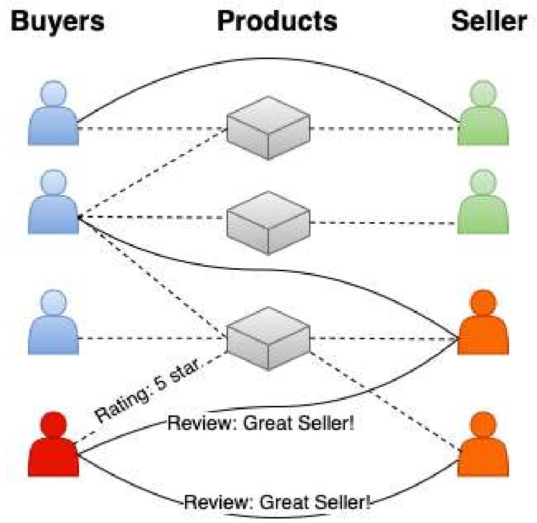
The Nature of Graph Neural Network Workloads

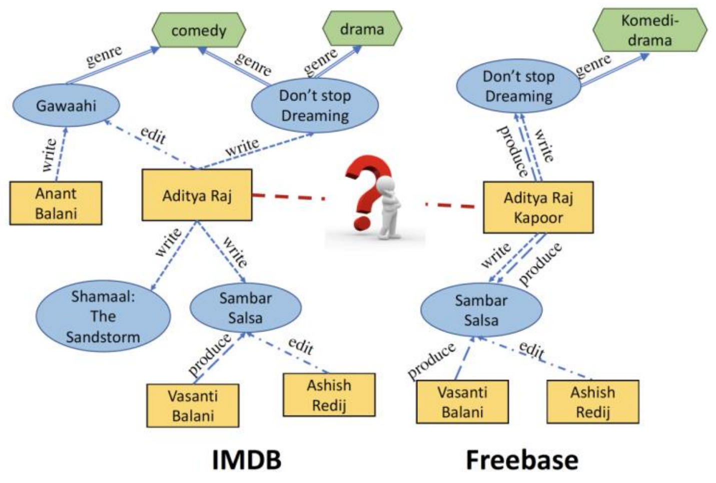
Graphs & applications

- Large: up to billions of nodes
- Many graphs are Heterogeneous
- Rich node/edge attributes

Social networks



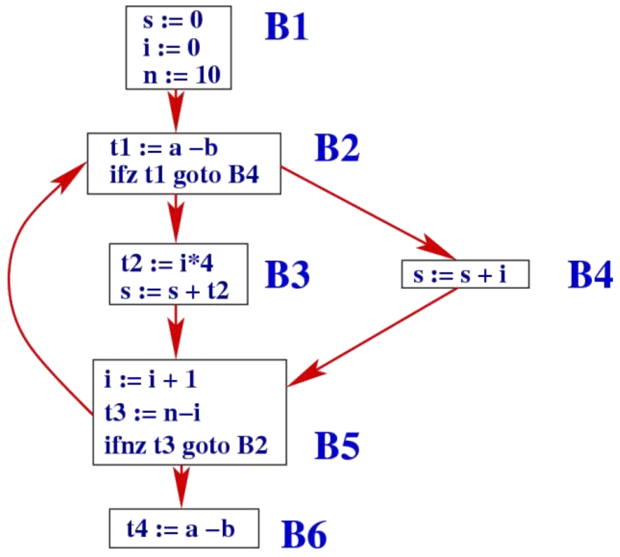
Amazon graph



Knowledge graphs

Graphs & applications

- Many small graphs



```
s := 0
i := 0
n := 10
```

```
t1 := a –b
ifz t1 goto B4
```

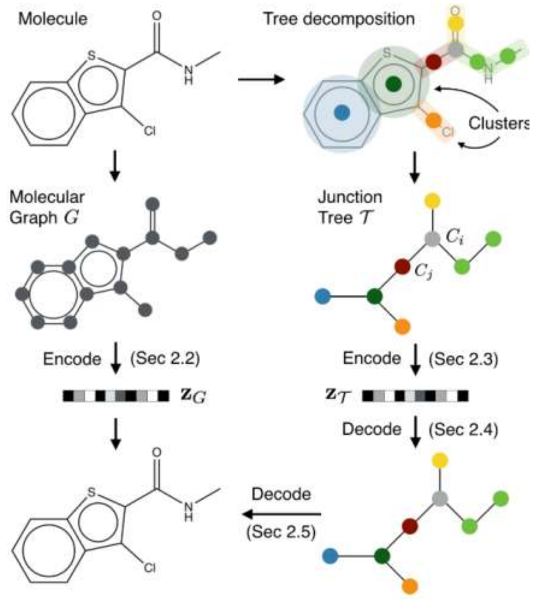
```
t2 := i*4
s := s + t2
```

```
s := s + i
```

```
i := i + 1
t3 := n–i
ifnz t3 goto B2
```

```
t4 := a –b
```

Code graph

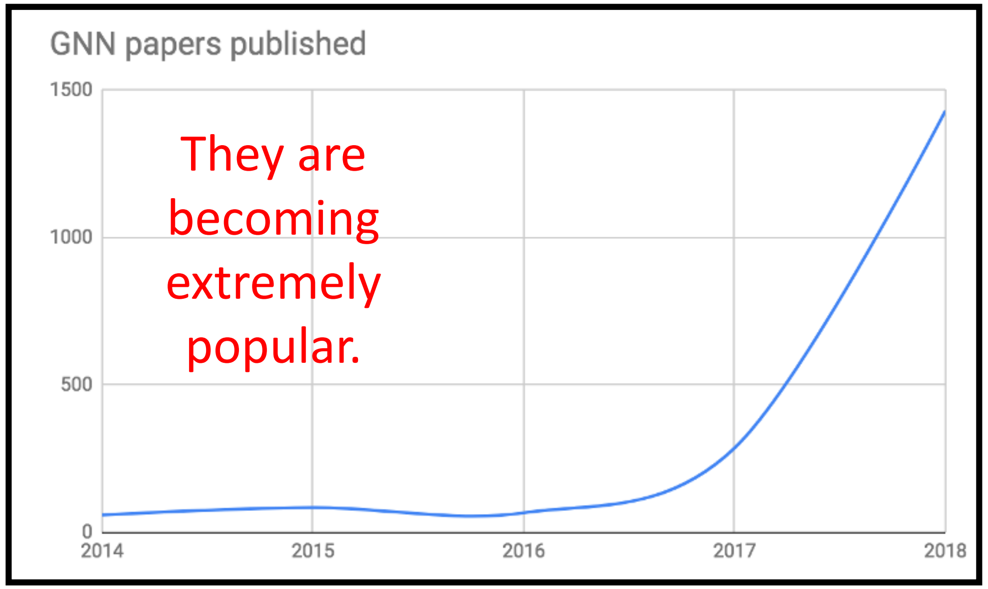


Chemistry compounds

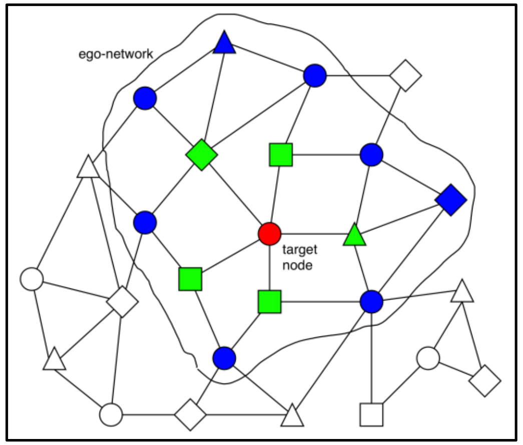
One graph has 100-1000 nodes, but there are many graphs.

Graph Neural Network

A family of (deep) neural networks that learn node, edge, and graph embeddings.



How do GNNs work?



An ego-network around each node is used to learn an embedding that captures task-specific information.

The embeddings use both the structure of the graph and the features of the nodes and edges.

The embeddings are learned in an end-to-end fashion; thus, the predictions are a function of the target node's ego-network.

A general graph neural network formalism

Graph neural networks are based on ***message-passing***

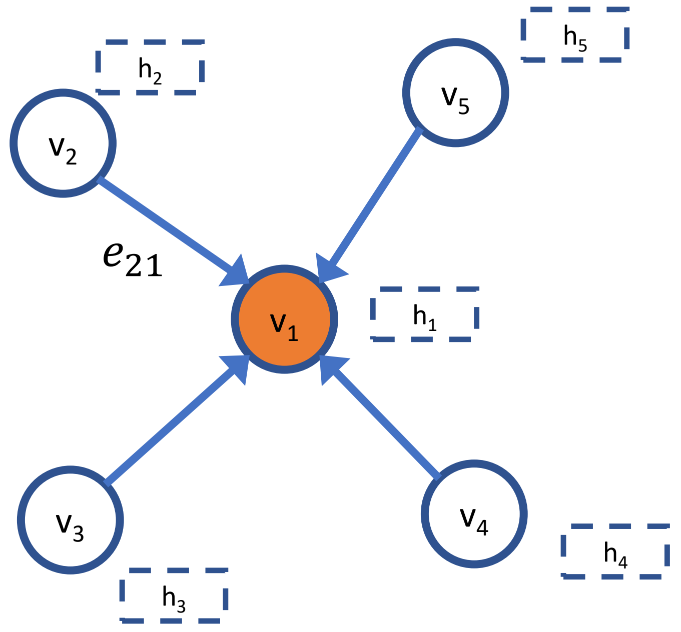
Message function

$$\text{Edge-wise: } \mathbf{m}_e^{(t+1)} = \phi\left(\mathbf{x}_v^{(t)}, \mathbf{x}_u^{(t)}, \mathbf{w}_e^{(t)}\right), (u, e, v) \in \mathcal{E}.$$

$$\text{Node-wise: } \mathbf{x}_v^{(t+1)} = \psi\left(\mathbf{x}_v^{(t)}, \rho\left(\left\{\mathbf{m}_e^{(t+1)} : (u, e, v) \in \mathcal{E}\right\}\right)\right)$$

Node update

aggregation

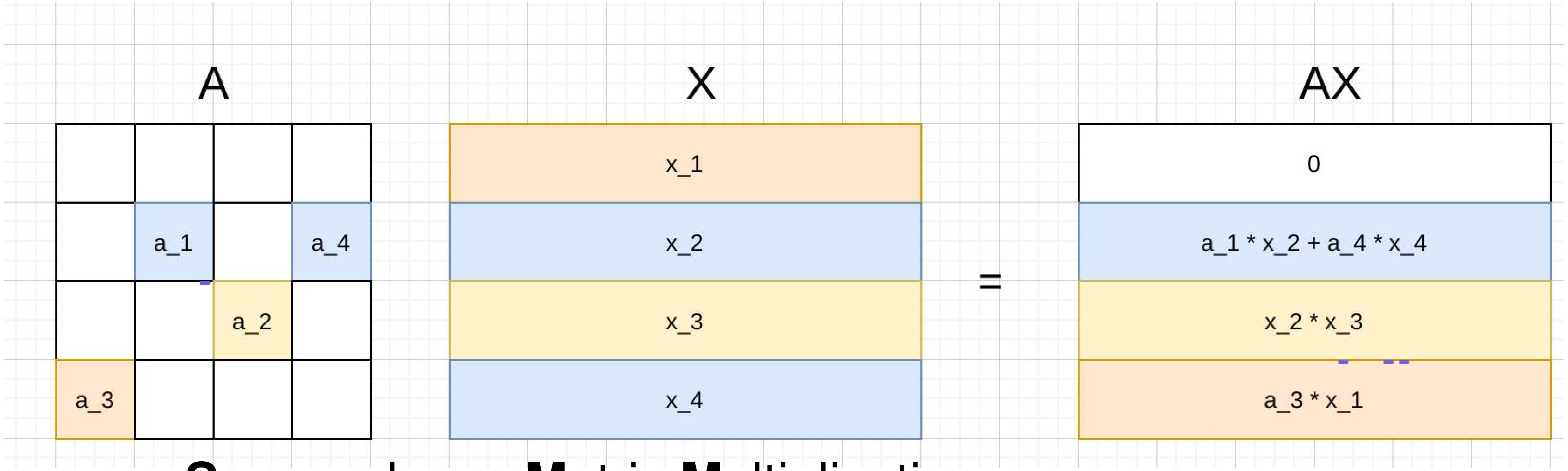


Message passing can be expressed with two sparse operators: SpMM and SDDMM.

[Neural Message Passing for Quantum Chemistry]

SpMM

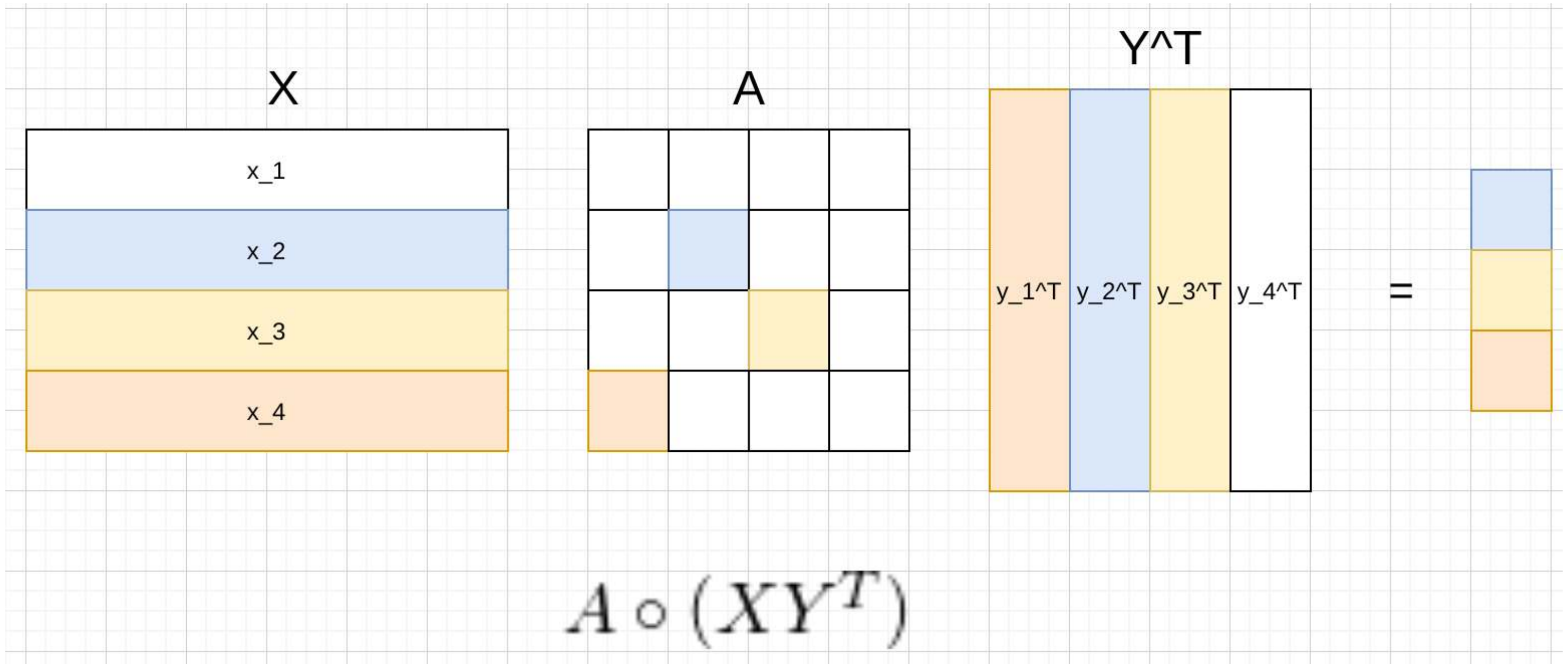
- Edge-wise: $m_e^{(t+1)} = x_u^{(t)}$
- Node-wise: $x_v^{(t+1)} = \sum_{(u,e,v)\in\mathcal{E}} m_e^{(t+1)}$



Sparse-dense **M**atrix **M**ultiplication

SDDMM

- Edge-wise: $m_e^{(t+1)} = x_u^{(t)} {x_v^{(t)}}^T, (u, e, v) \in \mathcal{E}$



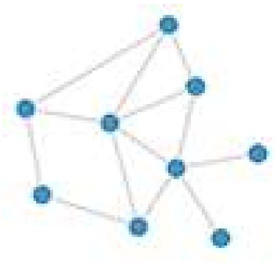
Sampled **D**ense-**D**ense **M**atrix **M**ultiplication

Generalized SpMM and SDDMM

- g-SpMM
  - Edge-wise: $m_e^{(t+1)} = \phi\left(x_u^{(t)}, x_v^{(t)}, w_e^{(t)}\right)$
  - Node-wise: $x_v^{(t+1)} = \rho(\{m_e^{(t+1)}: (u, e, v) \in \mathcal{E}\})$
- g-SDDMM
  - Edge-wise:$m_e^{(t+1)} = \phi\left(x_u^{(t)}, x_v^{(t)}, w_e^{(t)}\right), (u, e, v) \in \mathcal{E}$

Model 1: GraphSage

Handles graphs with one node type and one edge type.



$$M_{vw}^{(l)} = \frac{h_w^{(l-1)}}{d_v+1}$$

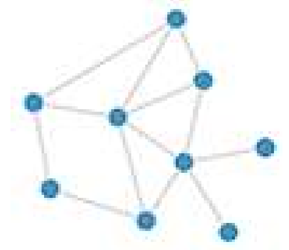
$$m_v^{(l)} = \sum_{w \in N(v) \cup \{v\}} M_{vw}^{(l)}$$

$$h_v^{(l)} = \phi(m_v^{(l)} W1^{(l)} + h_v^{(l-1)} W2^{(l)})$$

```
M = **SpMM**(A, H)/deg(A)
H = ReLU(matmul(M, W1) + b1 +
            matmul(H, W2) + b2)
H = Dropout(H)
```

Model 2: Graph attention networks (GAT)

Handles graphs with one node type and one edge type.



$$\alpha_{vw} = \frac{\exp(\text{LeakyReLU}(\vec{a}^T[W\vec{h}_v || W\vec{h}_w]))}{\sum_{k \in N_v} \exp(\text{LeakyReLU}(\vec{a}^T[W\vec{h}_v || W\vec{h}_k]))}$$

$$M_{vw}^{(l)} = \alpha_{vw} h_w^{(l-1)}$$

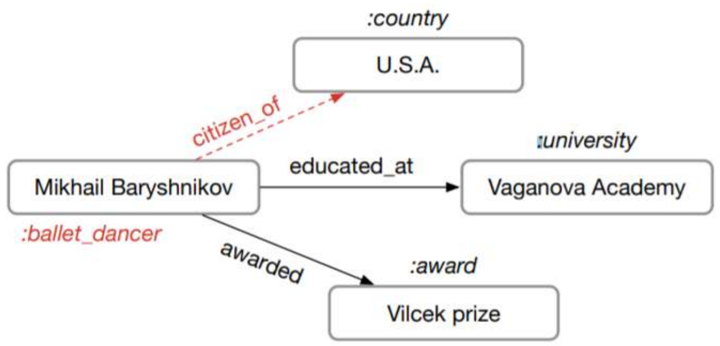
$$m_v^{(l)} = \sum_{w \in N(v) \cup \{v\}} M_{vw}^{(l)}$$

$$h_v^{(l)} = \phi(m_v^{(l)})$$

H = matmul(W, H)
El = matmul(Wl_a, H)
Er = matmul(Wr_a, H)
E = LeakyReLU(**SDDMM**(El, Er, A))
E = edge_softmax(E)
M = **SpMM**(E, H)
H = ReLU(M, W)

Model 3: Relational graph convolution networks (RGCN)

Handles graphs whose nodes are connected with different relations.



$$M_{vw}^{(l)} = \frac{1}{c_{v,r}} W_r^{(l)} h_w^{(l-1)}, \ r \text{ is the relation of } e_{vw}.$$

$$m_v^{(l)} = \sum_{w \in N(v) \cup \{v\}} M_{vw}^{(l)}$$

$$h_v^{(l)} = \sigma(m_v^{(l)})$$

```
M = []
# Sort the edges based on the edge type
src_id, dst_id, etype = g.edges()
etype, idx = sort(etype)
src_id, dst_id = src_id[idx], dst_id[idx]
H = H[src_id]
# Perform per-etype matrix multiplication
H_list = split(H, etypes)
for r in range(num_relations):
   M.append(matmul(H_list[r], W[r])/deg)
M = cat(M)
# Perform the final aggregation
A' = coo_matrix((range(num_edges), dst_id))
M = SpMM(A', M)
H = ReLU(M)
```

GNN benchmark

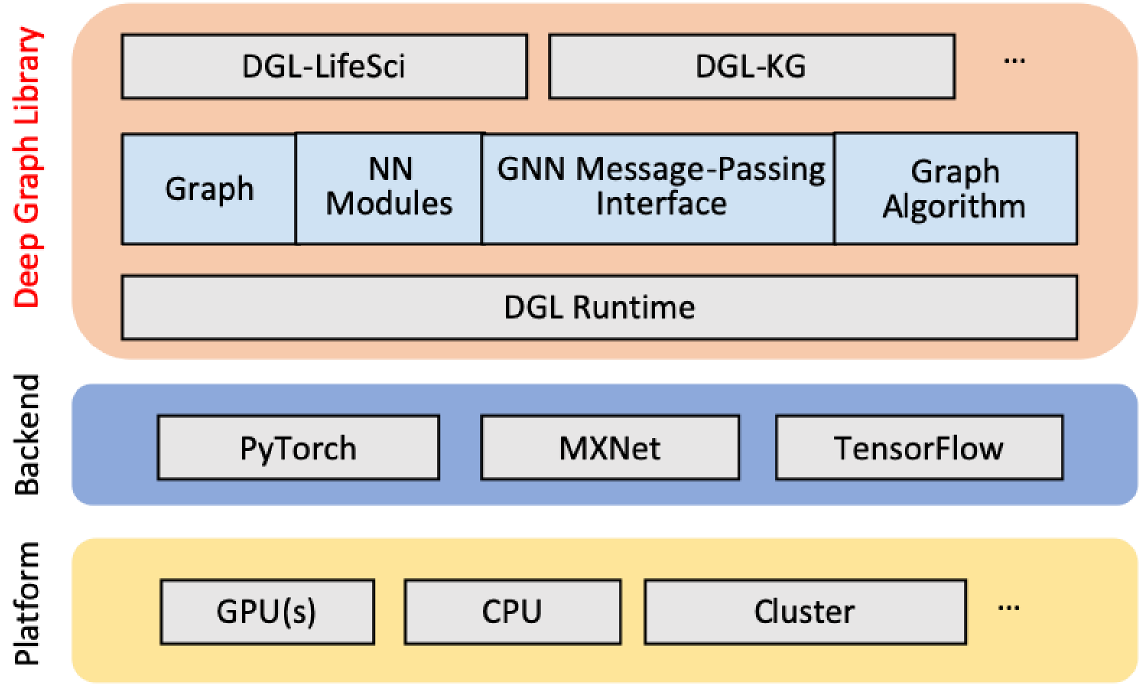
- Graph types:
  - Homogeneous
  - Heterogeneous
- Graph size:
  - Small
  - Large
- Training methods:
  - Full batch training
  - Mini-batch training

	Graph type	Graph Size	Training method
OGBN-arxiv	Homogeneous	|V|=169K, |E|=1M	Full graph
OGBN-products	Homogeneous	|V|=2.4M, |E|=62M	Mini-batch
MUTAG	Heterogeneous	|V|=27K, |E|=148K, |ETYPE|=50	Full graph
OGBN-MAG	Heterogeneous	|V|=1.9M, |E|=21M, |ETYPE|=8	Mini-batch

Benchmark framework: Deep Graph Library (DGL)

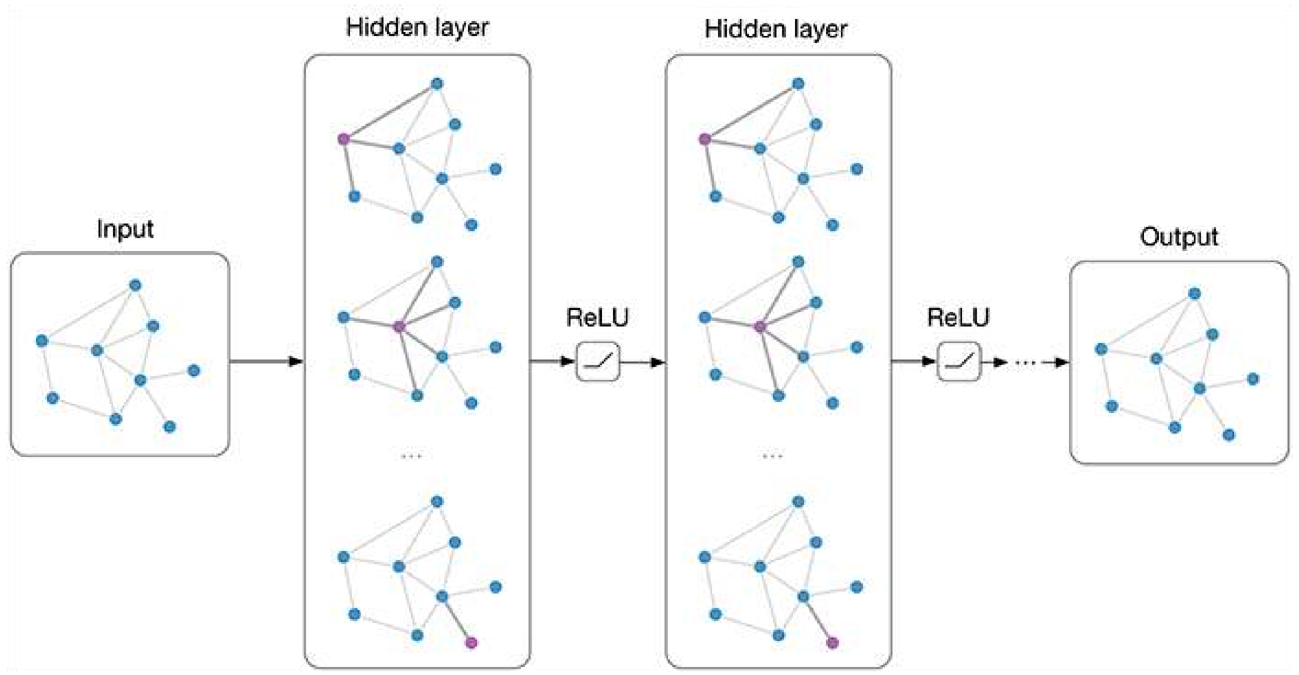
Benchmark framework: Deep Graph Library (DGL)

- DGL provides sparse operators:
  - SpMM, SDDMM
  - Neighbor sampling for mini-batch training.
- Deep learning framework provides dense operators:
  - Matrix multiplication, element-wise operations, reduction, etc
- The benchmark uses DGL + Pytorch.



Full-graph training on small graphs

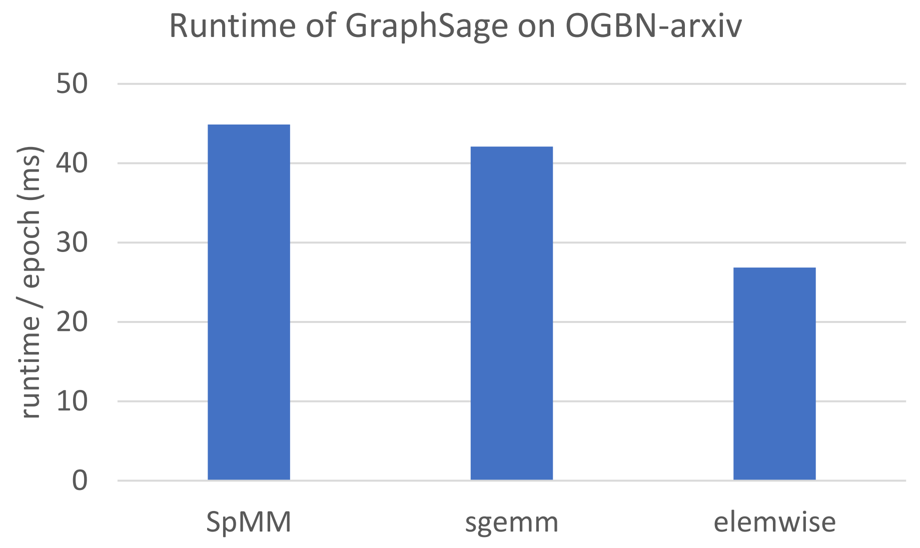
- Apply multiple layers of graph neural networks.



GraphSage full-batch training

Setup	
Dataset	OGBN-arxiv
#layers	3
Hidden dimensions	256
Hardware	Nvidia T4
Model size	217K

```
M = SpMM(A, H)/deg(A)
H = ReLU(matmul(M, W1) + b1 +
            matmul(H, W2) + b2)
H = Dropout(H)
```

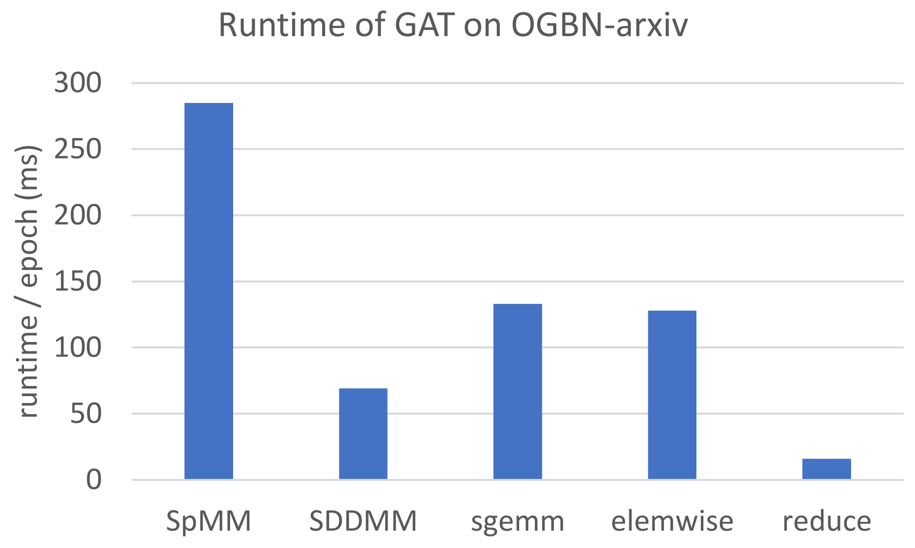


Both sparse operations and dense operations have roughly the same amount of overhead.

GAT full-batch training

Setup	
Dataset	OGBN-arxiv
#layers	3
Hidden dimensions	256
#attention heads	3
Hardware	Nvidia T4
Model size	1.4M

```
H = matmul(W, H)
El = matmul(Wl_a, H)
Er = matmul(Wr_a, H)
E = LeakyReLU(SDDMM(El, Er, A))
E = edge_softmax(E)
M = SpMM(E, H)
H = ReLU(M, W)
```

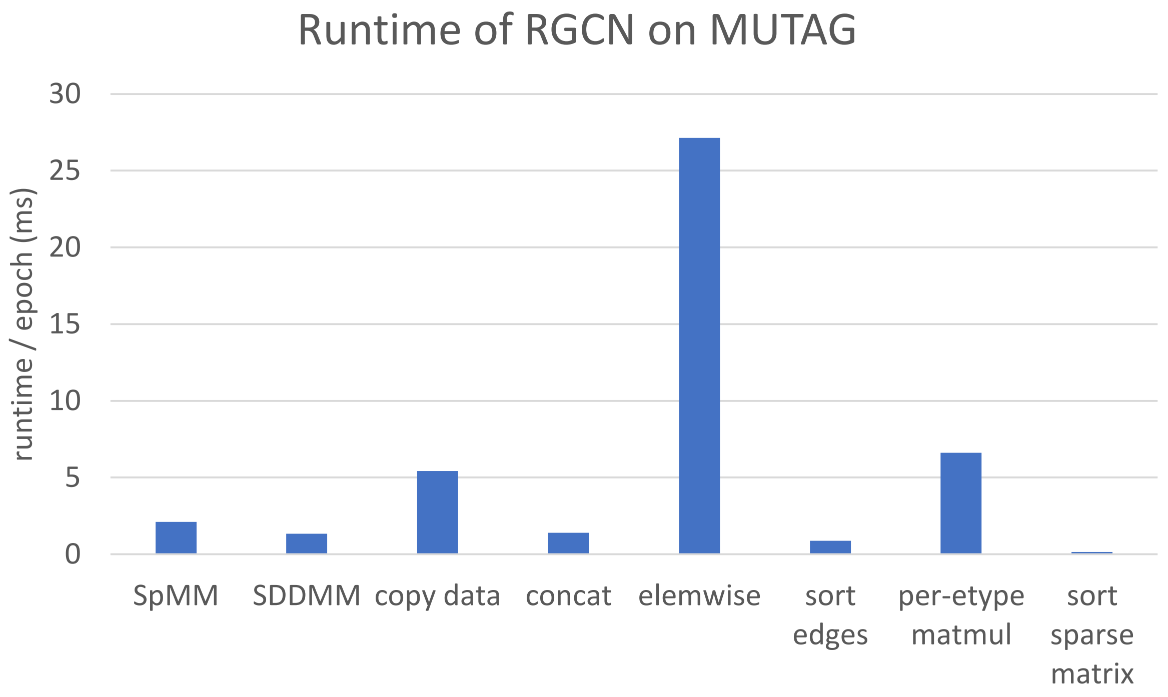


Both sparse operations and dense operations have roughly the same amount of overhead.

RGCN full-batch training

Setup	
Dataset	mutag
#layers	2
Hidden dimensions	256
Hardware	Nvidia T4
Model size	10M

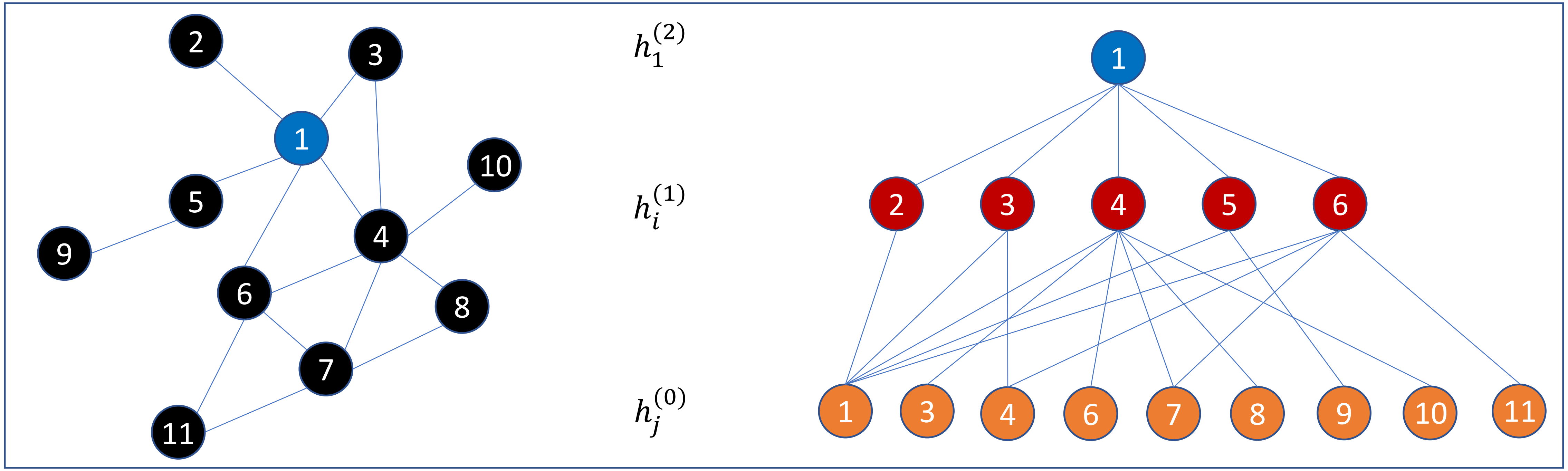
```
M = []
src_id, dst_id, etype = g.edges()
etype, idx = sort(etype)
src_id, dst_id = src_id[idx], dst_id[idx]
H = H[src_id]
H_list = split(H, etypes)
for r in range(num_relations):
   M.append(matmul(H_list[r], W[r])/deg)
M = cat(M)
A' = coo_matrix((range(num_edges), dst_id))
M = SpMM(A', M)
H = ReLU(M)
```



Dense operations dominate the model computation.

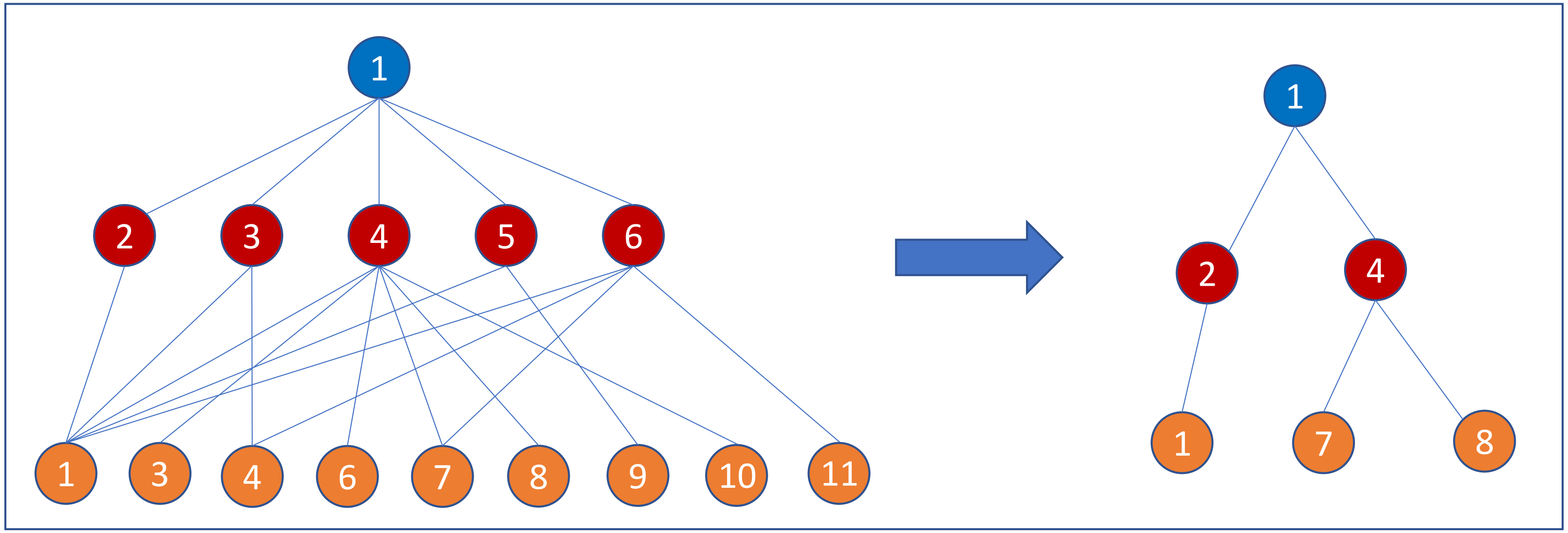
Mini-batch training on large graphs

- Another view of computing node embeddings.
- A mini batch represents the computation graph for target nodes.
- Small-world graphs lead to a huge computation graph.



Neighbor sampling

- Prune the computation graph:
  - Sample neighbors from a neighbor list of a vertex.



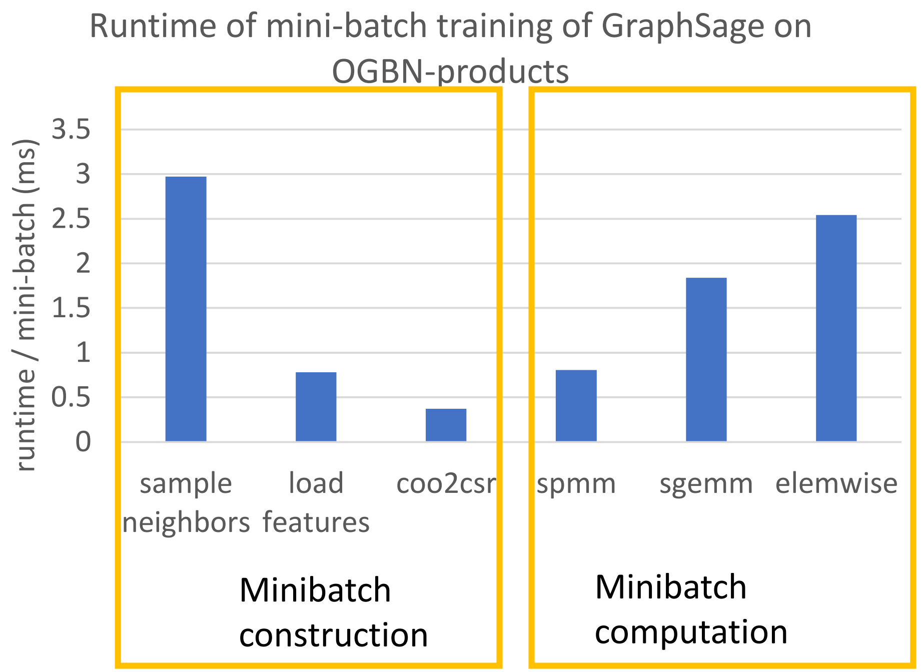
Mini-batch training on GPU

- Two ways of performing mini-batch training.
  - Pure GPU training: all data in GPU.
  - Mixed CPU-GPU training: the whole graph data in CPU and mini-batch computation in GPU.
- The benchmark covers pure GPU mini-batch training.

GraphSage mini-batch training

Setup	
Dataset	OGBN-products
#layers	2
Hidden dimensions	256
fanout	25,10
Batch size	1000
Hardware	Nvidia T4
Model size	217K

```
M = SpMM(A, H)/deg(A)
H = ReLU(matmul(M, W1) + b1 +
            matmul(H, W2) + b2)
H = Dropout(H)
```

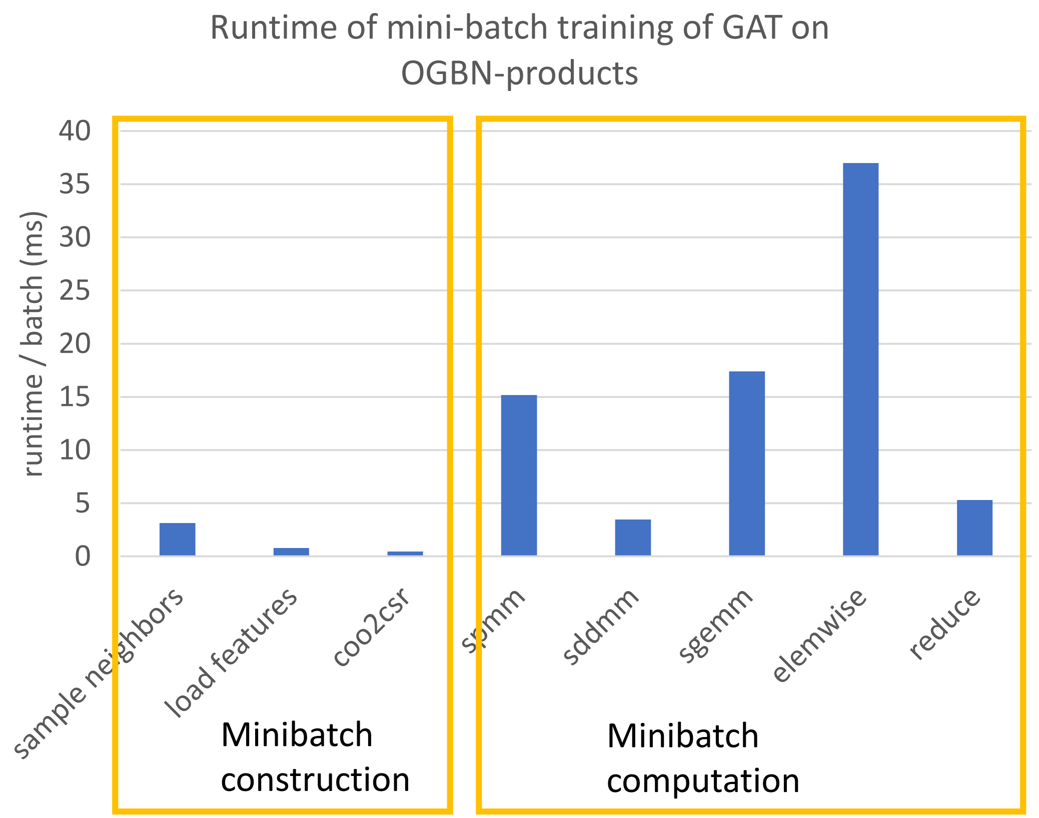


- Mini-batch construction is very expensive;
- Dense operations are much more expensive than sparse operations in mini-batch training.

GAT mini-batch training

Setup	
Dataset	OGBN-products
#layers	3
Hidden dimensions	256
fanout	5,10,15
#attention heads	3
Batch size	1000
Hardware	Nvidia T4
Model size	1.4M

```
H = matmul(W, H)
El = matmul(Wl_a, H)
Er = matmul(Wr_a, H)
E = LeakyReLU(SDDMM(El, Er, A))
E = edge_softmax(E)
M = SpMM(E, H)
H = ReLU(M, W)
```

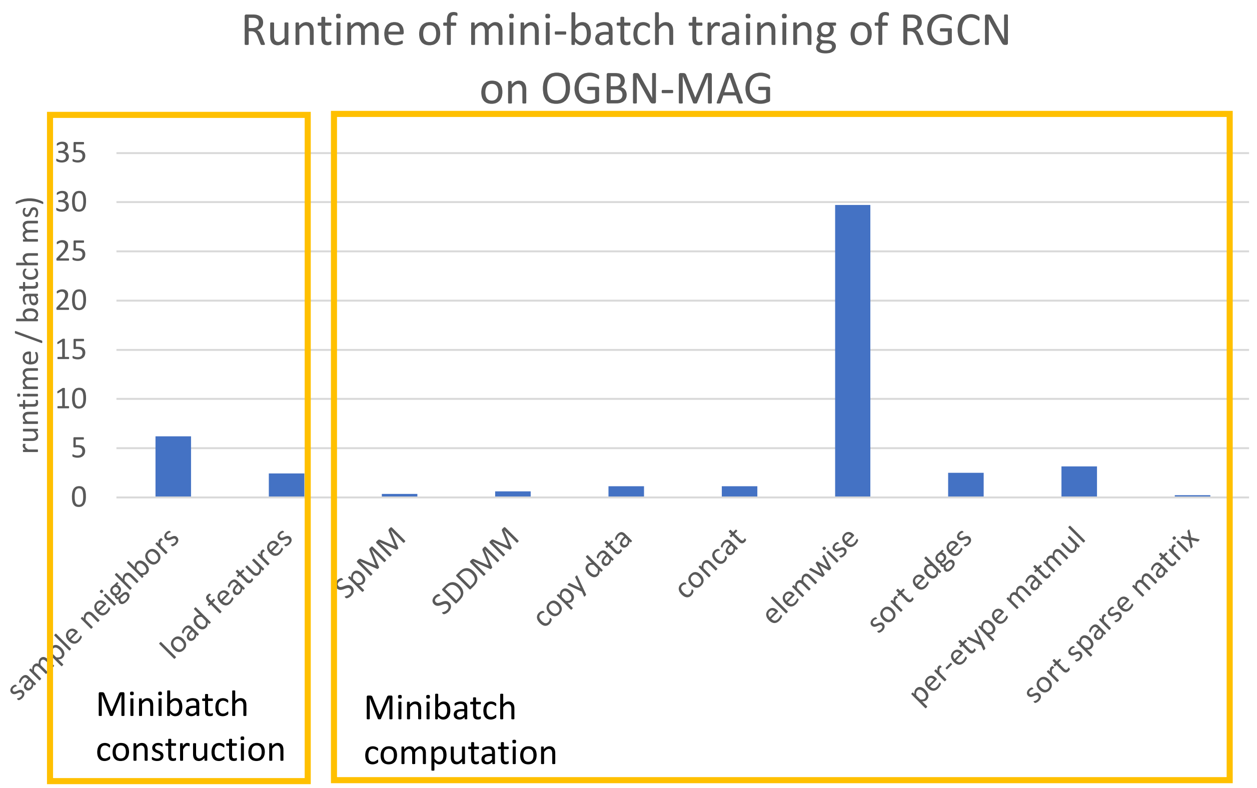


- Mini-batch computation of GAT is much more expensive; mini-batch construction is relatively cheap.
- Dense operations are much more expensive than sparse operations in mini-batch training.

RGCN mini-batch training

Setup	
Dataset	OGBN-MAG
#layers	2
Hidden dimensions	64
Batch size	512
Fanout	25,30
Hardware	Nvidia T4
Model size	309M

```
M = []
src_id, dst_id, etype = g.edges()
etype, idx = sort(etype)
src_id, dst_id = src_id[idx], dst_id[idx]
H = H[src_id]
H_list = split(H, etypes)
for r in range(num_relations):
   M.append(matmul(H_list[r], W[r])/deg)
M = cat(M)
A' = coo_matrix((range(num_edges), dst_id))
M = SpMM(A', M)
H = ReLU(M)
```



Summary

- For GNN workloads, both sparse and dense operations are important.
- Training methods have large impact on GNN workloads.
    - For full-graph training, both sparse and dense operations account for half of runtime.
    - For mini-batch training, runtime are more dominated by dense operations.
    - Mini-batch sampling may cause significant effort during training.