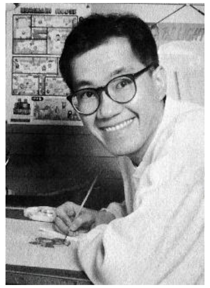
University of Science and Technology of China
Data Darkness Laboratory
SIGMOD PODS 2025 BERLIN GERMANY


# Capsule: An Out-of-Core Training Mechanism for Colossal GNNs

Yongan Xiang, Zezhong Ding, Rui Guo,
Shangyou Wang, Xike Xie, and S. Kevin Zhou

June. 26, 2025

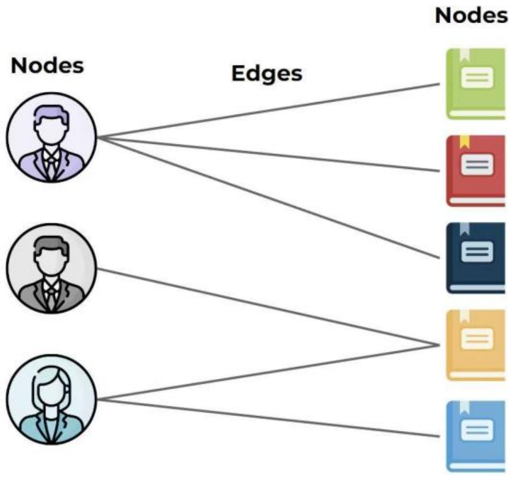


*Akira Toriyama*

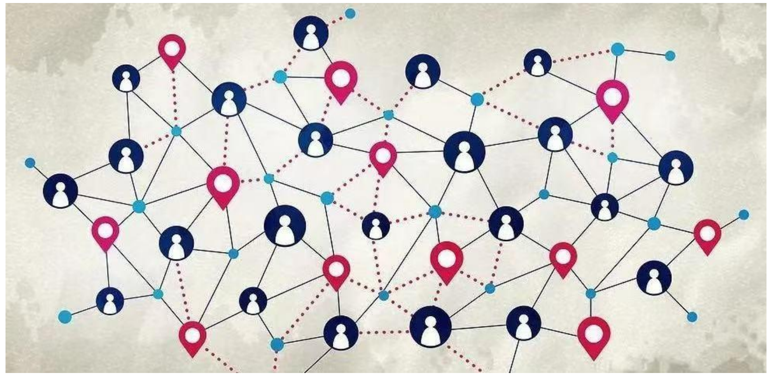
*In the Dragon Ball series by **Akira Toriyama** (1955-2024), Capsule Corporation designs **capsules** that can accommodate huge items like motorcycles, houses, and spaceships.*

# GNN Training System Background

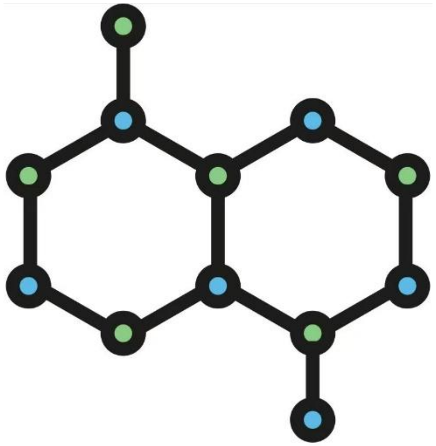
- Graph Neural Networks (GNNs) have been widely applied.




Recommendation Systems



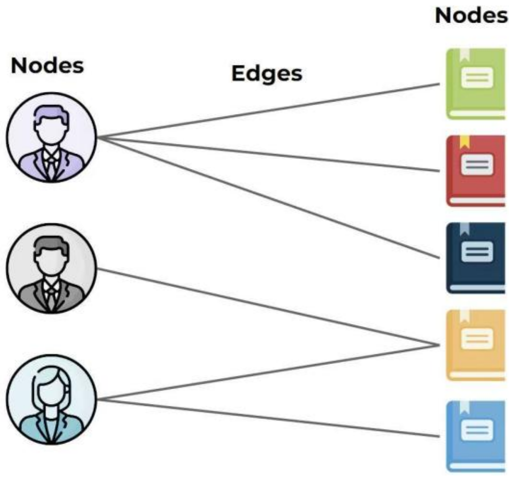
Social Network Analysis



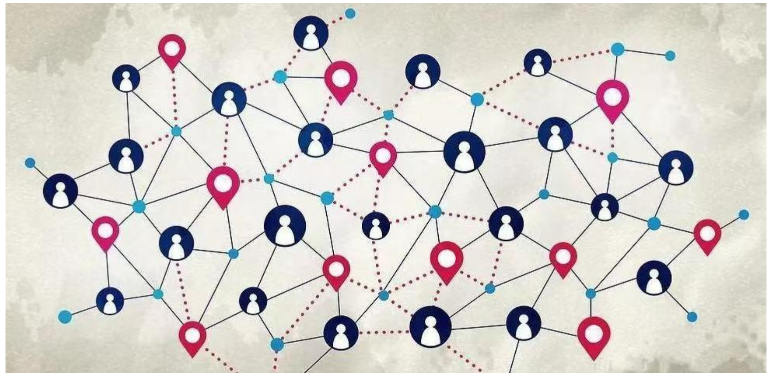
Chemistry and Bioinformatics

# GNN Training System Background

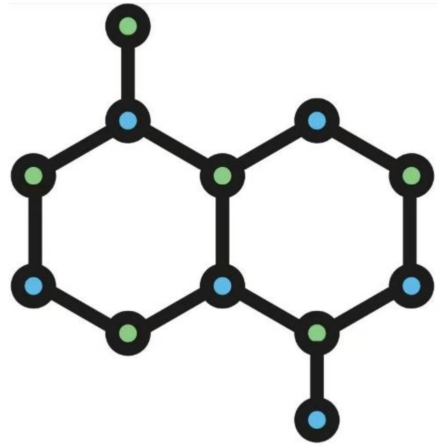
- Graph Neural Networks (GNNs) have been widely applied.




Recommendation Systems



Social Network Analysis



Chemistry and Bioinformatics

- How to train large graphs?
  - Only CPU?
  - GPU w/ main memory to store the graphs?
  - GPU w/ secondary memory (e.g., SSD) to store the graphs? (Out-of-Core)

# GNN Training System Background

- Only CPU?
  - Poor Parallelism and Slow Computation

# GNN Training System Background

- Only CPU?
  - Poor Parallelism and Slow Computation

- GPU (w/ main memory to store the graphs)?
  - ~~Poor Parallelism and Slow Computation~~
  - High Main Memory Cost
  - Significant Data Transfer Between Main Memory and GPU Memory

# GNN Training System Background

- Only CPU?
  - Poor Parallelism and Slow Computation

- GPU (w/ main memory to store the graphs)?
  - ~~Poor Parallelism and Slow Computation~~
  - High Main Memory Cost
  - Significant Data Transfer Between Main Memory and GPU Memory

- GPU (w/ secondary memory to store the graphs)? ------ Out-of-Core
  - ~~Poor Parallelism and Slow Computation~~
  - ~~High Main Memory Cost~~
  - Significant Data Transfer Among Secondary Memory, Main Memory and GPU Memory.

# GNN Training System Background

- GPU (w/ secondary memory to store the graphs)? ------ Out-of-Core
- SOTA GNN training systems

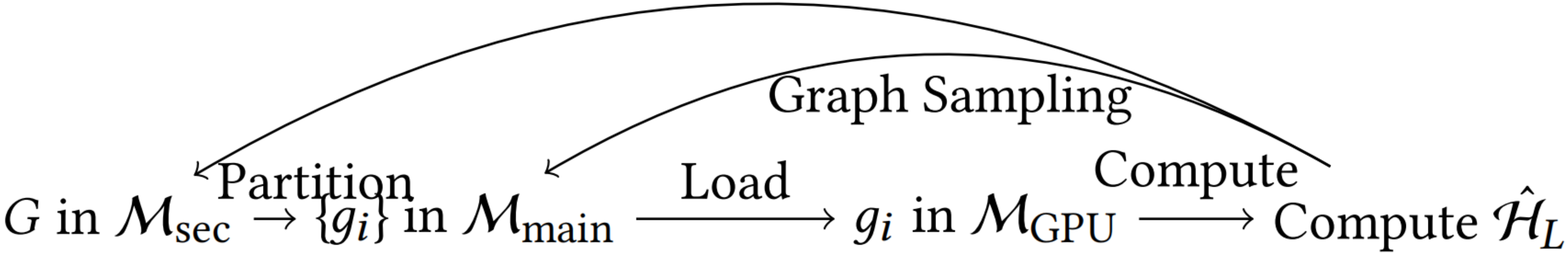
- DGL (AWS, arXiv'2018)
- PyG (ICLR'2019)

- MariusGNN (Eurosys'2023)
  Graph Partitioning
- Ginex (VLDB'2022)
  Caching

# GNN Training System Background

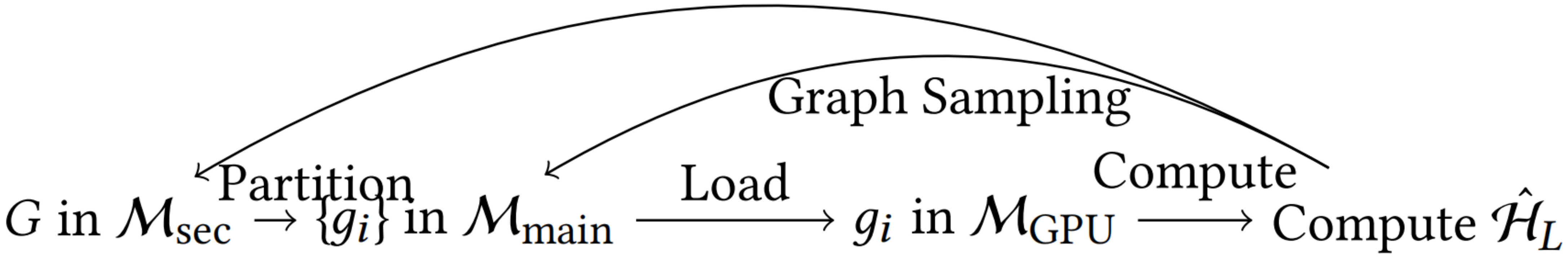
- GPU (w/ secondary memory to store the graphs)? ------ Out-of-Core
- SOTA GNN training systems

- DGL (AWS, arXiv'2018)
- PyG (ICLR'2019)
- MariusGNN (Eurosys'2023)
  Graph Partitioning
- Ginex (VLDB'2022)
  Caching




During graph sampling, **frequent data transfers** often occur between GPU memory and main memory or even secondary memory.

# GNN Training System Background (Cont.)




- ☐ Frequent data transfers, **extensive** computation and I/O overhead
- ☐ Sampling size is user-defined, not **fully** utilizing **GPU** parallelization

# Capsule

- **Traditional Out-of-Core GNN Training:**

Graph Sampling

$$G \text{ in } \mathcal{M}_{\text{sec}} \xrightarrow{\text{Partition}} \{g_i\} \text{ in } \mathcal{M}_{\text{main}} \xrightarrow{\text{Load}} g_i \text{ in } \mathcal{M}_{\text{GPU}} \xrightarrow{\text{Compute}} \text{Compute } \hat{\mathcal{H}}_L$$

- **Capsule Out-of-Core GNN Training:**
  - The entire sampling process is conducted on the GPU memory.
  - We can eliminate the I/O cost during sampling.

Sampling

$$G \text{ in } \mathcal{M}_{\text{sec}} \xrightarrow{\text{Partition}} \{g_i\} \text{ in } \mathcal{M}_{\text{main}} \xrightarrow[\text{Trim}]{\text{Load}} g_i \text{ in } \mathcal{M}_{\text{GPU}} \xrightarrow[\text{Compute}]{} \text{Compute } \hat{\mathcal{H}}_L$$

# Challenges

- It is difficult to streamline data transferring from secondary storage to main memory and finally to GPU memory.

# Challenges

- **It is difficult to streamline data transferring from secondary storage to main memory and finally to GPU memory.**

- **Challenge 1**: Traditional partitioning is different from partitioning designed for GNNs.

Traditional Graph Partitioning
(Minimizing Vertex Replication)

→

Graph Partitioning for GNNs
(Minimizing training metric)

$$minimize \ \mathrm{RF} \quad s.t. \ \frac{k \max_{i=1,\dots k} |P_i|}{|E|} \le \tau$$

$$\mathrm{RF} = \underbrace{\frac{\sum_{v \in V} |P(v)|}{|V|}}_{\textit{Vertex Replication Form}}$$

$$\text{minimize Metric} := \underbrace{\sum_{p_i \in P} |p_i|}_{\text{Raw Cost}} + \underbrace{\sum_{p_i \in P} |\Psi(p_i)|}_{\text{Completion Cost}}$$

# Propagation-based Graph Partitioning (Algorithm 1)

- **Challenge 1**: Traditional graph partitioning is different from graph partitioning for GNNs.

- Graph Partitioning Metric for GNNs

$$\text{minimize Metric} := \underbrace{\sum_{p_i \in P} |p_i|}_{\text{Raw Cost}} + \underbrace{\sum_{p_i \in P} |\Psi(p_i)|}_{\text{Completion Cost}}$$

Bitwise Optimization (GPU Parallel Acceleration)

*Please refer to our manuscript for more details about this optimization*

- Graph Partitioning Algorithm for GNNs

①Constructing the $L$-hop neighbor induced subgraphs of the training nodes (Propagation-based)

②Merging the subgraphs based on the heuristic strategy (Estimated size of resulting graph < $B_{GPU}$)

# Challenges

- After Algorithm 1, we **cannot** ensure that the subgraphs can be completely loaded into GPU memory.
  - The resulting subgraph size is estimated.
  - The training graph might have high connectivity and density.

- **Challenge 2**: It is non-trivial to fit subgraphs into GPU memory.

Subgraph Size > GPU Memory Budget

↓

Out of GPU Memory Issue

# Vertex Importance-based Subgraph Trimming (Algorithm 2)

- **Challenge 2:** It is non-trivial to fit subgraphs into GPU memory.

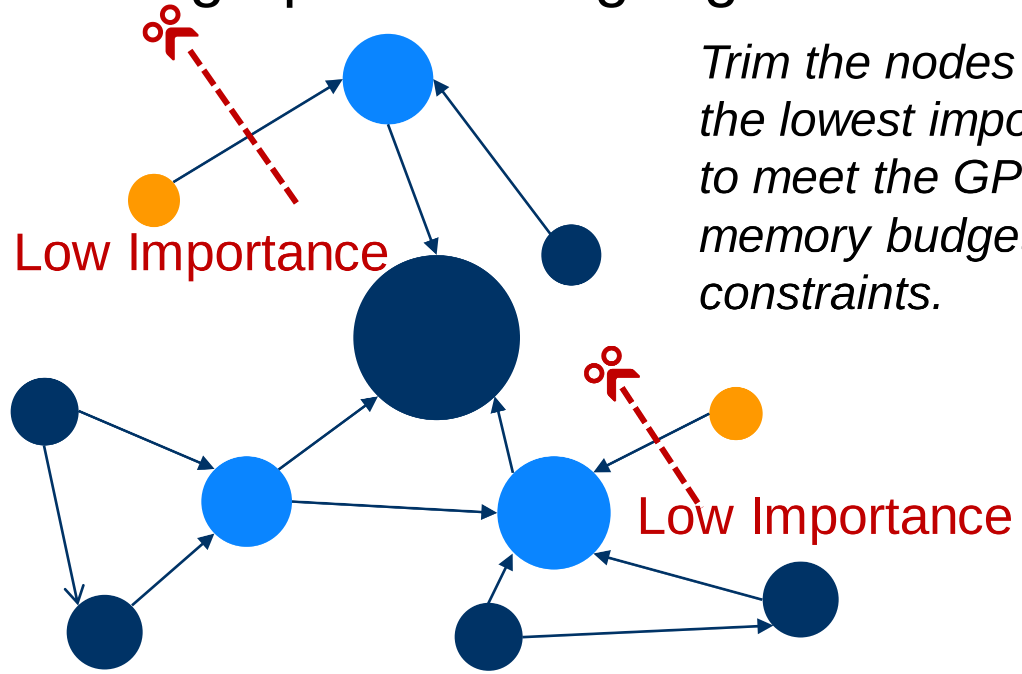
- Vertex Importance

$$I(v) = S_\ell(v) = \beta \sum_{e(u,v)\in E} \frac{S_{\ell-1}(u)}{d_{out}(u)}, \quad where \quad S_0(v) = \begin{cases} 1 & if\, v \in p_i^* \\ 0 & if\, v \notin p_i^* \end{cases}$$

**GNN Training Characteristics**

- Subgraph Trimming Algorithm

*Trim the nodes with the lowest importance to meet the GPU memory budget constraints.*




- Quality Guarantee

$$\text{Capsule: } \mathbb{E}[\|\hat{Z}_{p_i,*} - Z_{p_i,*}\|_F^2] \leq \frac{\delta \|Q_{p_i,\mathcal{D}_i}\|_F^2}{\rho}$$

Capsule can achieve a **better** bound than existing SOTA sampling methods (e.g., BNS-GCN MLsys 2022)

*Please refer to our manuscript for more details about theoretical analysis*

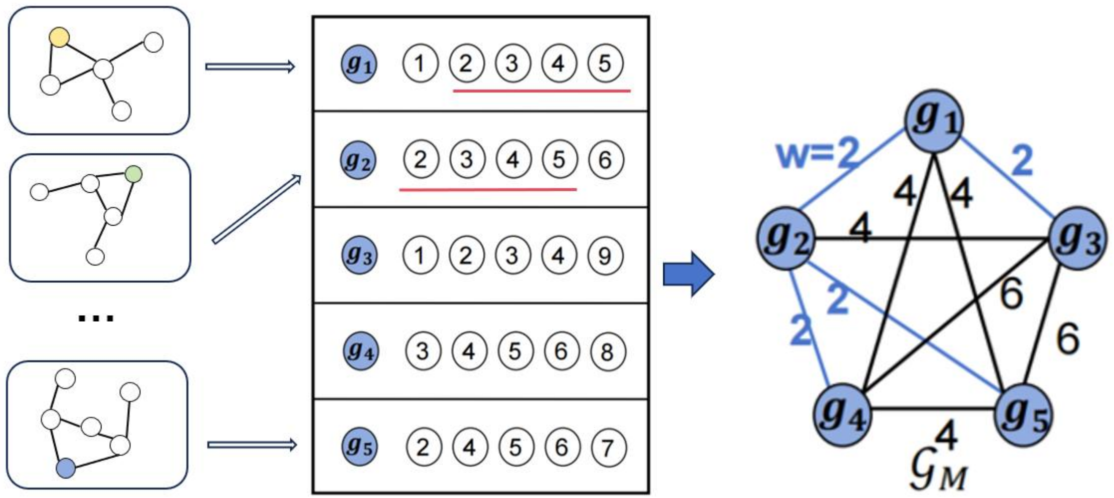
## Comparison of Partitioning Algorithm Performance

| Dataset / Method | PA $\mathcal{T}_p$ | PA $\mathcal{M}_{main}$ | PA Metric | PA $\mathcal{T}_t$ | FR $\mathcal{T}_p$ | FR $\mathcal{M}_{main}$ | FR Metric | FR $\mathcal{T}_t$ | UK $\mathcal{T}_p$ | UK $\mathcal{M}_{main}$ | UK Metric | UK $\mathcal{T}_t$ | WB $\mathcal{T}_p$ | WB $\mathcal{M}_{main}$ | WB Metric | WB $\mathcal{T}_t$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| METIS | N/A | OOM | N/A | N/A | N/A | OOM | N/A | N/A | N/A | OOM | N/A | N/A | 1,138 | 235,783 | 1.64 | **23** |
| 2PS-L | 1,455 | 105,692 | 2.54 | 31 | 1,391 | 53,722 | 3.85 | 54 | 1,670 | 48,803 | 1.35 | 18 | 799 | 70,145 | **1.63** | **23** |
| HDRF | 830 | 96,050 | 3.43 | 47 | 688 | 44,515 | 3.46 | 49 | 1,671 | 46,994 | 1.98 | 25 | 789 | 63,095 | 2.82 | 36 |
| Random | 992 | 126,324 | 6.11 | 88 | 611 | 55,915 | 4.80 | 67 | 959 | 77,184 | 2.99 | 37 | 809 | 93,647 | 5.52 | 70 |
| BGL | 3,867 | 243,712 | 4.51 | 66 | 3,373 | 155,648 | 3.11 | 46 | N/A | OOM | N/A | N/A | 2,520 | 163,840 | 3.36 | 43 |
| Capsule | **111** | **19,935** | **1.22** | **22** | **268** | **28,051** | **2.30** | **33** | **77** | **29,373** | **0.51** | **13** | **150** | **14,888** | 1.67 | 24 |

**Quality** **Efficiency**

# Subgraph Incremental Loading (Further Optimization)

- Subgraph loading problem can be transformed into the shortest Hamiltonian cycle problem




- Distance

$$Dist(g_i, g_j) = |g_i.V| + |g_j.V| - 2|g_i.V \cap g_j.V|$$

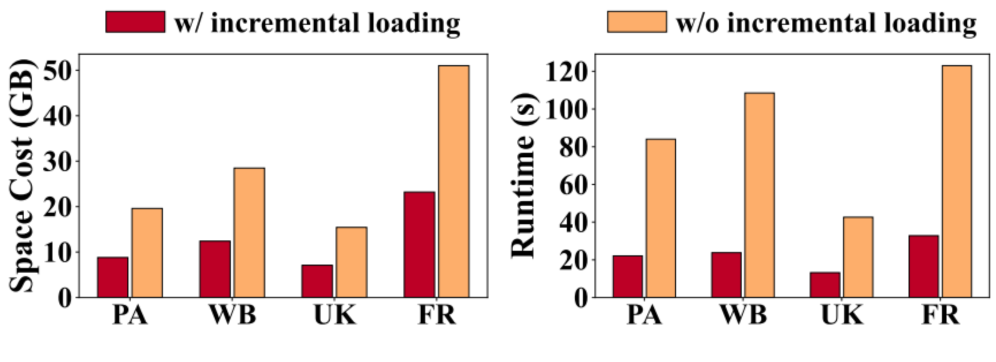
- Total Cost

$$Cost = Dist(g'_k, g_1) + \sum_{i=1}^{k-1} Dist(g_i, g_{i+1})$$

*Please refer to our manuscript for more details about this modeling*

Christofides Algorithm → **1.5 approximation ratio**

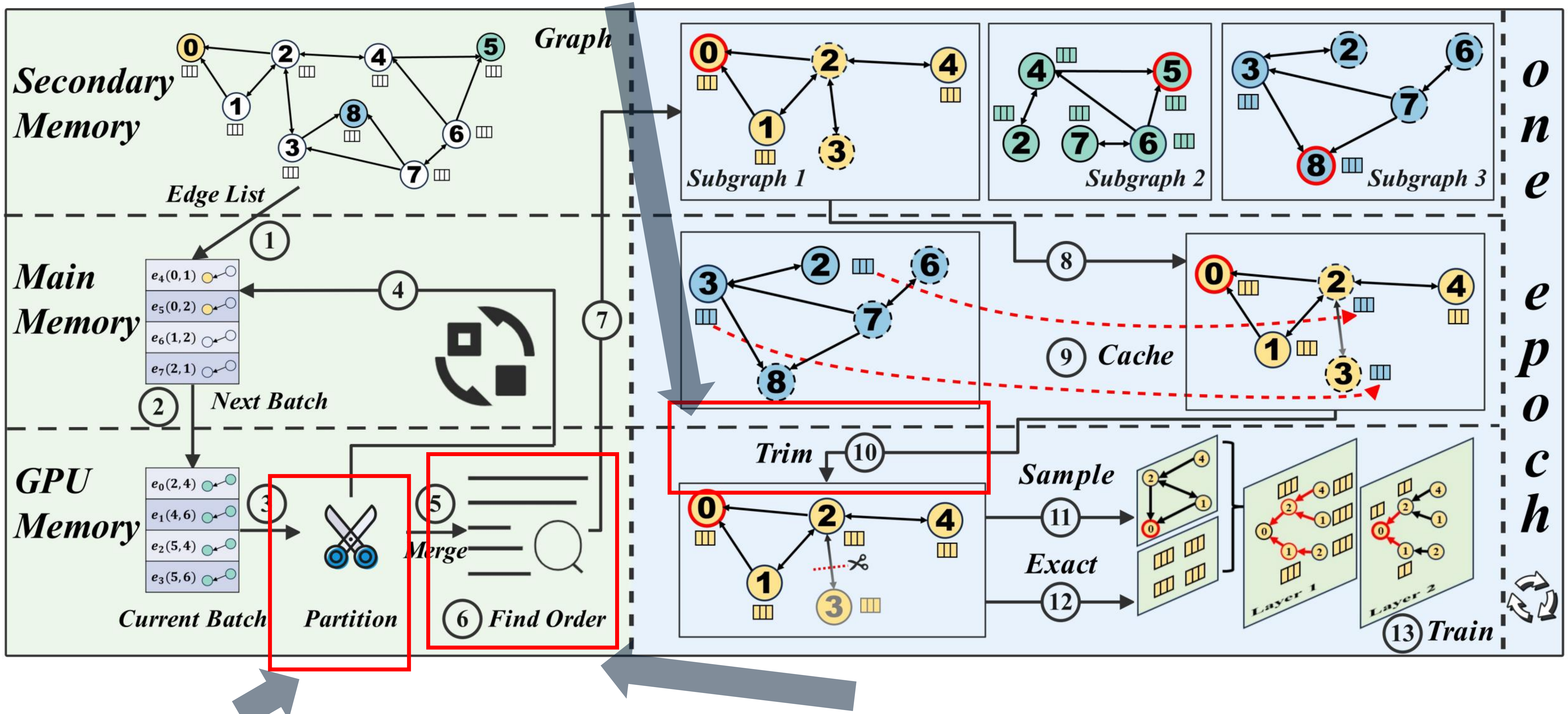
Analysis on Subgraph Loading




(a) Space Cost in $\mathcal{M}_{main}$ (b) Runtime in Training

# Capsule Framework

Vertex Importance-based Subgraph Trimming (Algorithm 2)




Propagation-based Graph Partitioning (Algorithm 1)

Subgraph Incremental Loading (Further Optimization)

# Evaluations

## ➢ Baseline Framework

- DGL, PyG, MariusGNN, Ginex

## ➢ GNN Algorithms

- GraphSage, GCN, GAT

## ➢ Training Task

- Node Classification, Link Prediction

## ➢ Real-world Graphs

- 3 labeled Graphs from OGB and DGL for node classification (RD, PD, PA)
- 3 w/o labeled Graphs from SNAP for System performance testing (UK, FR, WB)
- 2 labeled Graphs from OGB for link prediction task (CT, VS)

# Performance (For more information, please refer to our manuscipt)

Runtime and Space Cost Performance on Different Graphs (20 epochs, time/sec, Mmain/GB)

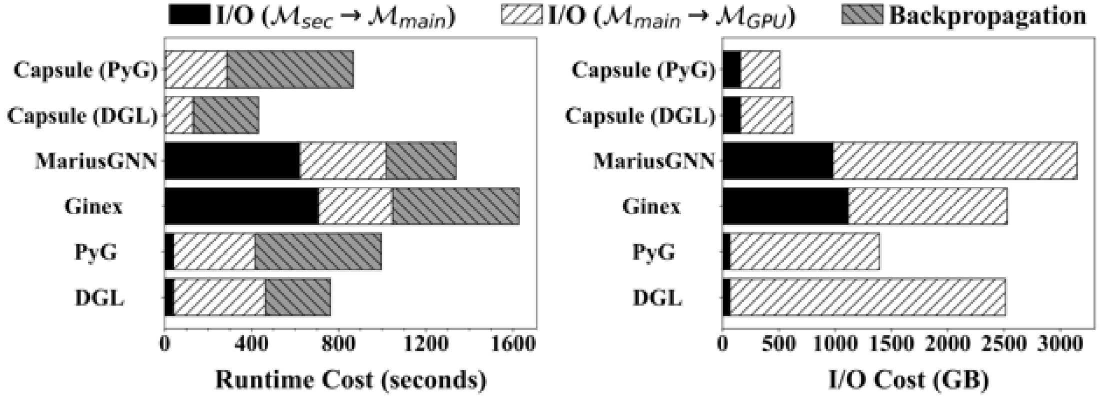
| Stage | Method \ Dataset | RD time | RD $\mathcal{M}_{main}$ | PD time | PD $\mathcal{M}_{main}$ | PA time | PA $\mathcal{M}_{main}$ | FR time | FR $\mathcal{M}_{main}$ | UK time | UK $\mathcal{M}_{main}$ | WB time | WB $\mathcal{M}_{main}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Preprocessing | MariusGNN | 28 | 7.7 | 38 | 8.4 | 955 | 108.7 | 754 | 118.4 | 1,153 | 192.7 | 618 | 71.6 |
| | Ginex | 40 | 7.0 | 86 | 8.3 | 1,845 | 197.0 | 807 | 144.0 | N/A | OOM | 2,019 | 146.0 |
| | Capsule | **25** | **3.5** | **27** | **4.3** | **208** | **20.7** | **413** | **26.3** | **136** | **28.3** | **270** | **14.3** |
| Training (GraphSAGE) | DGL | 879 | 9.1 | 879 | 12.3 | 2,139 | 223.5 | 1,106 | 73.5 | 899 | 204.5 | 818 | 95.7 |
| | PyG | 512 | 7.5 | 893 | 8.6 | 1,620 | 152.0 | 1,176 | 109.5 | 1,720 | 157.0 | 1,333 | 92.1 |
| | MariusGNN | 242 | 13.0 | 249 | 10.2 | 2,874 | 21.5 | 3,495 | 27.0 | 3,935 | 35.0 | [illegible] | [illegible] |
| | Ginex | 1,800 | 8.4 | 1,940 | 9.4 | 2,872 | 11.7 | 2,681 | 30.2 | N/A | OOM | 5,674 | 58.9 |
| | Capsule (DGL) | **106** | **2.7** | **94** | **3.0** | **442** | **9.1** | **656** | **23.7** | **264** | **7.3** | 476 | 13.2 |
| | Capsule (PyG) | 292 | **2.7** | 456 | **3.0** | 802 | **9.1** | 972 | **23.7** | 554 | **7.3** | **472** | **13.1** |
| Training (GCN) | DGL | 899 | 11.4 | 835 | 12.3 | 2,159 | 224.5 | 1,104 | 73.5 | 877 | 190.0 | 810 | 110.4 |
| | PyG | 504 | 7.7 | 929 | 8.8 | 1,628 | 152.5 | 1,211 | 109.5 | 2,094 | 157.0 | 1,357 | 91.8 |
| | MariusGNN | Fail | Fail | Fail | Fail | Fail | Fail | Fail | Fail | Fail | Fail | Fail | Fail |
| | Ginex | 1,560 | 8.4 | 1,640 | 9.4 | 2,804 | 11.7 | 2,601 | 30.2 | N/A | OOM | 5,522 | 53.9 |
| | Capsule (DGL) | **110** | **2.7** | **106** | **3.0** | **456** | **9.1** | **674** | **23.7** | **286** | **7.3** | **510** | 13.2 |
| | Capsule (PyG) | 254 | **2.7** | 480 | **3.0** | 840 | **9.1** | 1,100 | **23.7** | 604 | **7.3** | 526 | **13.1** |
| Training (GAT) | DGL | 359 | 9.0 | 253 | 12.2 | 951 | 142.3 | **442** | 117.8 | 501 | 175.4 | 690 | 111.4 |
| | PyG | 312 | 13.0 | 357 | 8.3 | 1,379 | 147.0 | 698 | 107.0 | 816 | 163.2 | 1,298 | 91.8 |
| | MariusGNN | 668 | 12.1 | 547 | 6.8 | 1,544 | 156.6 | 2,008 | 136.5 | 2,038 | 212.7 | 1,300 | 111.7 |
| | Ginex | Fail | Fail | Fail | Fail | Fail | Fail | Fail | Fail | Fail | Fail | Fail | Fail |
| | Capsule (DGL) | 112 | **2.7** | **128** | **2.9** | 580 | 9.3 | 940 | **23.3** | **362** | **7.7** | **640** | **12.8** |
| | Capsule (PyG) | **109** | 7.2 | 134 | 11.8 | **501** | **9.0** | 1,374 | 23.4 | 366 | **7.7** | 848 | 13.0 |

**12x improvement in Runtime Efficiency**

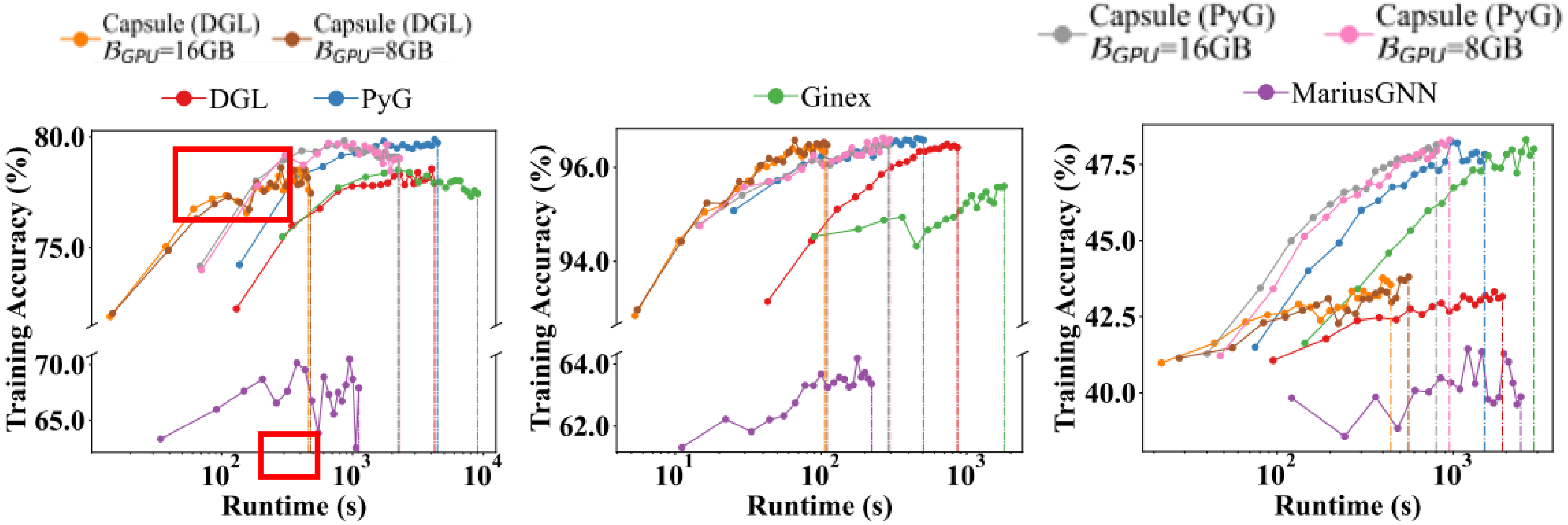
**4.5 × reduction in main memory usage**

**Play-and-plug**

# Runtime and I/O Cost Analysis (Node Classification)




(a) PA (20 epochs)

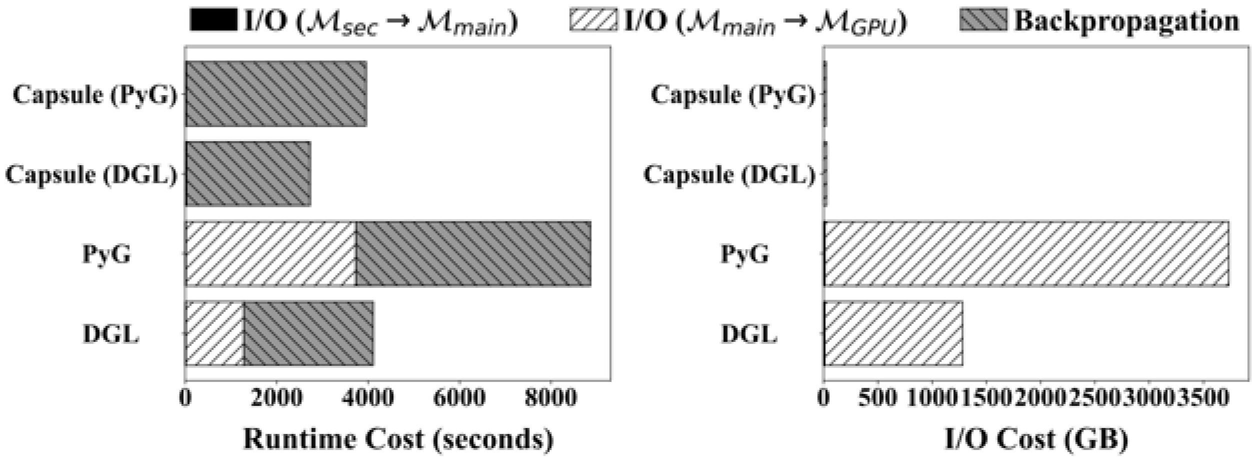



(a) Node Classification (100 epochs in PD)

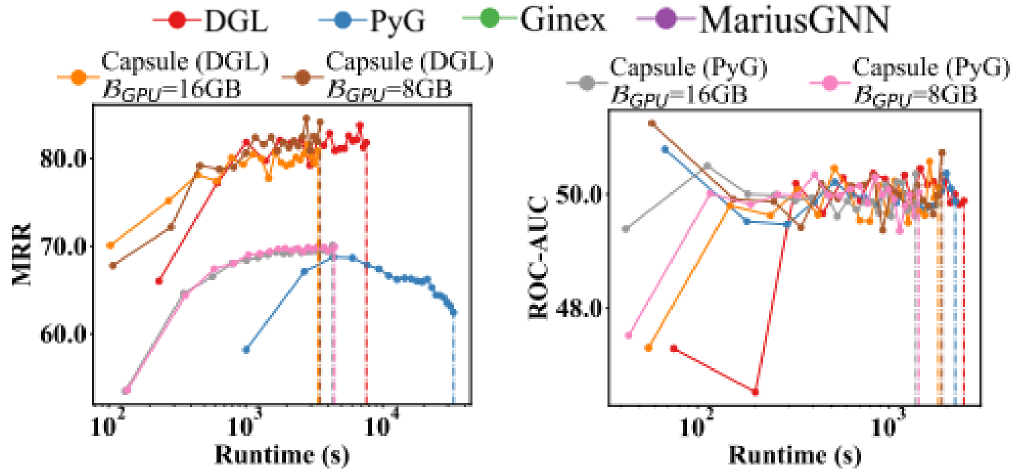
(b) Node Classification (20 epochs in RD)

(c) Node Classification (20 epochs in PA)

# Runtime and I/O Cost Analysis (Link Prediction)




(b) CT (100 epochs)




(d) Link Prediction
(100 epochs in CT)

(e) Link Prediction
(100 epochs in VS)

# Different Hardware Configurations

### Device 1: GeForce RTX 4080 (16G), $M_{main}$ = 128G

| Stage | Method \ Dataset | PA time | PA $M_{main}$ | UK time | UK $M_{main}$ | WB time | WB $M_{main}$ |
|---|---|---|---|---|---|---|---|
| **Prep.** | MariusGNN | 1,046 | 107 | N/A | OOM | 792 | 71.5 |
| | Ginex | N/A | OOM | N/A | OOM | N/A | OOM |
| | Capsule | **199.1** | **19.6** | **315.0** | **28.4** | **237.6** | **13.9** |
| **Sage** | DGL | N/A | OOM | N/A | OOM | N/A | OOM |
| | PyG | N/A | OOM | N/A | OOM | 84.3 | 92.1 |
| | MariusGNN | N/A | OOM | N/A | OOM | Fail | Fail |
| | Ginex | N/A | OOM | N/A | OOM | N/A | OOM |
| | Capsule (DGL) | **370** | **9.0** | **212** | **7.0** | **430** | **13.0** |
| | Capsule (PyG) | **630** | **9.0** | **414** | **6.0** | **392** | **13.0** |

### Device 2: GeForce RTX 3060 (12G), $M_{main}$ = 64G

| Stage | Method \ Dataset | PA time | PA $M_{main}$ | UK time | UK $M_{main}$ | WB time | WB $M_{main}$ |
|---|---|---|---|---|---|---|---|
| **Prep.** | MariusGNN | N/A | OOM | N/A | OOM | N/A | OOM |
| | Ginex | N/A | OOM | N/A | OOM | N/A | OOM |
| | Capsule | **456.6** | **19.8** | **362.9** | **28.6** | **574.4** | **14.5** |
| **Sage** | DGL | N/A | OOM | N/A | OOM | N/A | OOM |
| | PyG | N/A | OOM | N/A | OOM | N/A | OOM |
| | MariusGNN | N/A | OOM | N/A | OOM | N/A | OOM |
| | Ginex | N/A | OOM | N/A | OOM | N/A | OOM |
| | Capsule (DGL) | **538** | **9.3** | **344** | **6.4** | **1,060** | **23.1** |
| | Capsule (PyG) | **1,514** | **9.6** | **1,000** | **7.1** | **1,120** | **23.1** |

Most systems encounter out-of-memory issues

# Conclusions

- We propose Capsule, an innovative out-of-core mechanism to tackle the scalability challenges of GNN Training.
- **Algorithms**
  - ✓ Propagation-based partitioning algorithm optimized for GNN training
  - ✓ Vertex importance-based subgraph trimming algorithm to fit subgraphs within GPU memory
  - ✓ Modeling of the subgraph loading problem as the shortest Hamiltonian cycle problem to optimize loading order
- **Future work**
  - ✓ In the future, we plan to extend the training mechanism of Capsule to dynamic GNN training scenarios.

中国科学技术大学
University of Science and Technology of China

Data Darkness Laboratory

# Thank you!

Zezhong Ding

June. 26, 2025

**Capsule Source Code**

**Manuscript**