Evaluation of Distributed Graph Neural Networks Training for Particle Tracking

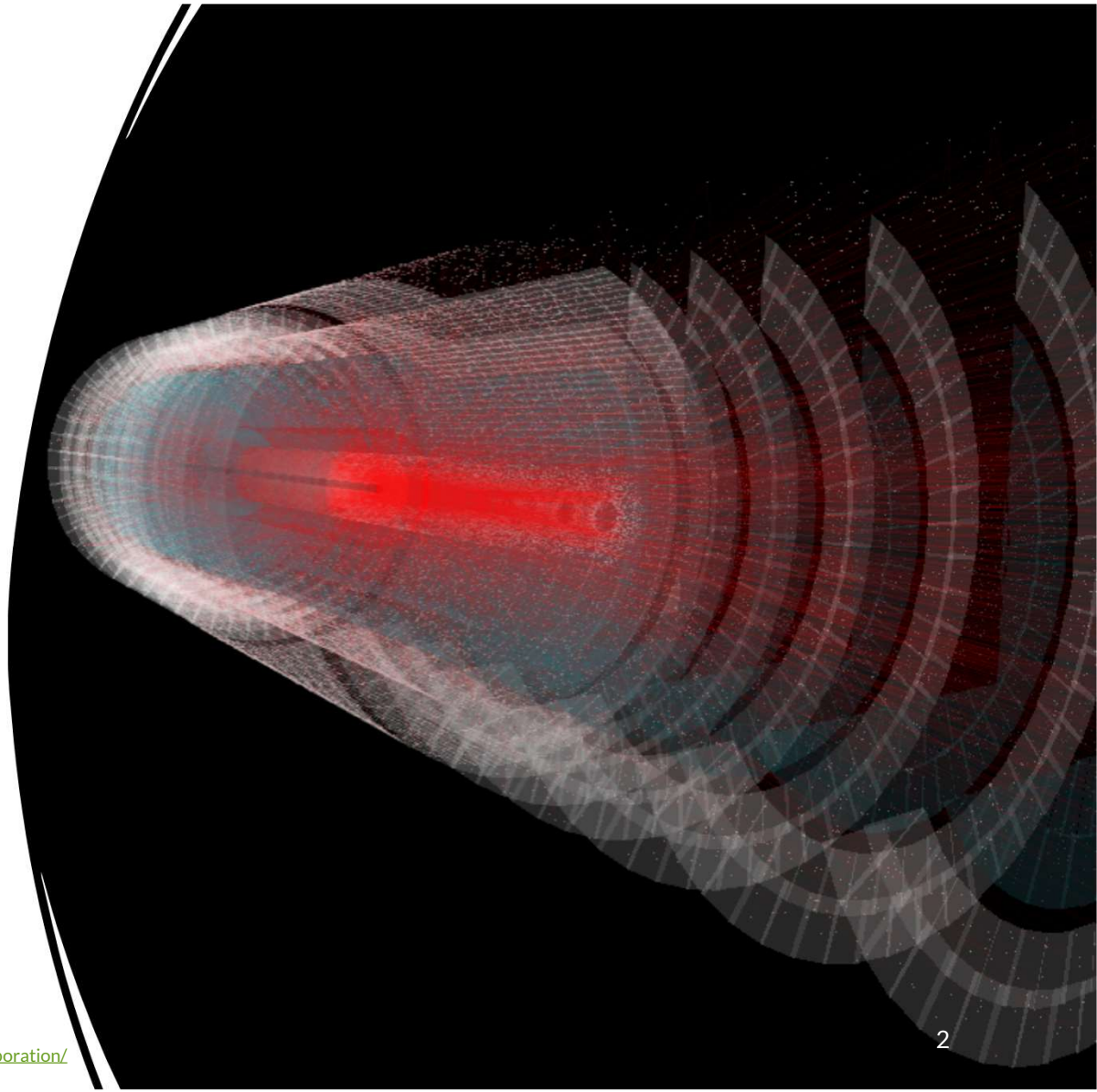
Alina Lazar

Youngstown State University

YOUNGSTOWN STATE UNIVERSITY

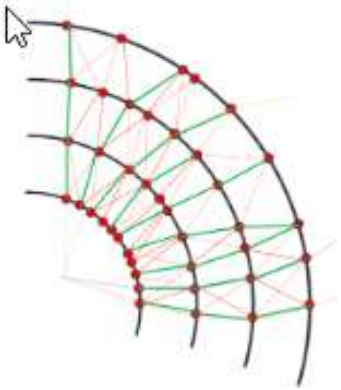
Graph Neural Networks (GNNs) for Particle Tracking

- GNN-based track pattern reconstruction is becoming the tool for track reconstruction.
- Focus: **Scaling GNN models**
- Training GNNs is challenging due to the irregular nature of graph data
- It takes weeks to train
- Scaling to large graphs that exceed the **memory** capacity of a single device



https://lgm.fri.uni-lj.si/research/high-energy-physics-event-visualisation-cern-collaboration/

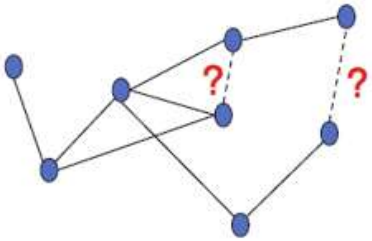
Particle Physics Applications as Graph Tasks

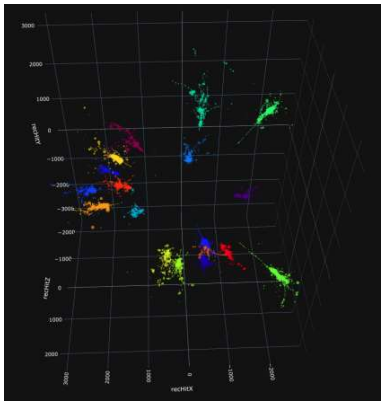


GNN Tutorial Part 1 - Fundamentals.pptx - Google Slides

Charged Particle Tracking Task

Edge Prediction

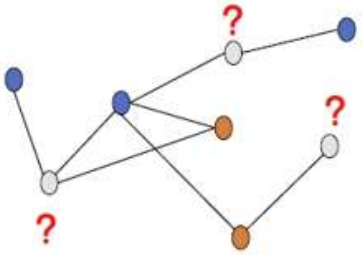


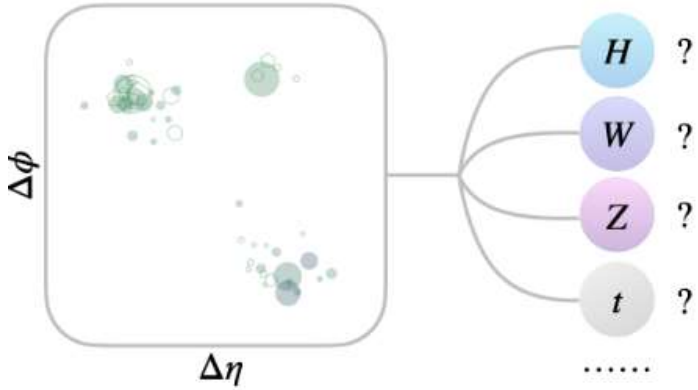


https://openreview.net/pdf?id=PYcp183GBL

Particle Flow Reconstruction

Node Classification

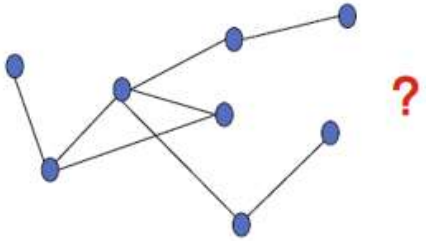




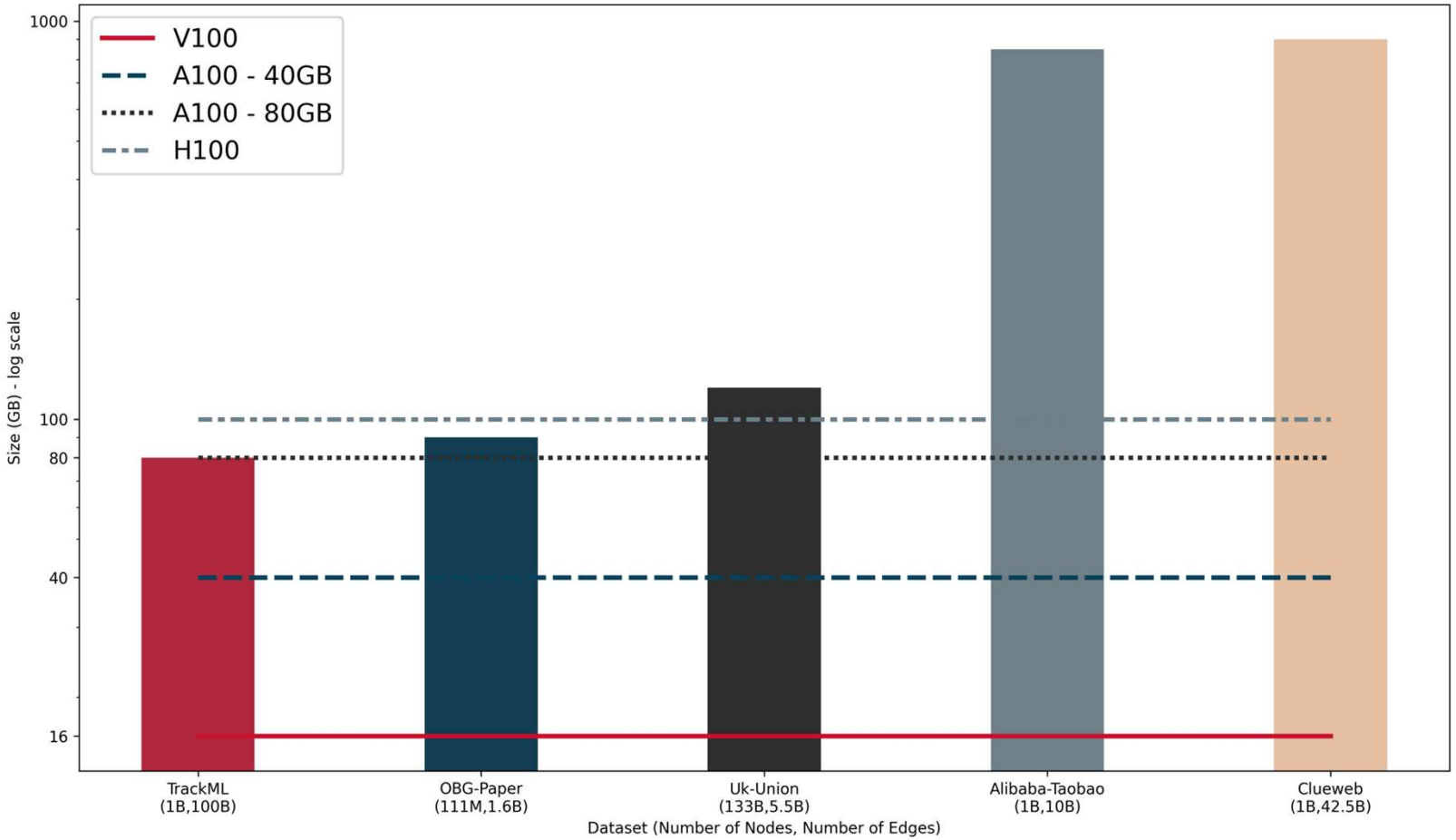
Particle Transformer for Jet Tagging

Jet Tagging and Event Classification

Graph-level Prediction



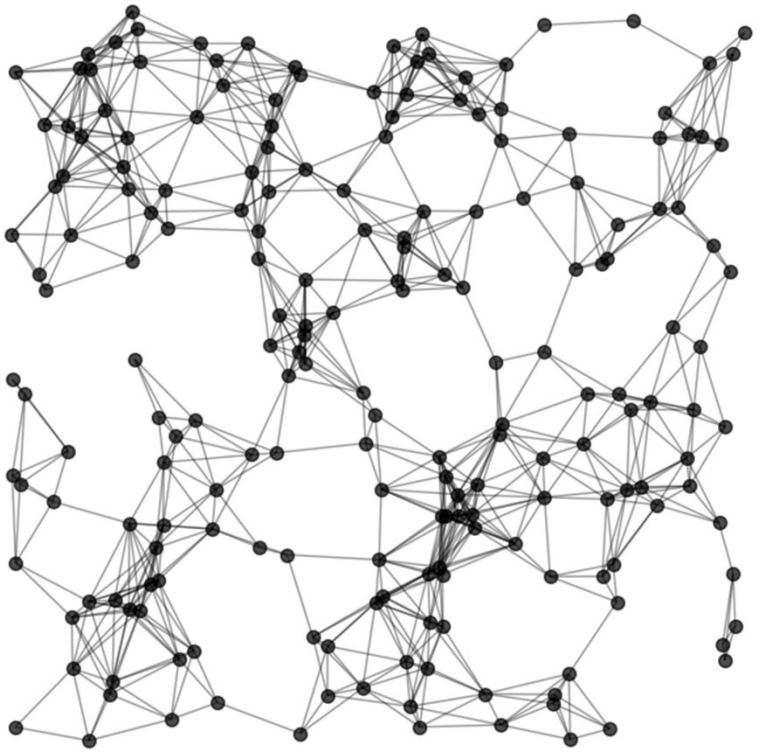
Memory Requirements for Training GNNs on Large Graphs



Legion: Automatically Pushing the Envelope of Multi-GPU System for Billion-Scale GNN Training

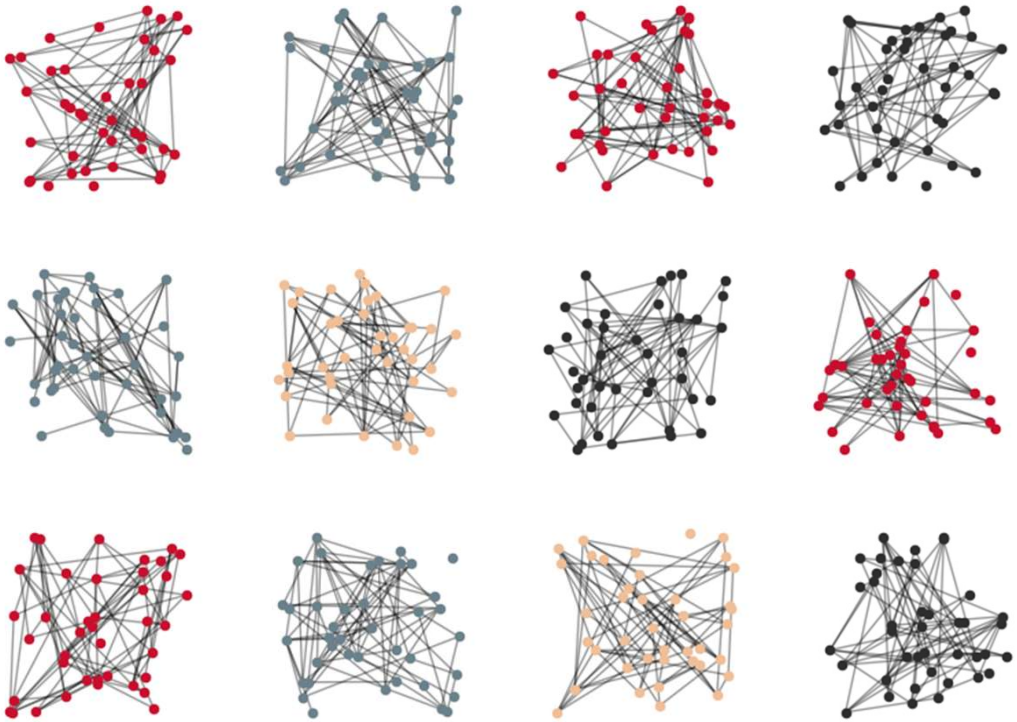
Training GNNs on Large Graphs

ClueWeb (1B nodes, 42.5B edges)

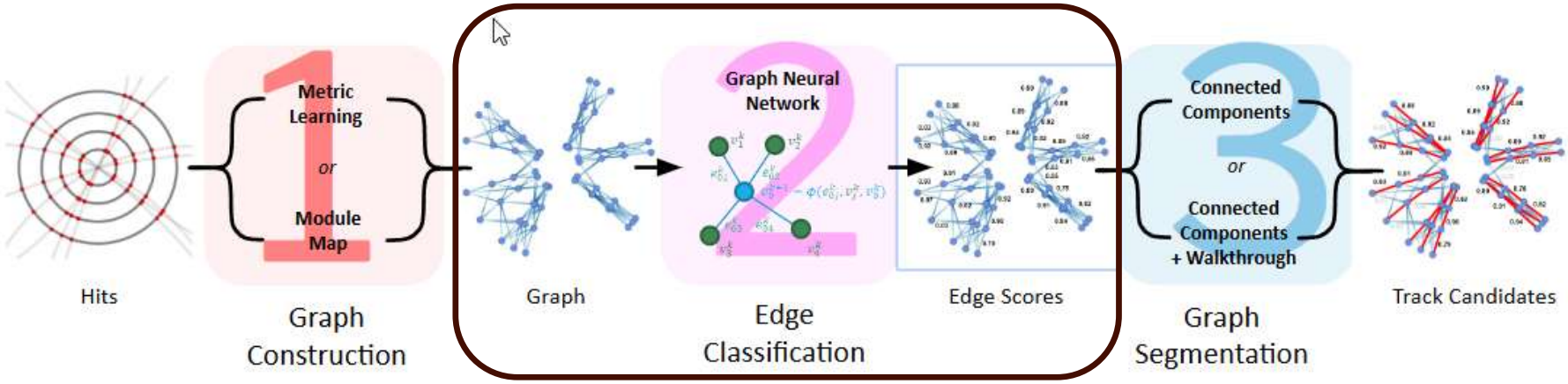


The ClueWeb22 Dataset (lemurproject.org)

TrackML (1B nodes, 100B edges)
10k events, 100k nodes, 10 million edges



GNN-based Track Reconstruction Pipeline

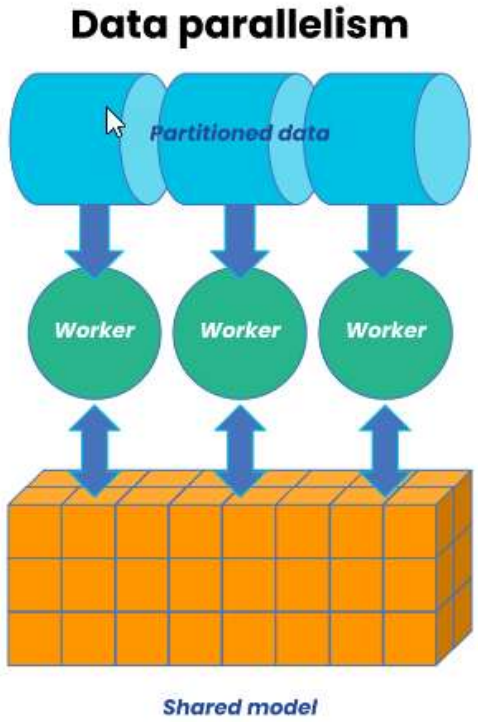


GNN Tutorial Part 1 - Fundamentals.pptx - Google Slides

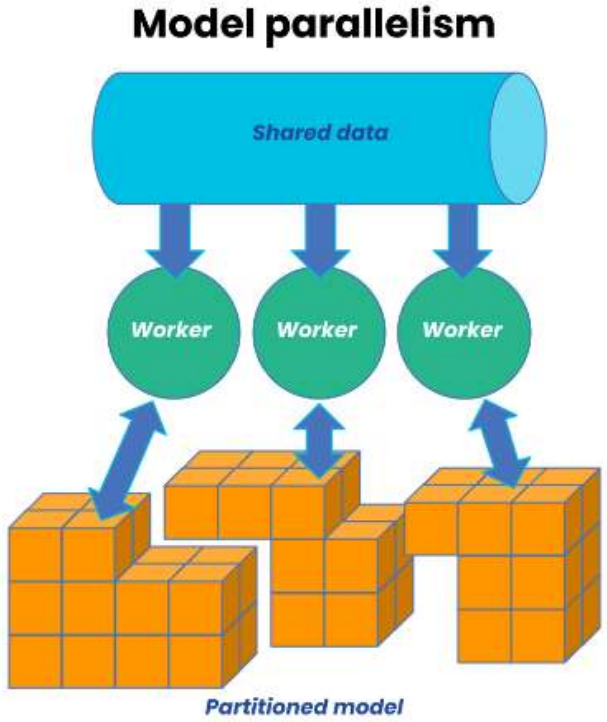
- **Data Loading and Sampling**
  - How to store the large-scale graphs?
  - How to effectively and efficiently sample the subgraphs?
- **Memory**
  - Model parameters and hidden states
  - How to reduce the memory usage during training?
- **Computation**
  - Feature transformation and aggregation
  - How to reduce the training or inference computation?
- **Communication**
  - Distributed training
  - How to efficiently exchange data (graph data and model data)?

Methods to Scale up Training

- **Data Parallel Training**
  - Model agnostic. Supported by DL frameworks: Tensorflow, PyTorch, etc.
  - Requires tuning the batch size and learning rate.
  - Gets complicated for GNNs

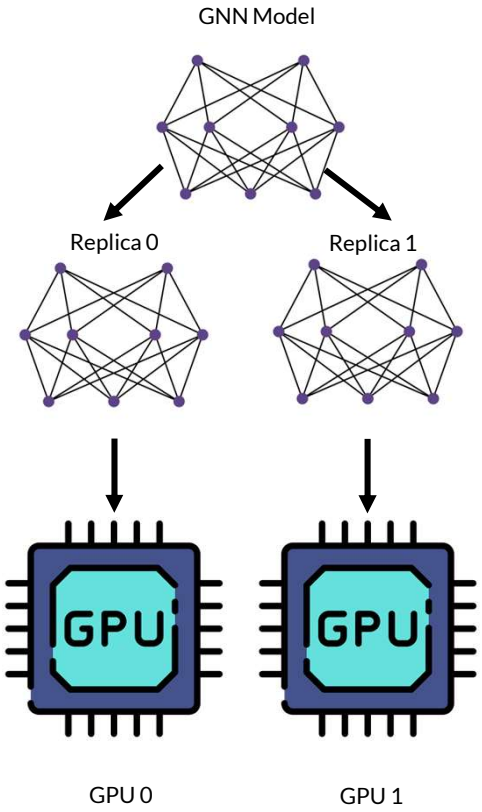


- **Model Parallel Training**
  - Model specific, simple to implement
  - Harder to optimize (dependencies)
  - Good when large models don't fit into a single GPU
  - Streams and input pipelines for speed

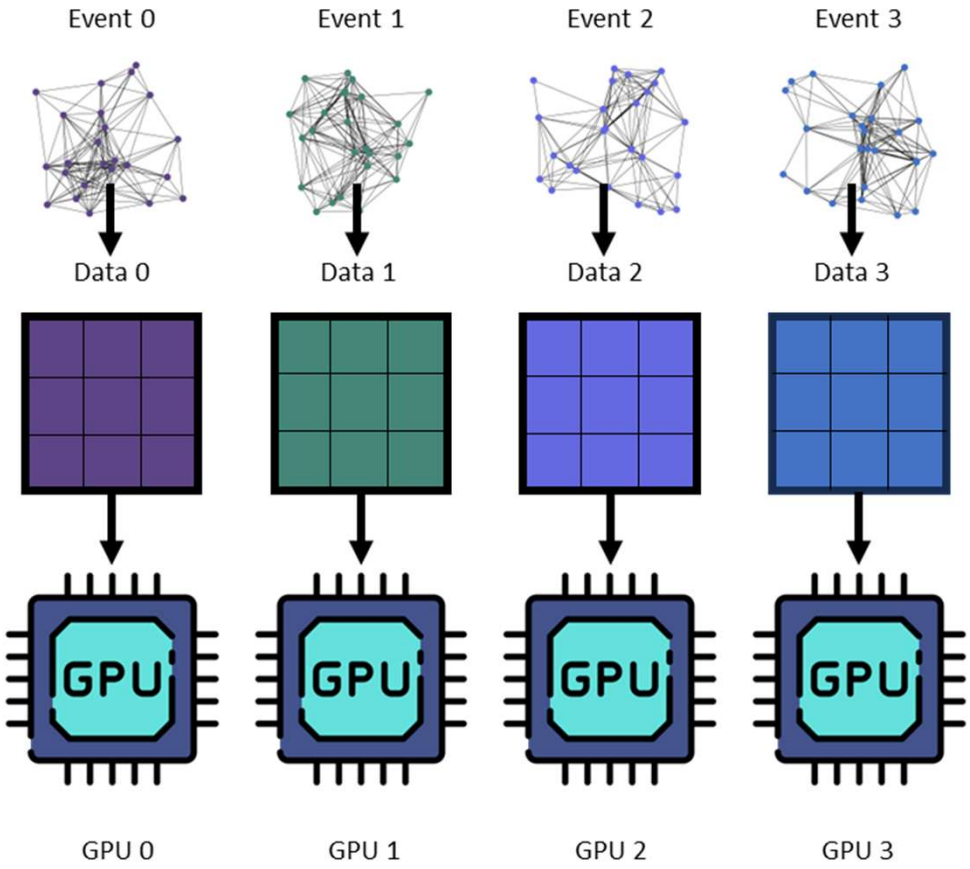


https://www.anyscale.com/blog/what-is-distributed-training

Parallelization Schemes – Distributed Data Parallelism (DDP)



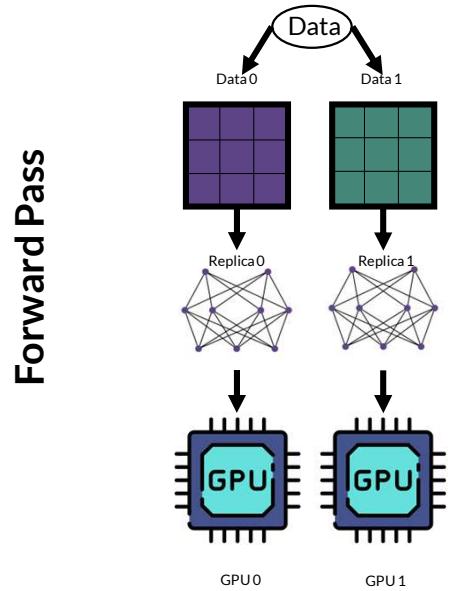
DDP Initialization Model

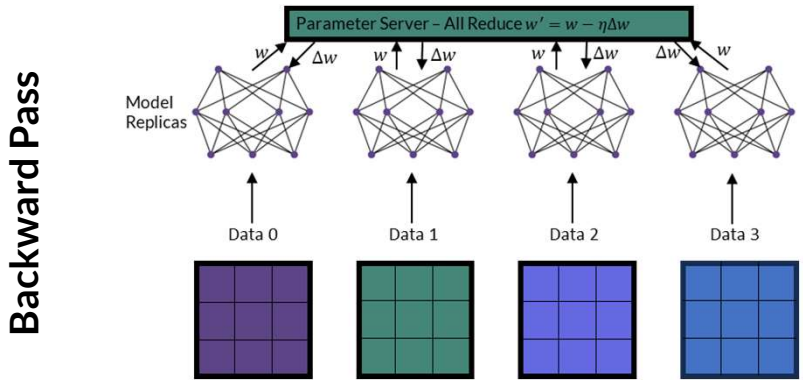


DDP Initialization Data

Data Parallelism Update Strategies

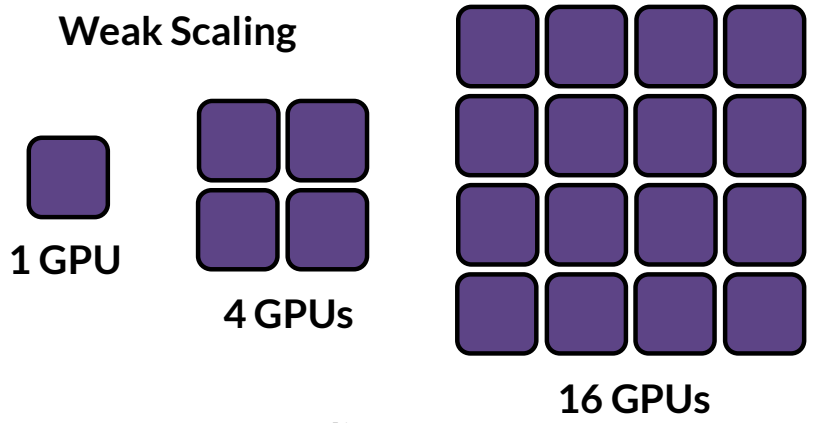
- **Synchronous** updates
  - stable convergence
  - can be decentralized (all-reduce)
  - computation may be blocked by communication
- **Asynchronous** updates
  - no waiting for gradients
  - state gradients affect convergence
  - parameter server can be a bottleneck
- Delayed-synchronous updates
  - Lagged gradients allow better comms overlap
  - Stale gradients affect convergence



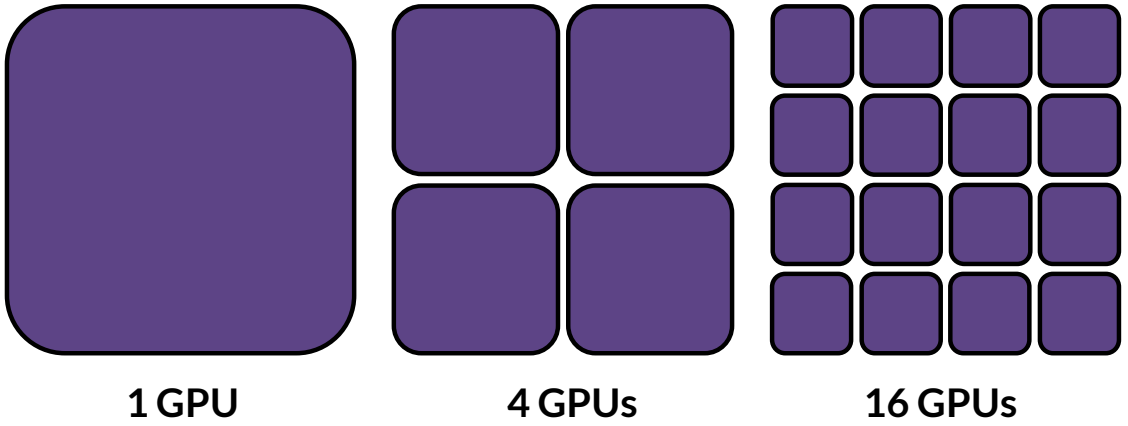


Distributed Data Parallel Scaling

- **Weak scaling** (fixed local batch size)
  - Global batch size grows with the number of workers
  - Local batch size stays constant
  - Computation grows with communication
  - Good scalability
  - Large batch sizes can negatively affect convergence
  - Learning Rate Scaling Rule: When the batch size is multiplied by k, multiply the learning rate by k.

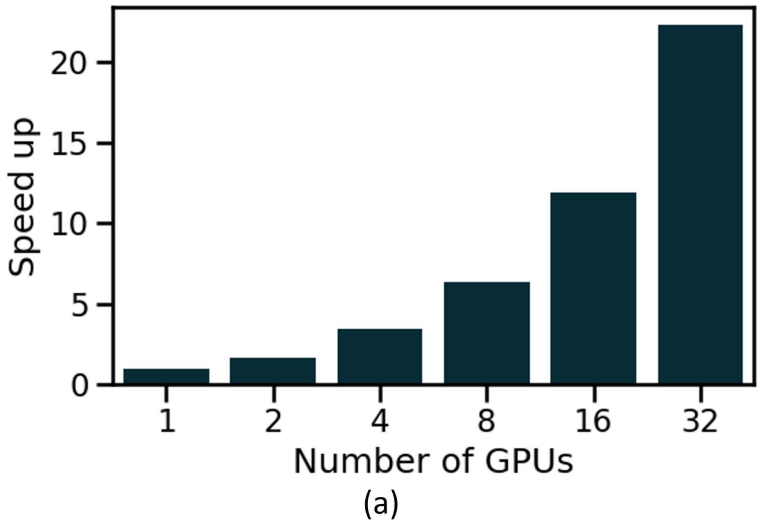


- **Strong scaling** (fixed global batch size)
  - Global batch size stays constant
  - Local batch size decreases with the number of workers
  - Convergence behavior unaffected
  - Communication can become a bottleneck

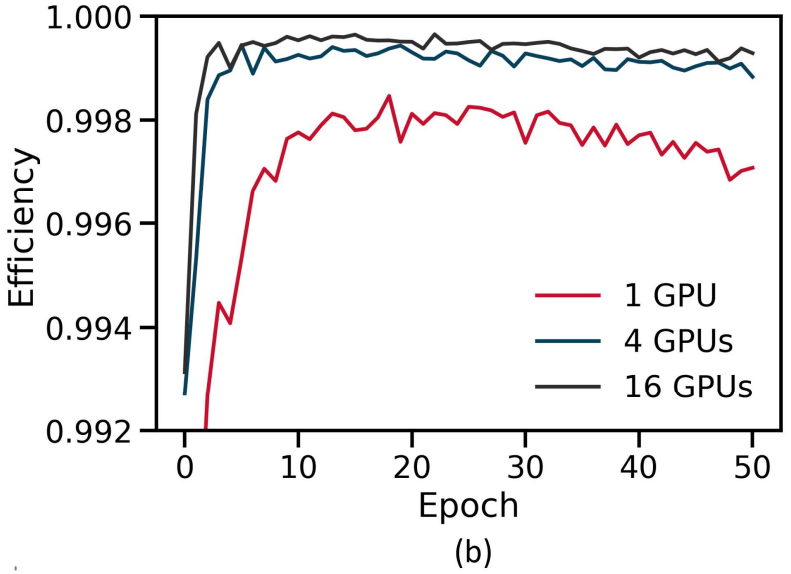


Weak Scaling DDP Experiments

- Experiments were run on A100s nodes with 4 GPUs per node and 80 GB of memory per GPU
- 100 events for training, 10 for validation and 10 for testing per GPU



(a)



(b)

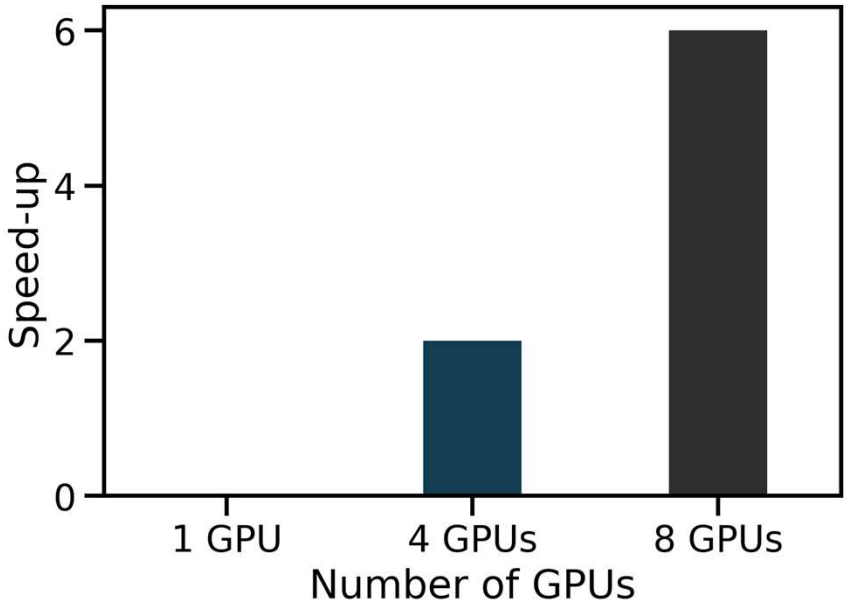
Table 1: Validation Performance Metrics during Training and Runtime

# of GPUs	Efficiency (%)	Target Purity v	Total Purity (%)	Purity (%)	AUC	Validation Loss	Epoch (s)	Batch (s)
1	99.6	87.831	99.249	99.152	98.577	0.00406	34.36	0.24
2	99.778	88.431	99.513	99.452	98.842	0.00226	39.63	0.29
4	99.838	88.508	99.587	99.536	98.846	0.00176	38.73	0.28
8	99.871	88.524	99.658	99.616	98.841	0.00141	42.12	0.31
16	99.903	88.499	99.674	99.633	98.884	0.00114	45.48	0.34
32	99.945	87.931	99.443	99.371	98.79	0.00124	47.6	0.37

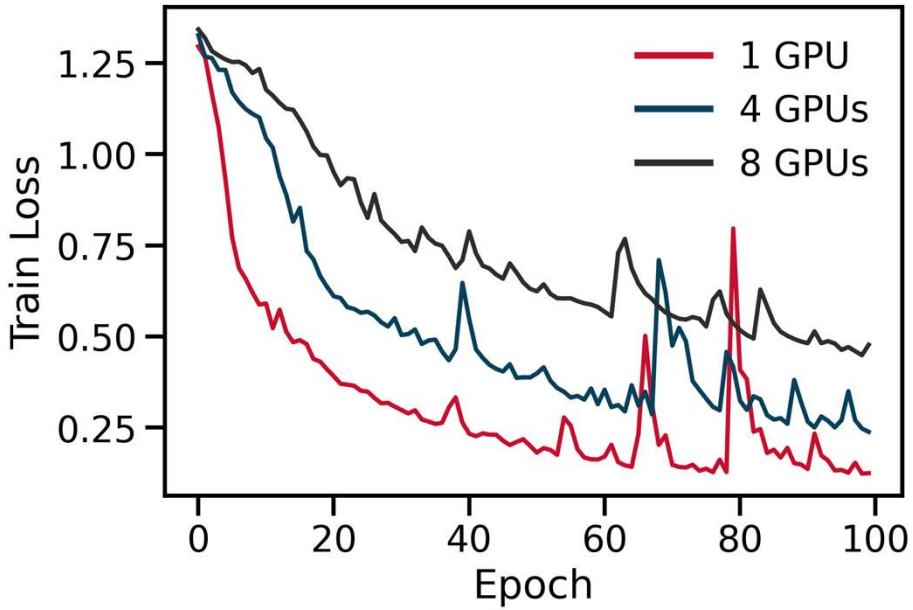
How is the performance of training on **32 GPUs** (100 events per GPU) compared to training on one GPU with **3200 events**?

Strong Scaling DDP Experiments

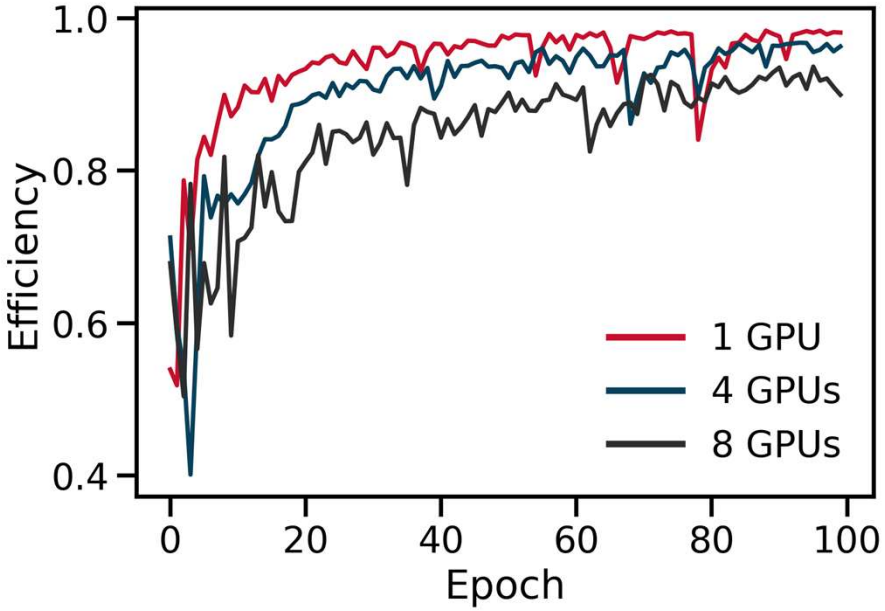
- Experiments were run on A100s nodes with 4 GPUs per node and 80 GB of memory per GPU
- GPU memory utilization of 88.65%

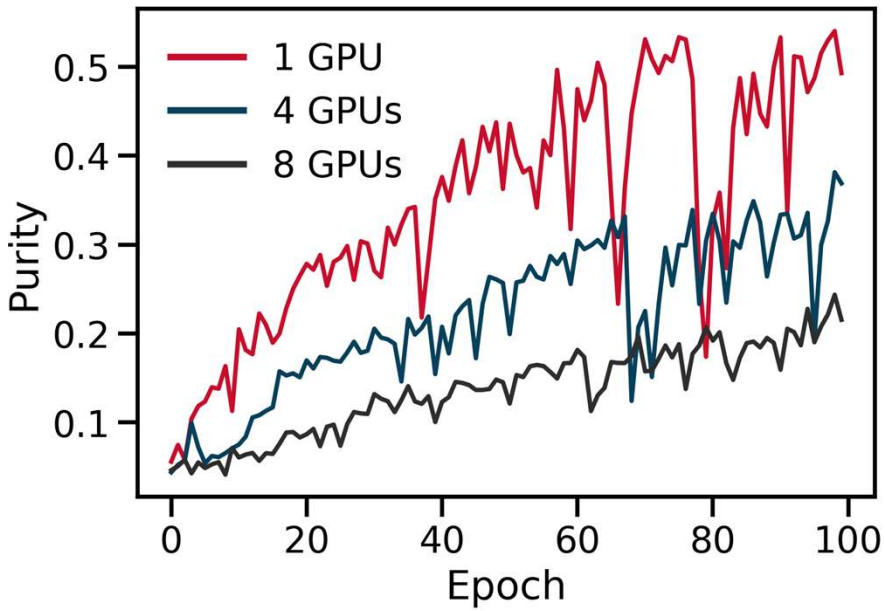


- 80 events for training, 10 for validation and 10 for testing
- Average number of nodes 84k ± 9k
- Average number of edges 2.6m ± 600k



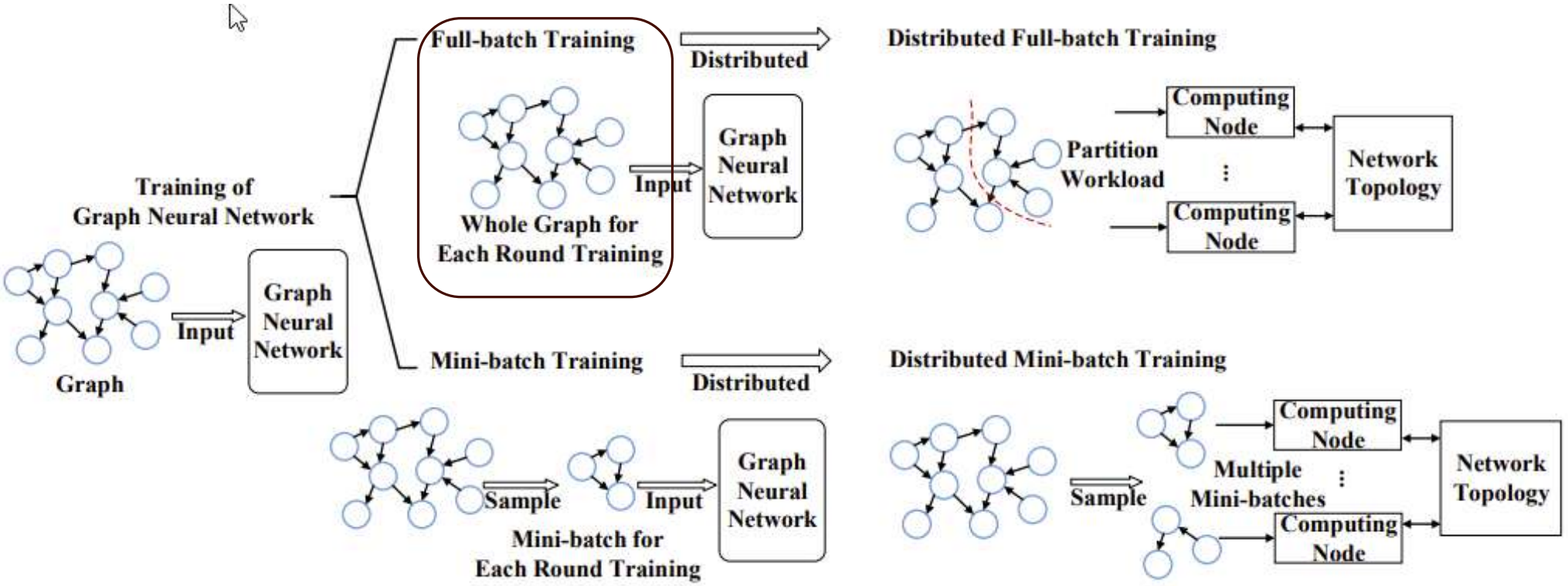
Strong Scaling DDP - Efficiency and Purity





- Using the strong scaling DDP at the **event level** degrades the physics performance in terms of efficiency and purity.
- Also, we want to scale to large event graphs that exceed the memory capacity of single GPUs.
- OpenAI model of noise scale indicates an optimum batch size lower than the full event
- **Solution:** breaking the graphs into smaller subgraphs that can fit in the memory of single GPUs.

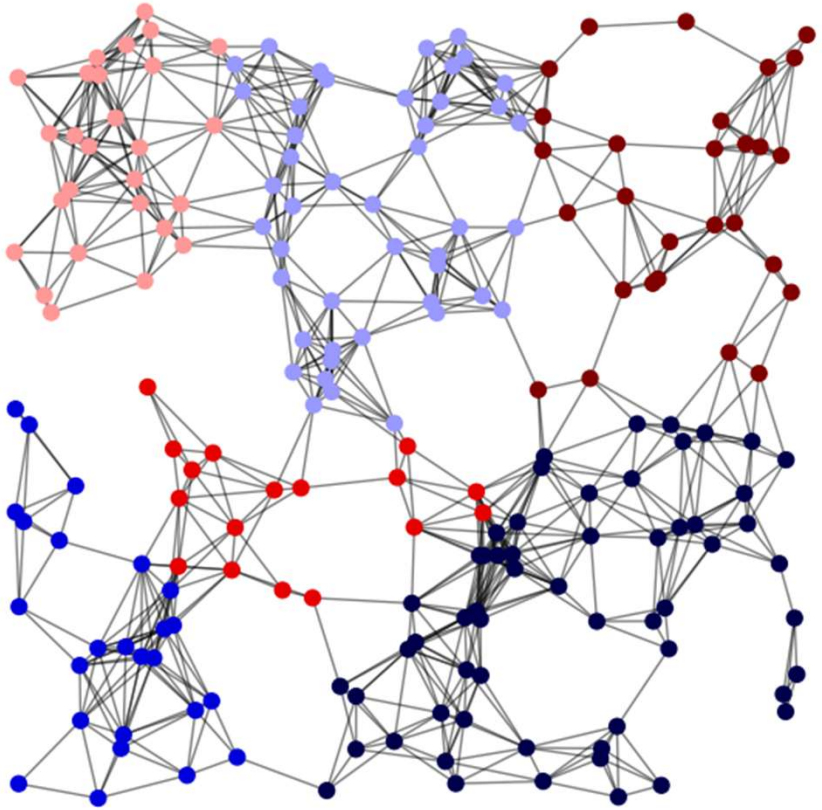
Partitioning versus Mini-Batch Schemes for GNN Training



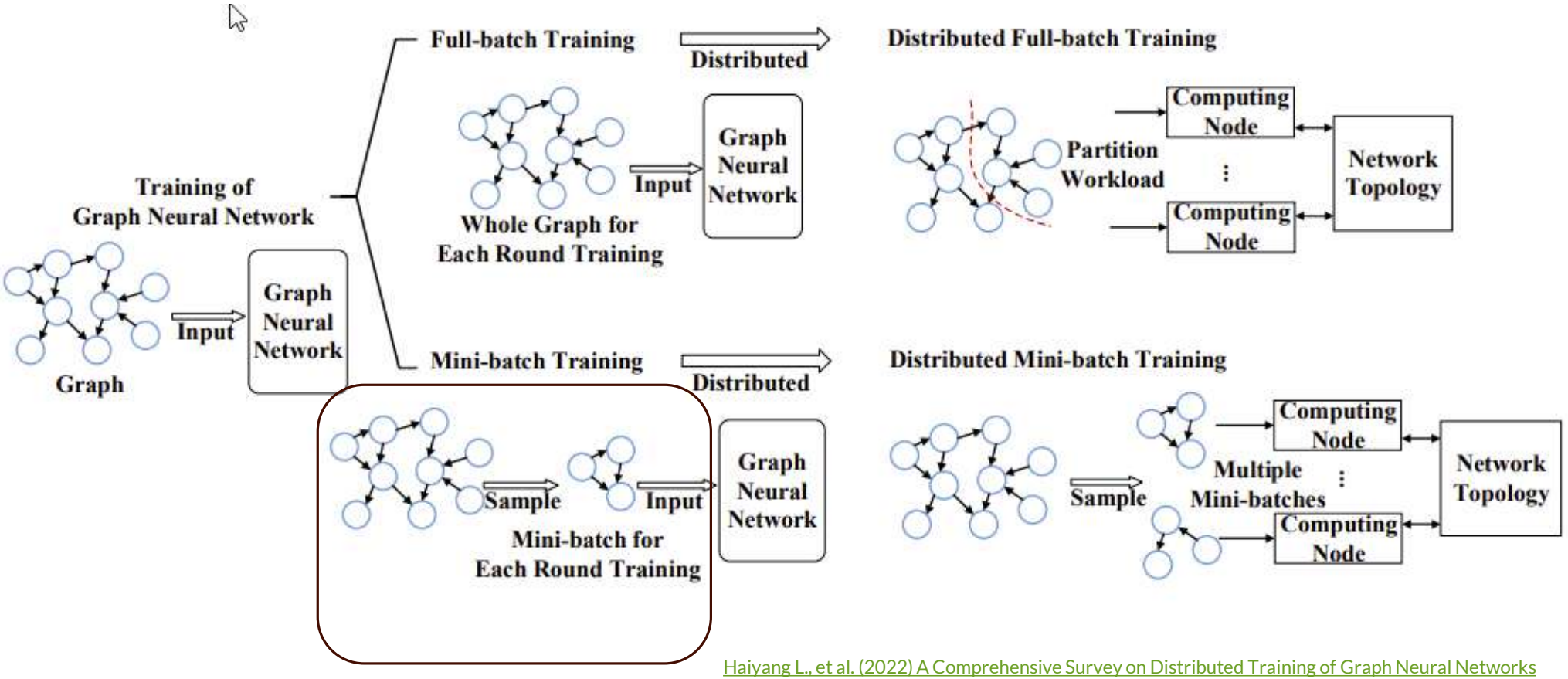
Haiyang L., et al. (2022) A Comprehensive Survey on Distributed Training of Graph Neural Networks

Graph Partitioning GNN Training

- Samples are partitioned across batches when the graph does not fit in the device's memory
- Each node and/or edge belongs to one partition
- There is no overlap between partitions
- Colors indicate partition



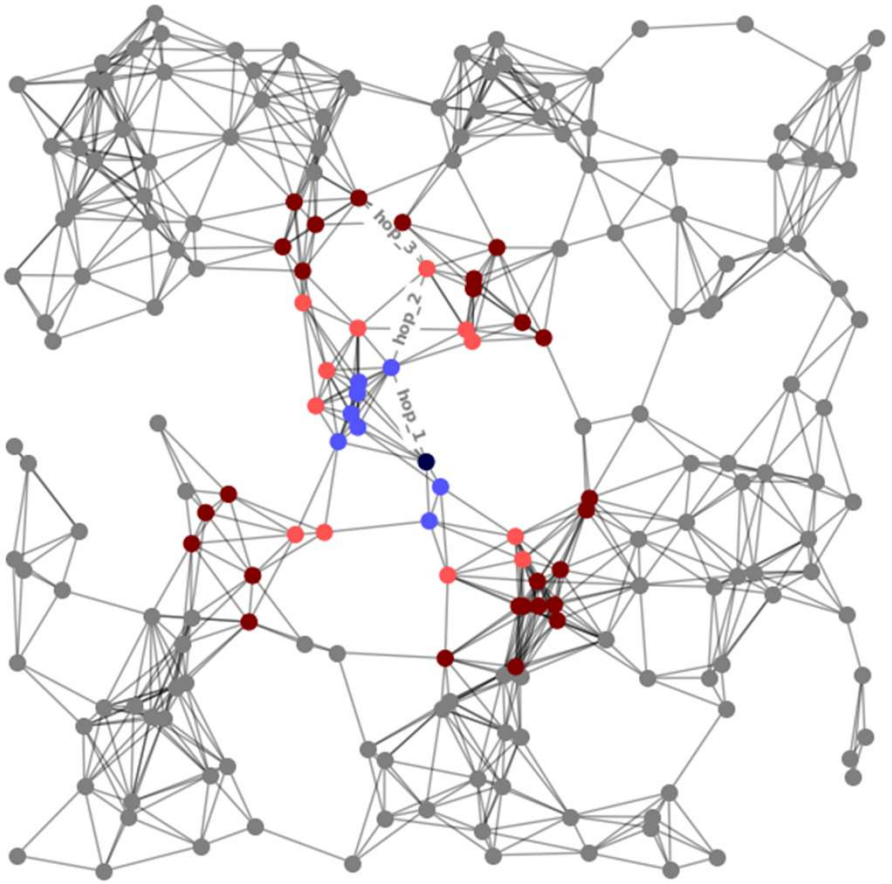
Partitioning versus Mini-Batch Schemes for GNN Training



Haiyang L., et al. (2022) A Comprehensive Survey on Distributed Training of Graph Neural Networks

Mini-batch GNN Training

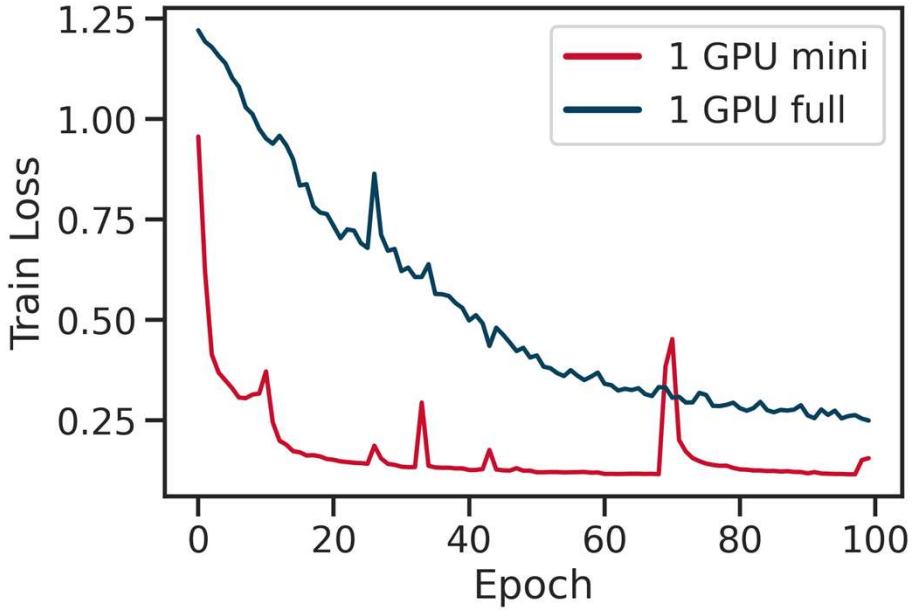
- Sample-based training first samples the graph to build mini-batches
- Sampling starts by selecting random subsets of nodes, edges, or subgraphs to be included in the mini-batch
- In a GNN model with *n* layers, each mini-batch includes the input features of the **n-hop neighborhood** of those target nodes
- There is overlap between the mini-batches
- Once the mini-batches are generated, distributed training can be applied

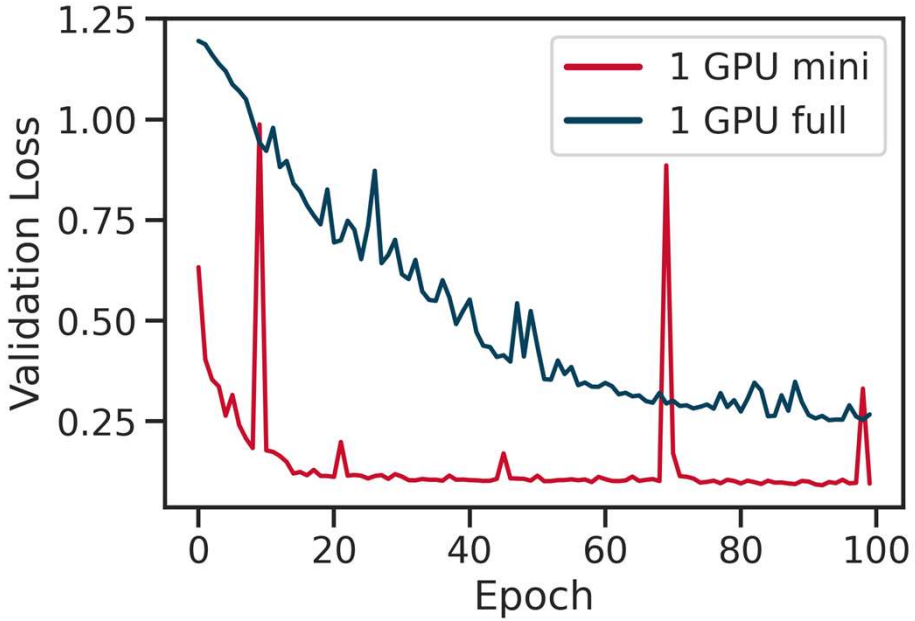


Training and Validation Loss Results for Mini-batch GNN training

- 80 events for training, 10 for validation, and 10 for testing
- Average number of nodes 84k ± 9k
- Average number of edges 150k ± 30k
- Number of nodes in subgraph: 2048

Full-batch – 80 batches
Mini-batch – 3294 batches

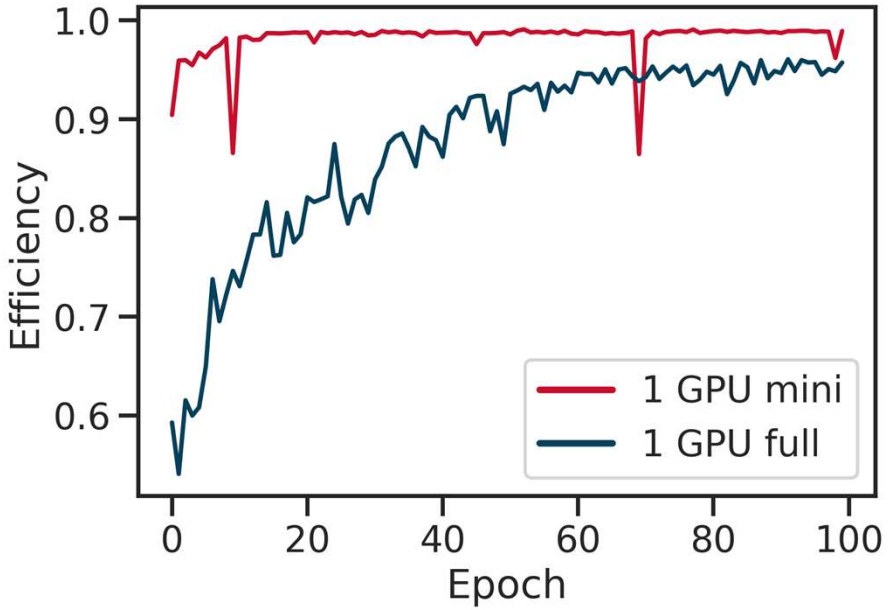




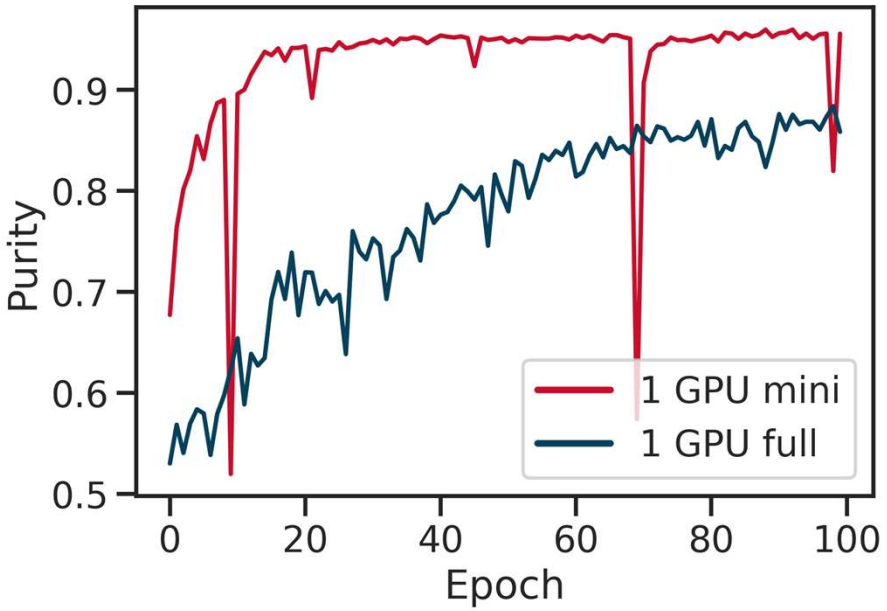
Efficiency and Purity - Full vs Mini-batch GNN training

Mini-batch training produces better models than full-batch training

Full-batch best efficiency: 0.957
Mini-batch best efficiency: 0.989



Full-batch best purity: 0.856
Mini-batch best purity: 0.956

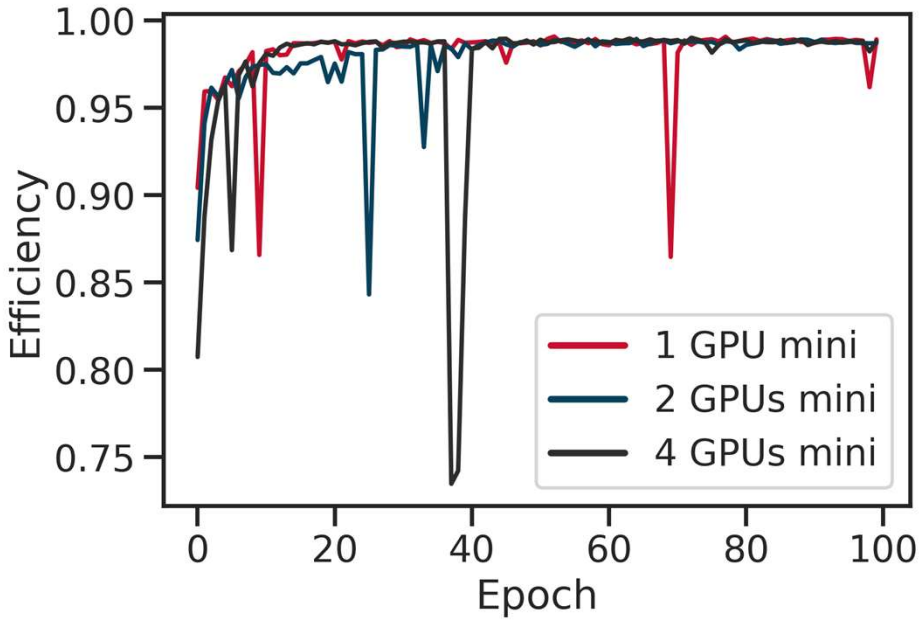


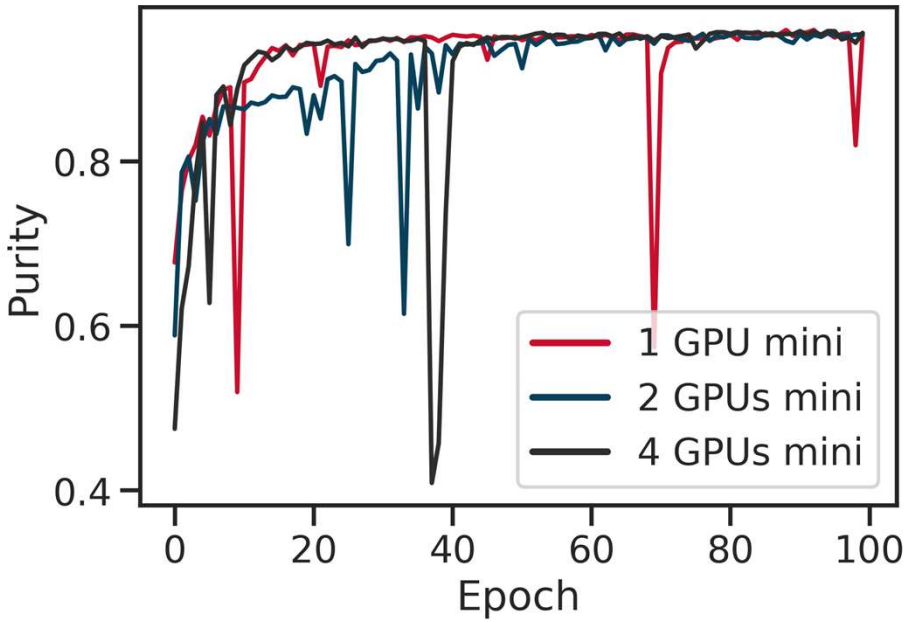
Efficiency and Purity - Mini-batch DDP GNN training

Mini-batch best efficiency:
- 1 GPU – 0.989
- 2 GPUs – 0.987
- 4 GPUs – 0.988

Mini-batch sizes:
- 1 GPU – 3.2k
- 2 GPUs – 1.6k
- 4 GPUs – 0.8k

Mini-batch best purity:
- 1 GPU – 0.956
- 2 GPUs – 0.954
- 4 GPUs – 0.956





Mini-batch training scales better to multi-GPUs than full-batch training

Recipe to Scale Training of GNNs

Start with a model which trains well on a single GPU

Optimize the single-node / single-GPU performance

- Using performance analysis tools
- **Tuning and optimizing the data pipeline (HPO)**
- Make effective use of the hardware (e.g. mixed precision)

Distribute the training across multiple processors

- Multi-GPU, multi-node training: data and/or model parallel
- Use best practices for large scale training and convergence
- Use best optimized libraries for communication, tune settings

Conclusions and Future Work

- **Scaling GNN training is challenging!**
- **Weak** scaling can be done at event level
- **Strong** scaling degrades **physics performance** and requires graph sampling

- Graph sampling (mini-batches) improves the performance of GNN training
- Scaling graph sampling-based training requires:
  - Sampling algorithms that can form mini-batches without losing information
  - Systems that can execute these algorithms efficiently

- Further testing and tuning of the sampling and partitioning methods is needed
- Sampling and data loading are expensive
- Sampling only works for node and edge-level tasks and requires special implementation for long-range dependencies

- How to distribute and store the graph data?
- How to transfer batches in and out of the GPUs to minimize the data transfers?

Thanks to:

- Ivan Ladutska
- Brenden Reeves
- Tuan Minh Pham
- Jay Chan
- Daniel Murnane
- Xiangyang Ju
- Paolo Calafiura

Thank you!

The Relationship between Batch Size and Learning Rate

- Batch size is a **hyperparameter**
- A larger batch size allows computational **speedups** from the parallelism of GPUs
- Too large of a batch size leads to poor generalization
- A batch equal to the entire dataset guarantees convergence to the global optima
- A smaller batch size has been shown to have **faster convergence**
- The downside of using a smaller batch size is that the model is not guaranteed to converge to the global optima

ImageNet Dataset

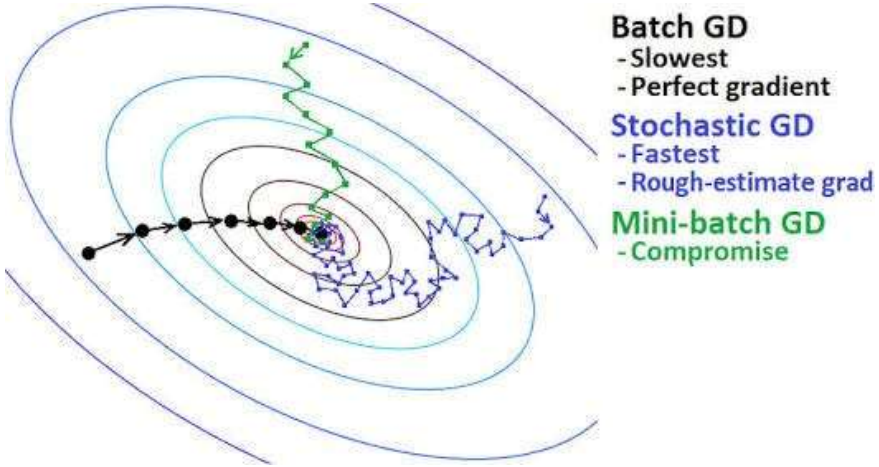
Learning Rate Scaling Rule

Learning Rate Scaling Rule: When the batch size is multiplied by *k*, multiply the learning rate by *k*.

Mini-batch Stochastic Gradient Descent:

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

η is the learning rate
$\mathcal{B}$ is the mini-batch



Typical practice/suggestion:

- Keep local batch size per worker the same
- Increase the global batch size linearly with the number of devices
- Increase the learning rate proportionally: $lr_{scale} = lr * num\_devices$

McCandlish, Sam, et al. "An empirical model of large-batch training." *arXiv preprint arXiv:1812.06162* (2018).