", model="gcn", hidden_size=32, max_epoch=200)

# run over multiple models on different seeds
experiment(task="node_classification", dataset="cora", model=["gcn", "gat"], seed=[1, 2])

# automl usage
def func_search(trial):
    return {
        "lr": trial.suggest_categorical("lr", [1e-3, 5e-3, 1e-2]),
        "hidden_size": trial.suggest_categorical("hidden_size", [32, 64, 128]),
        "dropout": trial.suggest_uniform("dropout", 0.5, 0.8),
    }

experiment(task="node_classification", dataset="cora", model="gcn", seed=[1, 2], func_search=func_search)
```

易用接口 – 自定义模型

```python
@register_model("mygcn")
class GCN(BaseModel):

    @staticmethod
    def add_args(parser):
        parser.add_argument("--hidden-size", type=int, default=64)
        parser.add_argument("--dropout", type=float, default=0.5)

    @classmethod
    def build_model_from_args(cls, args):
        return cls(args.num_features, args.hidden_size, args.num_classes, args.dropout)

    def __init__(self, in_feats, hidden_size, out_feats, dropout):
        super(GCN, self).__init__()
        self.conv1 = GraphConvolution(in_feats, hidden_size)
        self.conv2 = GraphConvolution(hidden_size, out_feats)
        self.dropout = nn.Dropout(dropout)

    def forward(self, graph):
        graph.sym_norm()
        h = graph.x
        h = F.relu(self.conv1(graph, self.dropout(h)))
        h = self.conv2(graph, self.dropout(h))
        return h

if __name__ == "__main__":
    experiment(task="node_classification", dataset="cora", model="mygcn")
```

1. 定义/传入模型的超参

2. 定义模型中的模块

3. 定义模型的执行逻辑

易用接口 – 自定义数据集

```python
@register_dataset("mydataset")
class MyDataset(BaseDataset):
    def __init__(self):
        super(MyDataset, self).__init__()

        num_nodes = 200
        num_edges = 500
        num_labels = 10
        edge_index = torch.randint(0, num_nodes, (2, num_edges))
        y = torch.randint(0, 2, (num_nodes, num_labels))
        self.data = Graph(edge_index=edge_index, y=y)

if __name__ == "__main__":
    experiment(task="unsupervised_node_classification", dataset="mydataset", model="prone")
```

只需要加载自定义的数据集即可

可复现性 - 节点分类任务（网络表示学习）

- 不同类型的数据集：
  - 蛋白质交互数据集PPI；词共现网络数据集Wikipedia
  - 社交网络数据集Blog和Flickr；学术引用数据集DBLP
- 两类网络表示学习方法：
  - 基于矩阵分解的方法：NetMF, ProNE, NetSMF, ...
  - 基于skip-gram的方法：DeepWalk, LINE, Node2vec, ...

Rank	Method	PPI (50%)	Wikipedia (50%)	Blogcatalog (50%)	DBLP (5%)	Flickr (5%)	*Reproducible*
1	NetMF [37]	23.73 ± 0.22	57.42 ± 0.56	42.47 ± 0.35	56.72 ± 0.14	36.27 ± 0.17	*Yes*
2	ProNE [66]	24.60 ± 0.39	56.06 ± 0.48	41.16 ± 0.26	56.85 ± 0.28	36.56 ± 0.11	*Yes*
3	NetSMF [36]	23.88 ± 0.35	53.81 ± 0.58	40.62 ± 0.35	59.76 ± 0.41	35.49 ± 0.07	*Yes*
4	Node2vec [17]	20.67 ± 0.54	54.59 ± 0.51	40.16 ± 0.29	57.36 ± 0.39	36.13 ± 0.13	*Yes*
5	LINE [45]	21.82 ± 0.56	52.46 ± 0.26	38.06 ± 0.39	49.78 ± 0.37	31.61 ± 0.09	*Yes*
6	DeepWalk [35]	20.74 ± 0.40	49.53 ± 0.54	40.48 ± 0.47	57.54 ± 0.32	36.09 ± 0.10	*Yes*
7	SpectralClustering [48]	22.48 ± 0.30	49.35 ± 0.34	41.41 ± 0.34	43.68 ± 0.58	33.09 ± 0.07	*Yes*
8	Hope [33]	21.43 ± 0.32	54.04 ± 0.47	33.99 ± 0.35	56.15 ± 0.22	28.97 ± 0.19	*Yes*
9	GraRep [5]	20.60 ± 0.34	54.37 ± 0.40	33.48 ± 0.30	52.76 ± 0.42	31.83 ± 0.12	*Yes*

可复现性 – 节点分类任务（图神经网络）

- 学术引用数据集：
  - Cora, Citeseer, Pubmed
- 两类图神经网络模型：
  - 监督模型：GCN, GAT, GRAND, …
  - 无监督模型：MVGRL, DGI

Rank	Method	Cora	Citeseer	Pubmed	*Reproducible*
1	GRAND [12]	84.8	75.1	82.4	*Yes*
2	GCNII [7]	85.1	71.3	80.2	*Yes*
3	MVGRL [20]	83.6 ↓	73.0	80.1	*Partial*
4	APPNP [26]	**84.3** ↑	72.0	80.0	*Yes*
5	Graph-Unet [15]	83.3 ↓	71.2 ↓	79.0	*Partial*
6	GDC [27]	82.5	72.1	79.8	*Yes*
7	GAT [53]	82.9	71.0	78.9	*Yes*
8	DropEdge [38]	82.1	72.1	79.7	*Yes*
9	GCN [25]	**82.3** ↑	**71.4** ↑	79.5	*Yes*
10	DGI [52]	82.0	71.2	76.5	*Yes*
11	JK-net [58]	81.8	69.5	77.7	*Yes*
12	Chebyshev [8]	79.0	69.8	68.6	*Yes*

可复现性 - 图分类任务

两类不同类型的图数据

- 生物信息数据集：MUTAG, PTC, NCI1, PROTEINS
- 社交网络数据集：IMDB-B/M, COLLAB, REDDIT-B

两类图分类模型

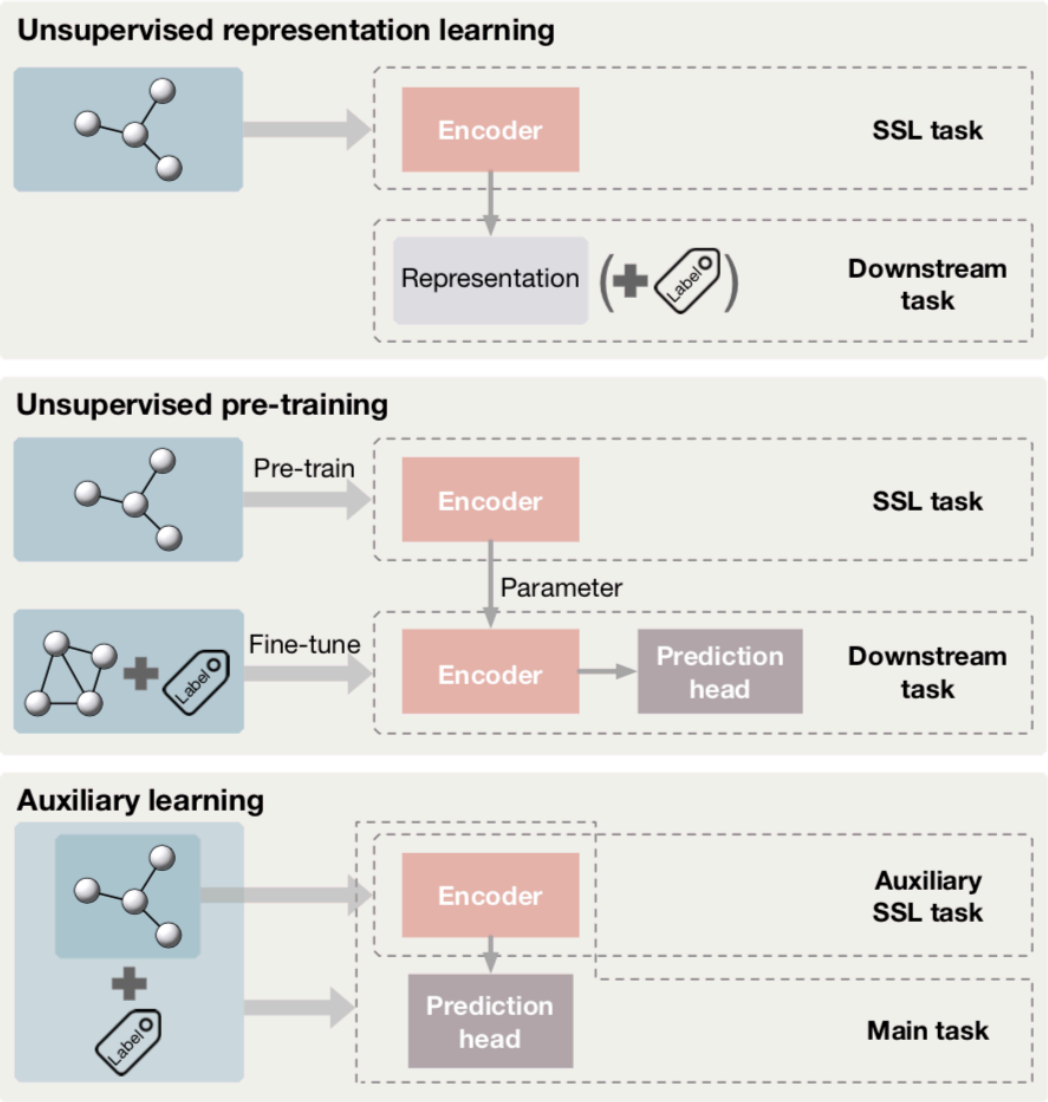
- 无监督模型：InfoGraph, graph2vec, DGK
- 有监督模型：GIN, DiffPool, SortPool, ...

Algorithm	MUTAG	PTC	NCI1	PROTEINS	IMDB-B	IMDB-M	COLLAB	REDDIT-B	*Reproducible*
GIN [57]	92.06	67.82	81.66	75.19	76.10	51.80	79.52	83.10 ↓	*Yes*
InfoGraph [42]	88.95	60.74	76.64	73.93	74.50	51.33	79.40	76.55	*Yes*
DiffPool [62]	85.18	58.00	69.09	75.30	72.50	50.50	79.27	81.20	*Yes*
SortPool [67]	87.25	62.04	**73.99** ↑	74.48	75.40	50.47	**80.07** ↑	78.15	*Yes*
graph2vec [31]	83.68	54.76 ↓	71.85	73.30	73.90	52.27	**85.58** ↑	91.77	*Yes*
PATCHY_SAN [32]	86.12	61.60	69.82	75.38	**76.00** ↑	46.40	74.34	60.61	*Yes*
DGCNN [56]	83.33	56.72	65.96	66.75	71.60	49.20	77.45	86.20	*Yes*
SAGPool [28]	71.73 ↓	59.92	72.87	74.03	74.80	51.33	/	89.21	*Yes*
DGK [59]	85.58	57.28	/	72.59	55.00 ↓	40.40 ↓	/	/	*Partial*

可复现性 – 图自监督学习

图自监督学习的主要构成要素有：

- 训练方式：
  - 无监督表示学习
  - 无监督预训练
  - 联合训练
- 自监督任务：
  - 生成式（generative）
  - 对比式（contrastive）
- 图编码器：GCN，GAT，GIN
- 下游任务：节点分类、图分类



可复现性 – 图自监督学习

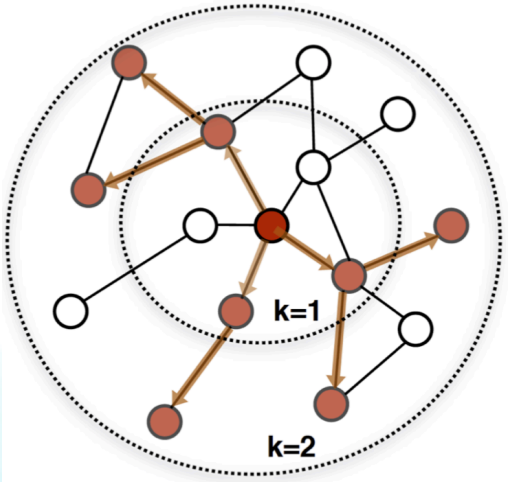
训练方式：监督学习（SL），联合训练（JL），无监督表示学习（URL）

半监督数据集：Cora, Citeseer, PubMed；监督数据集：Flickr, Reddit

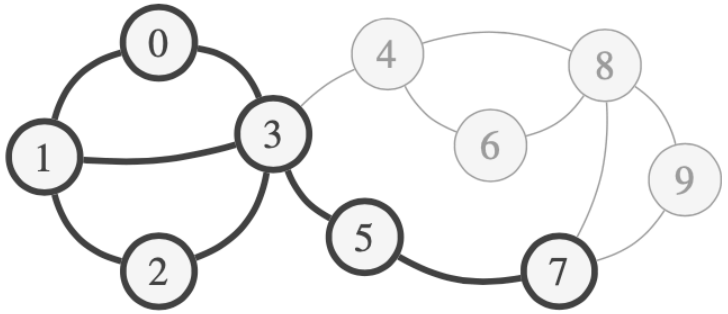
Model	Training Scheme	Cora	Citeseer	PubMed	Flickr	Reddit
Linear	SL	47.86±0.02	49.25±0.04	69.17±0.03	45.81±0.00	67.95±0.01
GCN	SL	81.53± 0.26	71.75±0.05	79.30±0.31	52.74±0.13	95.16±0.01
EdgeMask	JL	81.28±0.31	71.53±0.24	79.55±0.11	52.55±0.06	95.07±0.01
AttributeMask	JL	81.20±0.32	71.45±0.35	78.93±0.25	52.45±0.15	95.00±0.01
S^2GRL	JL	83.42±0.63	72.37±0.11	81.20±0.37	52.59±0.05	95.17±0.00
Distance2Clusters	JL	82.47±0.48	71.55±0.30	81.53±0.22	52.24±0.07	/
SuperGAT	JL	82.77±0.53	72.25±0.52	80.30±0.31	52.80±0.07	95.42±0.01
MVGRL	URL	80.76±0.80	65.84±2.72	76.01±2.46	46.08±0.39	92.76±0.22
DGI	URL	81.91±0.17	70.01±0.87	76.49±1.04	46.35±0.11	93.12±0.18
EdgeMask	URL	75.23±1.14	68.96±0.96	79.41±1.15	50.48±0.06	93.68±0.01
AttributeMask	URL	76.12±0.91	70.69±0.44	75.16±1.71	51.57±0.15	93.37±0.01
S^2GRL	URL	81.56±0.49	69.48±0.91	80.83±0.57	50.85±0.03	93.88±0.05
Distance2Cluster	URL	73.86±0.29	66.53±0.29	79.44±0.34	50.24±0.07	/

高效计算 – 超大规模图训练

- 超大规模的社交网络、推荐系统（亿级节点）
- 超大规模训练的主要挑战在于图神经网络的空间复杂度高
- 可以通过mini-batch的方式来训练图神经网络

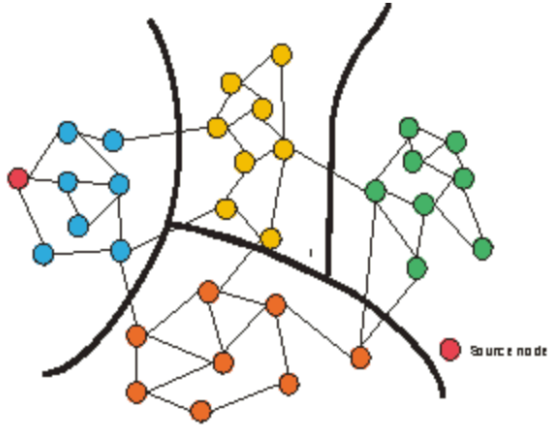


Neighbor Sampling
(NeurIPS ’17)



$\mathcal{G}_s = \texttt{SAMPLE}(\mathcal{G})$

GraphSAINT
(ICLR ’20)

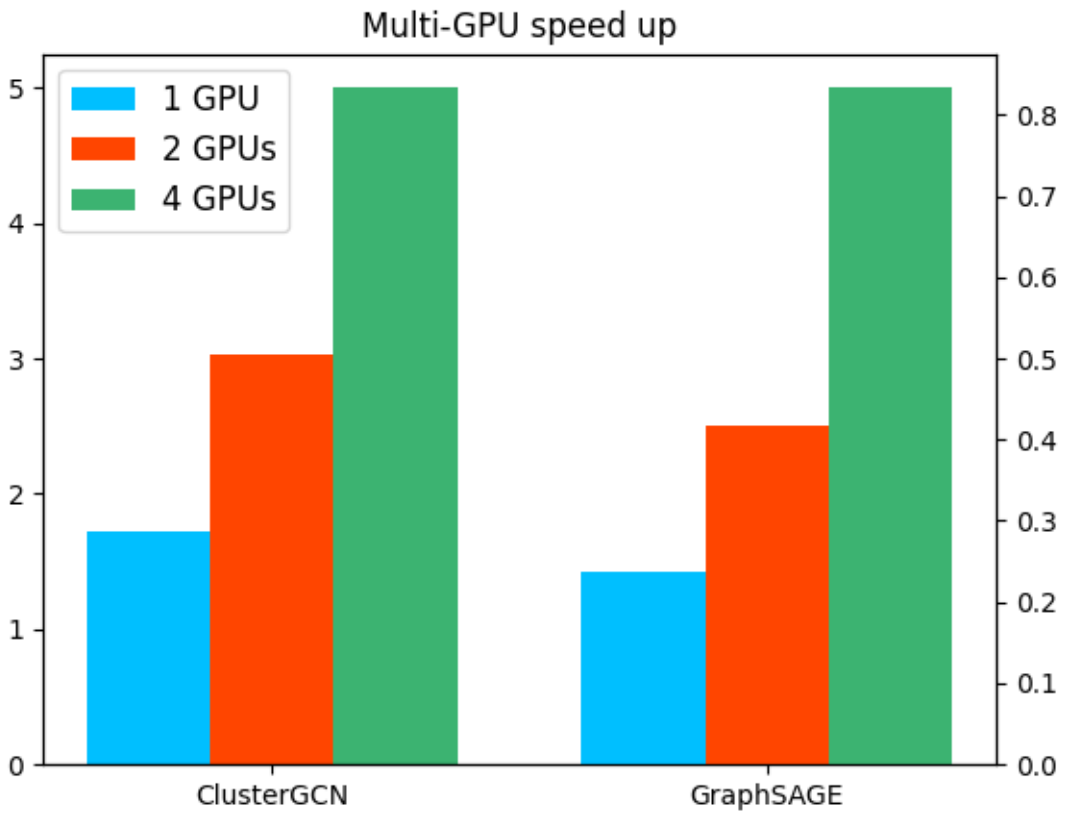


ClusterGCN
(KDD ’19)

1. Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In NeurIPS ’17.
2. Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In KDD ’19.
3. Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In ICLR ’20.

高效计算 - 多卡并行训练

- 常用大规模图训练采样算法并行实现
  - GraphSAGE、ClusterGCN等
- 4 GPUs ~ **3x**↑ 加速
- 指定trainer即可使用:
  - $clustergcn \rightarrow dist\_clustergcn$



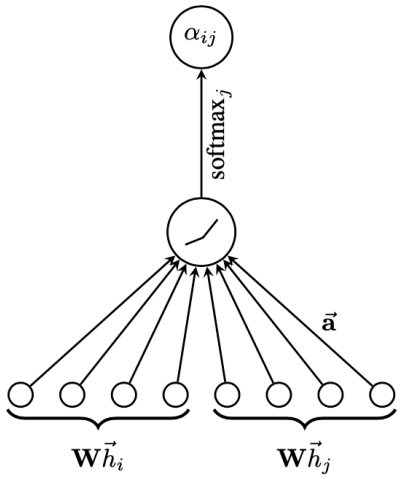
高效计算 - 图神经网络中的底层算子

- GCN（稀疏矩阵乘法 SpMM）

$$H^{(i+1)} = AH^{(i)}W$$

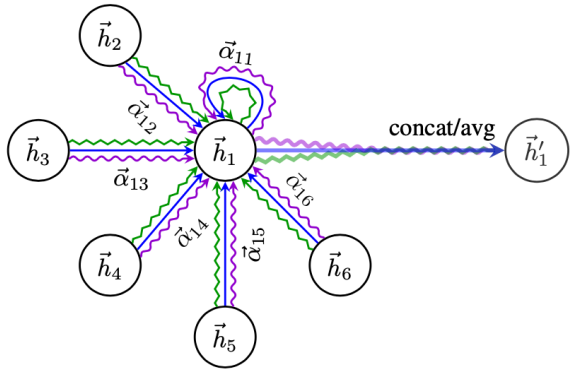
- GAT（ 注意力计算 Edge-wise-softmax）

$$a_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})}$$



- GAT（多头稀疏矩阵乘法 Multi-Head SpMM）

$$h_i = CONCAT\left(\sigma\left(\sum_{j \in N_i} a_{ij}^k \boldsymbol{W}^k h_j\right)\right)$$



高效计算 - CogDL中GCN/GAT模型的实现


```
# GCN Layer
# graph: cogdl.data.Graph
# x: node featurs
# weight: parameters

h = torch.mm(x, weight)
h = spmm(graph, h)
out = torch.relu(h)
```

```
# GAT Layer
# graph: cogdl.data.Graph
# h: node featurs
# h_score: importance score of edge

edge_attention = mul_edge_softmax(graph, edge_score)
h = mh_spmm(graph, edge_attention, h)
out = torch.cat(h, dim=1)
```

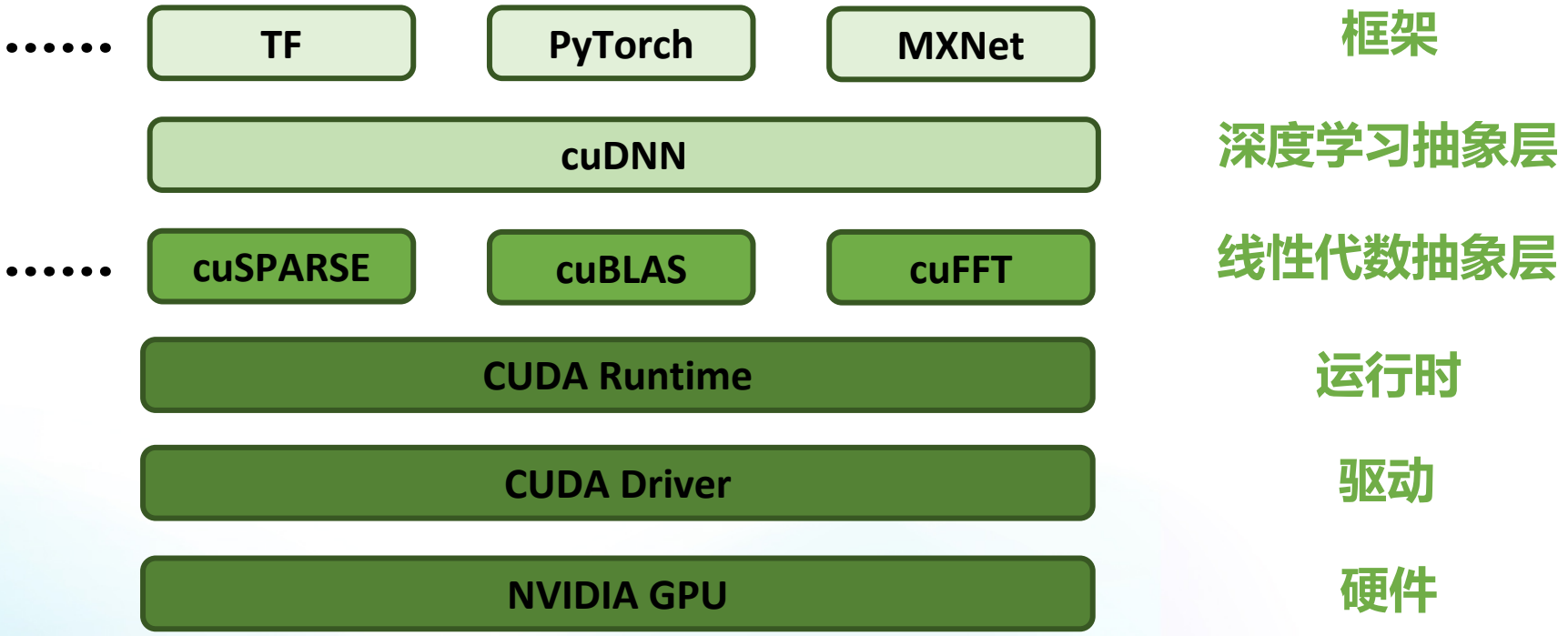
$$H^{(i+1)} = AH^{(i)}\boldsymbol{W}^{(\mathrm{i})}$$

$$a_{ij} = softmax(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in N_i}\exp(e_{ik})}$$

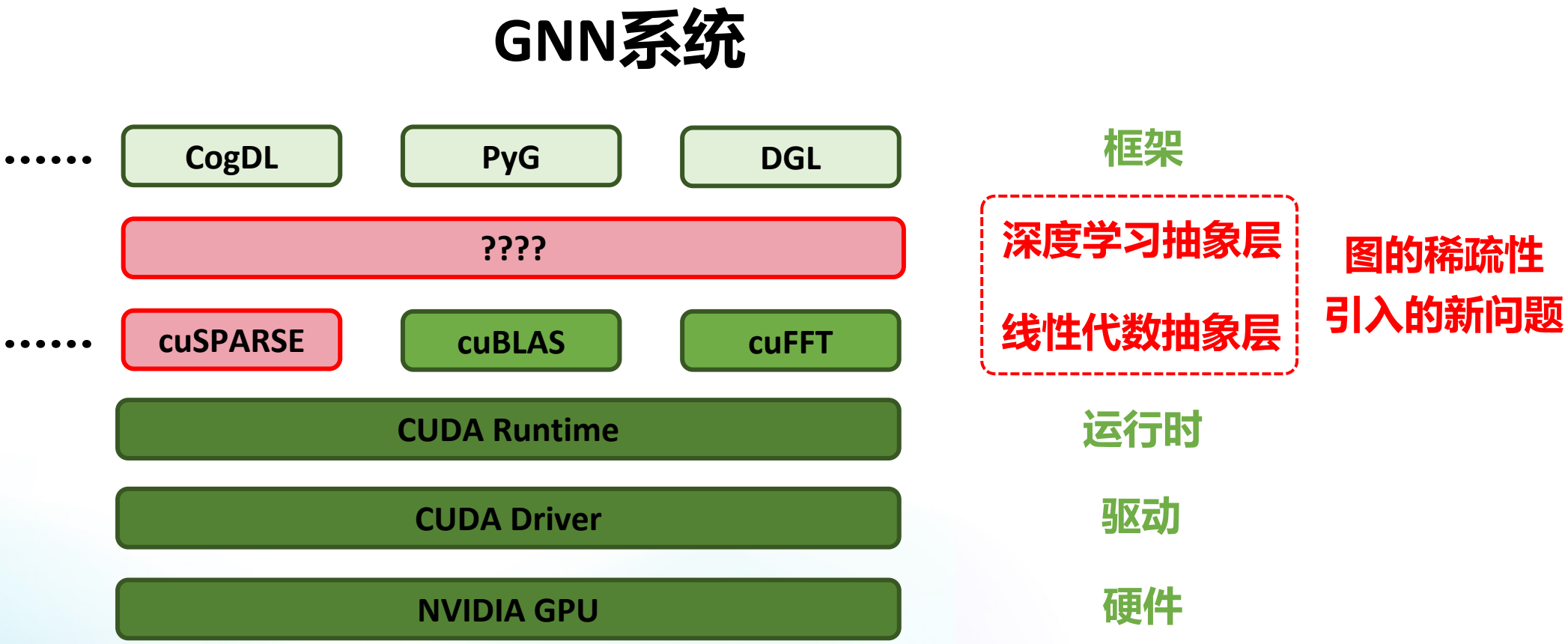
$$h_i = CONCAT\left(\sigma\left(\sum_{j\in N_i} a_{ij}^k \boldsymbol{W}^k h_j\right)\right)$$

高效计算 – 现有GNN系统的问题

传统DNN系统

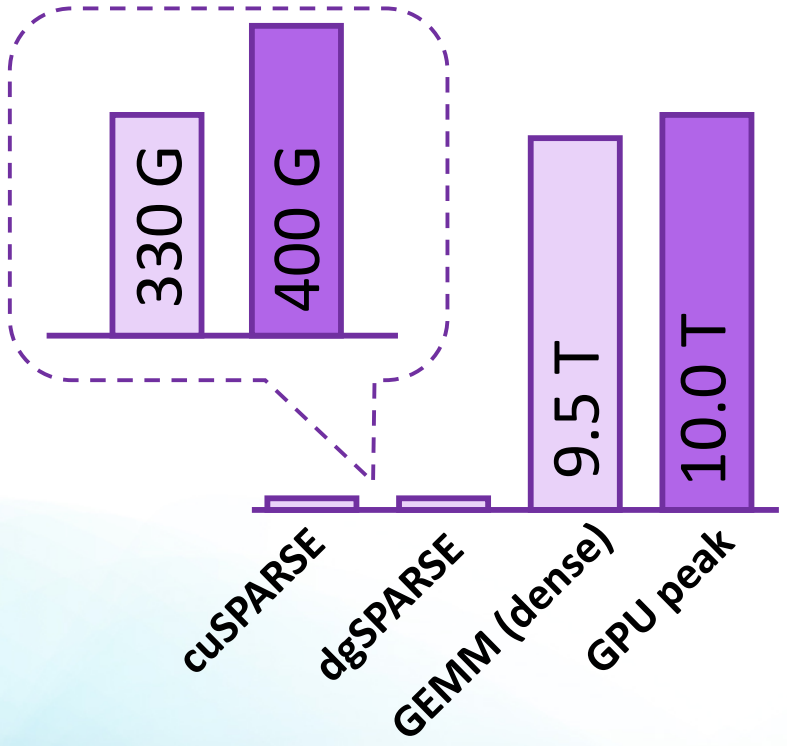


高效计算 – 现有GNN系统的问题

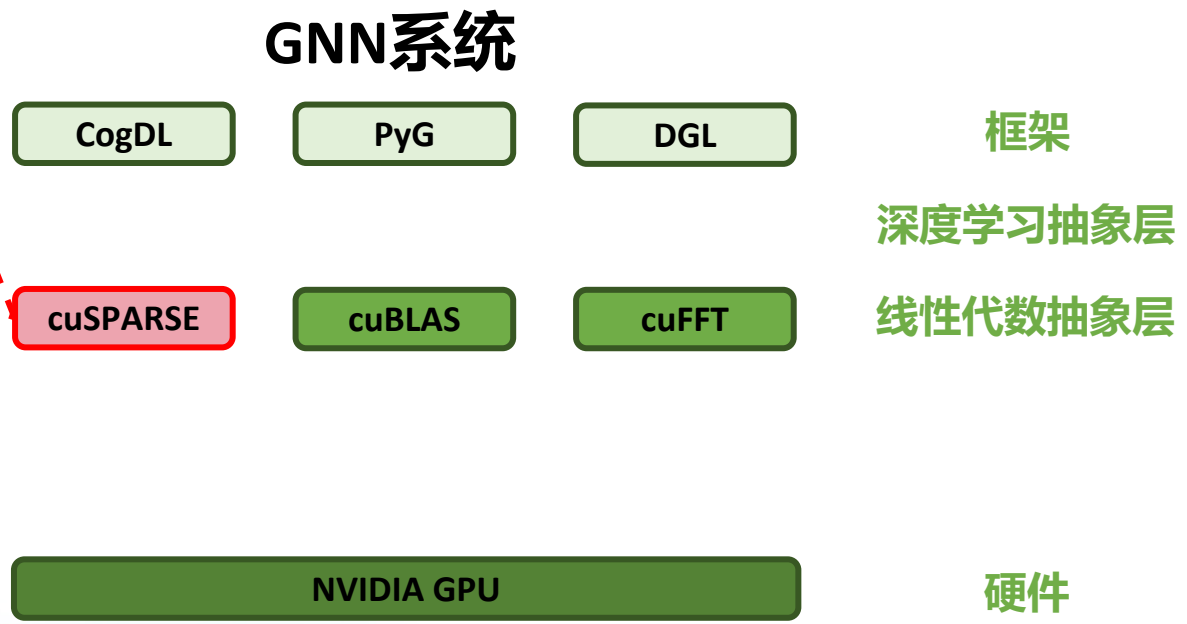


高效计算 – 现有GNN系统的问题

问题1：缺少高效稀疏算子



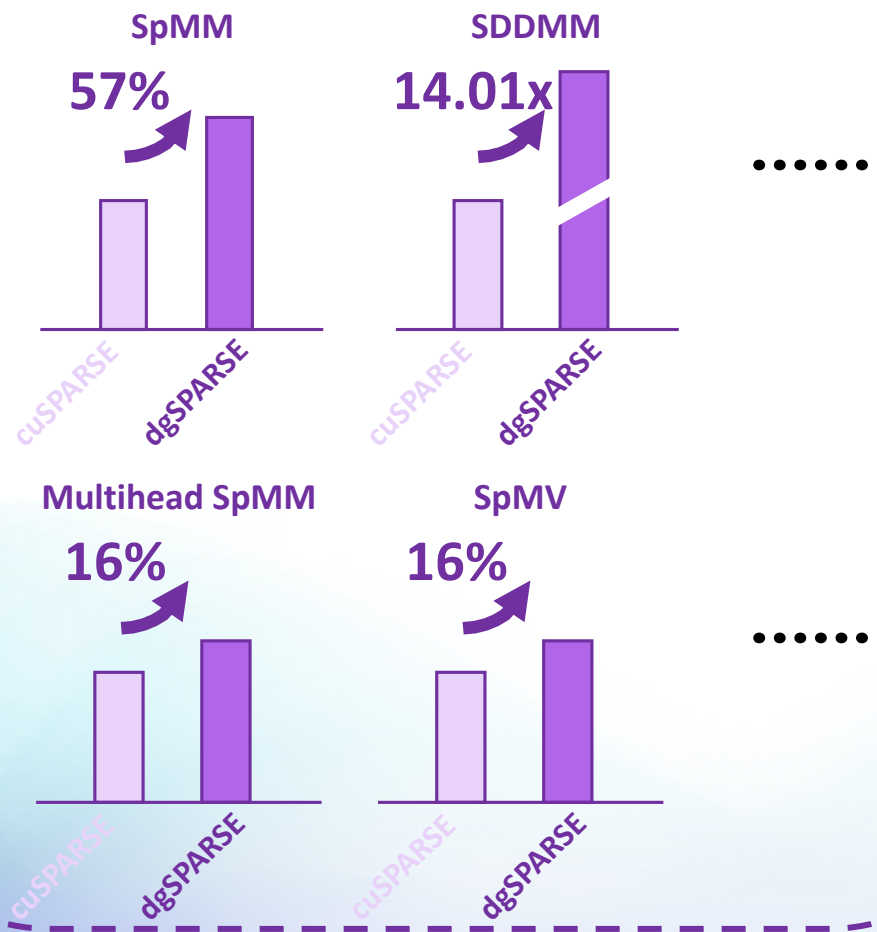
稀疏算力与GPU峰值算力存在巨大鸿沟
如cuSPARSE商用库性能存在优化空间

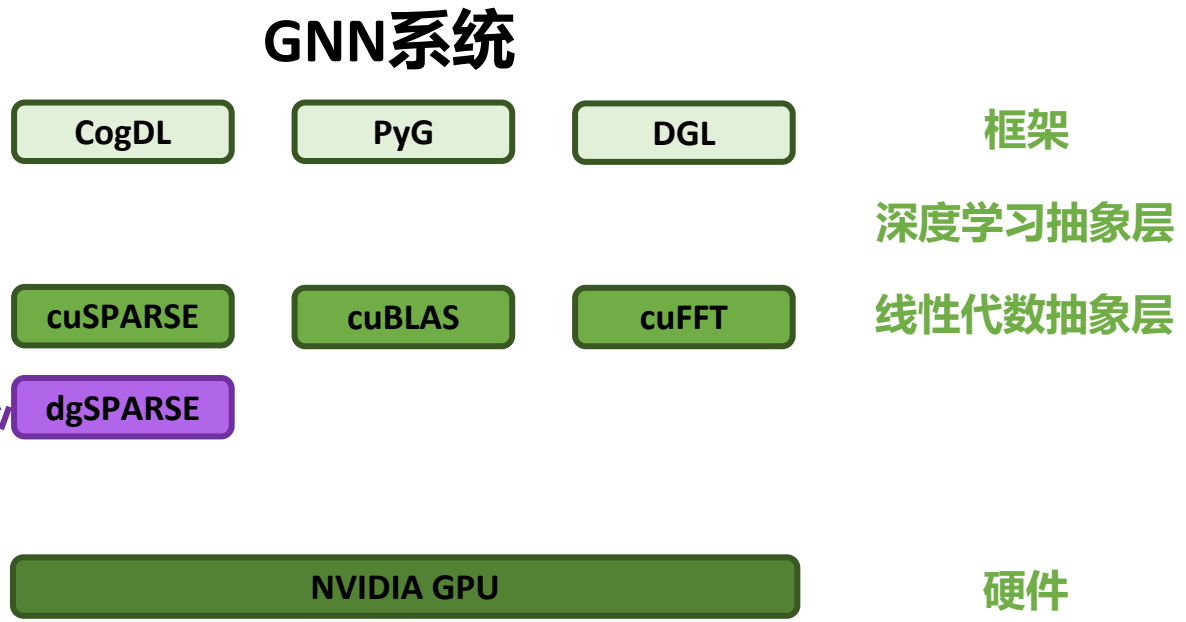


高效计算 – 面向稀疏图神经网络GPU加速库dgSPARSE

技术1：dgSPARSE, Deep Graph SPARSE

高效稀疏算子GPU实现





高效计算 – 现有GNN系统的问题

问题2：接口少不统一，难适应算法

cuSPARSE

```
// cuSPARSE
cusparseStatus_t
cusparseSpMM_bufferSize(cusparseHandle_t     handle,
                        cusparseOperation_t  opA,
                        cusparseOperation_t  opB,
                        const void*          alpha,
                        cusparseSpMatDescr_t matA,
                        cusparseDnMatDescr_t matB,
                        const void*          beta,
                        cusparseDnMatDescr_t matC,
                        cudaDataType         computeType,
                        cusparseSpMMAlg_t    alg,
                        size_t*              bufferSize)
```

- SpMM √
- SpMM-like ×

dgSPARSE

```
// GE-SpMM
void gespmm(int method,
            int tile_row,
            int A_nrows,
            int B_ncols,
            int* A_rowPtr,
            int* A_colInd,
            float *A_val,
            float *B_val,
            float C);
```

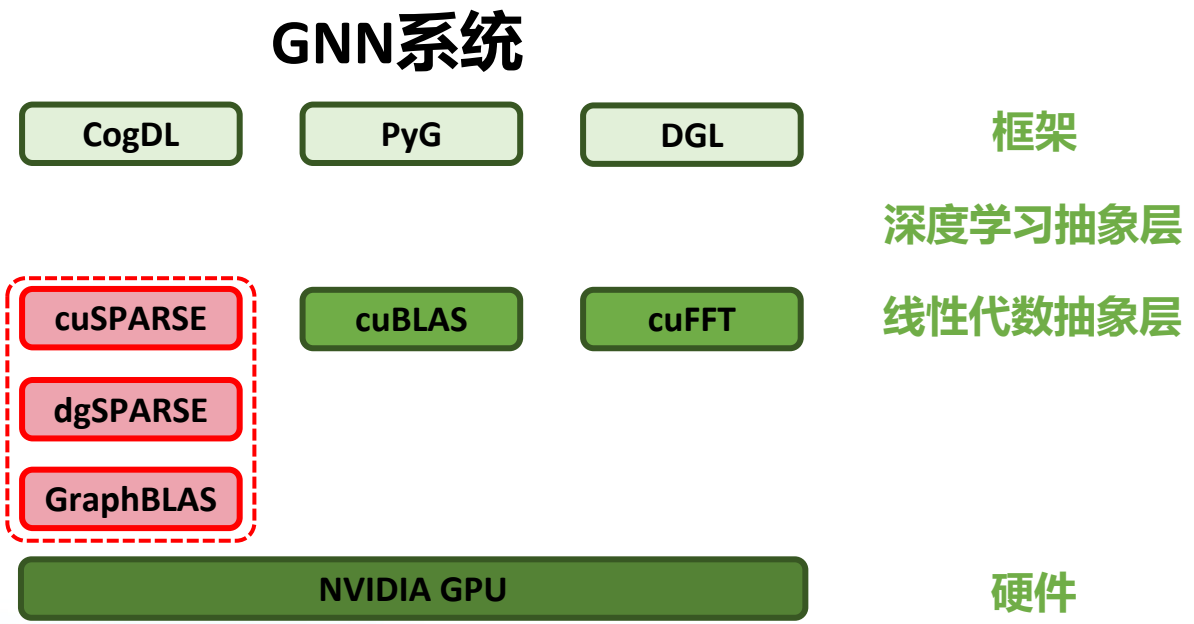
- SpMM √
- SpMM-like √

GraphBLAST

```
// GraphBLAST
Info spmm(  DenseMatrix<c>* C,
            const Matrix<m>* mask,
            const BinaryOpT* accum,
            const SemiringT* op,
            const SparseMatrix<a>* A,
            const DenseMatrix<b>* B,
            Descriptor* desc);
```

- SpMM √
- SpMM-like √

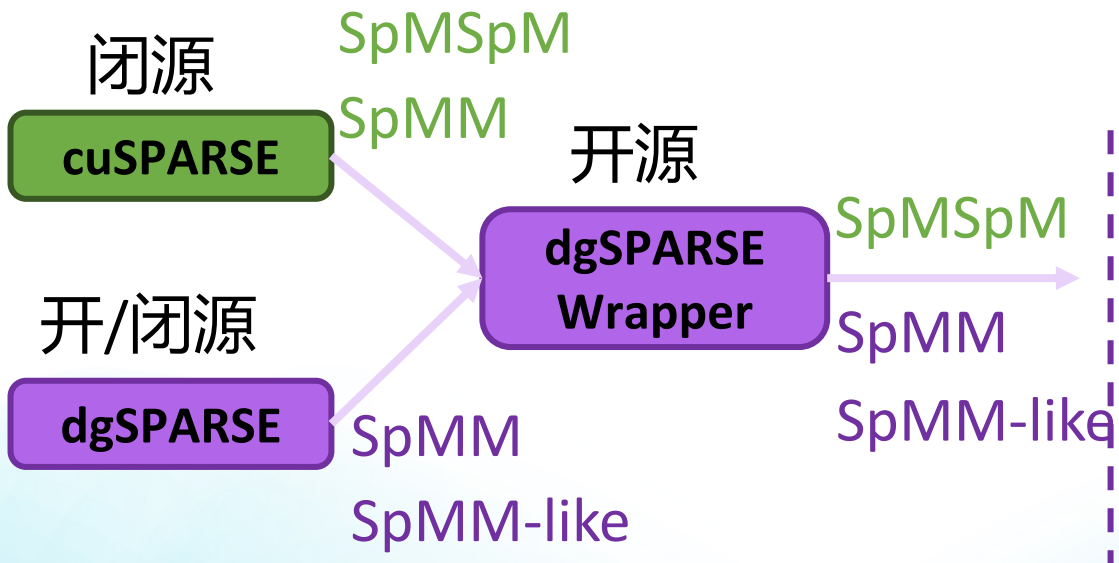
cuSPARSE提供了较少接口，其他稀疏算子加速工作接口各异，难以被框架使用

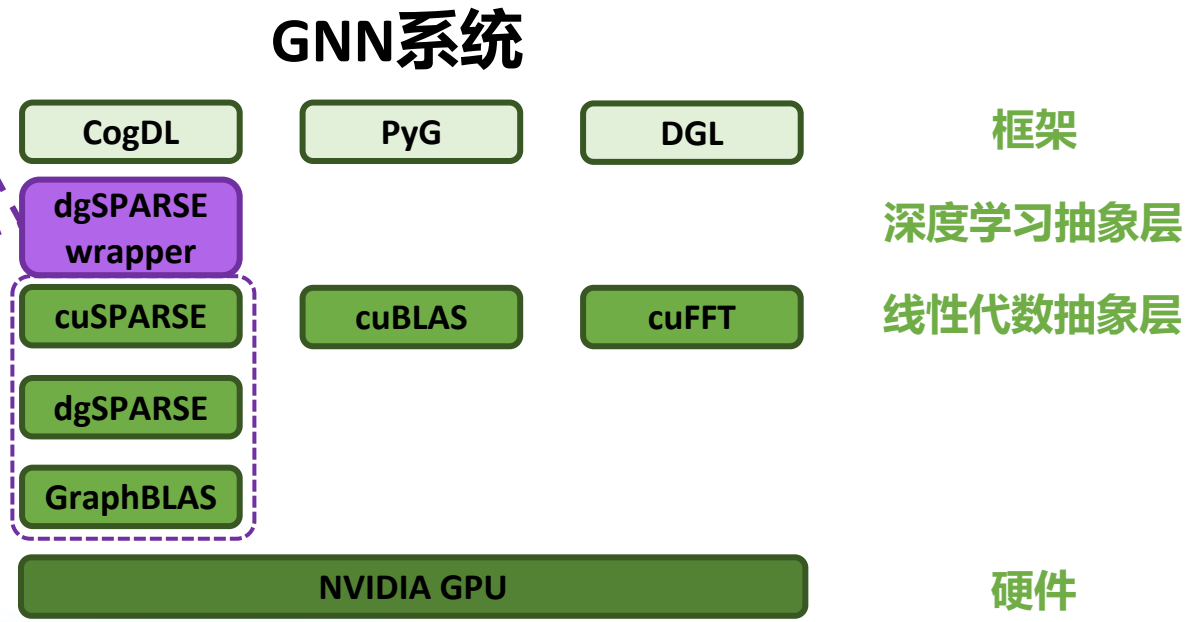


高效计算 – 面向稀疏图神经网络GPU加速库dgSPARSE

技术2：dgSPARSE wrapper

开源，兼容cuSPARSE且更多高效接口





高效计算 – 现有GNN系统的问题

问题3：无DL层抽象，开发效率低

CUDA code → Package → Framework call

```
__global__ void mhspmmSimple(
    int v, int nnz, int h, int f,
    int *rowptr, int *colind, float *attention /* E*H */,
    float *infeat /* V*H*F */,
    float *outfeat /* V*H*F */
)
{
  int rid = blockIdx.x;
  int hid = blockIdx.y;
  int lb = rowptr[rid];
  int hb = rowptr[(rid + 1)];
  float acc = 0;
  int offset1, offset2;
  for (int ptr = lb; ptr < hb; ptr++)
  {
    offset1 = colind[ptr] * f * h + hid * f + threadIdx.x;
    float att = attention[ptr * h + hid];
    acc = sum_reduce(acc, infeat[offset1] * att);
  }
  offset2 = rid * f * h + hid * f + threadIdx.x;
  outfeat[offset2] = acc;
}
```

```
torch::Tensor mhspmm_cuda(
    torch::Tensor rowptr,
    torch::Tensor colind,
    torch::Tensor attention,
    torch::Tensor infeat)
{
  const auto v = rowptr.size(0) - 1;
  const auto nnz = colind.size(0);
  const auto h = attention.size(1);
  const auto f = infeat.size(2);
  auto devid = infeat.device().index();
  auto options = torch::TensorOptions().\
    dtype(torch::kFloat32).device(torch::kCUDA, devid);
  auto outfeat = torch::empty({v, h, f}, options);
  mhspmmSimple<<<dim3(v, h, 1), dim3(f, 1, 1)>>>(v, nnz, h, f,
    rowptr.data_ptr<int>(), colind.data_ptr<int>(),
    attention.data_ptr<float>(), infeat.data_ptr<float>(),
    outfeat.data_ptr<float>());
  return outfeat;
}
```

```
def forward(self, graph, x):
    h = torch.matmul(x, self.W).view(-1, self.nhead, self.out_features)

    row, col = graph.edge_index
    # Self-attention on the nodes - Shared attention mechanism
    h_l = (self.a_l * h).sum(dim=-1)[row]
    h_r = (self.a_r * h).sum(dim=-1)[col]
    edge_attention = self.leakyrelu(h_l + h_r)
    # edge_attention: E * H
    edge_attention = mul_edge_softmax(graph, edge_attention)
    edge_attention = self.dropout(edge_attention)

    h_prime = mh_spmm(graph, edge_attention, h)
    out = h_prime.view(h_prime.shape[0], -1)

    out = torch.cat(h_prime, dim=1)

    return out
```

Python代码

```
def forward(self, graph, x):
    h = torch.matmul(x, self.W).view(-1, self.nhead, self.out_features)

    row, col = graph.edge_index
    # Self-attention on the nodes - Shared attention mechanism
    h_l = (self.a_l * h).sum(dim=-1)[row]
    h_r = (self.a_r * h).sum(dim=-1)[col]
    edge_attention = self.leakyrelu(h_l + h_r)
    # edge_attention: E * H
    edge_attention = mul_edge_softmax(graph, edge_attention)
    edge_attention = self.dropout(edge_attention)

    h_prime = mh_spmm(graph, edge_attention, h)
    out = h_prime.view(h_prime.shape[0], -1)

    out = torch.cat(h_prime, dim=1)

    return out
```

CUDA代码

```
__global__ void mhspmmSimple(
    int v, int nnz, int h, int f,
    int *rowptr, int *colind, float *attention /* E*H */,
    float *infeat /* V*H*F */,
    float *outfeat /* V*H*F */
)
{
  int rid = blockIdx.x;
  int hid = blockIdx.y;
  int lb = rowptr[rid];
  int hb = rowptr[(rid + 1)];
  float acc = 0;
  int offset1, offset2;
  for (int ptr = lb; ptr < hb; ptr++)
  {
    offset1 = colind[ptr] * f * h + hid * f + threadIdx.x;
    float att = attention[ptr * h + hid];
    acc = sum_reduce(acc, infeat[offset1] * att);
  }
  offset2 = rid * f * h + hid * f + threadIdx.x;
  outfeat[offset2] = acc;
}
```

框架开发者需写CUDA代码，并使用多个算子拼接成GNN中的一层，开发效率低

GNN系统

CogDL	PyG	DGL	框架
????			深度学习抽象层
cuSPARSE	cuBLAS	cuFFT	线性代数抽象层
dgSPARSE			
GraphBLAS			
NVIDIA GPU			硬件

高效计算 – 面向稀疏图神经网络GPU加速库dgSPARSE

技术3：dgNN库

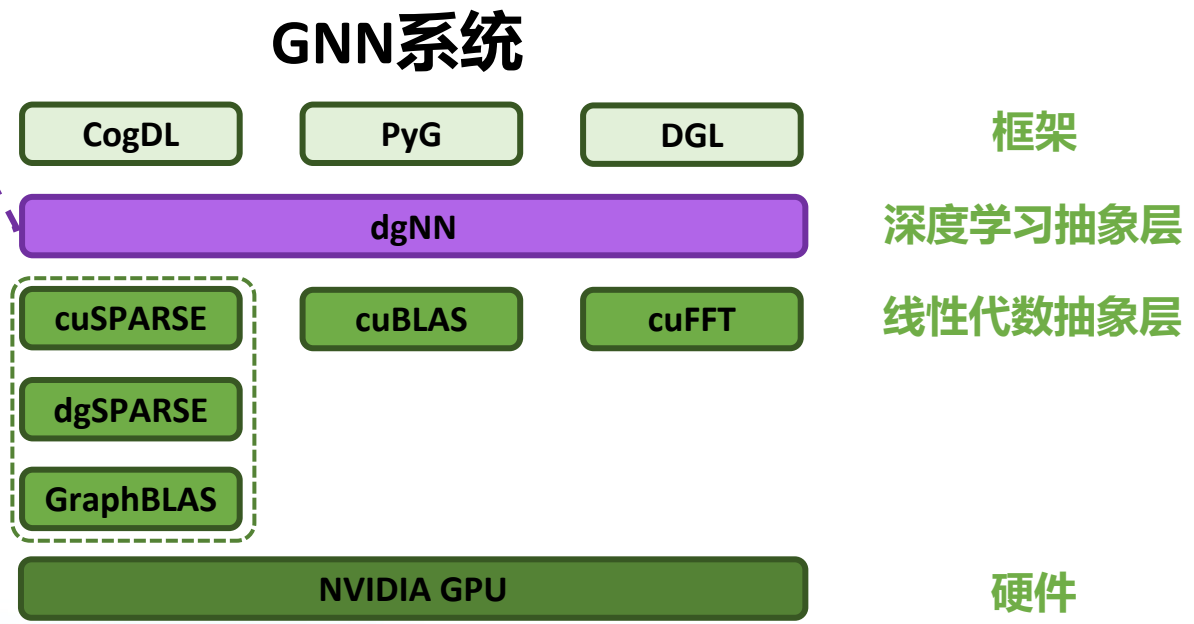
面向GNN的深度学习抽象层

```
# GAT Layer
# graph: cogdl.data.Graph
# h: node featurs
# h_score: importance score of edge

edge_attention = mul_edge_softmax(graph, edge_score)
h = mh_spmm(graph, edge_attention, h)
out = torch.cat(h, dim=1)
```

↓

```
dgNN.GATLayer.forward(graph, x, W, nhead, features)
```



高效计算 – 面向稀疏图神经网络GPU加速库dgSPARSE

图数据

高效并行

- 负载均衡划分
- 线程粗粒度化
- ……

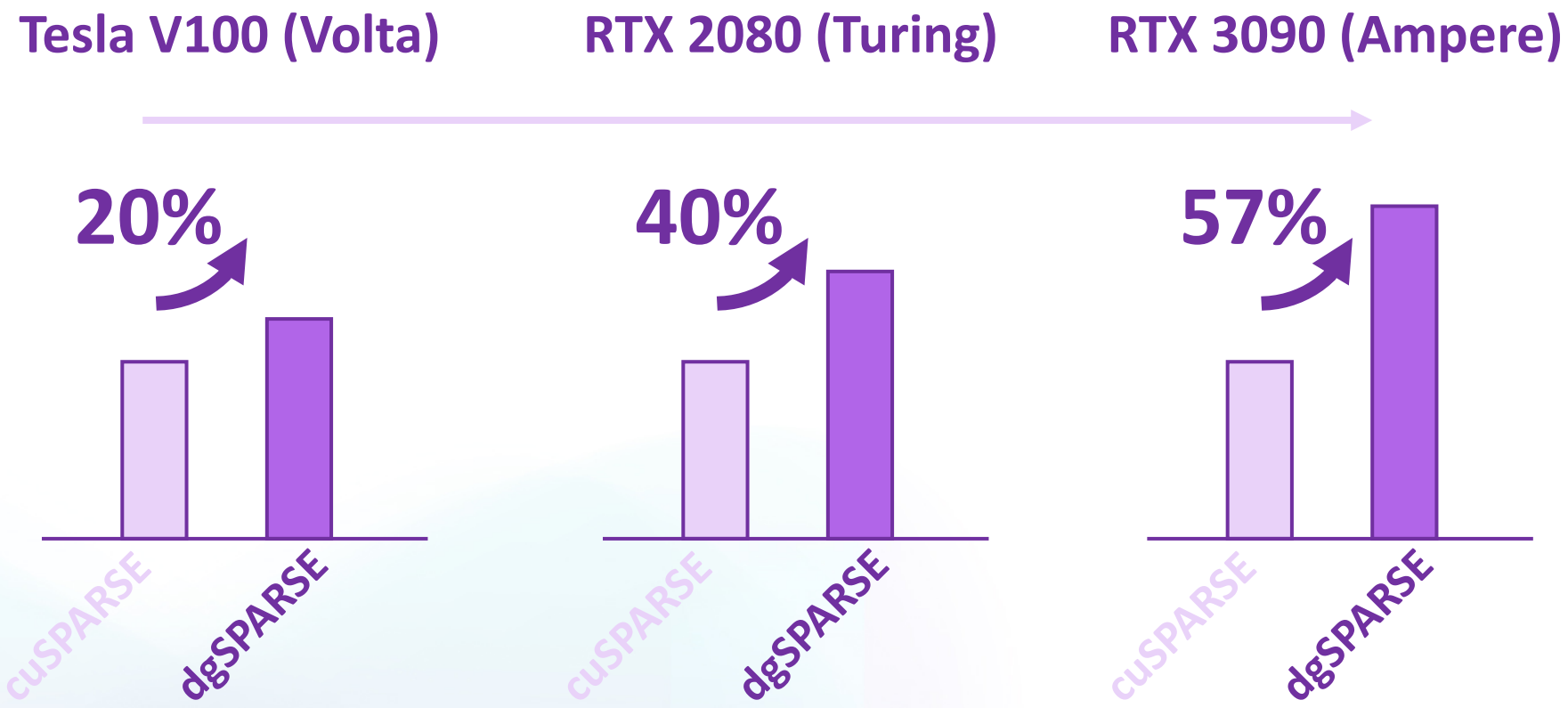
SIMT GPU架构

高效规约

- Warp内规约
- Warp间规约
- ……

图数据

简单但有效的设计思想，针对GPU的SIMT架构优化



CogDL与dgSPARSE深度配合

开发效率提升1个量级

```python
def forward(self, graph, x):
    h = torch.matmul(x, self.W).view(-1, self.nhead, self.out_features)

    row, col = graph.edge_index
    # Self-attention on the nodes - Shared attention mechanism
    h_l = (self.a_l * h).sum(dim=-1)[row]
    h_r = (self.a_r * h).sum(dim=-1)[col]
    edge_attention = self.leakyrelu(h_l + h_r)
    # edge_attention: E * H
    edge_attention = mul_edge_softmax(graph, edge_attention)
    edge_attention = self.dropout(edge_attention)

    h_prime = mh_spmm(graph, edge_attention, h)
    out = h_prime.view(h_prime.shape[0], -1)

    out = torch.cat(h_prime, dim=1)

    return out
```

Python代码

```python
def forward(self, graph, x):
    h = torch.matmul(x, self.W).view(-1, self.nhead, self.out_features)

    row, col = graph.edge_index
    # Self-attention on the nodes - Shared attention mechanism
    h_l = (self.a_l * h).sum(dim=-1)[row]
    h_r = (self.a_r * h).sum(dim=-1)[col]
    edge_attention = self.leakyrelu(h_l + h_r)
    # edge_attention: E * H
    edge_attention = mul_edge_softmax(graph, edge_attention)
    edge_attention = self.dropout(edge_attention)

    h_prime = mh_spmm(graph, edge_attention, h)
    out = h_prime.view(h_prime.shape[0], -1)

    out = torch.cat(h_prime, dim=1)

    return out
```

Python代码

```python
dgNN.GATLayer.forward(graph, x, W, nhead, features)
```

简单的Python代码

阶段三：使用dgNN

```cpp
__global__ void mhspmmSimple(
    int v, int nnz, int h, int f,
    int *rowptr, int *colind, float *attention /* E*H */,
    float *infeat /* V*H*F */,
    float *outfeat /* V*H*F */
)
{
  int rid = blockIdx.x;
  int hid = blockIdx.y;
  int lb = rowptr[rid];
  int hb = rowptr[(rid + 1)];
  float acc = 0;
  int offset1, offset2;
  for (int ptr = lb; ptr < hb; ptr++)
  {
    offset1 = colind[ptr] * f * h + hid * f + threadIdx.x;
    float att = attention[ptr * h + hid];
    acc = sum_reduce(acc, infeat[offset1] * att);
  }
  offset2 = rid * f * h + hid * f + threadIdx.x;
  outfeat[offset2] = acc;
}
```

CUDA代码

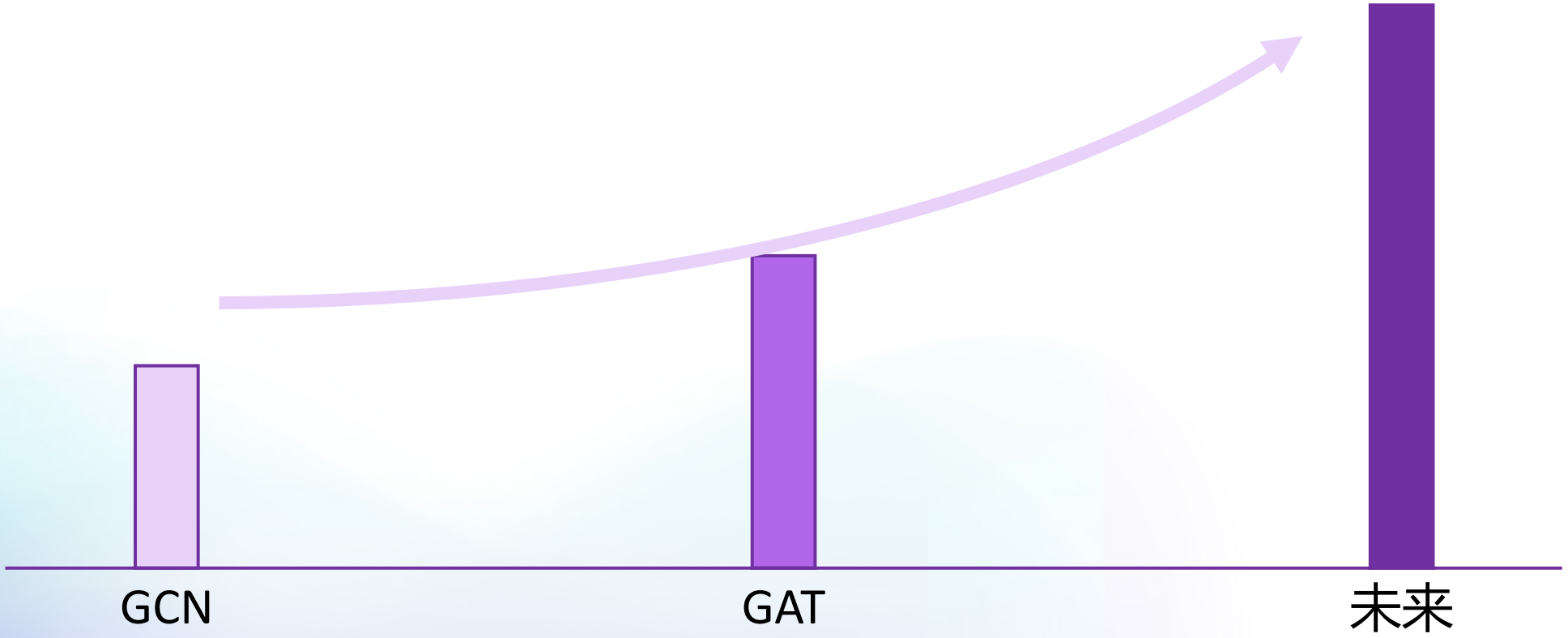
阶段二：使用dgSPARSE

阶段一：不使用dgSPARSE

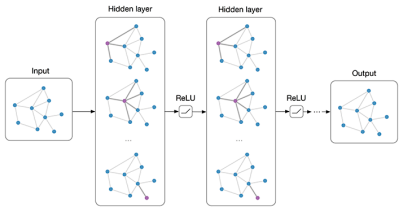
CogDL与dgSPARSE深度配合

模型速度提升1个量级

使用1个dgSPARSE算子	使用多个dgSPARSE算子	使用dgNN
3~5x	**4~6x**	**10x**



CogDL与dgSPARSE深度配合

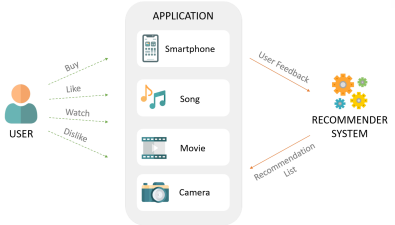


GNN研究者

- 易用实验接口
- 快速验证想法

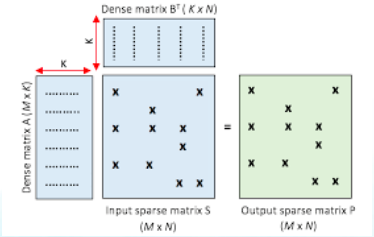
CogDL

应用开发者



- 易用应用接口
- 快速应用部署

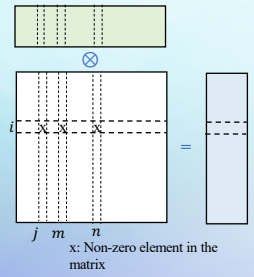
- 更快计算速度
- 更高开发效率



- 统一算子接口
- 快速框架部署

dgSPARSE

- 统一稀疏接口
- 硬件稀疏支持



算子开发者

硬件开发者

dgSPARSE开发进展

- 框架
- 算子
- 硬件

2020.11

- GE-SpMM paper on SC20

2021.06

- CogDL with dgSPARSE
- SDDMM paper to ICCD21

2021.07

- dgSPARSE wrapper release
- Cooperation with hardware developers

2021.08

- Fast GAT to ASPLOS22
- SpMMul paper to ASPLOS22

2021.10

- dgNN paper to MLSys22
- Wrapper with other SPARSE library

2021.12

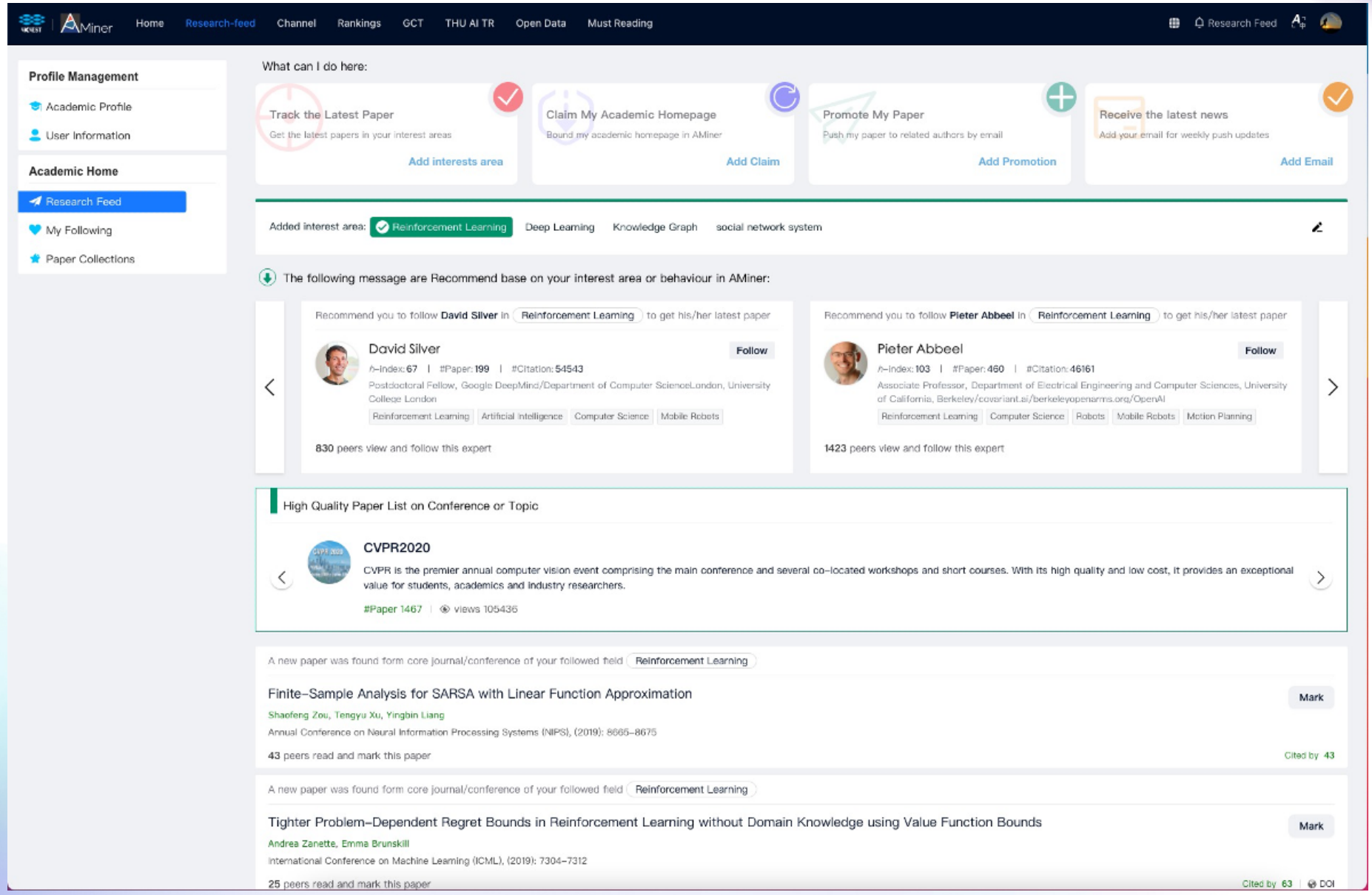
- More frameworks with dgSPARSE

2022~

- CUDA plugin
- GPU support for sparse

感谢dgSPARSE的开发者：清华大学电子工程系NICS实验室，戴国浩、黄古玥、于仲明、张恒瑞、杨上、汪玉

下游应用 - AMiner订阅



总结

- 背景介绍
  - 网络化的数据
  - 图表示学习、图神经网络
- CogDL简介
- CogDL特性：
  - 易用接口（“一行即训练”，自定义模型/数据集）
  - 可复现性（完全可复现的各类排行榜）
  - 高效计算（大规模图训练、 基于dgSPARSE的稀疏加速）
- CogDL应用

CogDL下一步计划

2021年8月 - CogDL v0.5

- 更进一步的性能优化（与dgSPARSE深度配合）
- 完善的图自监督学习与预训练

同时欢迎大家关注我们的KDD 2021 tutorial（https://kdd2021graph.github.io/）

Graph Representation Learning: Foundations, Methods, Applications and Systems

- Graph theory and Graph Fourier Analysis
- Foundations of Graph Neural Networks
- Network embedding theories and systems
- **Scalable Graph Neural Networks**
- **CogDL Toolkit for Graph Neural Networks**
- Heterogeneous Graph Neural Networks

欢迎大家成为 CogDL的贡献者！

谢谢!

感谢所有的合作者！

- 侯振宇、王岩、陈齐斌、罗弈桢、姚星丞、曾奥涵、郭世广、唐杰（清华KEG实验室）
- 戴国浩，黄古玥，于仲明，张恒瑞，杨上，汪玉（清华NICS实验室）
- 周畅、杨红霞（阿里巴巴）
- 张鹏（智谱.AI）

https://github.com/THUDM/cogdl