Scalable Graph Neural Network training using HPC and supercomputing facilities

Massimiliano (Max) Lupo Pasini

Jong Youl Choi

Pei Zhang

Oak Ridge National Laboratory

Who we are

Massimiliano (Max) Lupo Pasini
Data Scientist
lupopasinim@ornl.gov

Computational Sciences and Engineering Division (CSED)

Oak Ridge National Laboratory

Jong Youl Choi
Computer Scientist
choij@ornl.gov

Computer Science and Mathematics Division (CSMD)

Oak Ridge National Laboratory

Pei Zhang
Computational Scientist
zhangp1@ornl.gov

Computational Sciences and Engineering Division (CSED)

Oak Ridge National Laboratory

Outline

- Introduction
- Scalable GNN Training
- HydraGNN
- Hand-on Session
- Conclusion

Oak Ridge National Laboratory

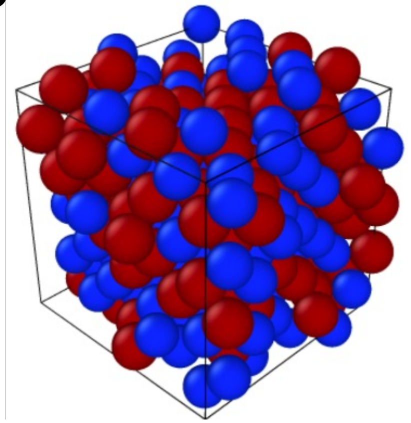
Introduction

Motivation – US DoE scientific applications

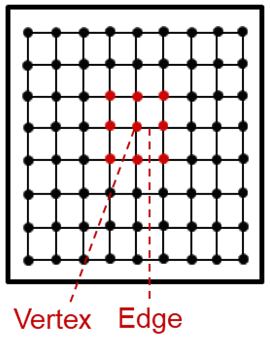
- **Scientific computing calculations can be computationally expensive** and take several wall-clock hours on distributed computing HPC platforms

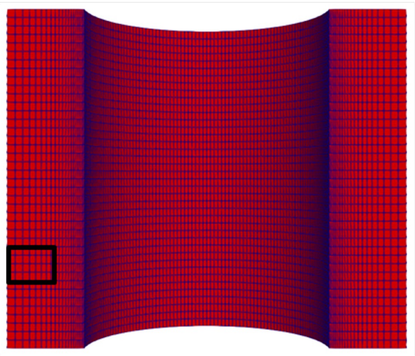
- **Surrogate models can mitigate the computational cost** of expensive large-scale scientific computing applications **while maintaining sufficient accuracy**

- For several **scientific computing** problems, the structure of the **physical system can be mapped onto a graph**



atomistic materials modeling





finite element simulations

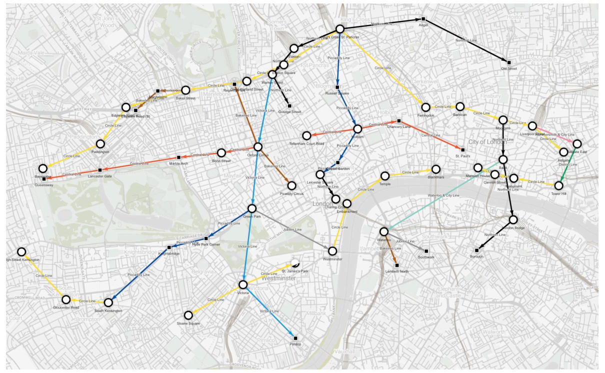


Image from https://memgraph.com/blog/modeling-visualizing-navigating-a-transportation-network-with-memgraph

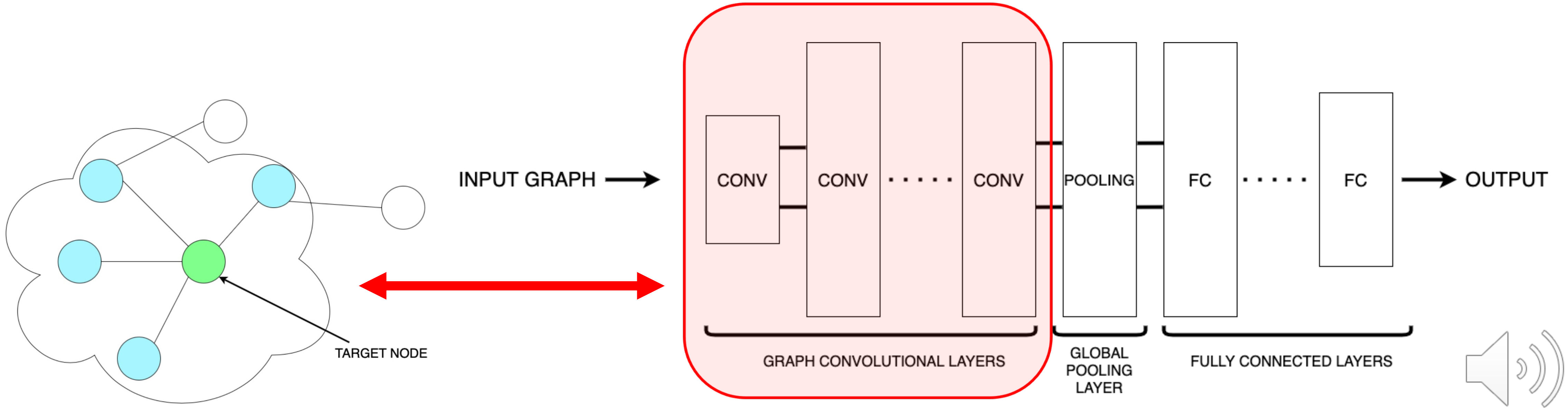
urban sciences
(e.g., transportation and power grid)

- Whenever the data can be expressed in the format of a graph, **graph neural networks (GNNs)** have been identified as promising tools to **extract relevant nodal and graph-level features** that describe the dynamics of the physical system

Graph Neural Networks (GNNs)

The architecture of a GNN is made of:

1. a graph embedding layer
2. hidden graph layers aim at capturing short range interactions between nodes in the graph
3. pooling layers interleaved with graph layers synthetize information related to adjacent nodes via aggregation
4. fully connected (FC) dense layers at the end of the architecture to capture effects that global features of the graph have over the target properties of interest



Convolutional operations aggregate information from neighboring nodes

Limitations of open-source GNN implementations

Popular open-source GNN implementations lack vital features, hindering their full-scale application to computational chemistry.

In particular these libraries do not simultaneously support:

(1) multi-task learning (MTL), which is used to effectively stabilize the training by taking advantage of implicit correlations between multiple target properties of interest;

(2) seamless replacement of MPNNs without drastically and disruptively re-implement a significantly large portion of the original code;

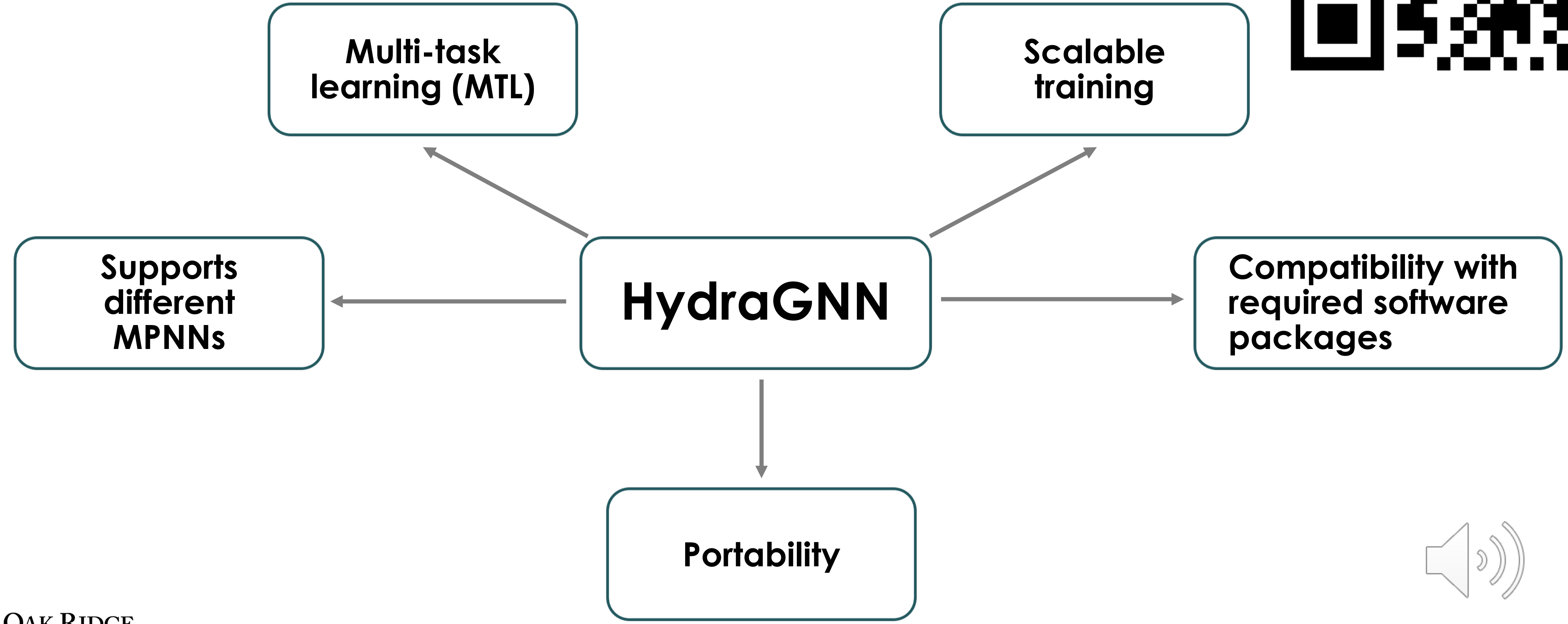
(3) distributed data parallelism (DDP) effectively implemented to address scaling challenges on large-scale supercomputing facilities;

(4) regular software maintenance to ensure appropriate updates of the software packages required to run the code.

(5) portability across diverse hardware architectures

HydraGNN: Distributed PyTorch Implementation of Multi-Headed GNNs

https://www.osti.gov/doecode/biblio/65891
https://github.com/ORNL/HydraGNN

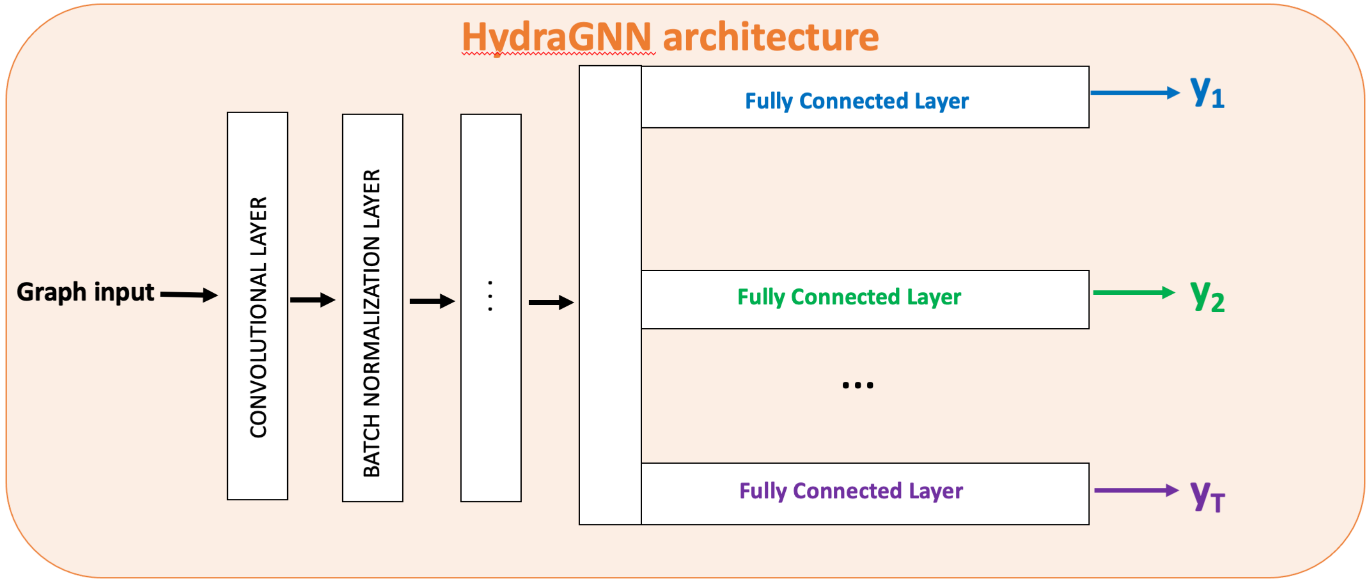

Multi-task learning (MTL)
Scalable training
Supports different MPNNs
HydraGNN
Compatibility with required software packages
Portability

HydraGNN: Multi-task learning (MTL) for stabilization by extracting physics correlations between multiple target properties of interest

Multi-Task Learning stabilizes predictions of multiple properties
Each property operates as a mutual regularizer to stabilize the prediction of other properties

Quantities simultaneously predicted:

- **Property y_1**
- **Property y_2**
- …
- **Property y_T**



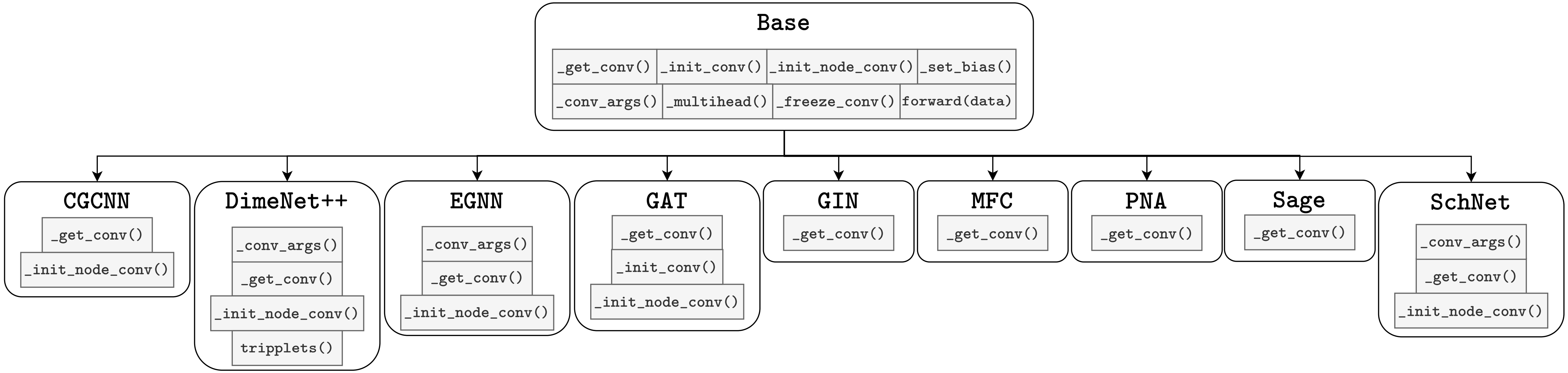
$\mathbf{w}$ = parameters of the neural network to optimize during the training

$$\underset{\mathbf{w}}{\operatorname{argmin}} \underbrace{\boxed{\|\mathbf{y}_{\text{predict},1}(\mathbf{w}) - \mathbf{y}_1\|_2^2} + \boxed{\|\mathbf{y}_{\text{predict},2}(\mathbf{w}) - \mathbf{y}_2\|_2^2} + \ldots + \boxed{\|\mathbf{y}_{\text{predict},T}(\mathbf{w}) - \mathbf{y}_T\|_2^2}}_{\text{Global Multi-Task Training Loss Function}}$$

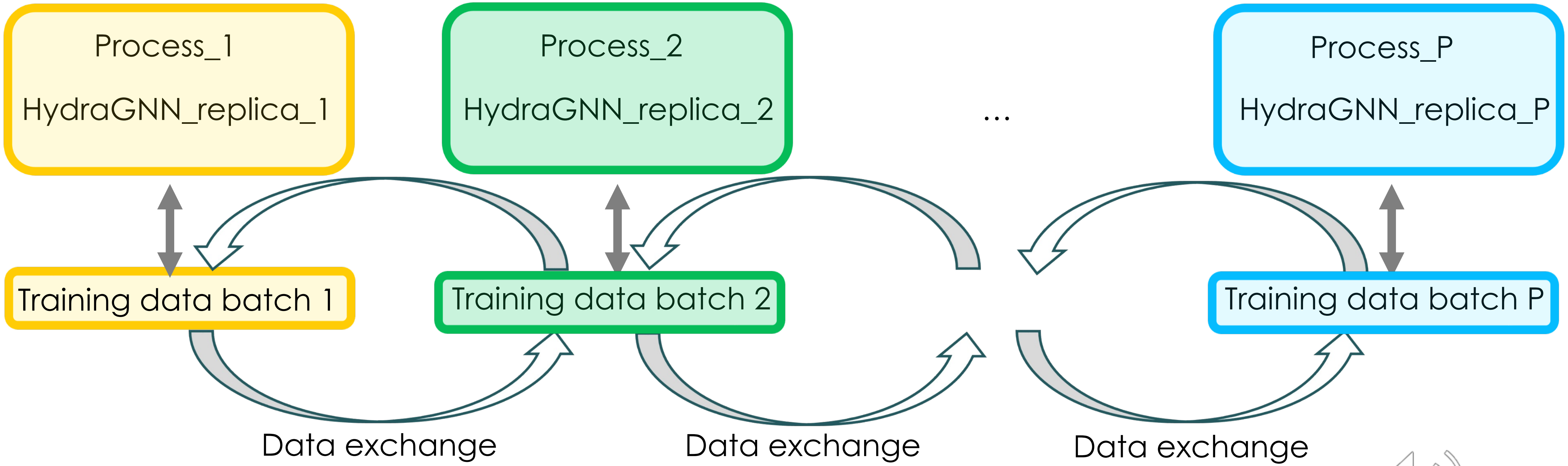
Global Multi-Task Training Loss Function

HydraGNN: Message passing layer treated as hyperparameter

Object-oriented programming enables seamless switch between different MPNN layers that can be treated as hyperparameters

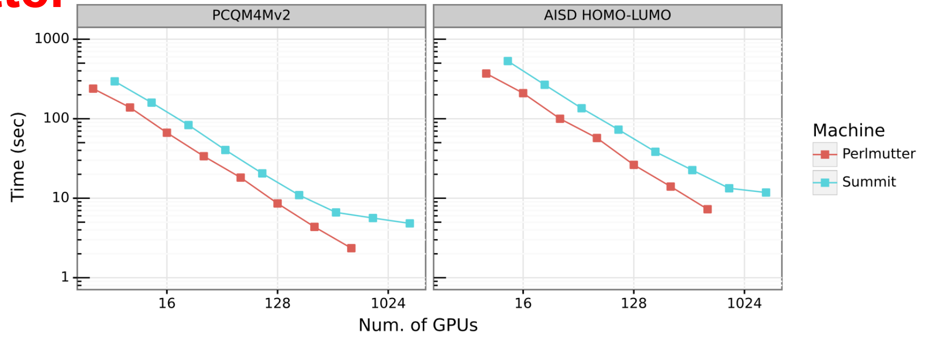


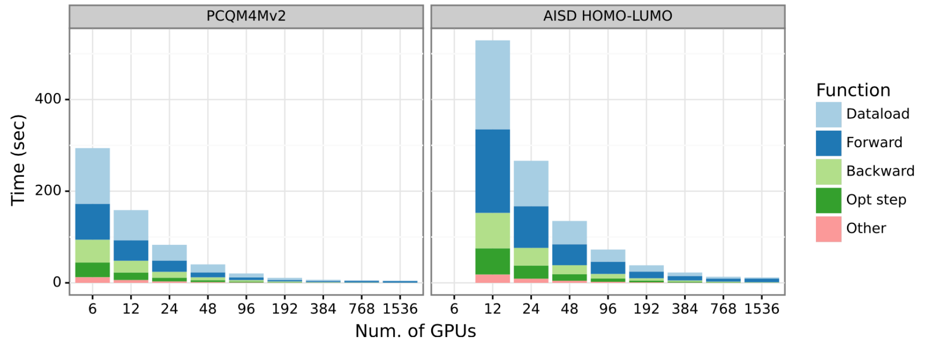
HydraGNN: Scalable training with Distribute Data Parallelism (DDP)



HydraGNN: Scalable training with Distribute Data Parallelism (DDP)

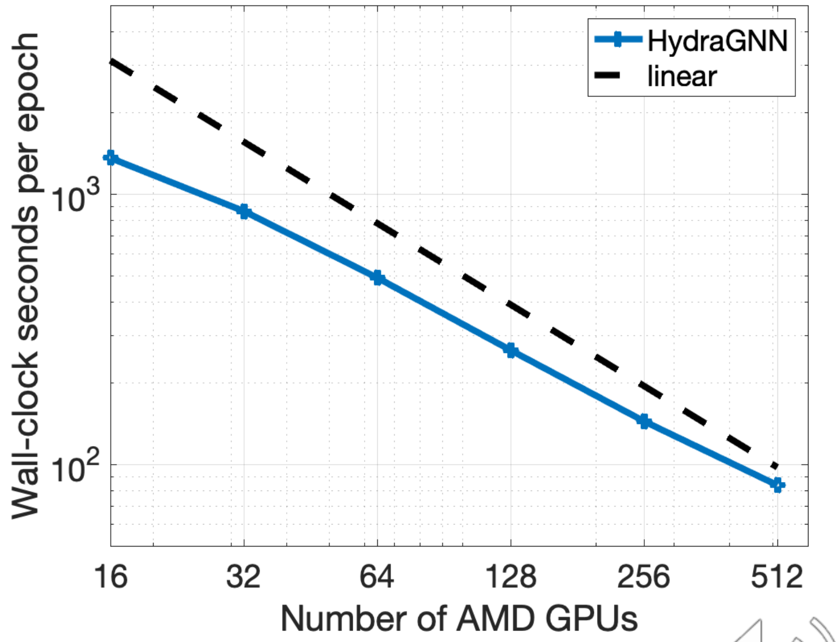
Results: linear scaling of data reading + training using up to 1,024 NVIDIA V100 GPUs on OLCF Summit and 1,024 NVIDIA GPUs on NERSC Perlmutter



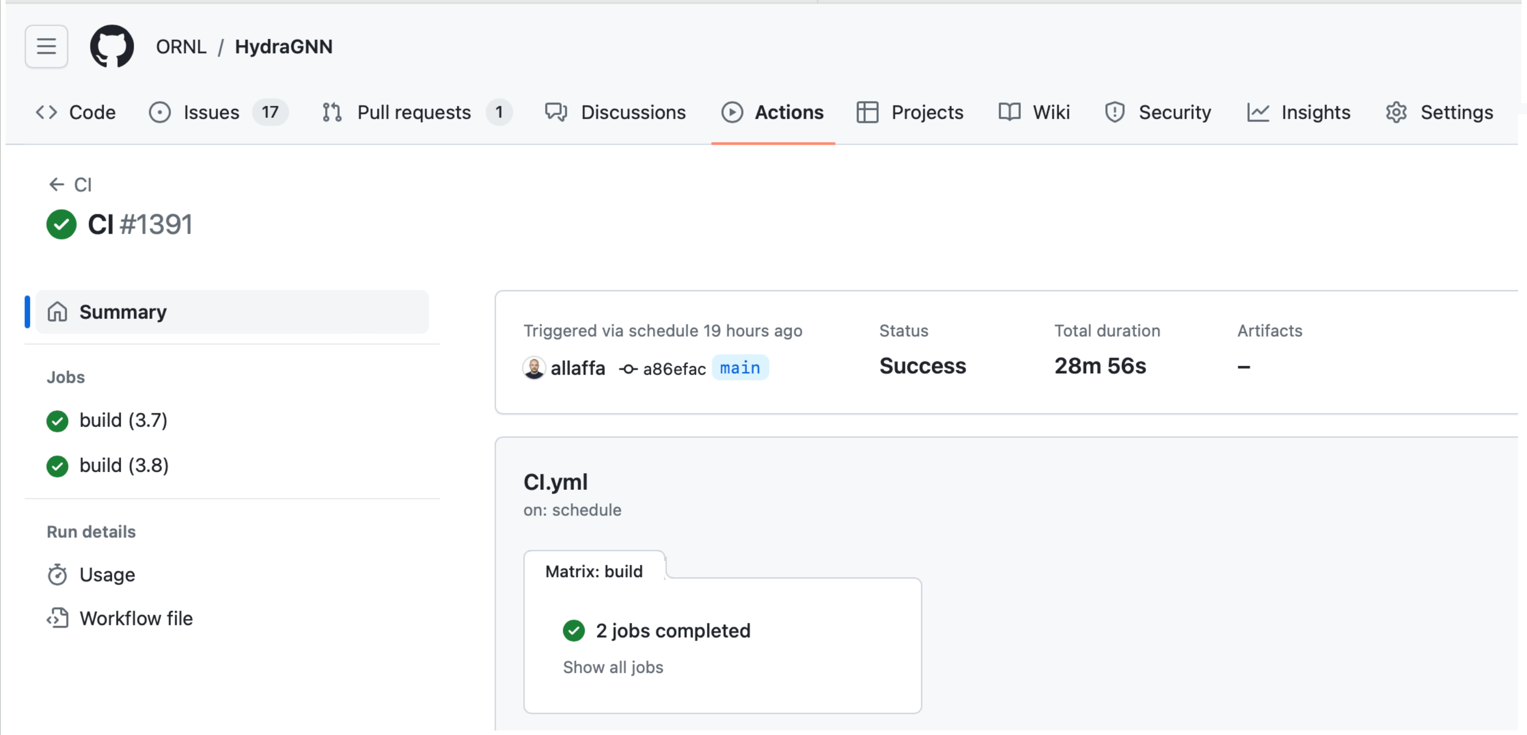


Result:
Scaling of HydraGNN almost linear using 512 AMD MI250X GPUs of OLCF-Crusher

PCQM4Mv2 dataset (> 3 million organic molecules)



HydraGNN: Compatibility with required software packages



ORNL / HydraGNN

Code | Issues 17 | Pull requests 1 | Discussions | Actions | Projects | Wiki | Security | Insights | Settings

← CI

CI #1391

Summary

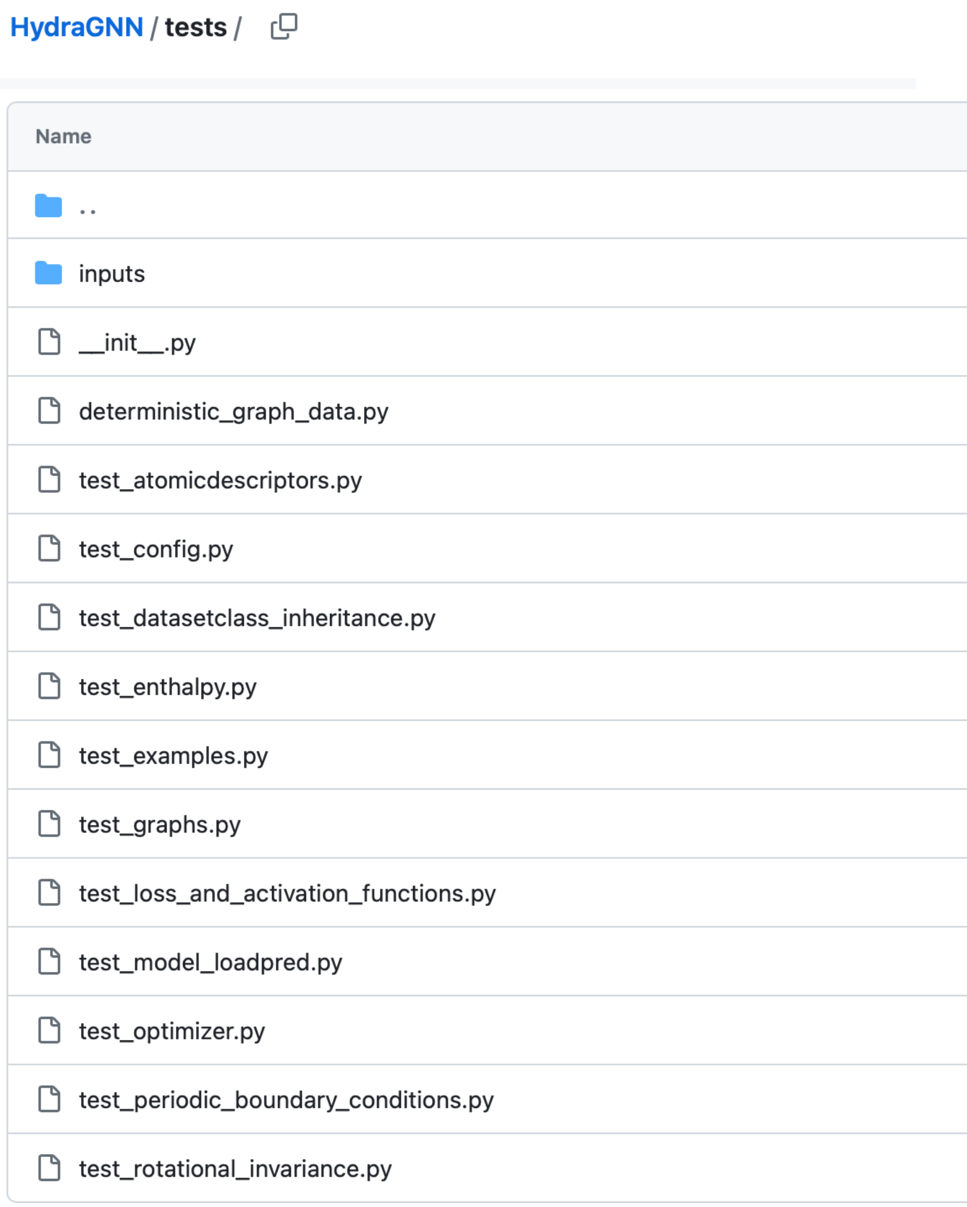
Jobs
- build (3.7)
- build (3.8)

Run details
- Usage
- Workflow file

Triggered via schedule 19 hours ago	Status	Total duration	Artifacts
allaffa a86efac main	Success	28m 56s	–

CI.yml
on: schedule

Matrix: build
2 jobs completed
Show all jobs



HydraGNN / tests /

Name
..
inputs
__init__.py
deterministic_graph_data.py
test_atomicdescriptors.py
test_config.py
test_datasetclass_inheritance.py
test_enthalpy.py
test_examples.py
test_graphs.py
test_loss_and_activation_functions.py
test_model_loadpred.py
test_optimizer.py
test_periodic_boundary_conditions.py
test_rotational_invariance.py

Continuous integrations tests on the GitHub repo ensure software sustainability

HydraGNN: Portability across Diverse Computing Platforms

HydraGNN functionalities are regularly tested on a broad set of computing architectures:

- Personal laptops for small scale training
- ORNL Edge Computing DGX boxes using docker containers
- OLCF CADES clusters using conda environments: https://www.olcf.ornl.gov/tag/cades/
- OLCF supercomputer Summit (NVIDIA V100 GPUs): https://www.olcf.ornl.gov/summit/
- NERSC supercomputer Perlmutter (NVIDIA A100 GPUs): https://docs.nersc.gov/systems/perlmutter/
- OLCF Crusher (AMD Instinct 250X GPUs): https://www.olcf.ornl.gov/tag/crusher/
- OLCF supercomputer Frontier (AMD Instinct 250X GPUs): https://www.olcf.ornl.gov/frontier/
- University of Tsukuba supercomputer Pegasus (NVIDIA H100 GPUs): https://www.ccs.tsukuba.ac.jp/wp-content/uploads/sites/14/Pegasus.pdf
- Groq technology: https://groq.com

Scalable GNN training

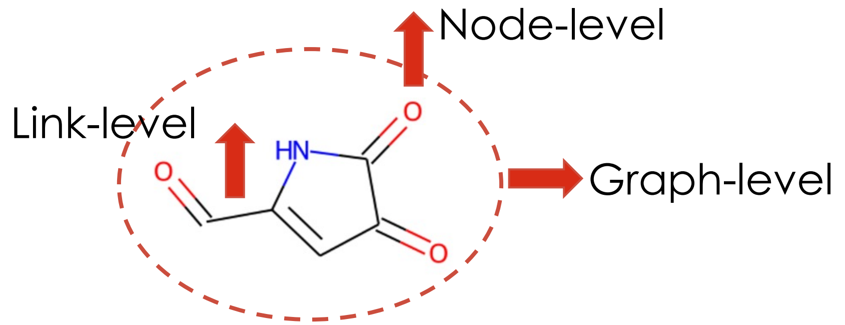
Motivation

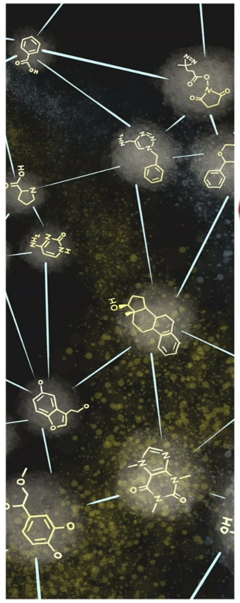
In scientific applications like atomistic materials modeling, the GCNN must be accurate and robust in a high-dimensional parameter space to model very diverse configurations.

Example - Atomistic materials modeling

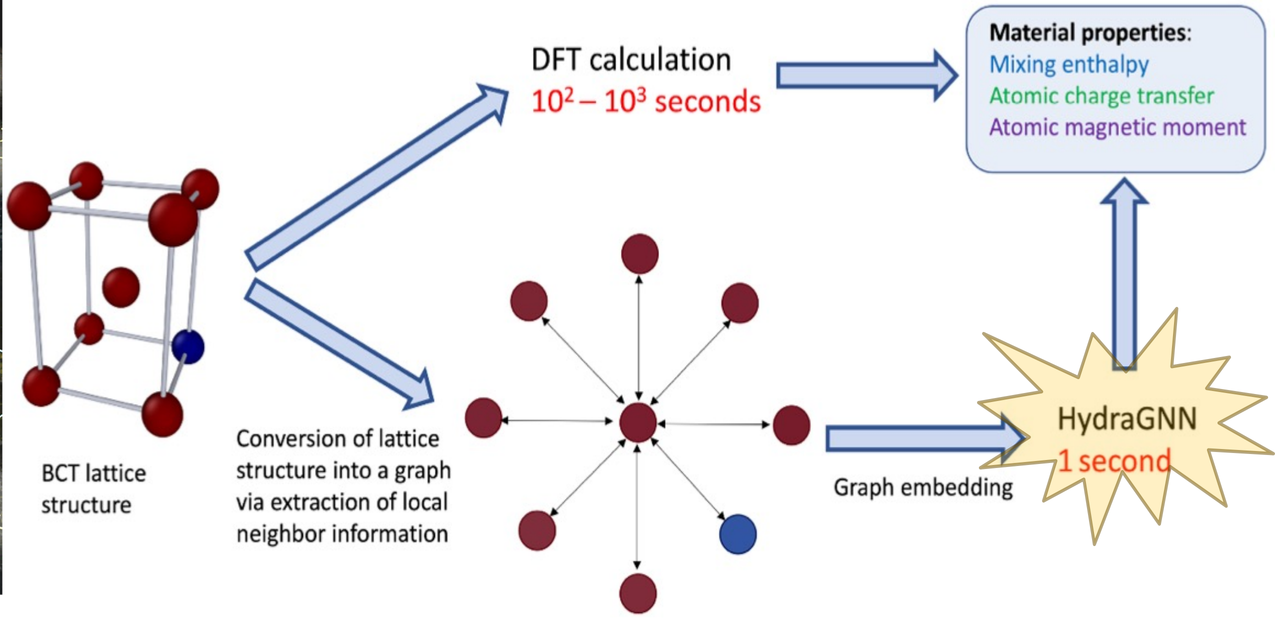
(1) chemical composition, and

(2) arrangement of atoms of different constituents





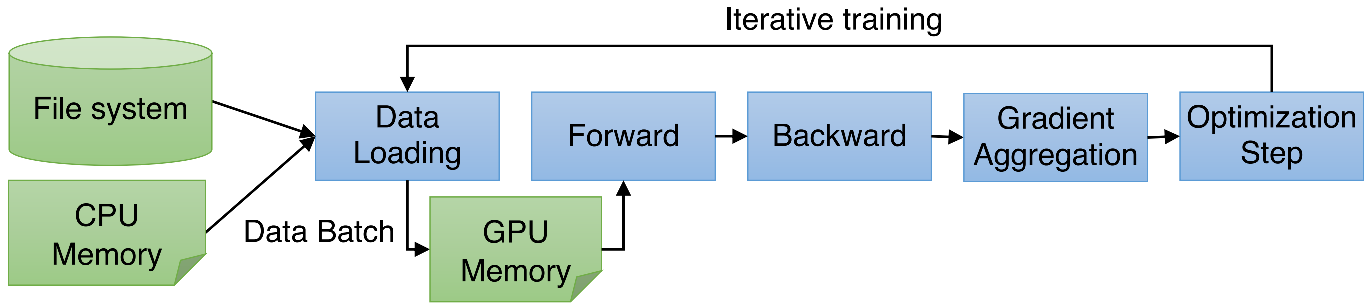
Chemical Space,
Naomi Johnson and Lee Cronin



This requires training the GCNN model on **large volumes of graph data,** which makes the training both computationally, memory, and I/O intensive.

GNN I/O Challenges

- GNN I/O characteristics
  - **Read-oriented**
  - **Frequent** access: (e.g., 100 epochs per hour)
  - **Shuffled** access to improve generalization or to avoid overfitting

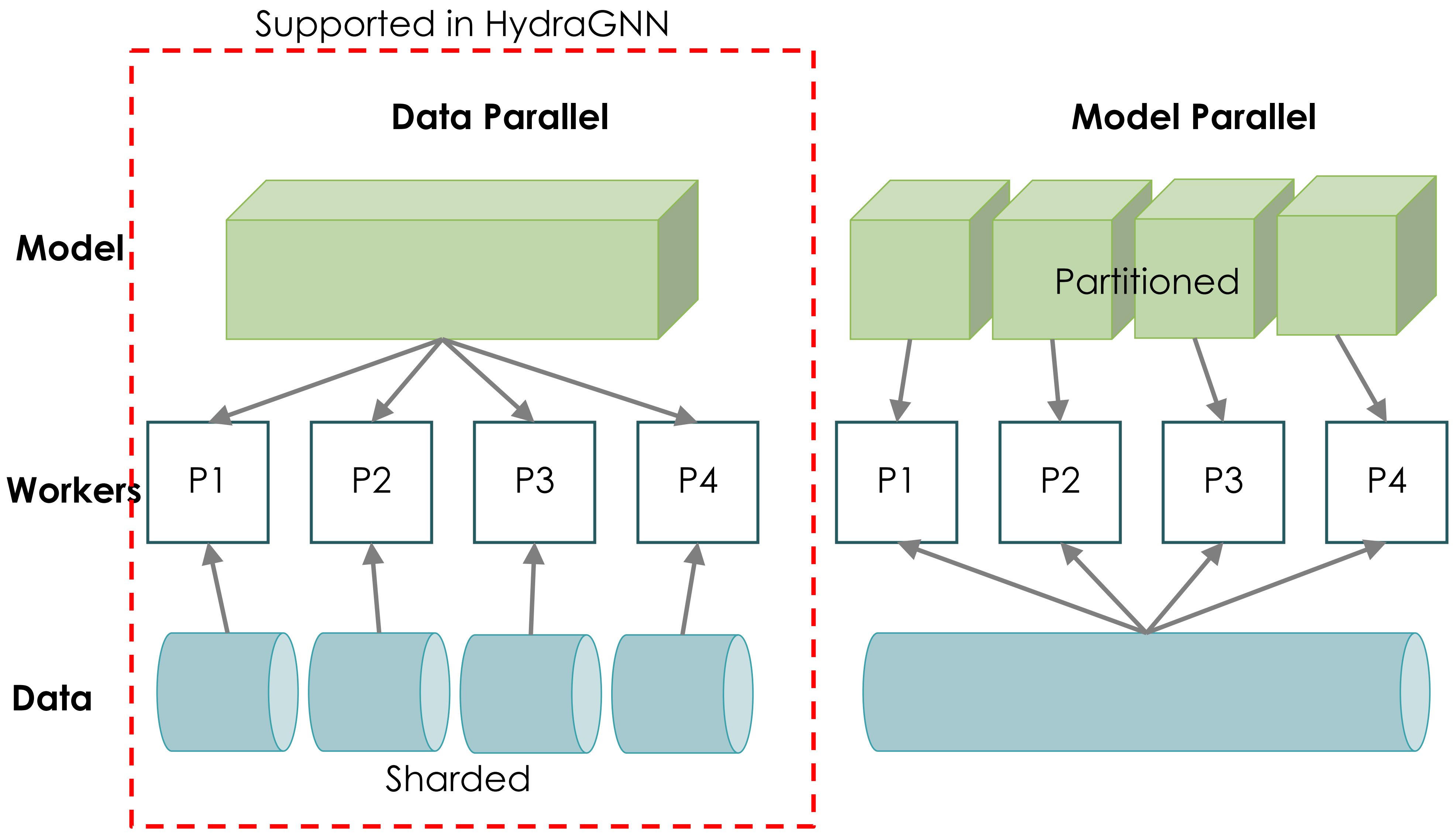


Iterative GNN training: Data loading is one of the key steps for performance

File per molecule	**NVME/Node-local SSD**	**Sharding**
• 10s of millions of files • Large meta data • Huge stress on filesystem • Multiple requests to increase space/node quotas	• Non-negligible setup time • Total (N nodes x data size) byte transfer	• Flexibility issue • May limit the quality of training

Open slide master to edit

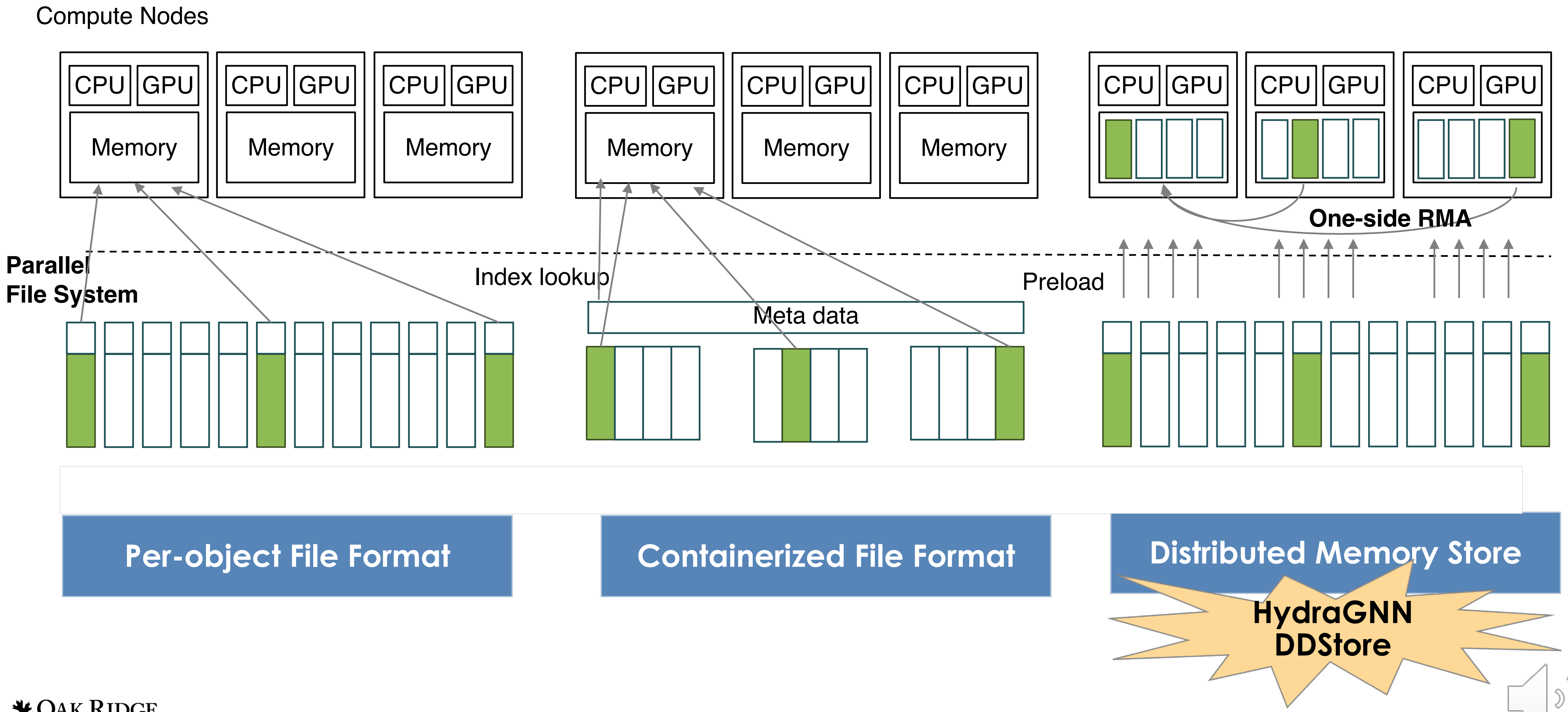
Parallelisms: Scalable GNN Training Strategies



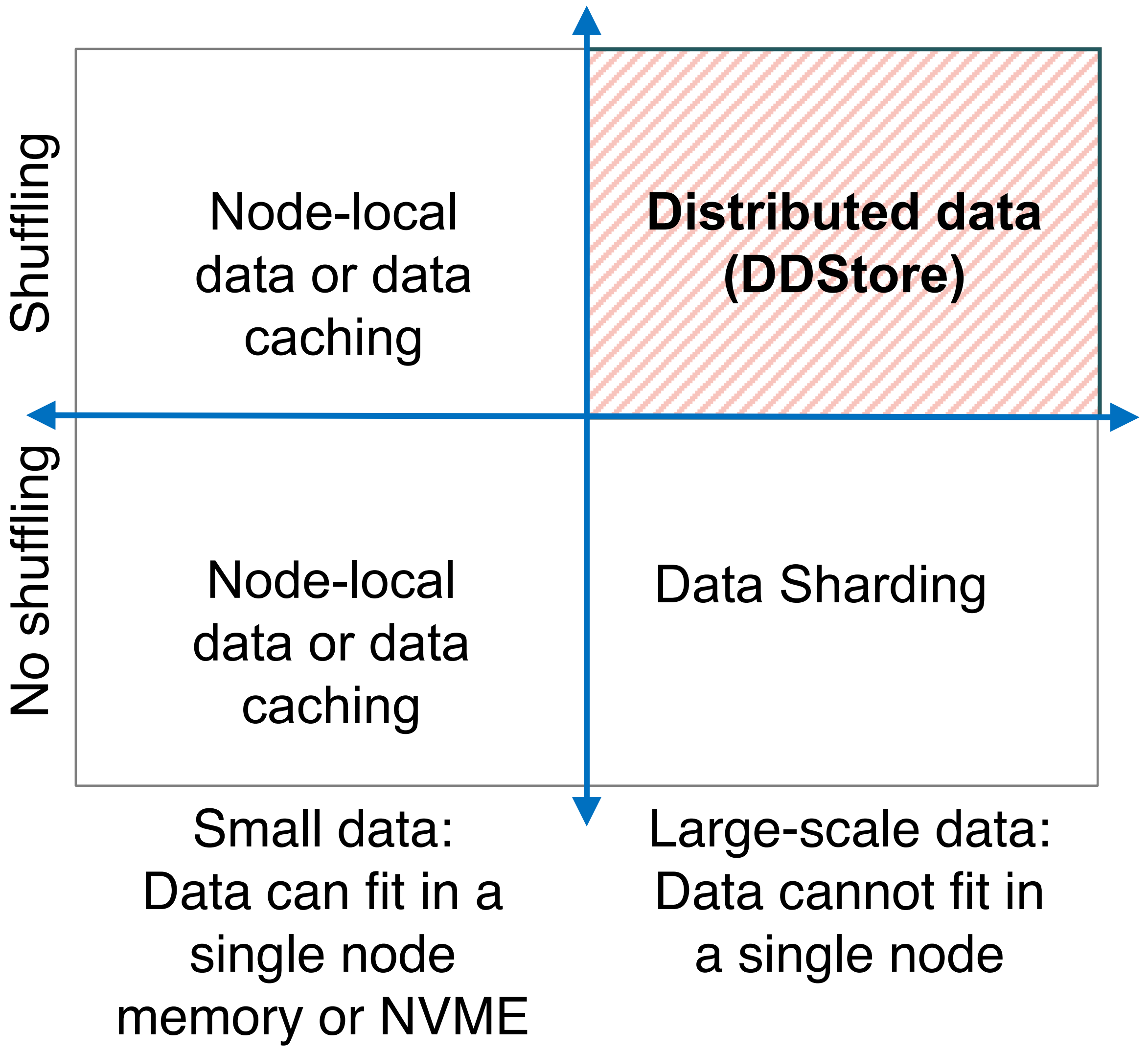
And many more:

- Pipeline
- Hybrid
- Tensor
- Spatial
- Layer
- Sequence
- …

Data Loading Strategy



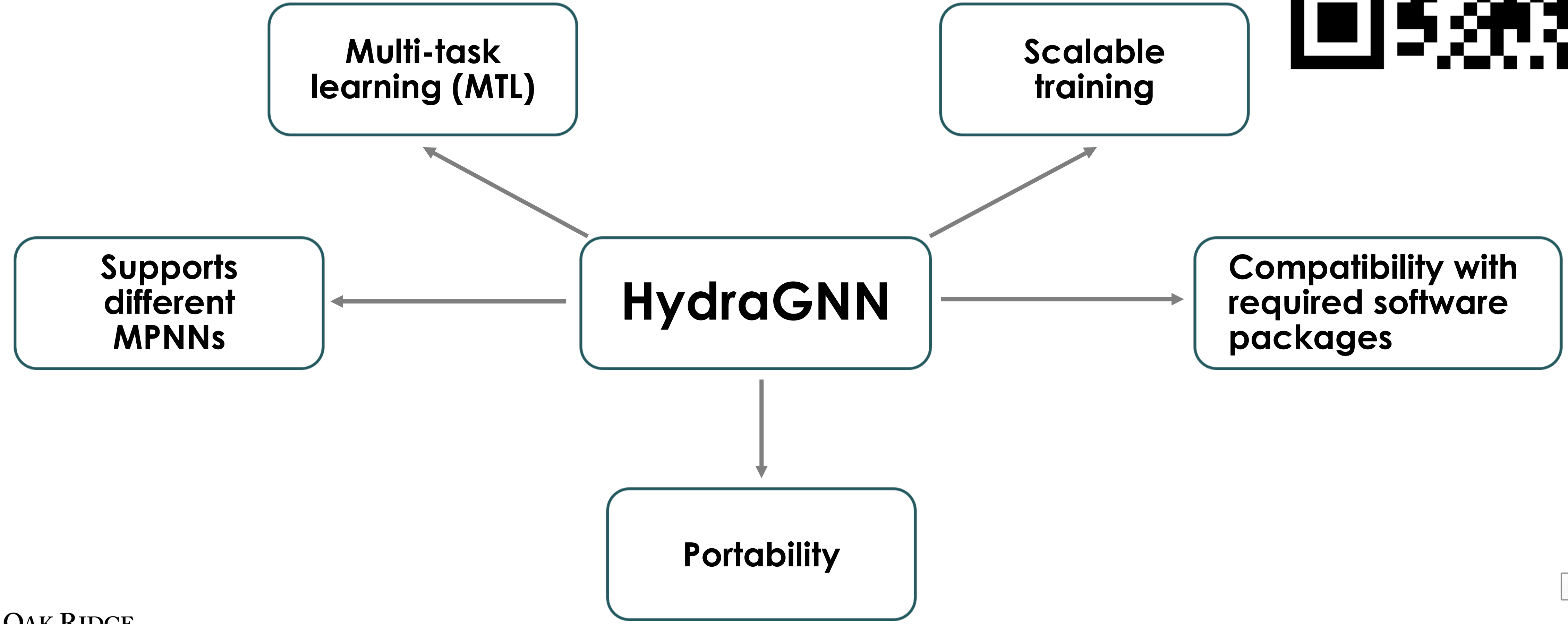
DDStore Mileage


Shuffling
No shuffling
Node-local data or data caching
Distributed data (DDStore)
Node-local data or data caching
Data Sharding
Small data: Data can fit in a single node memory or NVME
Large-scale data: Data cannot fit in a single node

HydraGNN

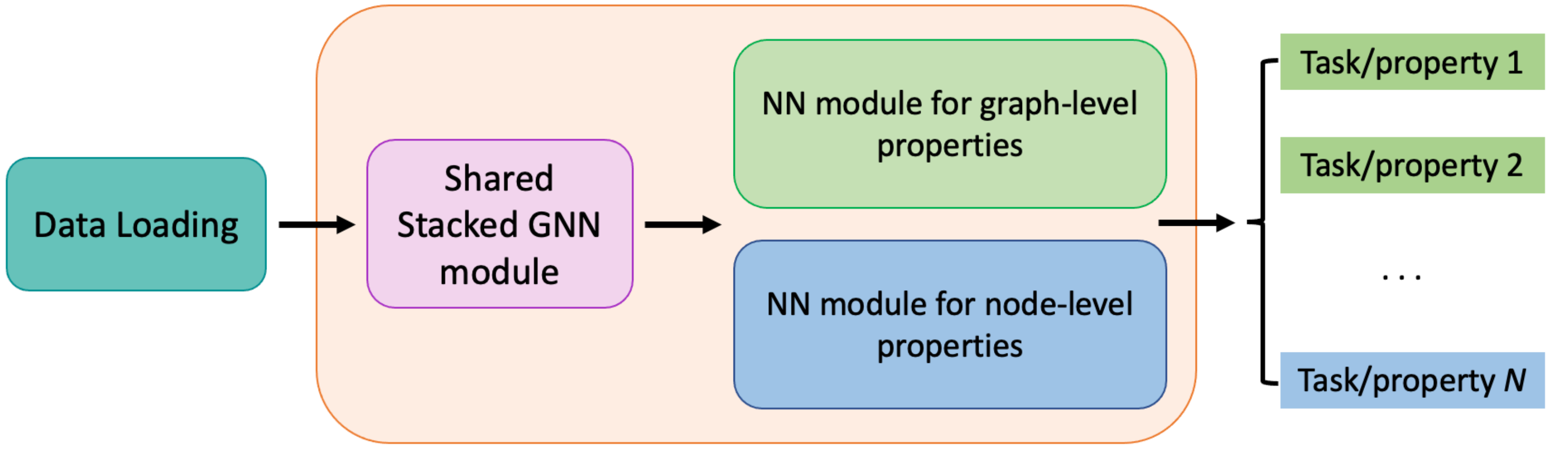
HydraGNN: enabling large-scale GNN training on HPC

https://www.osti.gov/doecode/biblio/65891
https://github.com/ORNL/HydraGNN


HydraGNN
Multi-task learning (MTL)
Scalable training
Supports different MPNNs
Compatibility with required software packages
Portability

Overview

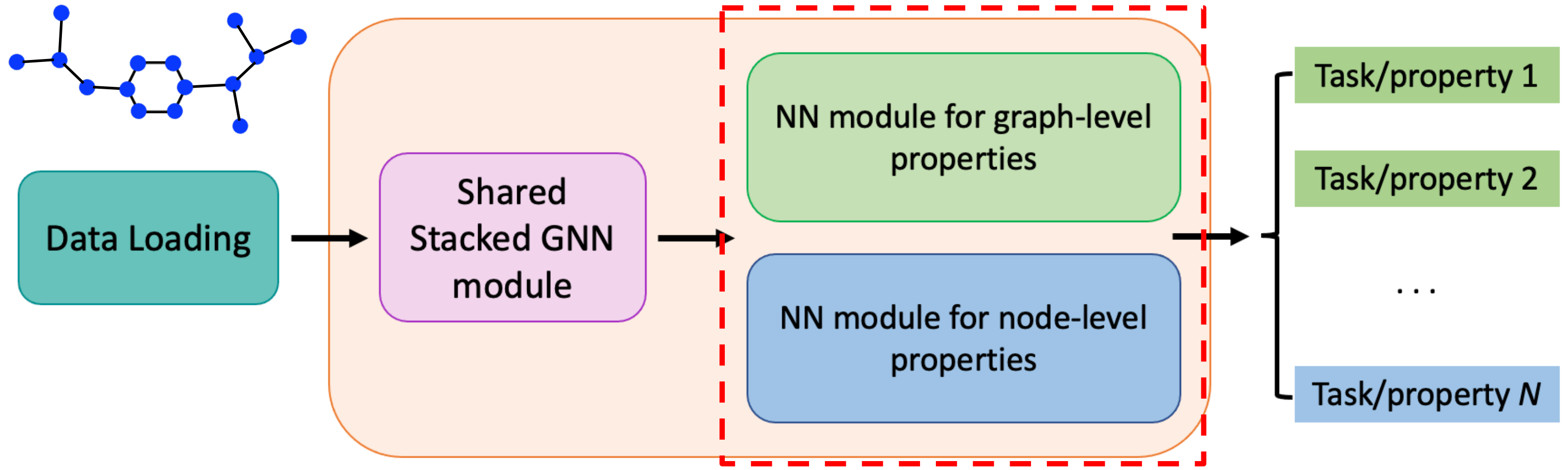
- **Multitasking** heads for better data efficiency
- **Message passing layers** treated as hyperparameter
- **Setting up** HydraGNN via a configuration json file
- Multiple **data** loaders/file format to support scalability
- **Data** file formats
- DDStore (Scalable Distributed **Data** Store)


Data Loading
Shared Stacked GNN module
NN module for graph-level properties
NN module for node-level properties
Task/property 1
Task/property 2
...
Task/property N

(Zhang et al., TMS, 2022)

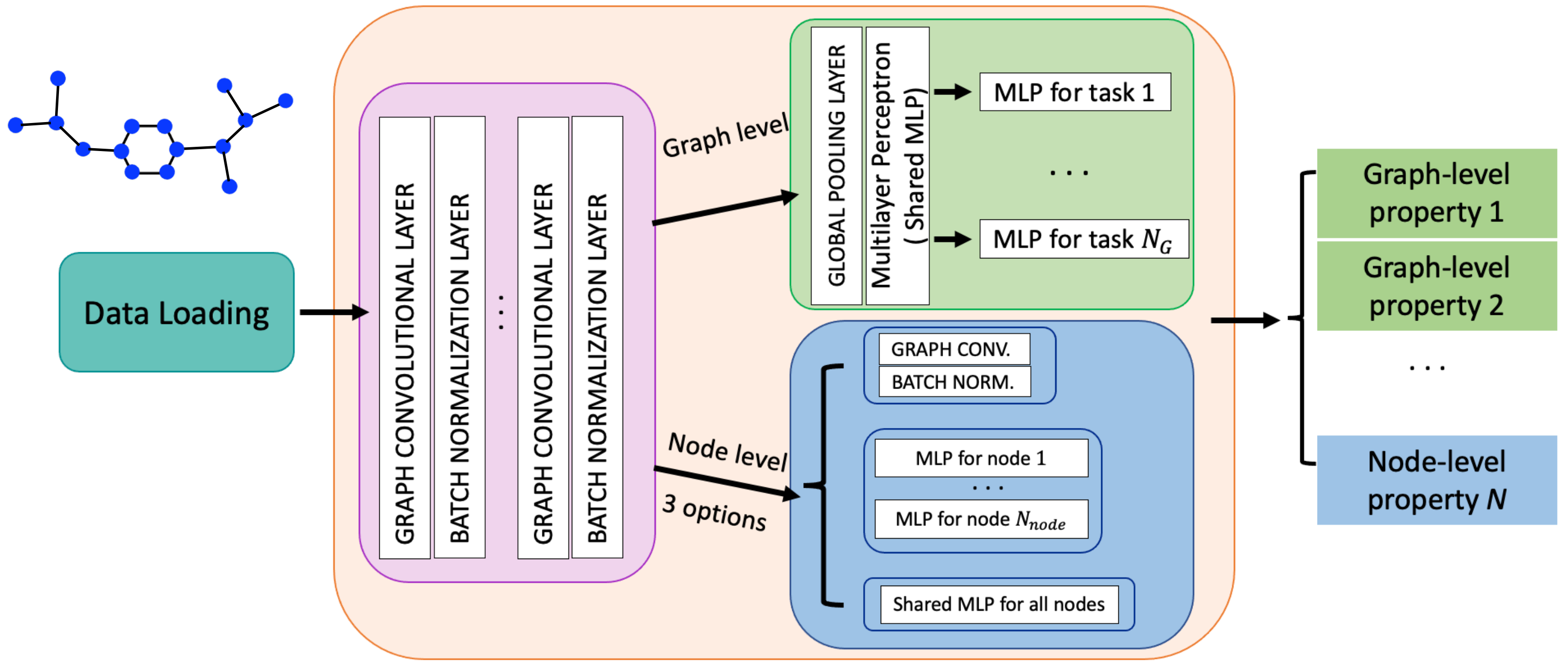
Multitasking for improved data efficiency

- Joint learning of multiple properties
  - **Input:** Graph representation (node feature, edge feature, adjacent matrix)
  - **Output**: Regression targets (node level, graph level)
- Inherently sharing features across learning tasks → Improved prediction accuracy
- Improved generalization/reduced overfitting
- Saved training time and improved training stability
- E.g., FePt (Lupo Pasini et al., 2022)
  - mixing enthalpy (global), charge transfer and magnetic moment (atomic/node) in FePt


Data Loading
Shared Stacked GNN module
NN module for graph-level properties
NN module for node-level properties
Task/property 1
Task/property 2
...
Task/property N

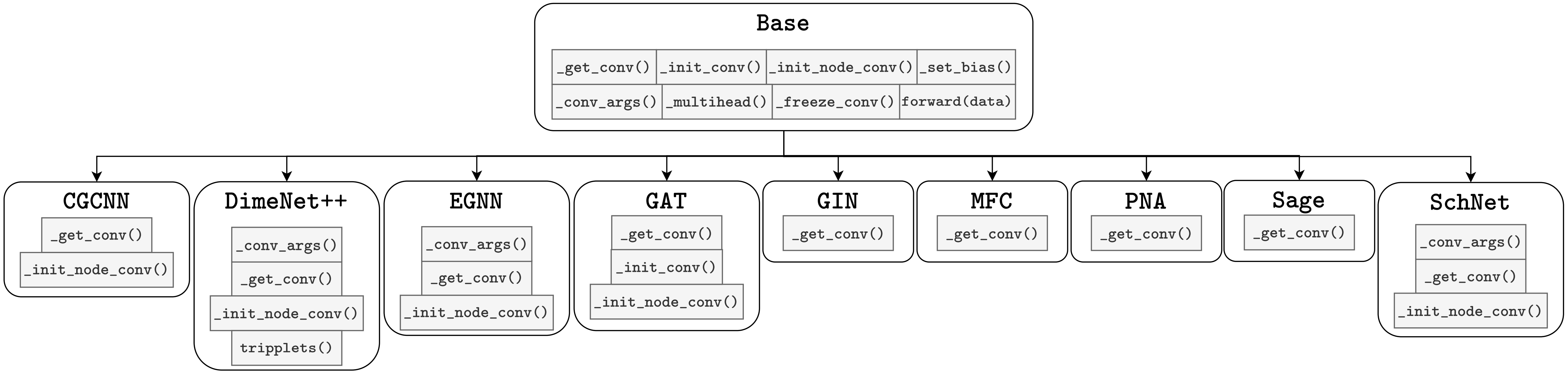
Multitasking for improved data efficiency

- Implementation
  - User-controlled task weights→ prioritize tasks
  - Graph-level module
    - Shared multilayer perceptron (MLP) + MLP for individual tasks
  - Node-level module (how to handle variable number of nodes?)
    - Padding to the largest graph (inefficient)
    - Graph convolutional layers
    - A shared MLP between all the nodes, mapping from extracted feature space to node-level properties



$$L = \sum_{i=1}^{N_G+N_N} \alpha_i l_i$$

Message passing layer treated as hyperparameter



- Object-oriented modules for each message passing layers
- Easy for extension/to include other GNN layers
- Convenient for user to find the optimal model for their applications
- How can users contribute by introducing additional MPNN layers?
  - Develop new class that inherits from "Base"
  - Implement the "get_conv()" method that defines the message passing policy

User set up HydraGNN case via a configuration json file

1. Define verbosity level

2. Define graph objects
   – Load data
   – Specify input features and regression targets

3. Design model architecture
   – Message passing method
   – Number of layers
   – Task weights

4. Specify training parameters
   – Loss function
   – Batch size, epochs
   – Optimizer, learning rate

5. Visualization of training/validation/testing results

Example can be found at
https://github.com/ORNL/HydraGNN/blob/LoG2023_tutorial/examples/lsms/lsms.json

User set up HydraGNN case via a configuration json file

Verbosity is used to handle amount of context printed in output by multiple processes during scalable HydraGNN training with distributed data parallelism

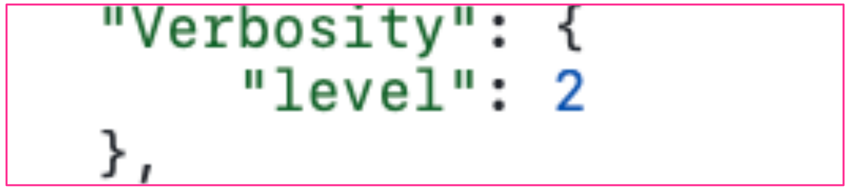
`'level: 0'`: nothing is printed on the screen

`'level: 1'`: only the process with rank 0 prints output at the end of each training epoch

`'level: 2'`: only the process with rank 0 prints output at each batched gradient update, showing the stage of the training on each epoch using a progression bar

`'level: 3'`: every process prints output at the end of each training epoch

`'level: 4'`: every process prints output at each batched gradient update, showing the stage of the training on each epoch using a progression bar



```
"Verbosity": {
    "level": 2
},
```

User set up HydraGNN case via a configuration json file

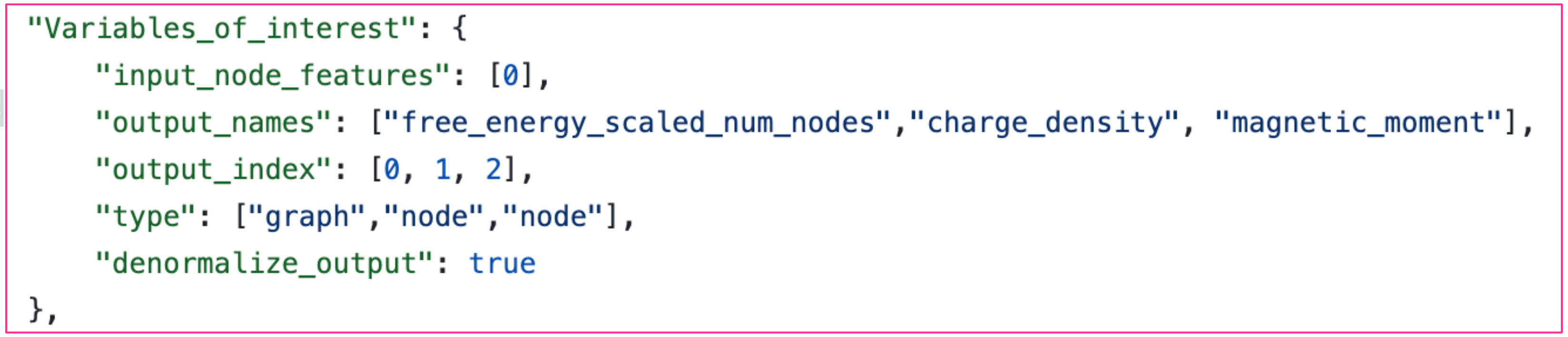
2. Define graph objects
   - Load data
   - Specify input features and regression targets
3. Design model architecture
   - Message passing method
   - Number of layers
   - Task weights
4. Specify training parameters
   - Loss function
   - Batch size, epochs
   - Optimizer, learning rate
5. Visualization of training/validation/testing results

```
"Dataset": {
    "name": "FePt_32atoms",
    "path": {"total": "./dataset/FePt_enthalpy"},
    "format": "LSMS",
    "compositional_stratified_splitting": true,
    "rotational_invariance": false,
    "node_features": {
        "name": ["num_of_protons","charge_density", "magnetic_moment"],
        "dim": [1,1,1],
        "column_index": [0,5,6]
    },
    "graph_features":{
        "name": [ "free_energy_scaled_num_nodes"],
        "dim": [1],
        "column_index": [0]
    }
},
```

Example can be found at https://github.com/ORNL/HydraGNN/blob/LoG2023_tutorial/examples/lsms/lsms.json

User set up HydraGNN case via a configuration json file

2. Define graph objects
   - Load data
   - Specify input features and regression targets
3. Design model architec
   - Message passing method
   - Number of layers
   - Task weights
4. Specify training parameters
   - Loss function
   - Batch size, epochs
   - Optimizer, learning rate
5. Visualization of training/validation/testing results



```
"Variables_of_interest": {
    "input_node_features": [0],
    "output_names": ["free_energy_scaled_num_nodes","charge_density", "magnetic_moment"],
    "output_index": [0, 1, 2],
    "type": ["graph","node","node"],
    "denormalize_output": true
},
```

Example can be found at https://github.com/ORNL/HydraGNN/blob/LoG2023_tutorial/examples/lsms/lsms.json

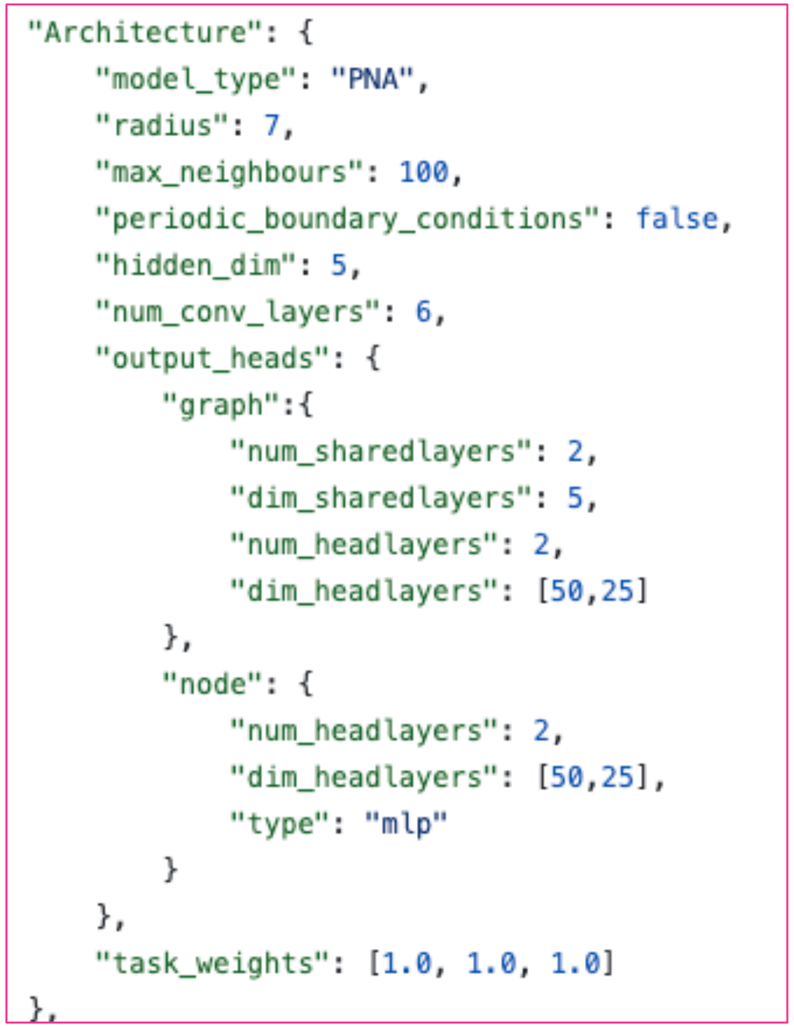
User set up HydraGNN case via a configuration json file

2. Define graph objects
   - Load data
   - Specify input features and regression targets

3. Design model architecture
   - Message passing method
   - Number of layers
   - Task weights

4. Specify training parameters
   - Loss function
   - Batch size, epochs
   - Optimizer, learning rate

5. Visualization of training/validation/testing results

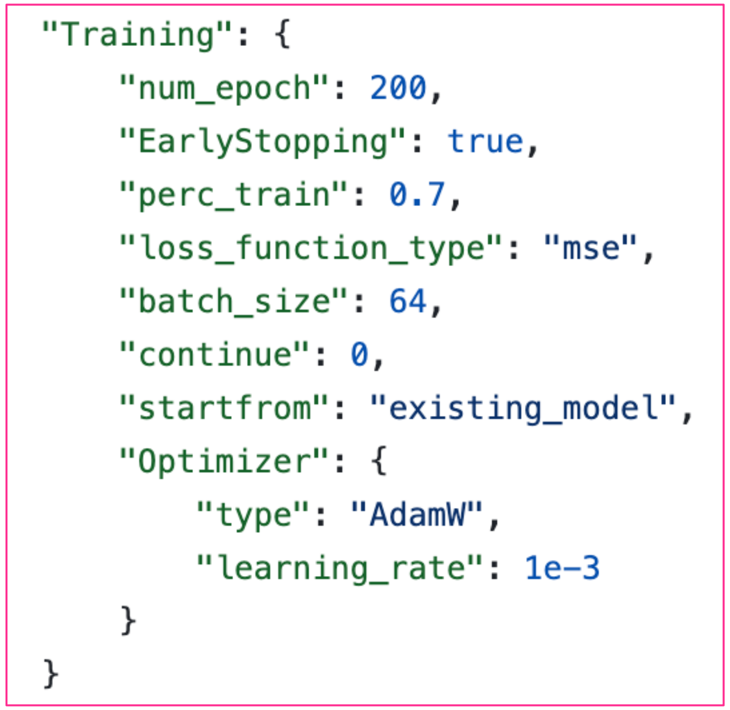


```json
"Architecture": {
    "model_type": "PNA",
    "radius": 7,
    "max_neighbours": 100,
    "periodic_boundary_conditions": false,
    "hidden_dim": 5,
    "num_conv_layers": 6,
    "output_heads": {
        "graph":{
            "num_sharedlayers": 2,
            "dim_sharedlayers": 5,
            "num_headlayers": 2,
            "dim_headlayers": [50,25]
        },
        "node": {
            "num_headlayers": 2,
            "dim_headlayers": [50,25],
            "type": "mlp"
        }
    },
    "task_weights": [1.0, 1.0, 1.0]
},
```

Example can be found at https://github.com/ORNL/HydraGNN/blob/LoG2023_tutorial/examples/lsms/lsms.json

User set up HydraGNN case via a configuration json file

2. Define graph objects
   - Load data
   - Specify input features and regression targets
3. Design model architecture
   - Message passing method
   - Number of layers
   - Task weights
4. Specify training parameters
   - Loss function
   - Batch size, epochs
   - Optimizer, learning rate
5. Visualization of training/validation/testing results



```
"Training": {
    "num_epoch": 200,
    "EarlyStopping": true,
    "perc_train": 0.7,
    "loss_function_type": "mse",
    "batch_size": 64,
    "continue": 0,
    "startfrom": "existing_model",
    "Optimizer": {
        "type": "AdamW",
        "learning_rate": 1e-3
    }
}
```

Example can be found at https://github.com/ORNL/HydraGNN/blob/LoG2023_tutorial/examples/lsms/lsms.json

User set up HydraGNN case via a configuration json file

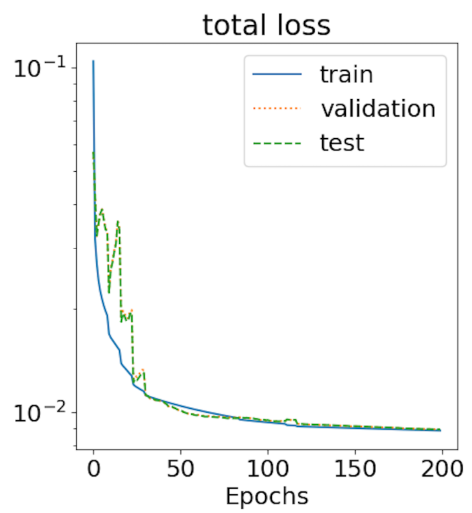
2. Define graph objects
   - Load data
   - Specify input features and regression targets
3. Design model architecture
   - Message passing method
   - Number of layers
   - Task weights
4. Specify training parameters
   - Loss function
   - Batch size, epochs
   - Optimizer, learning rate
5. Visualization of training/validation/testing results

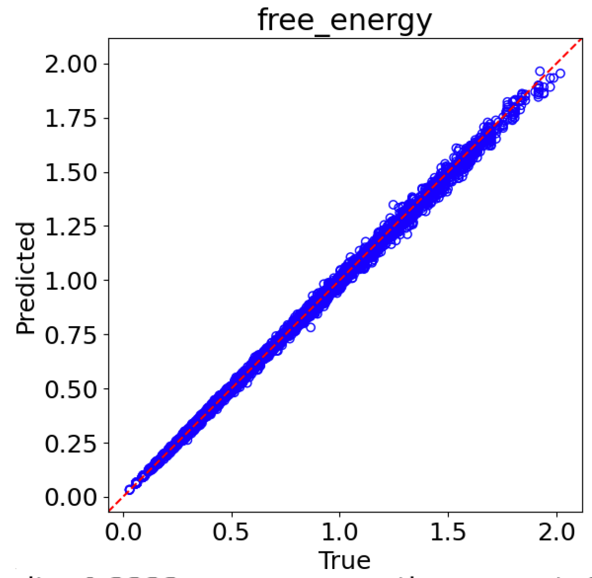
```
"Visualization": {
    "plot_init_solution": true,
    "plot_hist_solution": false,
    "create_plots": true
}
```

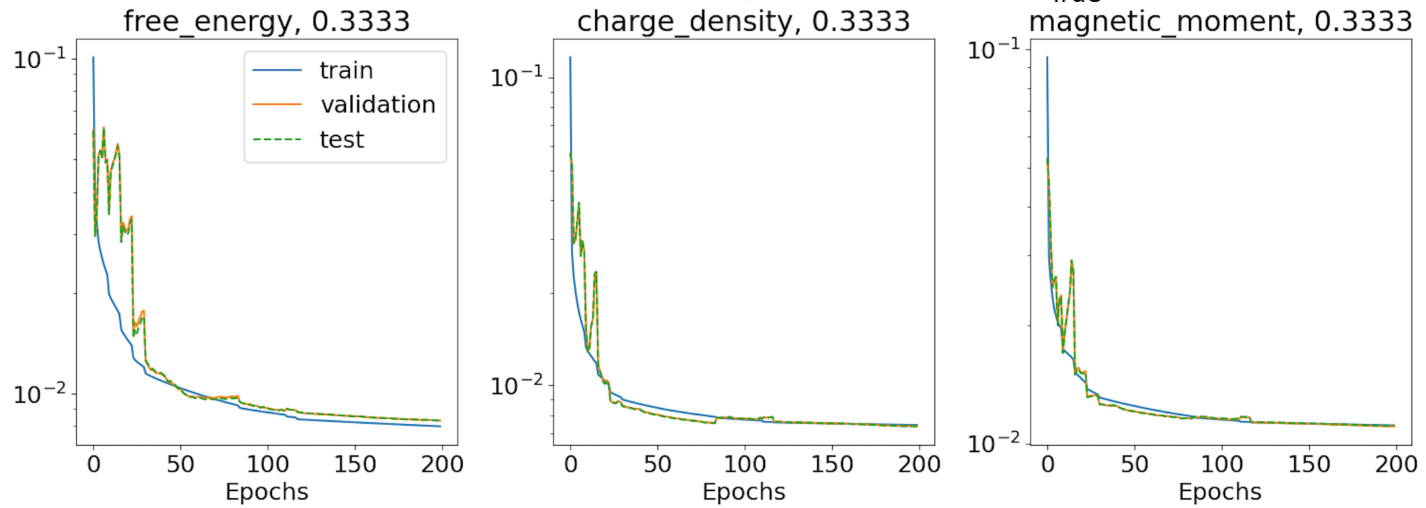
Example can be found at https://github.com/ORNL/HydraGNN/blob/LoG2023_tutorial/examples/lsms/lsms.json

User set up HydraGNN case via a configuration json file

2. Define graph objects
   - Load data
   - Specify input features and regres targets
3. Design model architecture
   - Message passing method
   - Number of layers
   - Task weights
4. Specify training parameters
   - Loss function
   - Batch size, epochs
   - Optimizer, learning rate
5. Visualization of training/validation/testing results







Example can be found at https://github.com/ORNL/HydraGNN/blob/LoG2023_tutorial/examples/lsms/lsms.json

Data file formats

The raw data can be converted into two pre-standardized formats:

- Pickle (preferrable for small/intermediate volumes of data)
- ADIOS2 https://adios2.readthedocs.io/en/v2.9.2/ (preferred for large volumes of data)

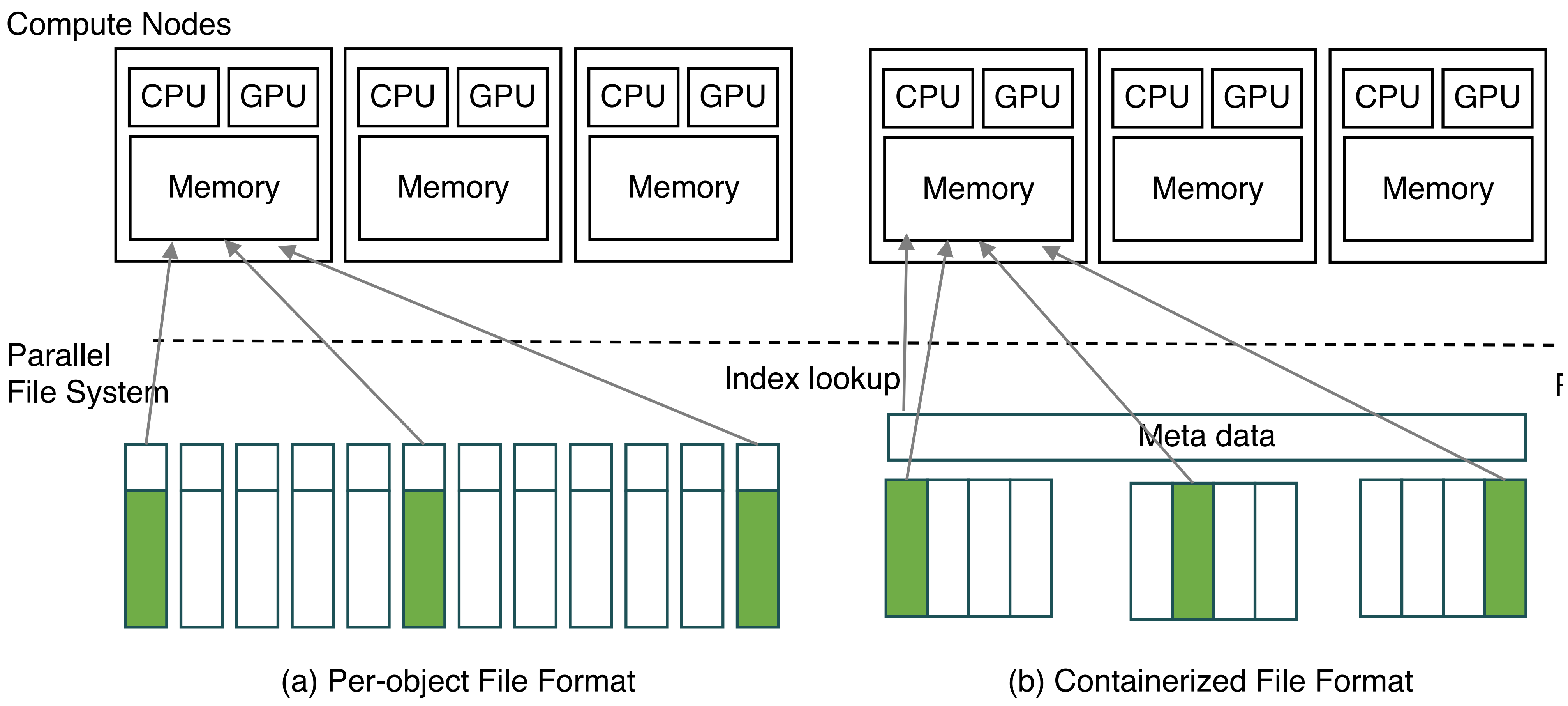
The user can choose the "degree of packing" to aggregate multiple data samples and avoid stressing the parallel file system of the HPC facility when HydraGNN is trained on large volumes of data.

Examples:

- If the data is "relatively" small in volume (i.e., < 50k data samples), storing one pickle file per data sample is fine → per-object file format (PFF)
- If the number of data samples 50k, then it is recommended to pre-package multiple data samples within the same file → containerized file format (CFF)

Data file formats

Traditional ways to read data from pickle and adios



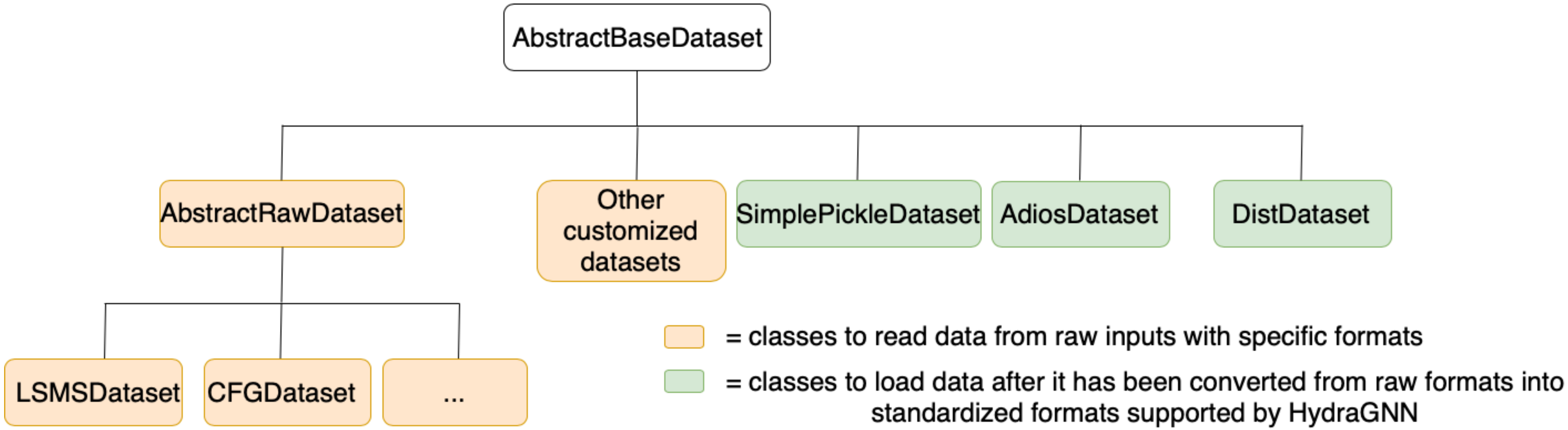
(a) Per-object File Format

(b) Containerized File Format

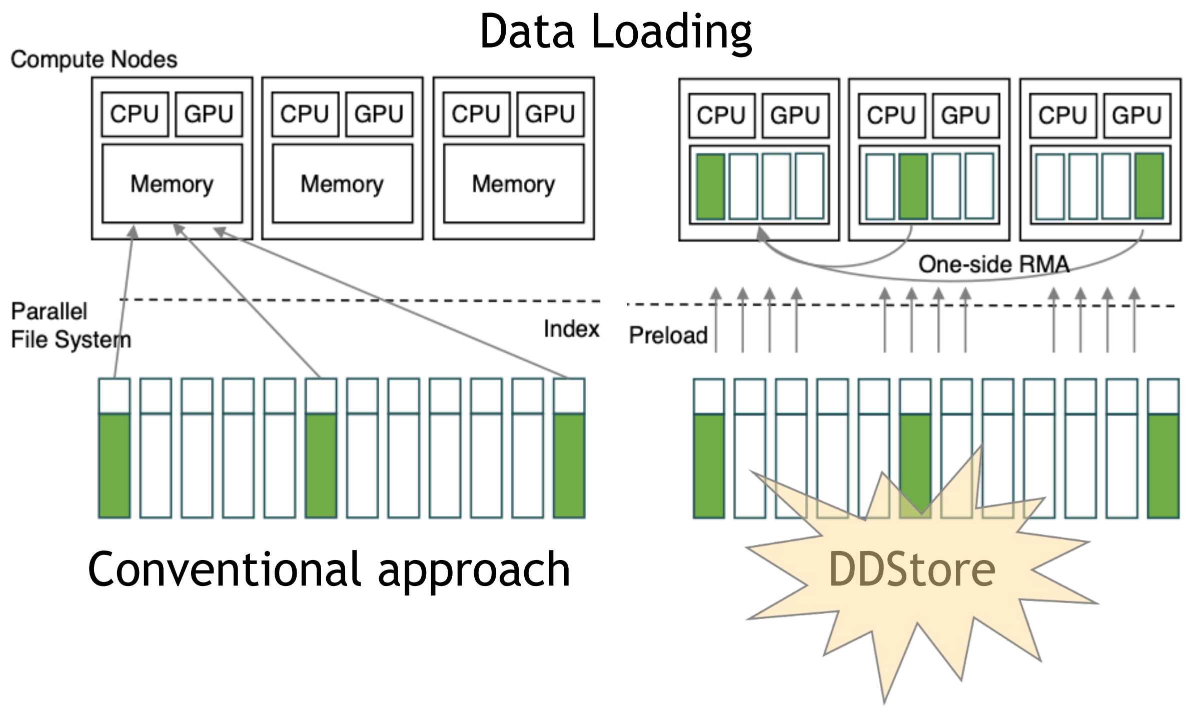
Data file formats

Object-oriented programming framework for data imports:

- Classes to read data from raw files and convert them into pickle or ADIOS files
- Classes to read pre-standardized data and feed it to HydraGNN for training

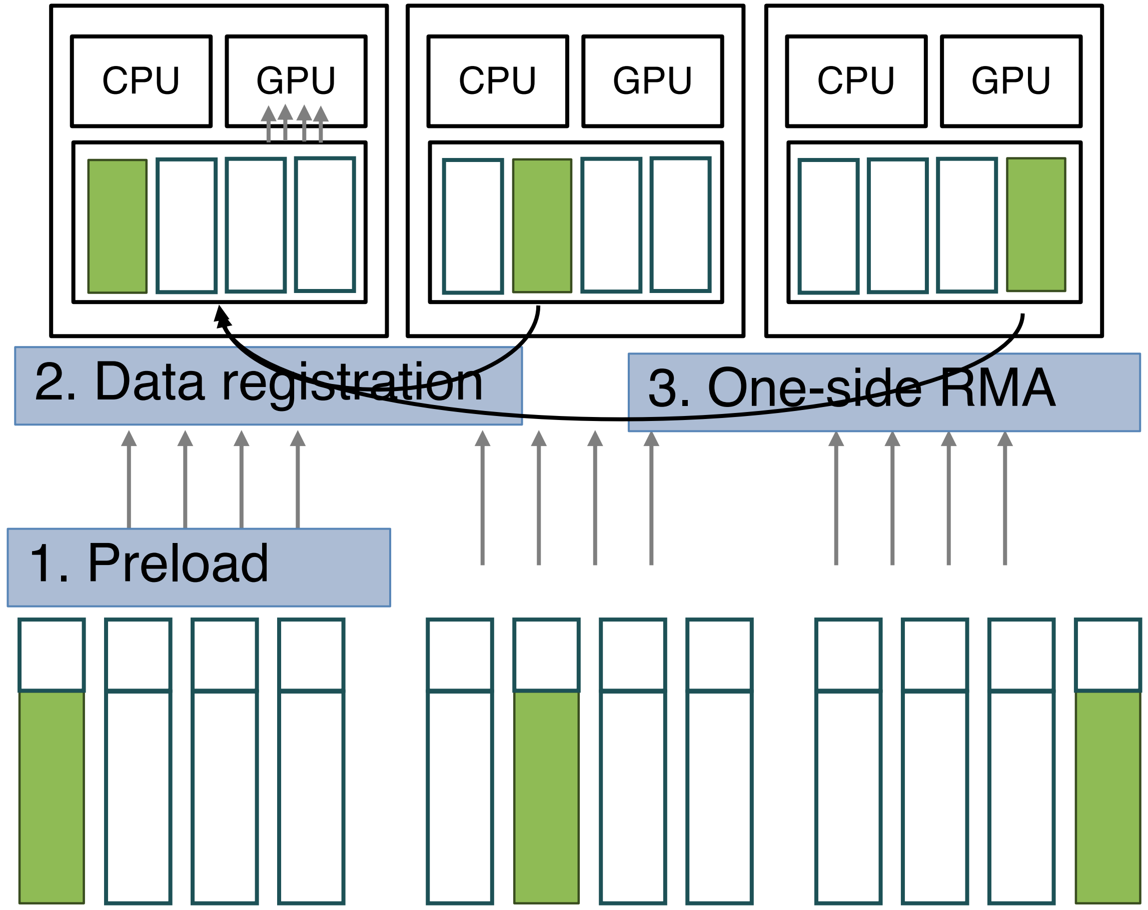

AbstractBaseDataset
AbstractRawDataset
Other customized datasets
SimplePickleDataset
AdiosDataset
DistDataset
LSMSDataset
CFGDataset
...
= classes to read data from raw inputs with specific formats
= classes to load data after it has been converted from raw formats into standardized formats supported by HydraGNN

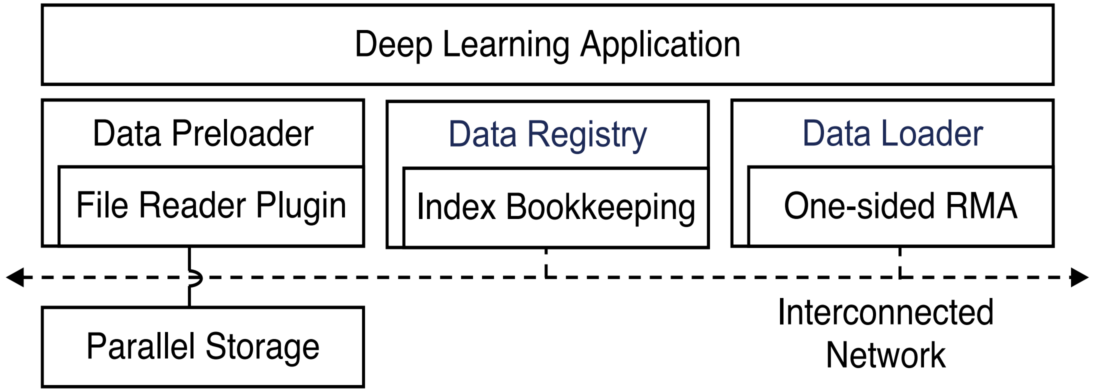
DDStore: Scalable Distributed Data Store



- **DDStore** specifically addresses **random, read-oriented, global shuffle** operations.
- **Memory-to-memory** distributed data access
- In-memory, **one-side remote memory (RMA) access**
- Minimize access to the file system during the shuffling steps and make in-memory data accessible to other nodes
- Utilize efficient, and portable communication on HPC

DDStore Procedures



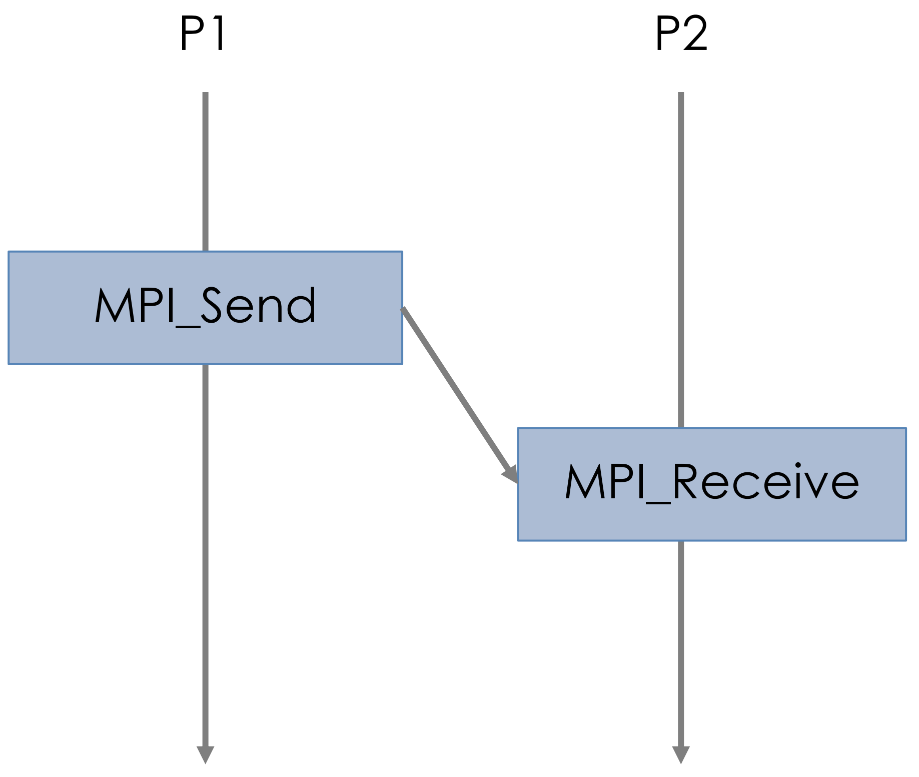


1 Preload
- Read data from file system
- Load in chunk

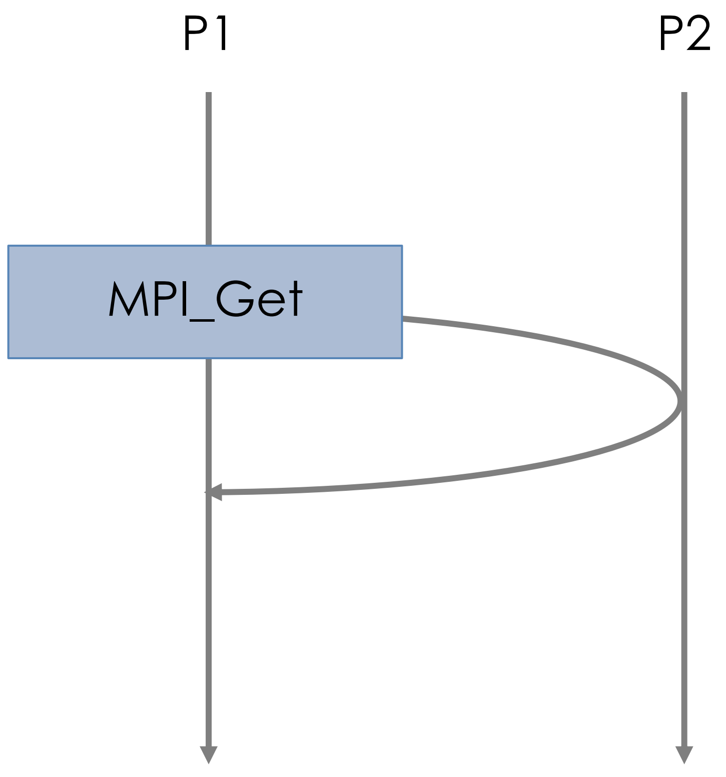
2 Data registration
- Create local index
- Share globally

3 Data loader
- Memory-to-memory data fetch
- Utilizing MPI RMA

MPI One-side Communication or RMA

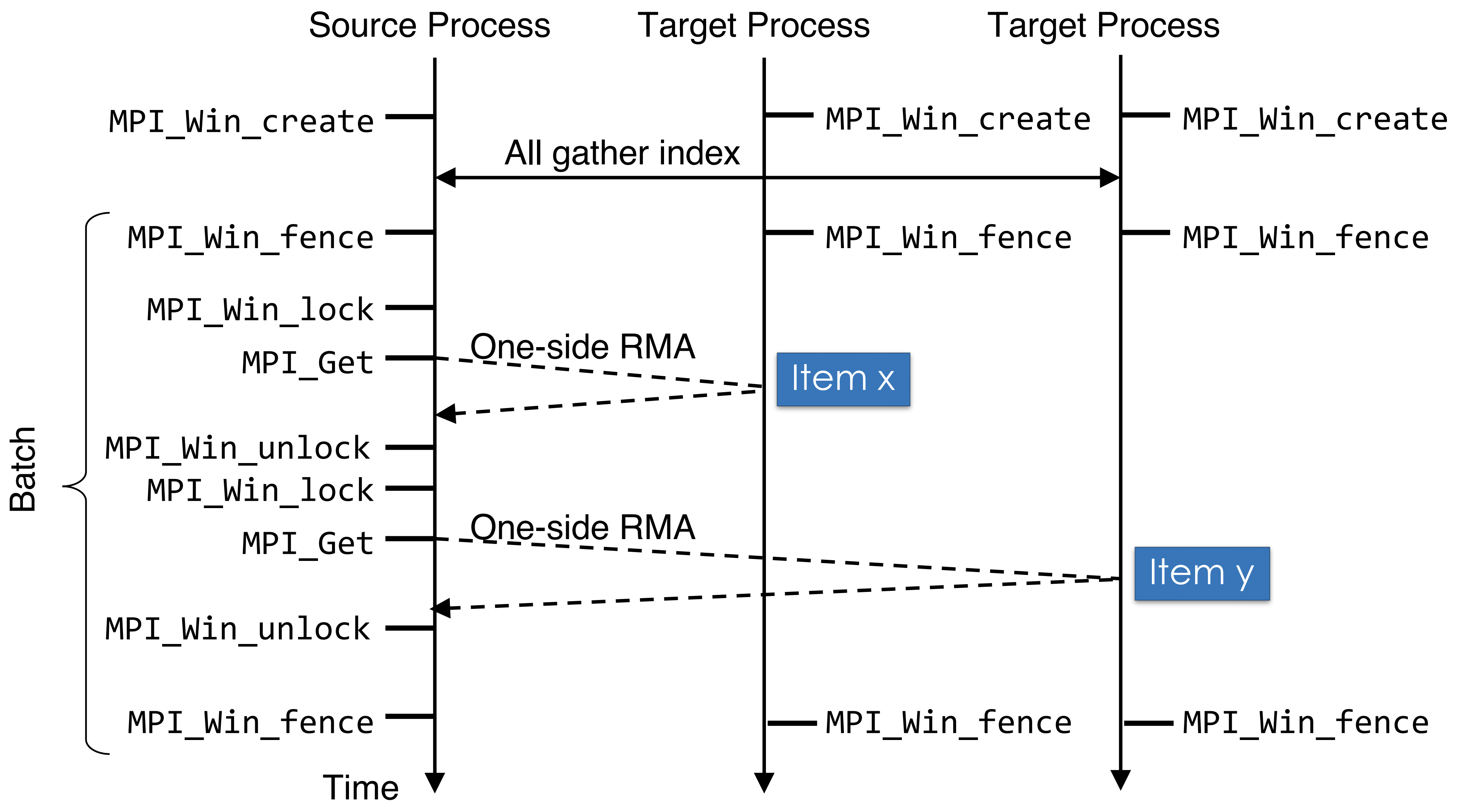


Two-sided communication



One-sided communication

DDStore Using MPI One-side Communication (RMA)



Hands-on session

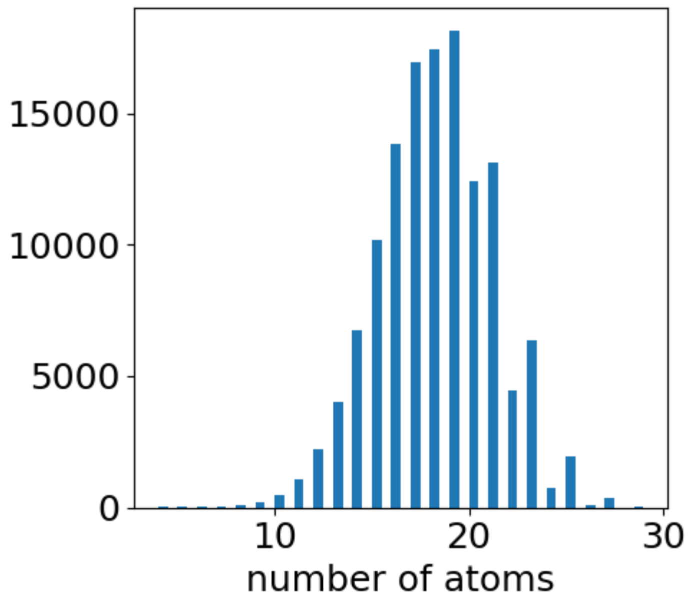
Overview

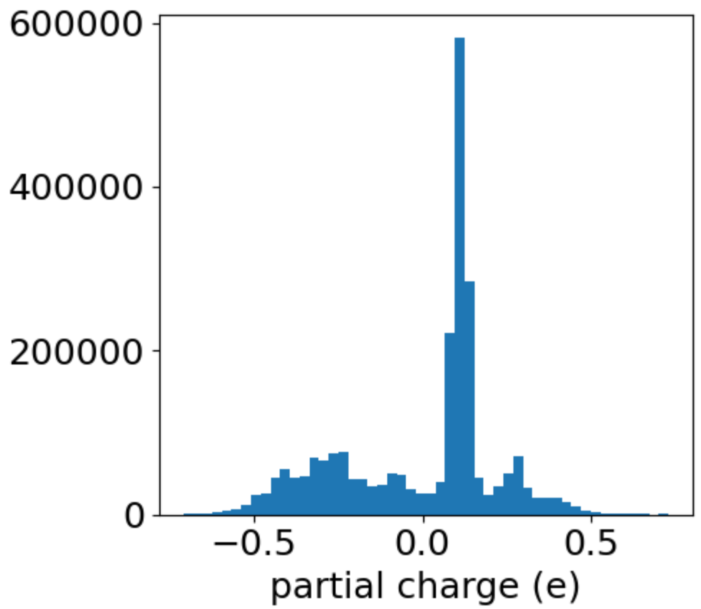
- Prerequisites
  - Setting up virtual environment
    - Activate your virtual environment
  - Downloading code (https://github.com/ORNL/HydraGNN)
    - *"git clone https://github.com/ORNL/HydraGNN"*
- Three examples (https://github.com/ORNL/HydraGNN/tree/LoG2023_tutorial)
  - QM9
    - Single tasking for a graph-level property
    - Multitasking regressions at both graph-level and node-level
  - LSMS
    - Customization of dataset/user dataset
  - AISD HOMO-LUMO
    - Scalability
    - DDStore/video record of OLCF-Frontier (due to access limitation)

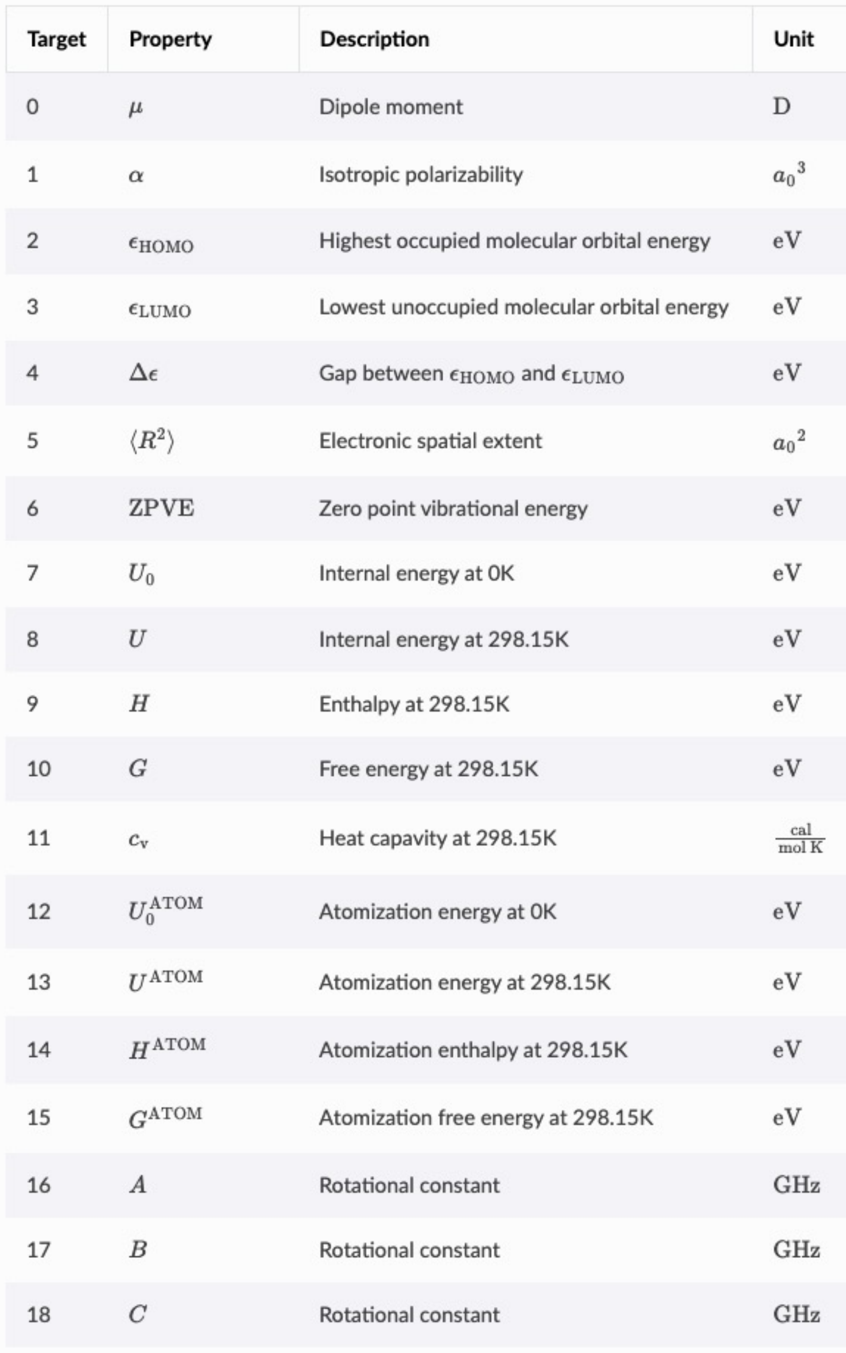
QM9 dataset

QM9 (Ramakrishnan et al., 2014)

- **torch_geometric.datasets.QM9**
- 130k molecules
- <u>**20**</u> regression targets
  - 19 original regression properties (**graph-level**)
    - geometric, energetic, electronic, and thermodynamic properties
  - Add Mulliken partial charge (**node-level**) from (Ramakrishnan al., 2014)







Target	Property	Description	Unit
0	μ	Dipole moment	D
1	α	Isotropic polarizability	$a_0{}^3$
2	ϵ_{HOMO}	Highest occupied molecular orbital energy	eV
3	ϵ_{LUMO}	Lowest unoccupied molecular orbital energy	eV
4	$\Delta\epsilon$	Gap between ϵ_{HOMO} and ϵ_{LUMO}	eV
5	$\langle R^2 \rangle$	Electronic spatial extent	$a_0{}^2$
6	ZPVE	Zero point vibrational energy	eV
7	U_0	Internal energy at 0K	eV
8	U	Internal energy at 298.15K	eV
9	H	Enthalpy at 298.15K	eV
10	G	Free energy at 298.15K	eV
11	c_{v}	Heat capacity at 298.15K	$\frac{\mathrm{cal}}{\mathrm{mol\,K}}$
12	U_0^{ATOM}	Atomization energy at 0K	eV
13	U^{ATOM}	Atomization energy at 298.15K	eV
14	H^{ATOM}	Atomization enthalpy at 298.15K	eV
15	G^{ATOM}	Atomization free energy at 298.15K	eV
16	A	Rotational constant	GHz
17	B	Rotational constant	GHz
18	C	Rotational constant	GHz

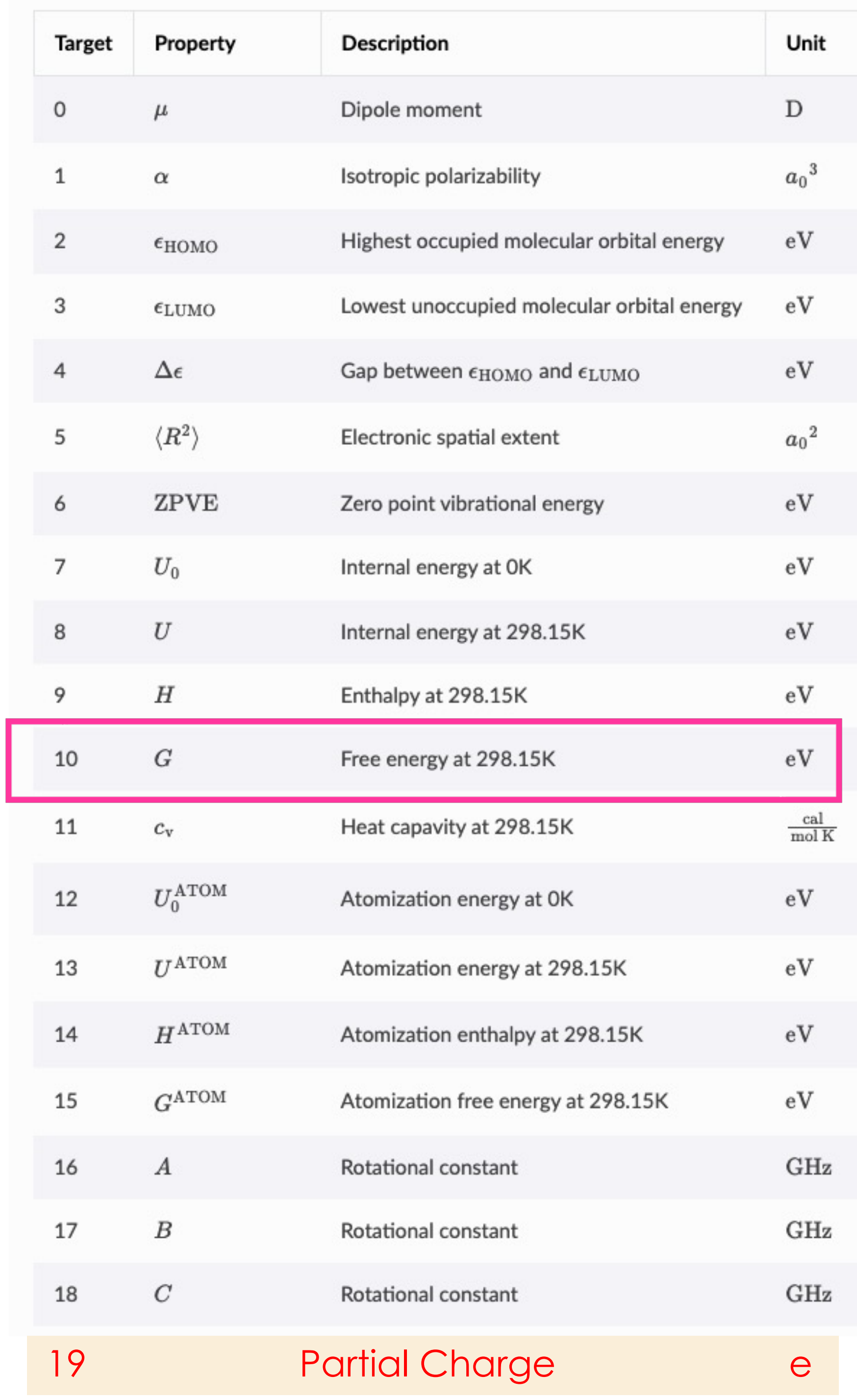
(PyTorch Geometric bult-in dataset)

Two examples

- https://github.com/ORNL/HydraGNN/tree/LoG2023_tutorial/examples/qm9

- Single-tasking on free energy, **G**
  - Files: **qm9.py** and **qm9.json**

- Multitasking on all 20 (=19+1) properties
  - Files: qm9_custom20.py and qm9_all20.json



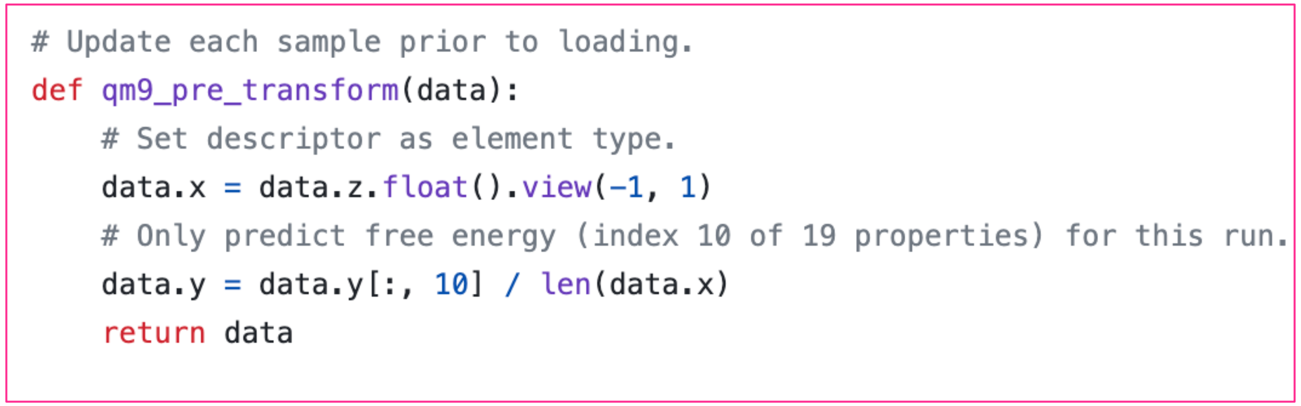
Target	Property	Description	Unit
0	μ	Dipole moment	D
1	α	Isotropic polarizability	${a_0}^3$
2	ϵ_{HOMO}	Highest occupied molecular orbital energy	eV
3	ϵ_{LUMO}	Lowest unoccupied molecular orbital energy	eV
4	$\Delta\epsilon$	Gap between ϵ_{HOMO} and ϵ_{LUMO}	eV
5	$\langle R^2 \rangle$	Electronic spatial extent	${a_0}^2$
6	ZPVE	Zero point vibrational energy	eV
7	U_0	Internal energy at 0K	eV
8	U	Internal energy at 298.15K	eV
9	H	Enthalpy at 298.15K	eV
10	G	Free energy at 298.15K	eV
11	c_{v}	Heat capavity at 298.15K	$\frac{\mathrm{cal}}{\mathrm{mol\,K}}$
12	U_0^{ATOM}	Atomization energy at 0K	eV
13	U^{ATOM}	Atomization energy at 298.15K	eV
14	H^{ATOM}	Atomization enthalpy at 298.15K	eV
15	G^{ATOM}	Atomization free energy at 298.15K	eV
16	A	Rotational constant	GHz
17	B	Rotational constant	GHz
18	C	Rotational constant	GHz
19		Partial Charge	e

Single-tasking on free energy, G

- Files: **qm9.py** and **qm9.json**
- *“python examples/qm9/qm9.py”*

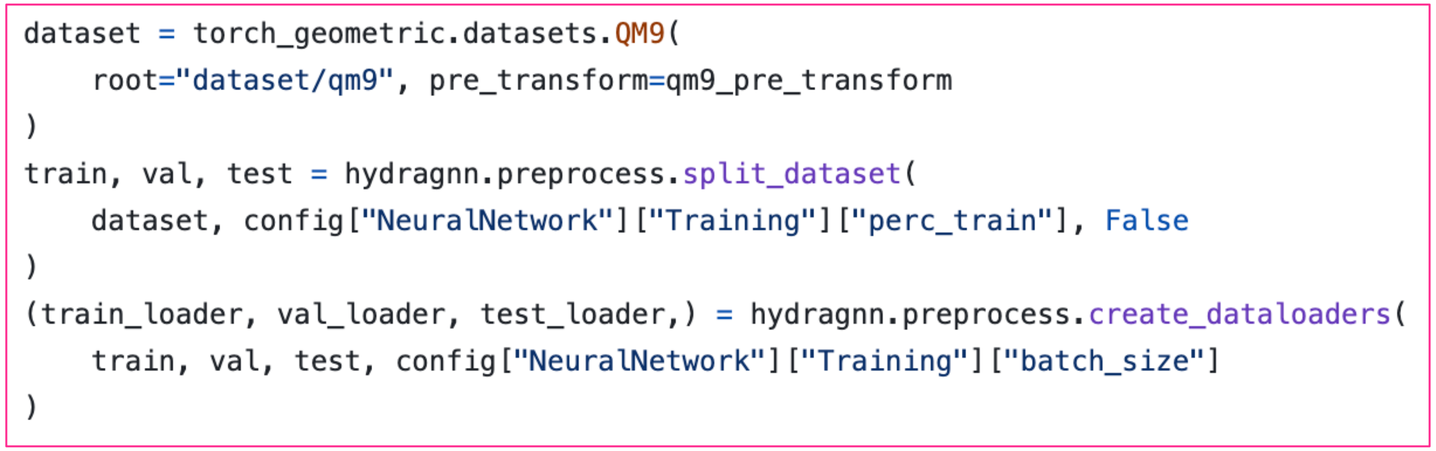
Single-tasking on free energy, G

- Loading **data**
  - torch_geometric.datasets.QM9
  - pre_transform function



```python
# Update each sample prior to loading.
def qm9_pre_transform(data):
    # Set descriptor as element type.
    data.x = data.z.float().view(-1, 1)
    # Only predict free energy (index 10 of 19 properties) for this run.
    data.y = data.y[:, 10] / len(data.x)
    return data
```

- Split dataset and create **dataloaders**



```python
dataset = torch_geometric.datasets.QM9(
    root="dataset/qm9", pre_transform=qm9_pre_transform
)
train, val, test = hydragnn.preprocess.split_dataset(
    dataset, config["NeuralNetwork"]["Training"]["perc_train"], False
)
(train_loader, val_loader, test_loader,) = hydragnn.preprocess.create_dataloaders(
    train, val, test, config["NeuralNetwork"]["Training"]["batch_size"]
)
```

Single-tasking on free energy, G

- Create **model** with **config** from qm9.json
- Set up optimizer
- Train the model
  - hydragnn.train.train_validate_test

```
"Architecture": {
    "model_type": "PNA",
    "hidden_dim": 5,
    "num_conv_layers": 6,
    "output_heads": {
        "graph":{
            "num_sharedlayers": 2,
            "dim_sharedlayers": 50,
            "num_headlayers": 2,
            "dim_headlayers": [50,25]
        }
    },
    "task_weights": [1.0]
},
```

```
model = hydragnn.models.create_model_config(
    config=config["NeuralNetwork"],
    verbosity=verbosity,
)
```

Single-tasking on free energy, G

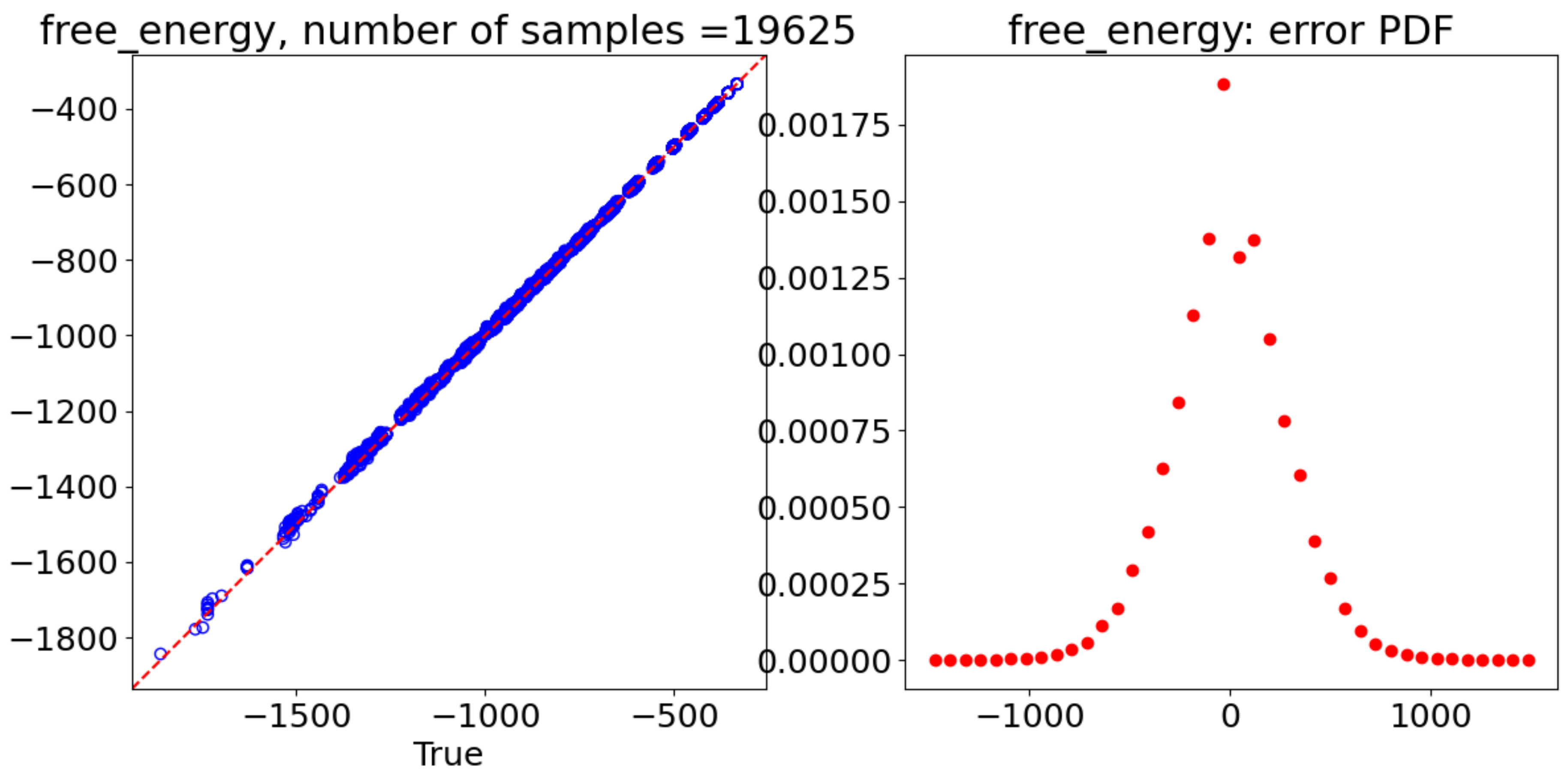
- Create model with **config** from qm9.json
- Set up optimizer
- Train the model
  - hydragnn.train.train_validate_test

```
learning_rate = config["NeuralNetwork"]["Training"]["Optimizer"]["learning_rate"]
optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode="min", factor=0.5, patience=5, min_lr=0.00001
)
```

```
"Training": {
    "num_epoch": 200,
    "perc_train": 0.7,
    "loss_function_type": "mse",
    "batch_size": 64,
    "continue": 0,
    "startfrom": "existing_model",
    "Optimizer": {
        "type": "AdamW",
        "learning_rate": 1e-3
```

```
hydragnn.train.train_validate_test(
    model,
    optimizer,
    train_loader,
    val_loader,
    test_loader,
    writer,
    scheduler,
    config["NeuralNetwork"],
    log_name,
    verbosity,
    create_plots=config["Visualization"]["create_plots"],
)
```

Single-tasking on free energy, G



Results of test set

"python examples/qm9/qm9.py"

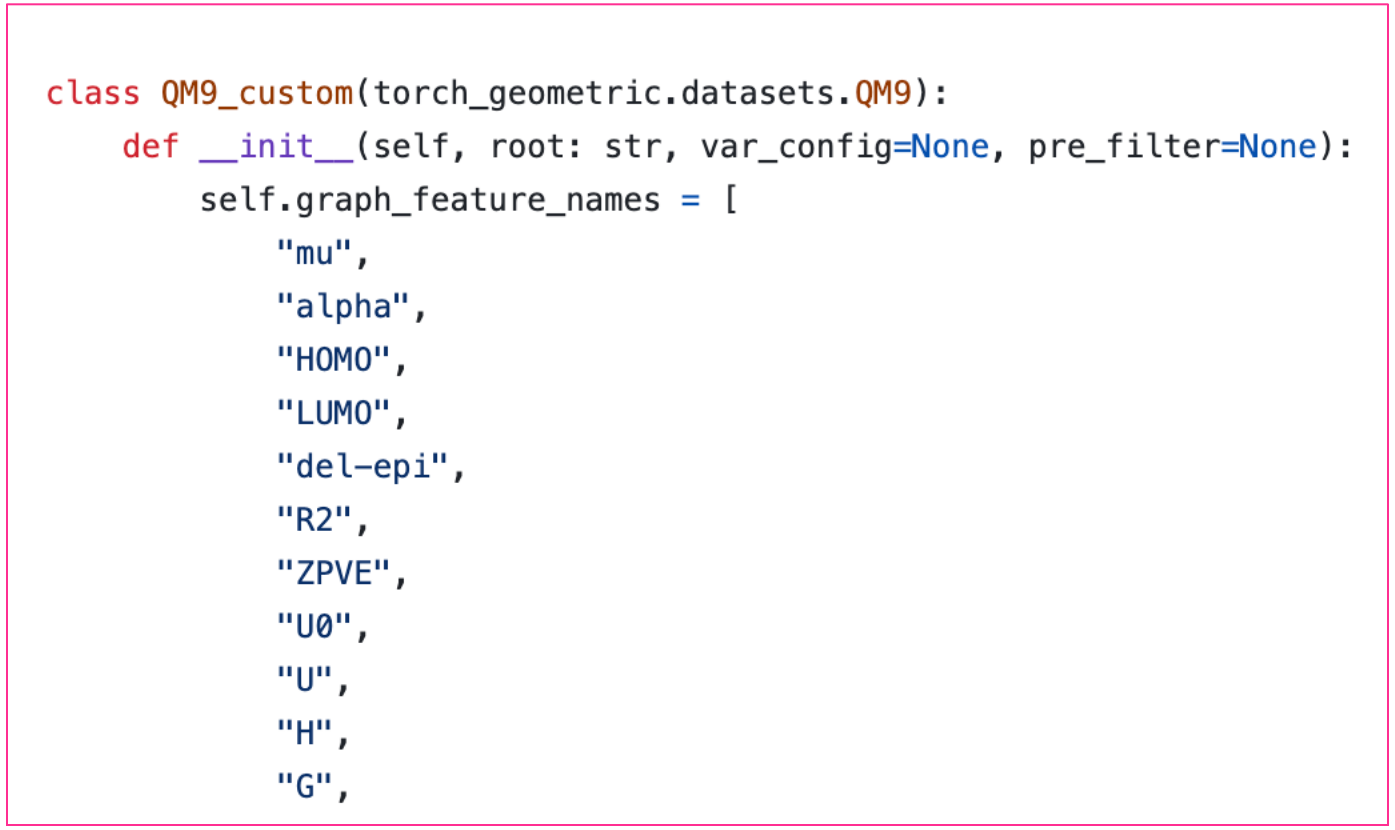
Multitasking on all 20 (=19+1) properties

- Files
  - **qm9_custom20.py** and **qm9_all20.json**
  - Pre-processed splits
    - **qm9_train_test_val_idx_lists.pkl**

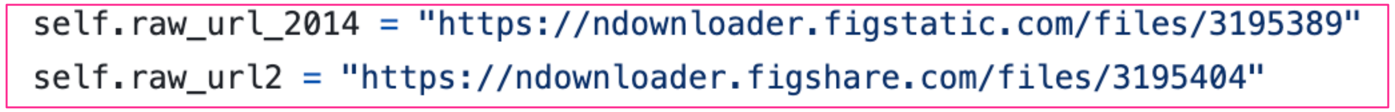
- “*python examples/qm9/qm9_custom20.py*”

Multitasking on all 20 (=19+1) properties

- Customized dataset
  - **QM9_custom(…)**
    - Download charge density
    - *get_charge(self, data)*
  - Pre-processed splits
    - **qm9_train_test_val_idx_lists.pkl**



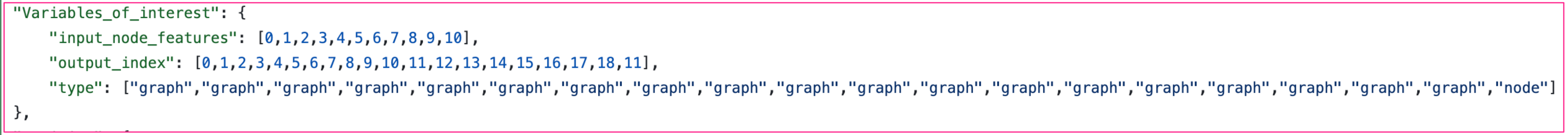
```
class QM9_custom(torch_geometric.datasets.QM9):
    def __init__(self, root: str, var_config=None, pre_filter=None):
        self.graph_feature_names = [
            "mu",
            "alpha",
            "HOMO",
            "LUMO",
            "del-epi",
            "R2",
            "ZPVE",
            "U0",
            "U",
            "H",
            "G",
```



```
self.raw_url_2014 = "https://ndownloader.figstatic.com/files/3195389"
self.raw_url2 = "https://ndownloader.figshare.com/files/3195404"
```

Multitasking on all 20 (=19+1) properties

- *data*.x for node feature
  - 11-dimension vector
  - atom type (i.e., "atomH", "atomC", "atomN", "atomO", "atomF"), atomic number, aromatic [or not], hybridization types (i.e., sp, sp2, or sp3), Hprop (i.e., number of hydrogen neighbors are used as features for each node)
- *data*.y for outputs/regression tasks
  - 19 graph-level + 1 node-level (number of nodes, varying across samples)



```
"Variables_of_interest": {
    "input_node_features": [0,1,2,3,4,5,6,7,8,9,10],
    "output_index": [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,11],
    "type": ["graph","graph","graph","graph","graph","graph","graph","graph","graph","graph","graph","graph","graph","graph","graph","graph","graph","graph","graph","node"]
},
```

Multitasking on all 20 (=19+1) properties

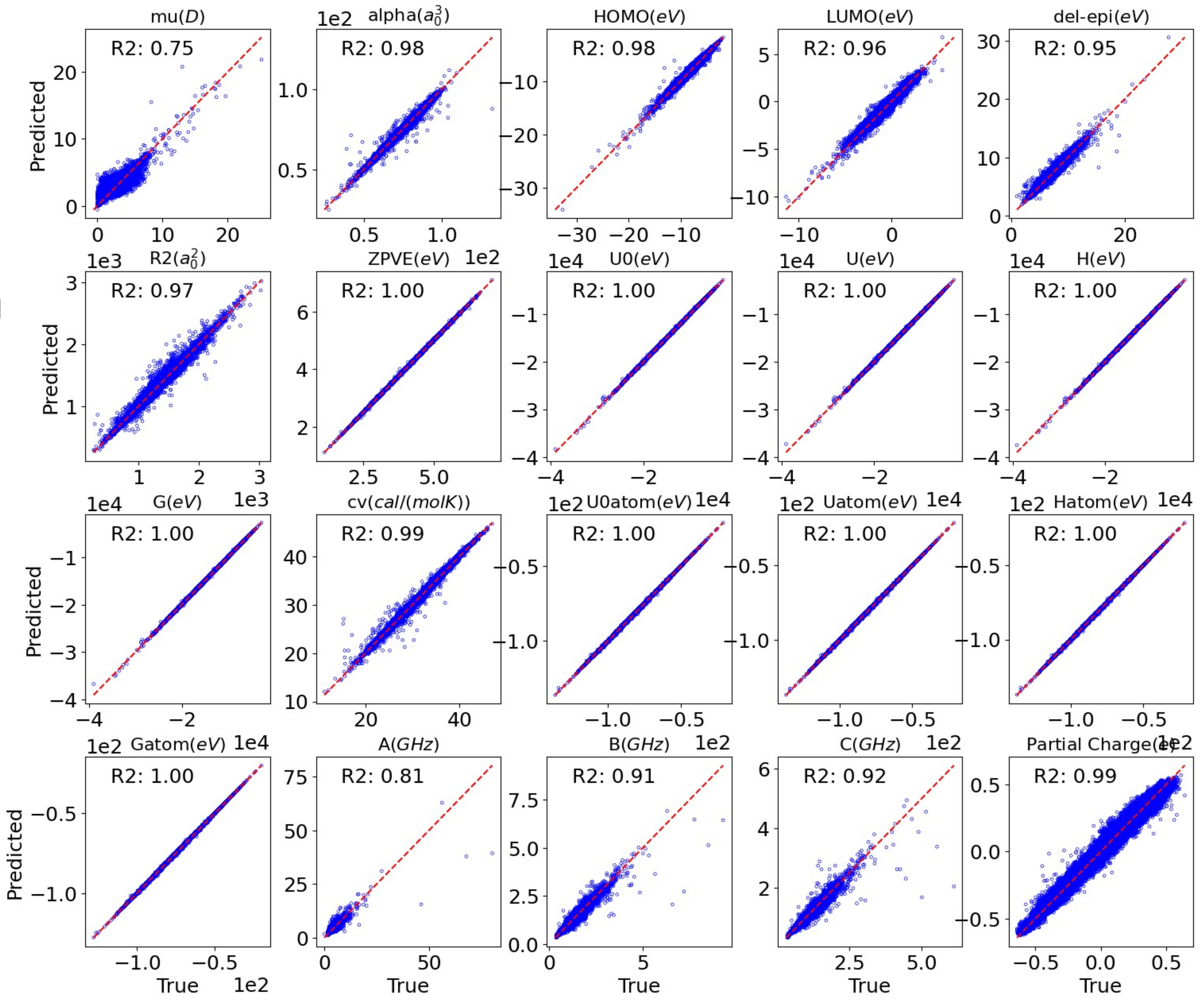
- Create dataloaders
- Create model
  - Graph heads
  - Node heads
- Set up optimizer
- Train the model
  - *task_weights*

```
"Architecture": {
    "model_type": "PNA",
    "hidden_dim": 30,
    "num_conv_layers": 6,
    "output_heads": {
        "graph":{
            "num_sharedlayers": 2,
            "dim_sharedlayers": 50,
            "num_headlayers": 2,
            "dim_headlayers": [50,25]
        },
        "node": {
            "num_headlayers": 3,
            "dim_headlayers": [50,50,25],
            "type": "mlp"
        }
    },
    "task_weights": [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
},
```

Multitasking

- Test HydraGNN in multitasking with hybrid graph-level and node-level properties
  - 19 graph-level properties
  - 1 node-level property

“python examples/qm9/qm9_custom20.py”



OAK RIDGE National Laboratory

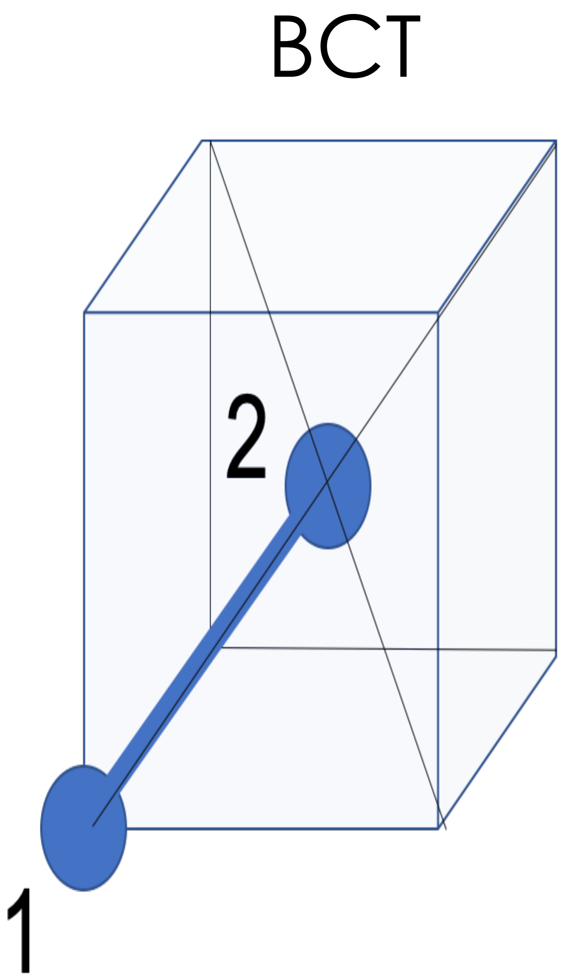
FePt binary alloy with 32 atoms LSMS-3 data

FePt binary alloy with 32 atoms - LSMS-3 data

Iron-Platinum (FePt) Open-Source Dataset binary alloy
https://doi.org/10.13139/OLCF/1762742

- **32 atoms** arranged in a body-centered tetragonal (BCT) structure
- The entire composition range is spanned
  (from 0% Fe-100%Pt through 100% Fe-0%Pt)
- 32,000 configurations

For each configuration, DFT calculations are performed to compute the total energy of the systems
DFT calculations are performed using the LSMS-3 code



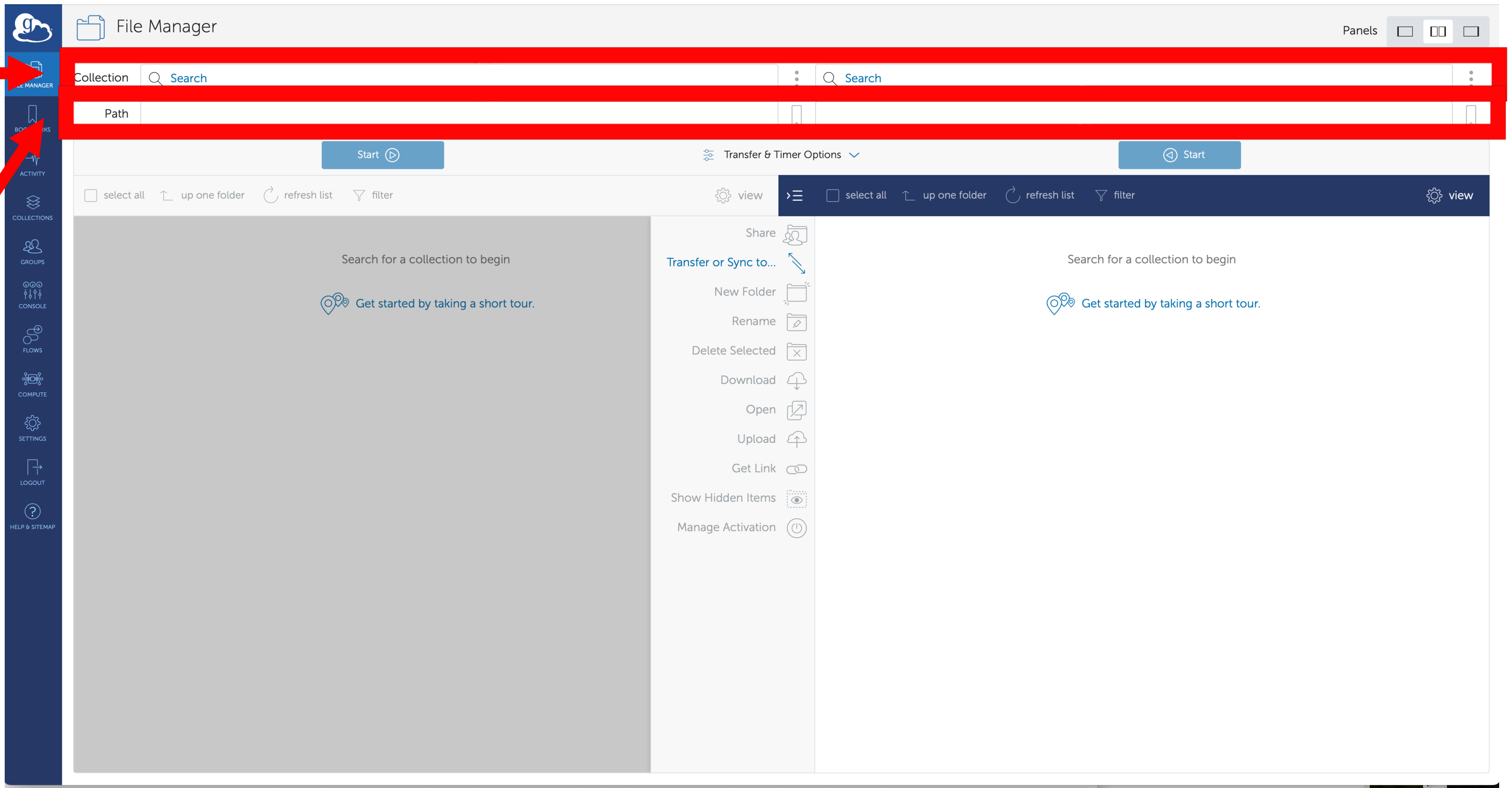
FePt binary alloy with 32 atoms - LSMS-3 data

Download dataset using Globus https://www.globus.org

- Create a Globus account and log-in

Specify the name of the **source** and **destination endpoints** among which the data transfer must be established

Specify the paths on the source endpoint where the data is available and the path on the destination of endpoint where the data must be transferred



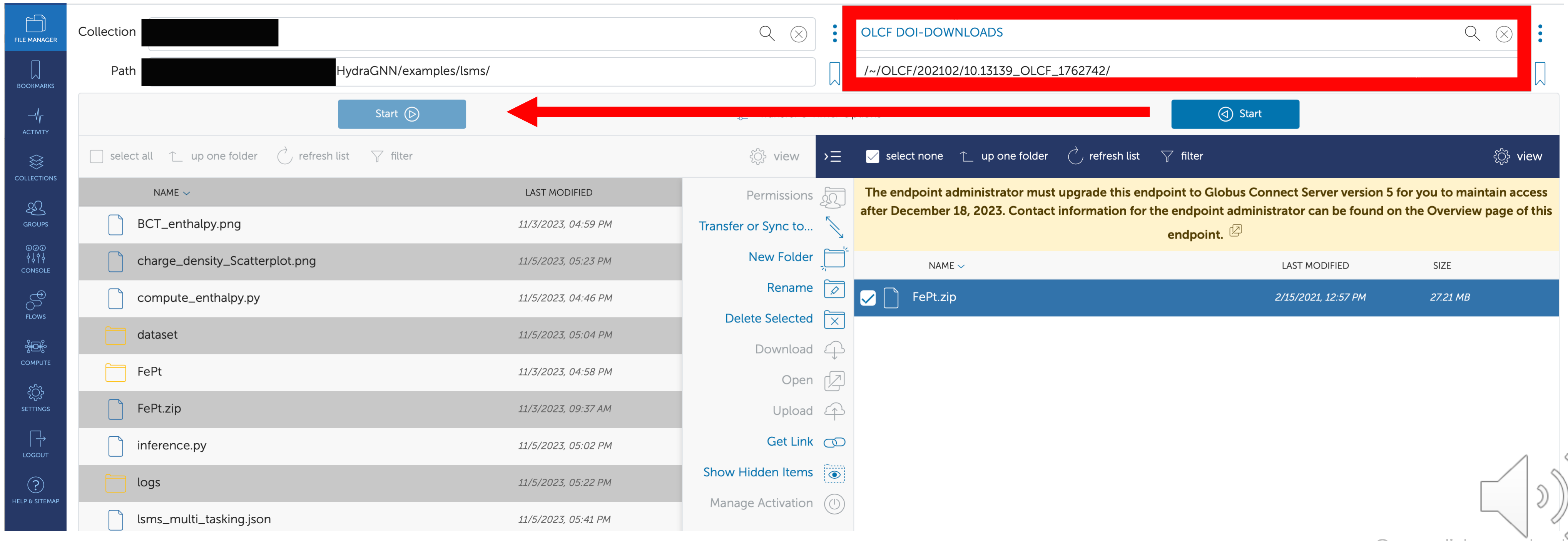
FePt binary alloy with 32 atoms - LSMS-3 data

Download dataset using Globus https://www.globus.org

Choice of endpoints:

- One endpoint must be where you want the dataset to be downloaded
- One endpoint must be where the data is available: **OLCF-DOI-DOWNLOADS**

Path: /~/OLCF/202102/10.13139_OLCF_1762742/



Collection [redacted]

Path [redacted] HydraGNN/examples/lsms/

NAME	LAST MODIFIED
BCT_enthalpy.png	11/3/2023, 04:59 PM
charge_density_Scatterplot.png	11/5/2023, 05:23 PM
compute_enthalpy.py	11/5/2023, 04:46 PM
dataset	11/5/2023, 05:04 PM
FePt	11/3/2023, 04:58 PM
FePt.zip	11/3/2023, 09:37 AM
inference.py	11/5/2023, 05:02 PM
logs	11/5/2023, 05:22 PM
lsms_multi_tasking.json	11/5/2023, 05:41 PM

OLCF DOI-DOWNLOADS

/~/OLCF/202102/10.13139_OLCF_1762742/

The endpoint administrator must upgrade this endpoint to Globus Connect Server version 5 for you to maintain access after December 18, 2023. Contact information for the endpoint administrator can be found on the Overview page of this endpoint.

NAME	LAST MODIFIED	SIZE
FePt.zip	2/15/2021, 12:57 PM	27.21 MB

Open slide master to edit

FePt binary alloy with 32 atoms - LSMS-3 data

Code for this example is available at the following GitHub fork:
https://github.com/ORNL/HydraGNN/tree/LoG2023_tutorial/examples/lsms

Python scripts to run for this example are available inside **HydraGNN/examples/lsms:**

- **compute_enthalpy.py** → data pre-processing
- **lsms.py** → data pre-loading and training
- **inference.py** → post-processing and analysis of results

FePt binary alloy with 32 atoms - LSMS-3 data

Code for this example is available at the following GitHub fork: https://github.com/allaffa/HydraGNN/tree/LoG2023_tutorial_lsms_example

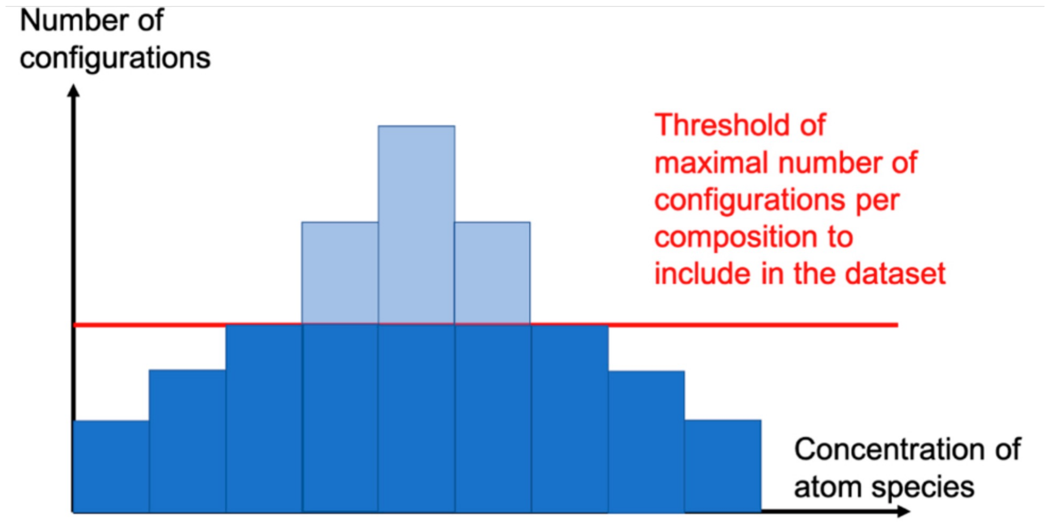
compute_enthalpy.py

1. Performs histogram cutoff to ensure that the atomic configurations are balanced across all chemical compositions.

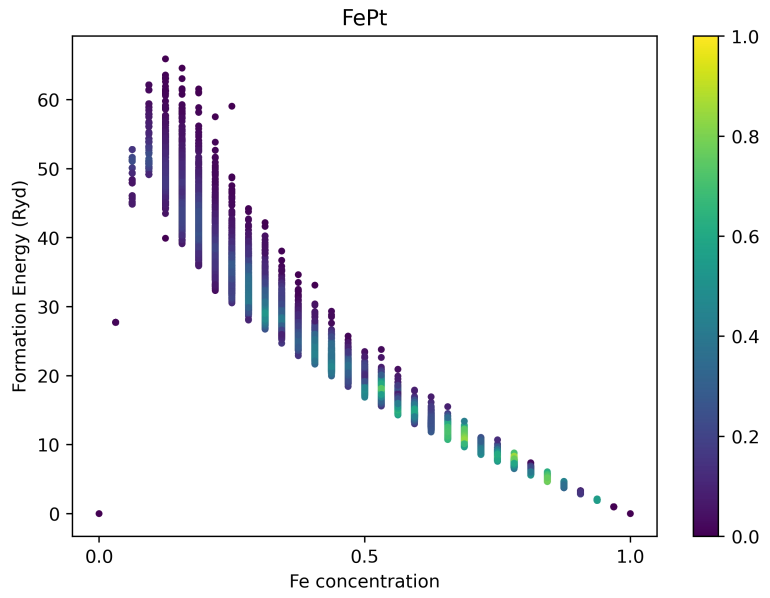
   We used 1,000 atomic configurations for thresholding

   From the original set of 32,017 configurations, only 28,058 configurations are retained

2. Computes mixing enthalpy by removing the linear mixina terms from the total enerav of each DFT calculation



Number of configurations
Threshold of maximal number of configurations per composition to include in the dataset
Concentration of atom species



FePt binary alloy with 32 atoms - LSMS-3 data

lsms.py

1. Dataset reading and pre-loading

Create 'dataset' folder inside the 'example directory
Move FePt_enthalpy into 'dataset'

Class inheritance for dataset classes

$$\begin{tikzcd}
\text{AbstractBaseDataset} \arrow[d] \\
\text{AbstractRawDataset} \arrow[d] \\
\text{LSMSDataset}
\end{tikzcd}$$

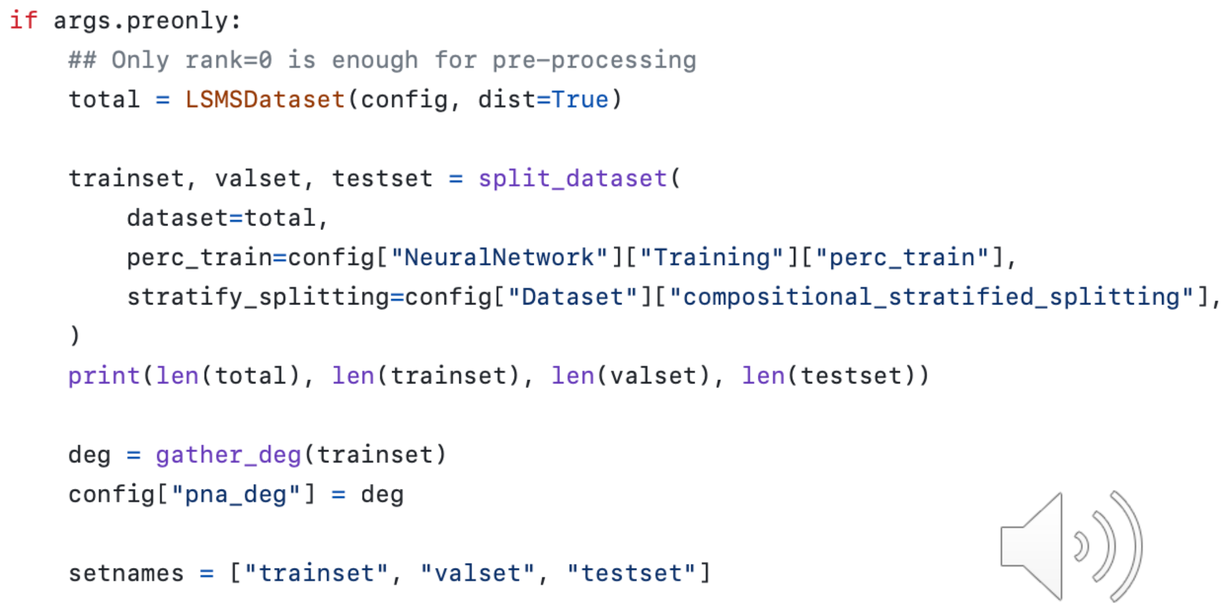
Intermediate layer in the class inheritance that implemented useful methods that can be used for data with diverse formats

Remark:

Your customized dataset does not need to inherit from Abs'tractRawDataset.

It can directly inherit from AbstractBaseDataset.

Inheriting from AbstractBaseDataset ensures that you can scale the data management using internal capabilities of HydraGNN
[Jong will provide more details in this regard]



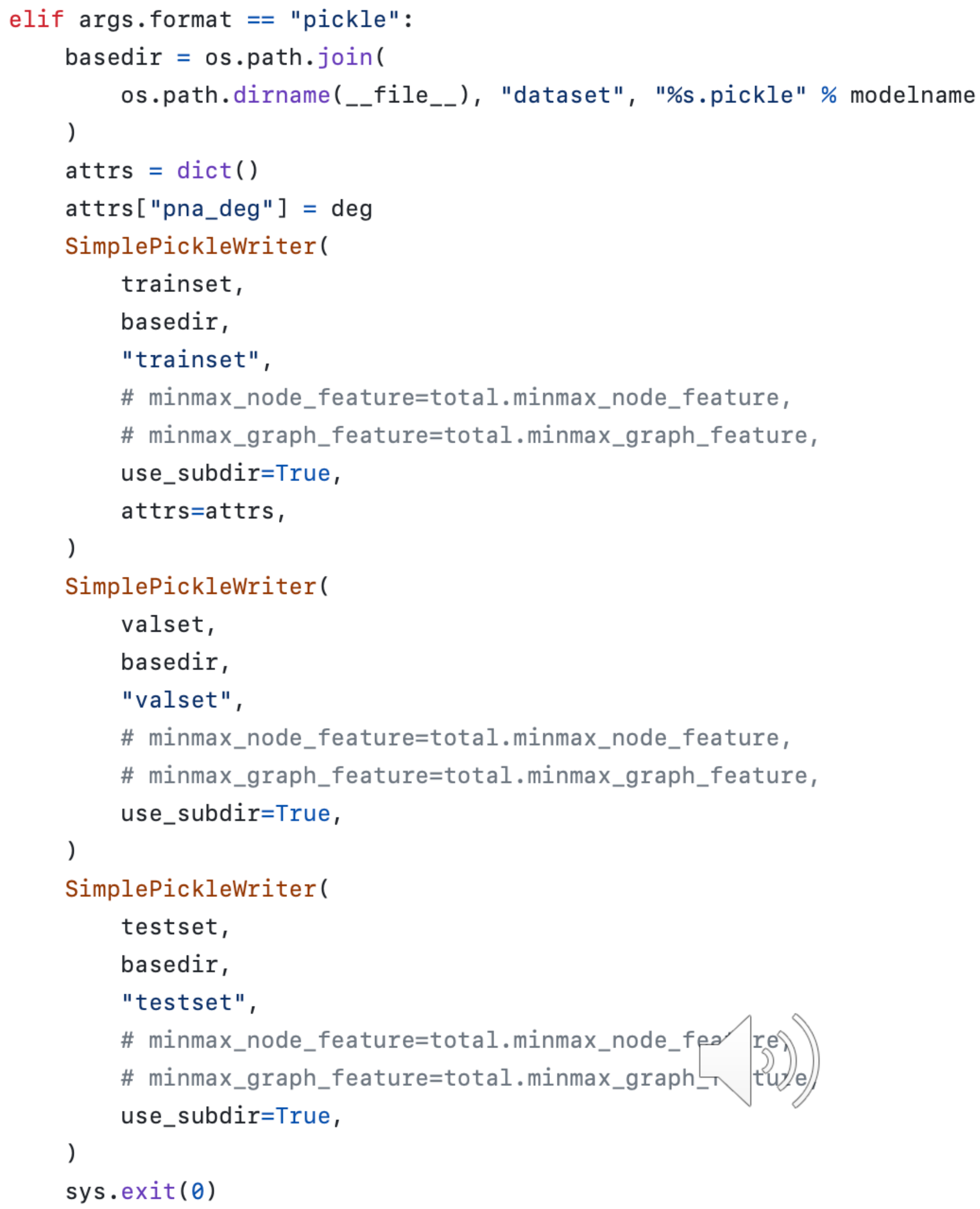
```
if args.preonly:
    ## Only rank=0 is enough for pre-processing
    total = LSMSDataset(config, dist=True)

    trainset, valset, testset = split_dataset(
        dataset=total,
        perc_train=config["NeuralNetwork"]["Training"]["perc_train"],
        stratify_splitting=config["Dataset"]["compositional_stratified_splitting"],
    )
    print(len(total), len(trainset), len(valset), len(testset))

    deg = gather_deg(trainset)
    config["pna_deg"] = deg

    setnames = ["trainset", "valset", "testset"]
```

FePt binary alloy with 32 atoms - LSMS-3 data

lsms.py

2. Dataset conversion into pickle format and storage in files

After every atomic structure is converted into a ‘torch.geometric.dataset’ object, which is ready to feed into HydraGNN for training and inferencing, the data samples are saved into individual pickle files

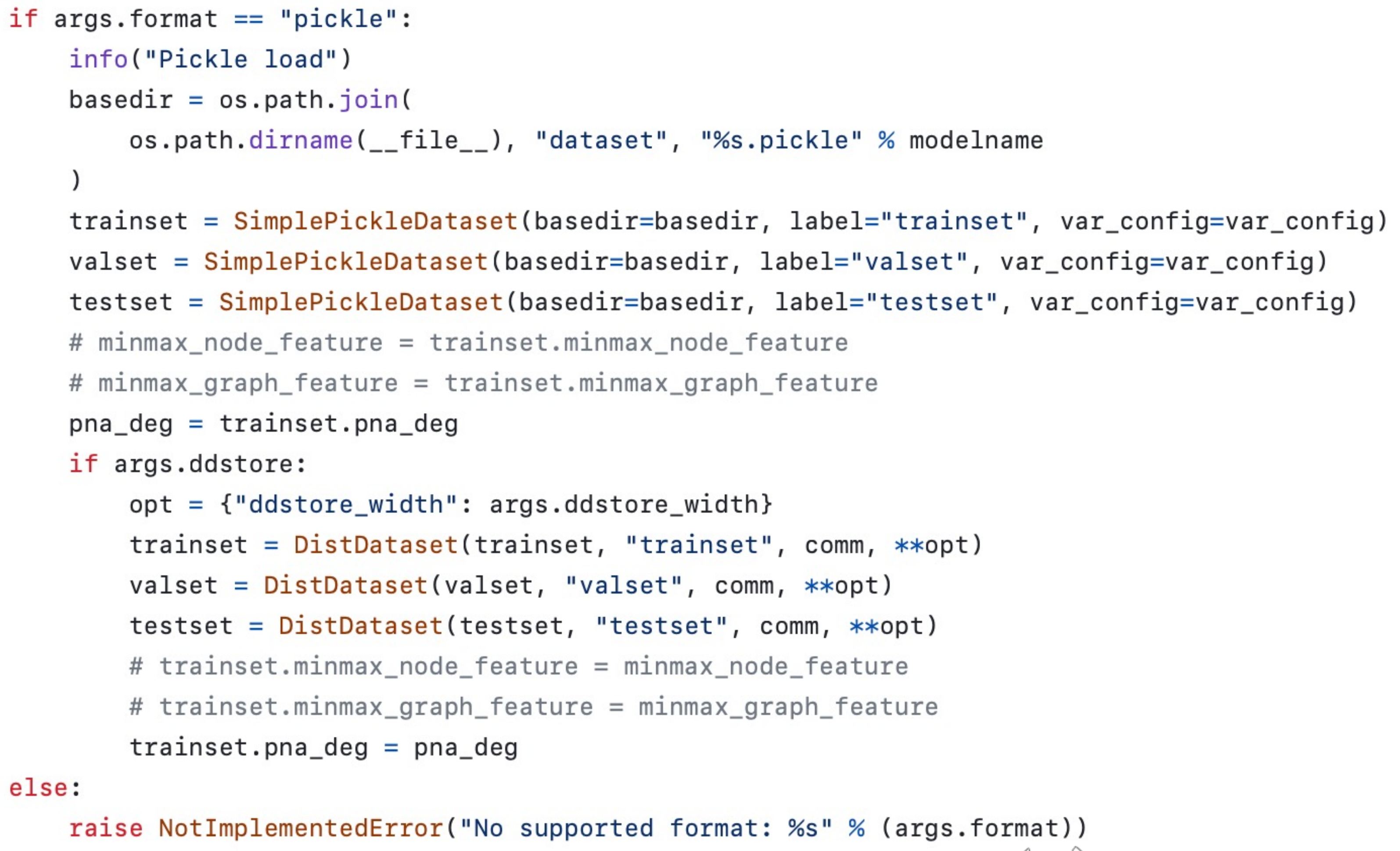


```
elif args.format == "pickle":
    basedir = os.path.join(
        os.path.dirname(__file__), "dataset", "%s.pickle" % modelname
    )
    attrs = dict()
    attrs["pna_deg"] = deg
    SimplePickleWriter(
        trainset,
        basedir,
        "trainset",
        # minmax_node_feature=total.minmax_node_feature,
        # minmax_graph_feature=total.minmax_graph_feature,
        use_subdir=True,
        attrs=attrs,
    )
    SimplePickleWriter(
        valset,
        basedir,
        "valset",
        # minmax_node_feature=total.minmax_node_feature,
        # minmax_graph_feature=total.minmax_graph_feature,
        use_subdir=True,
    )
    SimplePickleWriter(
        testset,
        basedir,
        "testset",
        # minmax_node_feature=total.minmax_node_feature,
        # minmax_graph_feature=total.minmax_graph_feature,
        use_subdir=True,
    )
    sys.exit(0)
```

FePt binary alloy with 32 atoms - LSMS-3 data

lsms.py

3. Data loading from pickle files for training



```
if args.format == "pickle":
    info("Pickle load")
    basedir = os.path.join(
        os.path.dirname(__file__), "dataset", "%s.pickle" % modelname
    )
    trainset = SimplePickleDataset(basedir=basedir, label="trainset", var_config=var_config)
    valset = SimplePickleDataset(basedir=basedir, label="valset", var_config=var_config)
    testset = SimplePickleDataset(basedir=basedir, label="testset", var_config=var_config)
    # minmax_node_feature = trainset.minmax_node_feature
    # minmax_graph_feature = trainset.minmax_graph_feature
    pna_deg = trainset.pna_deg
    if args.ddstore:
        opt = {"ddstore_width": args.ddstore_width}
        trainset = DistDataset(trainset, "trainset", comm, **opt)
        valset = DistDataset(valset, "valset", comm, **opt)
        testset = DistDataset(testset, "testset", comm, **opt)
        # trainset.minmax_node_feature = minmax_node_feature
        # trainset.minmax_graph_feature = minmax_graph_feature
        trainset.pna_deg = pna_deg
else:
    raise NotImplementedError("No supported format: %s" % (args.format))
```

Open slide master to edit

FePt binary alloy with 32 atoms - LSMS-3 data

lsms.py

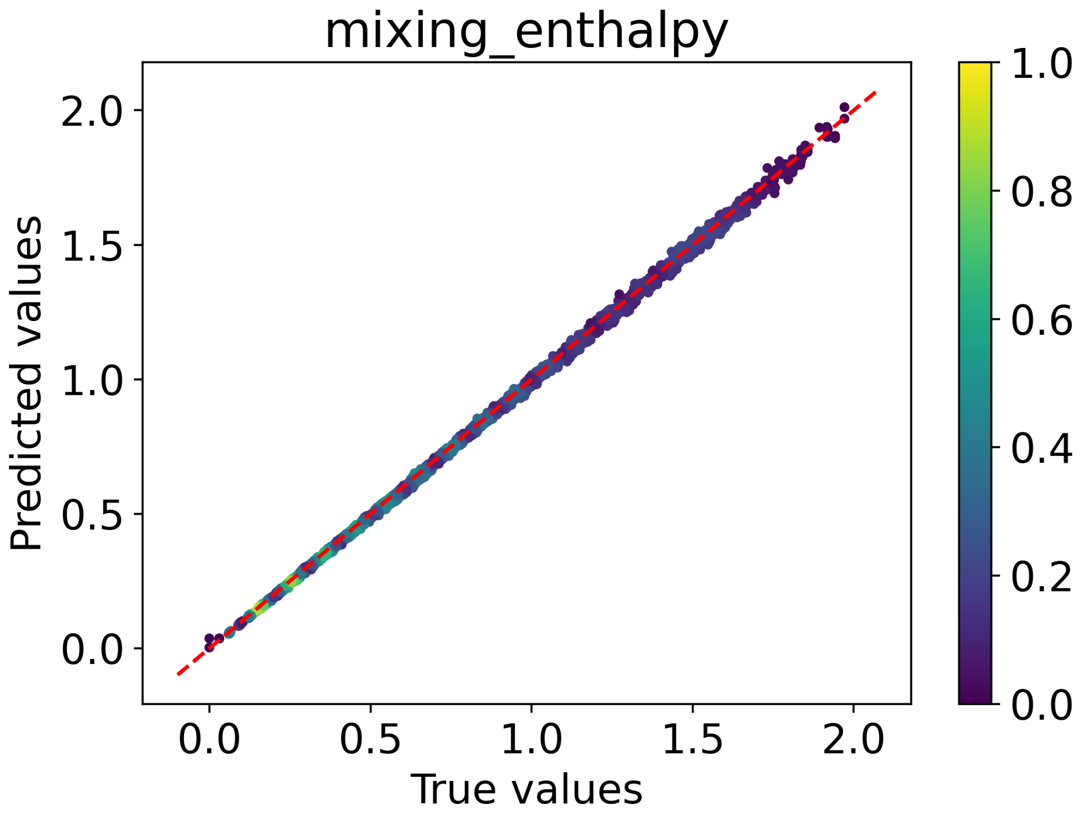
Single-task training for predictions of mixing enthalpy

inference.py

Load pre-trained model and run inference on testing data

Test MAE:

0.010 Rydberg

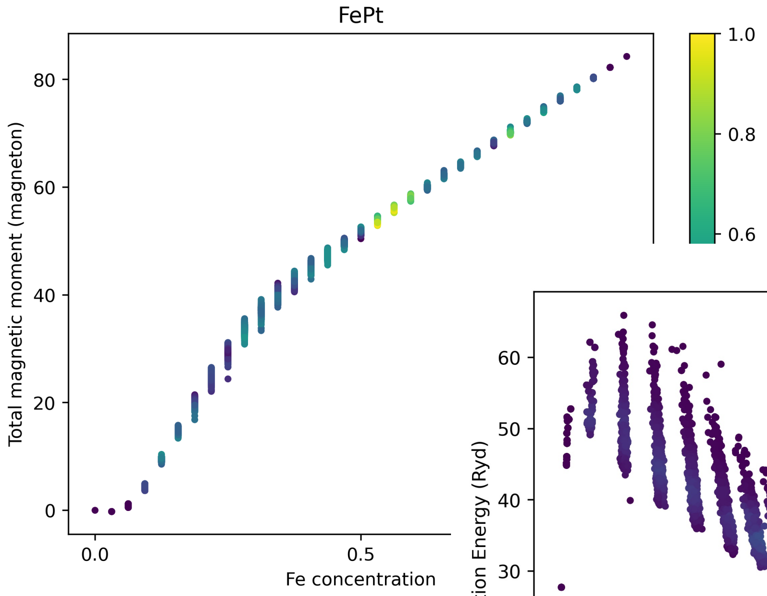


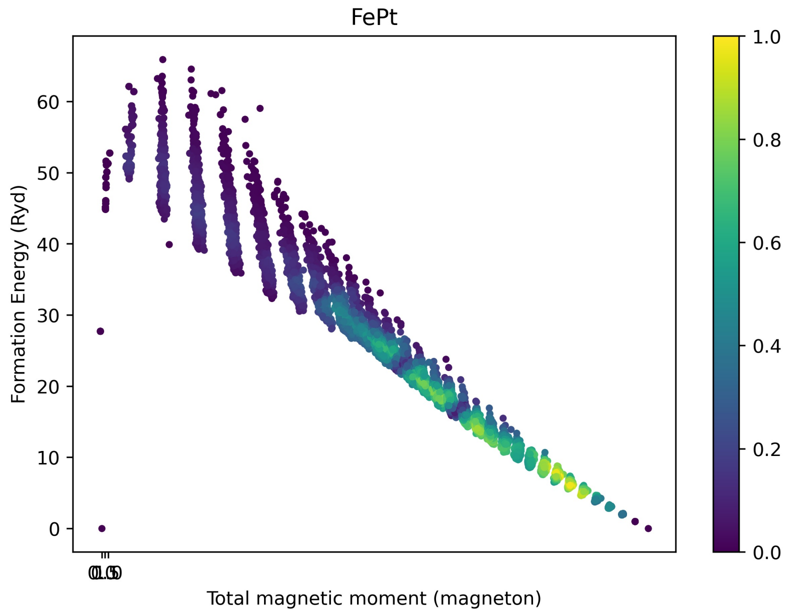
```
"NeuralNetwork": {
    "Architecture": {
        "model_type": "PNA",
        "radius": 7,
        "max_neighbours": 100,
        "periodic_boundary_conditions": false,
        "hidden_dim": 100,
        "num_conv_layers": 6,
        "output_heads": {
            "graph":{
                "num_sharedlayers": 2,
                "dim_sharedlayers": 5,
                "num_headlayers": 2,
                "dim_headlayers": [50,25]
            }
        },
        "task_weights": [1.0]
    },
    "Variables_of_interest": {
        "input_node_features": [0],
        "output_names": ["mixing_enthyalpy"],
        "type": ["graph"],
        "output_index": [0],
        "output_dim": [1],
        "denormalize_output": false
    },
```

FePt binary alloy with 32 atoms - LSMS-3 data

Multi-task learning (MTL) for predictions of mixing enthalpy, atomic charge transfer, and atomic magnetic moment

Magnetic moment and mixing enthalpy are strongly correlated, and MTL can use this correlation to stabilize the training





```
"NeuralNetwork": {
    "Architecture": {
        "model_type": "PNA",
        "radius": 7,
        "max_neighbours": 100,
        "periodic_boundary_conditions": false,
        "hidden_dim": 100,
        "num_conv_layers": 6,
        "output_heads": {
            "graph":{
                "num_sharedlayers": 2,
                "dim_sharedlayers": 5,
                "num_headlayers": 2,
                "dim_headlayers": [50,25]
            },
            "node": {
                "num_headlayers": 2,
                "dim_headlayers": [50,25],
                "type": "mlp"
            }
        },
        "task_weights": [1.0, 1.0, 1.0]
    },
    "Variables_of_interest": {
        "input_node_features": [0],
        "output_names": ["mixing_enthalpy","charge_density", "magnetic_moment"],
        "type": ["graph","node","node"],
        "output_index": [0, 1, 2],
        "output_dim": [1, 1, 1],
        "denormalize_output": false
    },
```

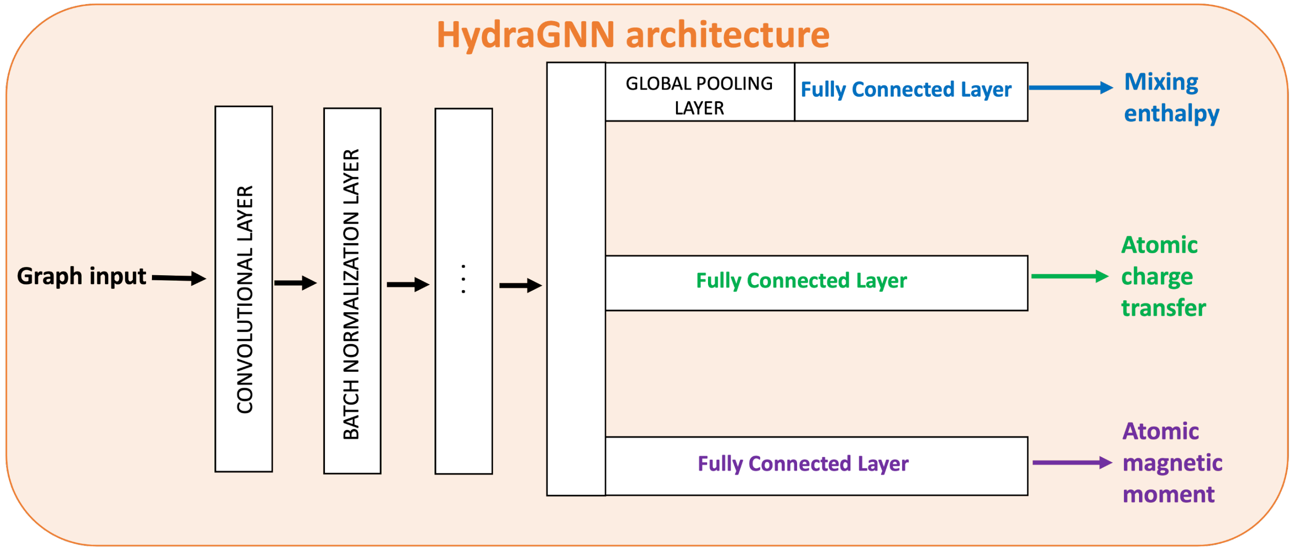
FePt binary alloy with 32 atoms - LSMS-3 data

lsms.py

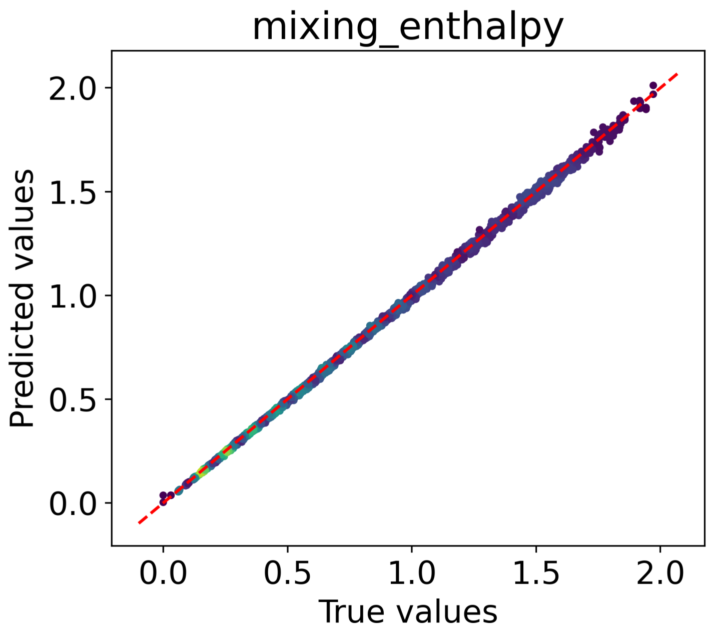
Multi-task training for predictions of mixing enthalpy, atomic charge density, and atomic magnetic moment

inference.py

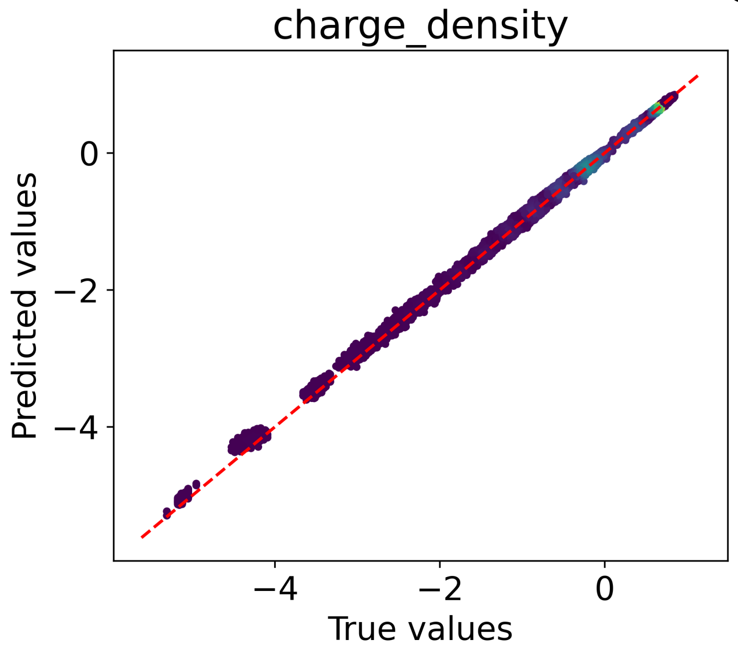
Load pre-trained model and run inference on testing data



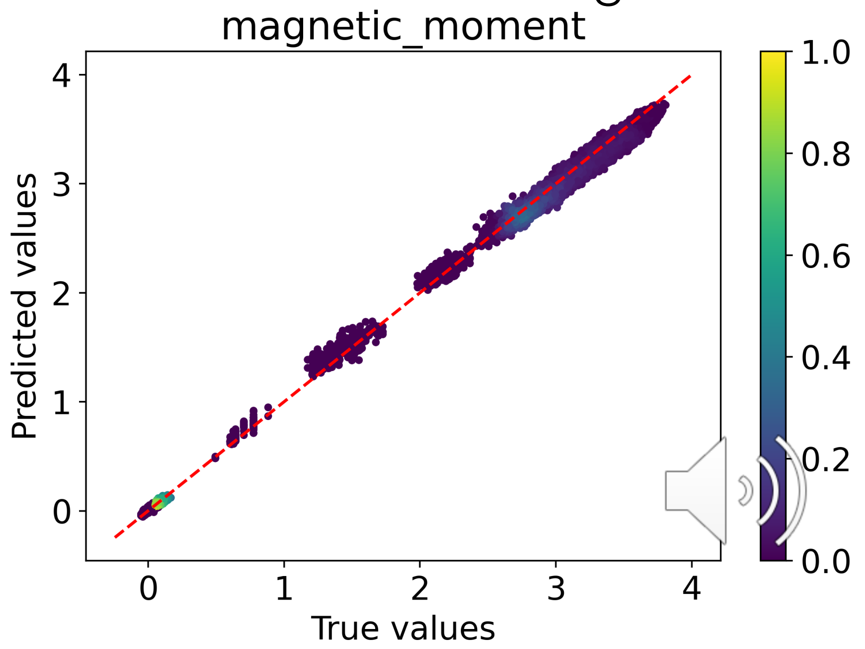
Test MAE: 0.010 Rydberg



Test MAE: 0.66 electron charges



Test MAE: 0.98 magnetons



AIDS HOMO-LUMO dataset

AISD HOMO-LUMO data with DDP

- Code for this example is available at the following GitHub fork: https://github.com/ORNL/HydraGNN/tree/LoG2023_tutorial

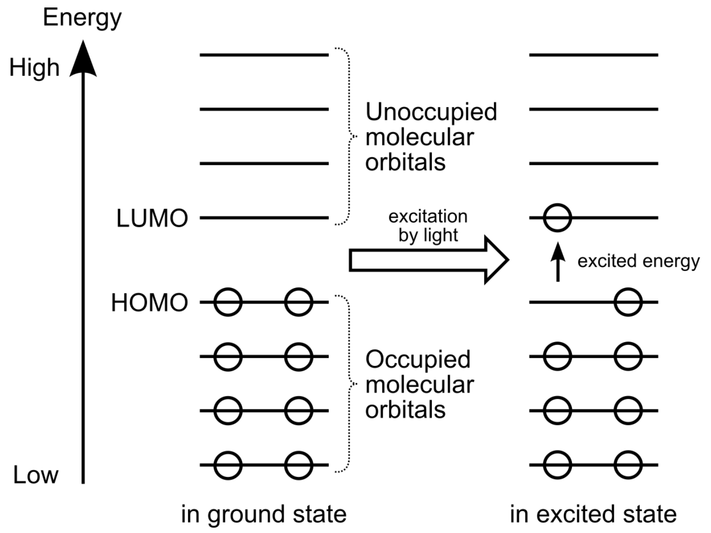
```
$ git clone -b LoG2023_tutorial
https://github.com/ORNL/HydraGNN.git
```

- Python scripts to run for this example are available inside **HydraGNN/examples/csce**
- Demonstrating how to perform **DDP** with **HydraGNN** using **DDStore** on **Frontier, ORNL**
- Main training steps
  - Pre-processing of raw data for DDP and DDStore
  - GNN training
    - With DDP
    - Training with DDStore

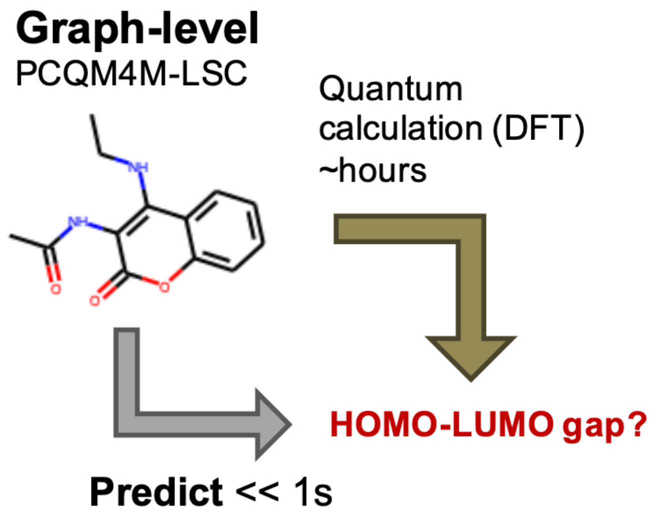
ORNL AISD HOMO-LUMO data

- Graph level prediction
  - Predicting energy gap of molecules given their 2D molecular graphs
  - Over 10.5 M molecules

```
$ cd examples/csce
$ mkdir dataset && cd dataset
$ wget https://users.nccs.gov/~jyc/csce_gap_synth.csv
```



The HOMO and LUMO of a molecule (Wikipedia)



W. Hu, et al., 2021

Frontier environment

- We have HydraGNN development environment on Frontier
  - Python environment
  - Custom build of **PyTorch** and **PyG** to utilize GPUs
  - **DDStore**
  - **mpi4py**
  - Adios

```
module purge
ml DefApps
ml gcc
module unload darshan-runtime

module use -a /gpfs/alpine/world-shared/lrn026/sw/modulefiles
ml anaconda3/2022.10
ml adios2/devel
```

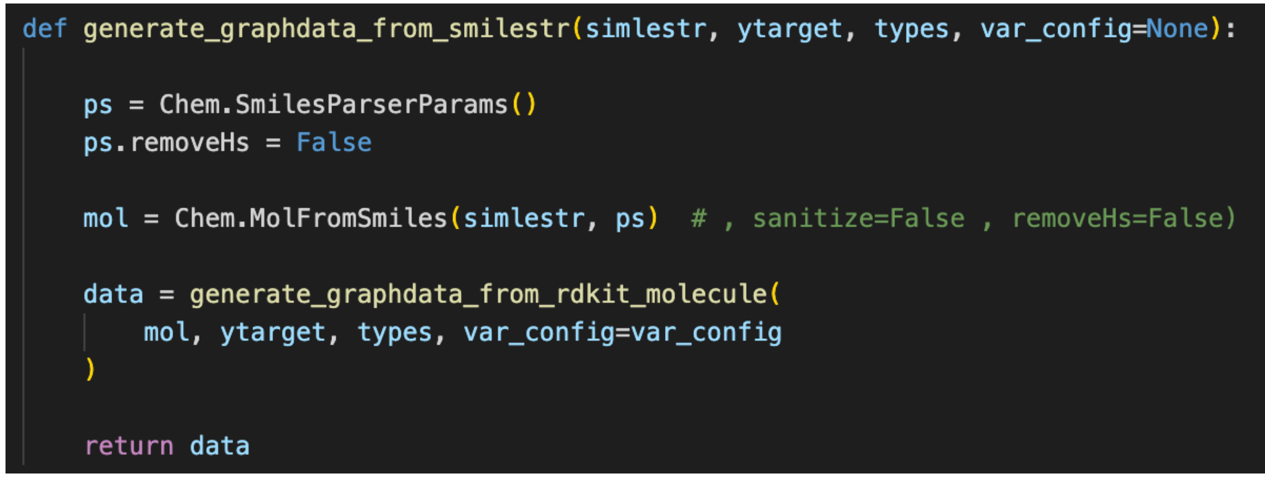
DDStore Setup and Use

1. Read raw data
2. Convert to PyG graph object (`Chem.MolFromSmiles`)
3. Create DDStore object (`PyDDStore`)
4. Register a list of graph objects (`PyDDStore.add`)
5. (Optional) Save as Adios or Pickle format
6. DDStore object is `DataSet`. Combine with `DataLoader` and `DistributedSampler`
7. Call `PyDDStore.get` to retrieve

Note 1: We provide various wrappers and functions

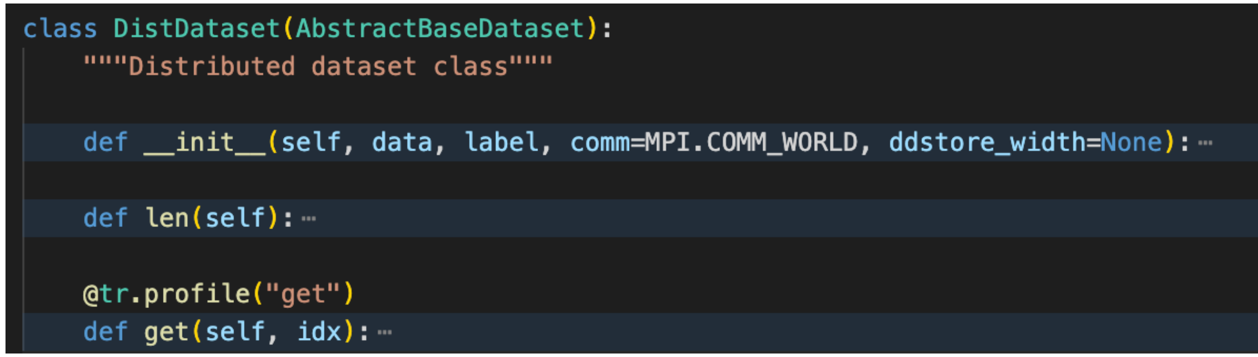
Note 2: We have examples

Hydragnn/utils/smiles_utils.py



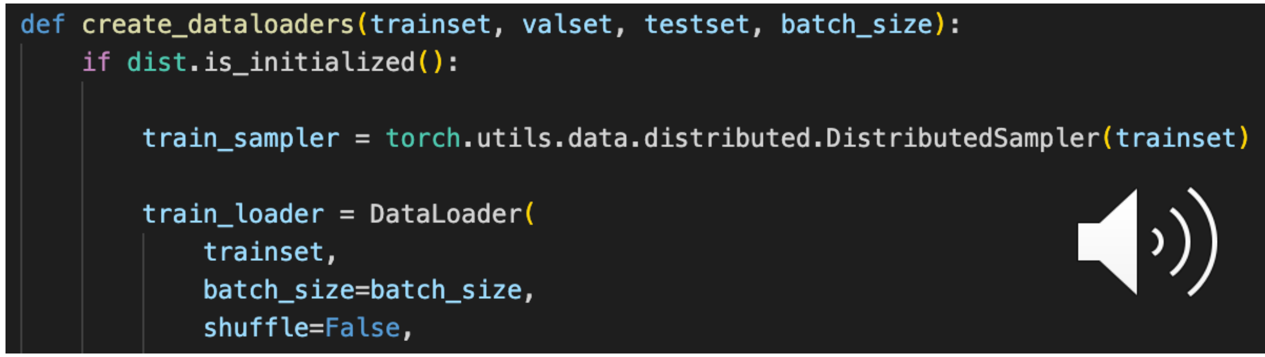
```python
def generate_graphdata_from_smilestr(simlestr, ytarget, types, var_config=None):

    ps = Chem.SmilesParserParams()
    ps.removeHs = False

    mol = Chem.MolFromSmiles(simlestr, ps)  # , sanitize=False , removeHs=False)

    data = generate_graphdata_from_rdkit_molecule(
        mol, ytarget, types, var_config=var_config
    )

    return data
```

Hydragnn/utils/distdaset.py



```python
class DistDataset(AbstractBaseDataset):
    """Distributed dataset class"""

    def __init__(self, data, label, comm=MPI.COMM_WORLD, ddstore_width=None):⋯

    def len(self):⋯

    @tr.profile("get")
    def get(self, idx):⋯
```

Hydragnn/preprocess/load_data.py

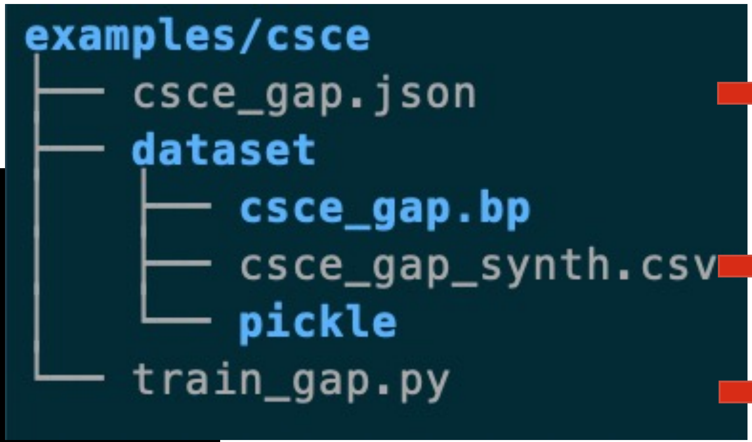


```python
def create_dataloaders(trainset, valset, testset, batch_size):
    if dist.is_initialized():

        train_sampler = torch.utils.data.distributed.DistributedSampler(trainset)

        train_loader = DataLoader(
            trainset,
            batch_size=batch_size,
            shuffle=False,
```

OAK RIDGE National Laboratory

Data pre-processing

```bash
#!/bin/bash
#SBATCH -A LRN026
#SBATCH -J HydraGNN
#SBATCH -t 00:30:00
#SBATCH -p batch
#SBATCH -N 2

export MPICH_ENV_DISPLAY=1
export MPICH_VERSION_DISPLAY=1
export MPICH_GPU_SUPPORT_ENABLED=1
export MPICH_GPU_MANAGED_MEMORY_SUPPORT_ENABLED=1
export MPICH_OFI_NIC_POLICY=GPU
export MIOPEN_DISABLE_CACHE=1
export NCCL_PROTO=Simple

export OMP_NUM_THREADS=7
export PYTHONPATH=$PWD:$PYTHONPATH

srun -n64 python -u examples/csce/train_gap.py --preonly
```



```
examples/csce
├── csce_gap.json
├── dataset
│   ├── csce_gap.bp
│   ├── csce_gap_synth.csv
│   └── pickle
└── train_gap.py
```

Configuration file

Raw data file

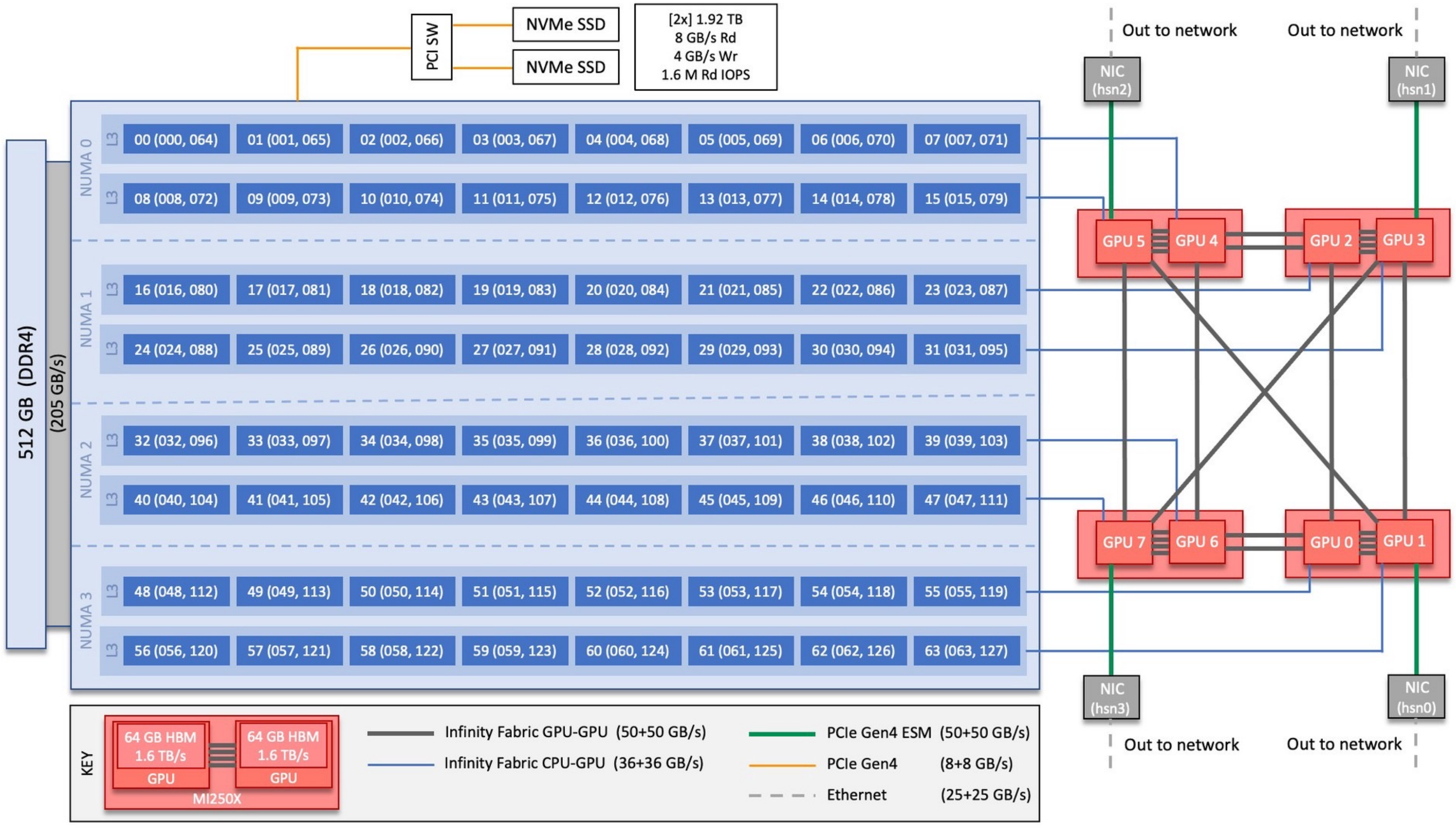
Train script

Oak Ridge National Laboratory

Open slide master to edit

Training

```
#!/bin/bash
#SBATCH -A LRN026
#SBATCH -J HydraGNN
#SBATCH -t 00:30:00
#SBATCH -p batch
#SBATCH -N 32

export MPICH_ENV_DISPLAY=1
export MPICH_VERSION_DISPLAY=1
export MPICH_GPU_SUPPORT_ENABLE
export MPICH_GPU_MANAGED_MEMORY
export MPICH_OFI_NIC_POLICY=GPU
export MIOPEN_DISABLE_CACHE=1
export NCCL_PROTO=Simple

export OMP_NUM_THREADS=7
export PYTHONPATH=$PWD:$PYTHONPATH

srun -n256 -c7 --gpus-per-task=1 --gpu-bind=closest \
    python -u examples/csce/train_gap.py
```

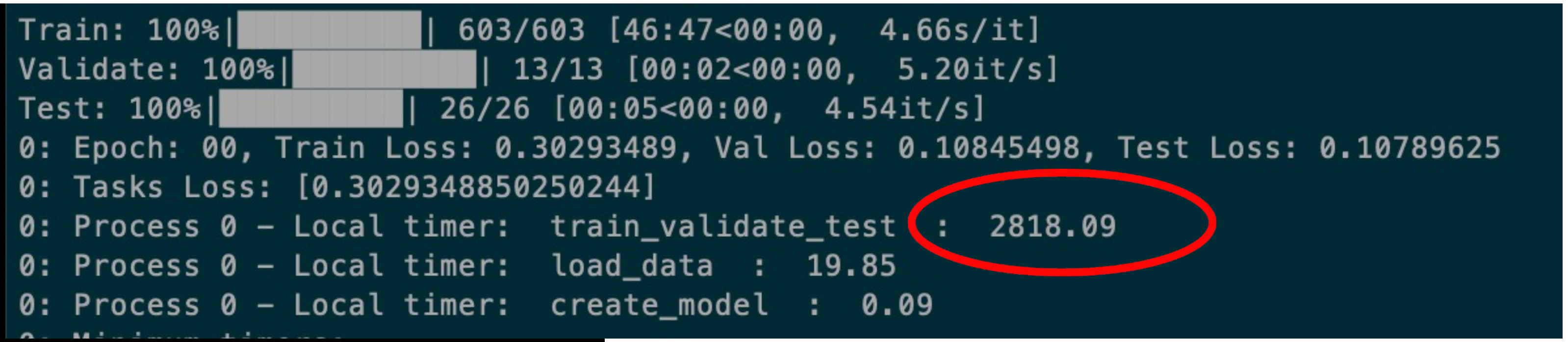


Frontier node layout: 8 GPUs per node

Training with DDStore

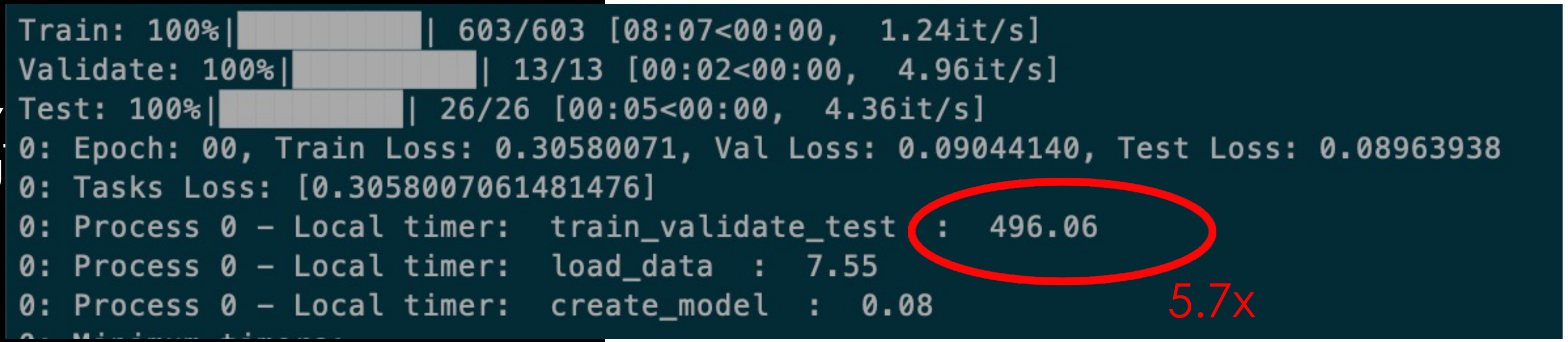
```
#!/bin/bash
#SBATCH -A LRN026
#SBATCH -J HydraGNN
#SBATCH -t 00:30:00
#SBATCH -p batch
#SBATCH -N 32

export MPICH_ENV_DISPLAY=1
export MPICH_VERSION_DISPLAY=1
export MPICH_GPU_SUPPORT_ENABLE
export MPICH_GPU_MANAGED_MEMORY
export MPICH_OFI_NIC_POLICY=GPU
export MIOPEN_DISABLE_CACHE=1
export NCCL_PROTO=Simple

export OMP_NUM_THREADS=7
export PYTHONPATH=$PWD:$PYTHONPATH

srun -n256 -c7 --gpus-per-task=1 --gpu-bind=closest \
    python -u examples/csce/train_gap.py --ddstore
```

No DDStore



```
Train: 100%|██████████| 603/603 [46:47<00:00,  4.66s/it]
Validate: 100%|██████████| 13/13 [00:02<00:00,  5.20it/s]
Test: 100%|██████████| 26/26 [00:05<00:00,  4.54it/s]
0: Epoch: 00, Train Loss: 0.30293489, Val Loss: 0.10845498, Test Loss: 0.10789625
0: Tasks Loss: [0.3029348850250244]
0: Process 0 - Local timer:  train_validate_test  :  2818.09
0: Process 0 - Local timer:  load_data  :  19.85
0: Process 0 - Local timer:  create_model  :  0.09
```

With DDStore



```
Train: 100%|██████████| 603/603 [08:07<00:00,  1.24it/s]
Validate: 100%|██████████| 13/13 [00:02<00:00,  4.96it/s]
Test: 100%|██████████| 26/26 [00:05<00:00,  4.36it/s]
0: Epoch: 00, Train Loss: 0.30580071, Val Loss: 0.09044140, Test Loss: 0.08963938
0: Tasks Loss: [0.3058007061481476]
0: Process 0 - Local timer:  train_validate_test  :  496.06
0: Process 0 - Local timer:  load_data  :  7.55
0: Process 0 - Local timer:  create_model  :  0.08
```

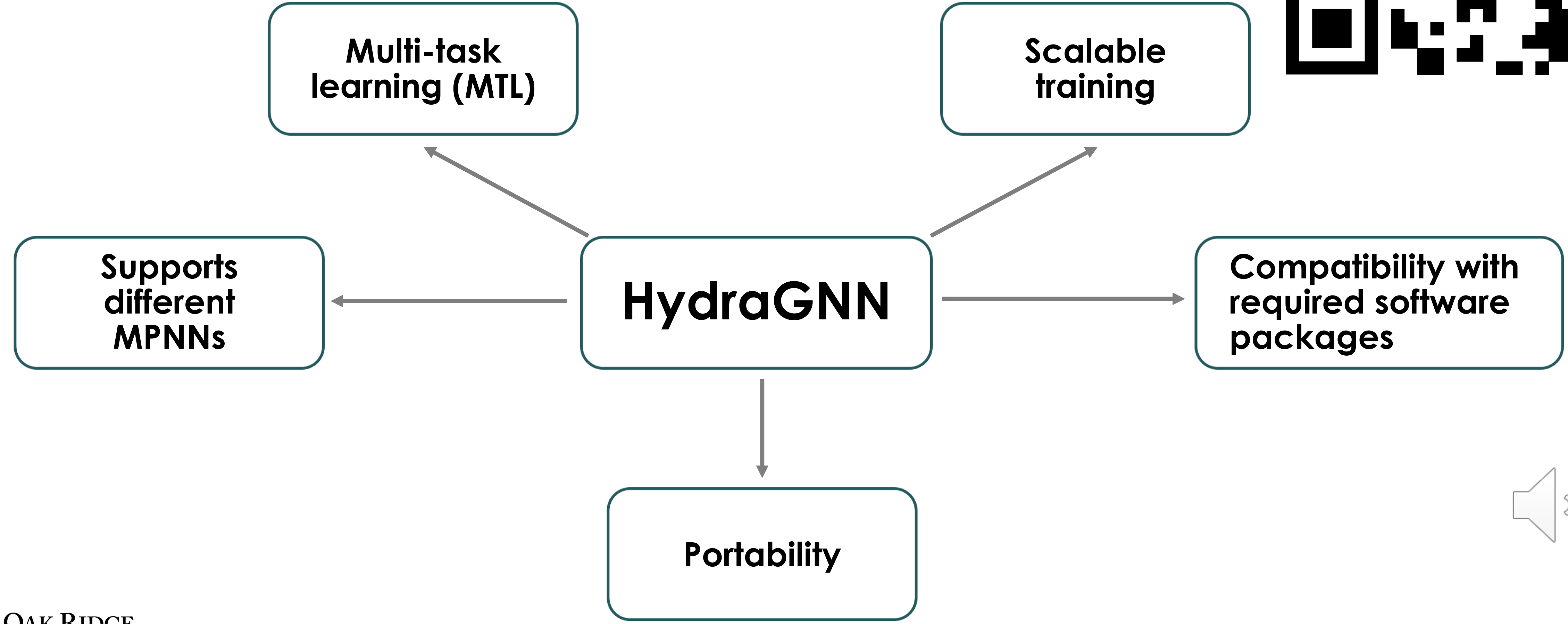
5.7x

Oak Ridge National Laboratory

Conclusions

HydraGNN: enabling large-scale GNN training on HPC

https://www.osti.gov/doecode/biblio/65891
https://github.com/ORNL/HydraGNN


Multi-task learning (MTL)
Scalable training
Supports different MPNNs
HydraGNN
Compatibility with required software packages
Portability

Past and present contributors

- Marko Burčul (master thesis at Politecnico di Milano, Italy)
- Samuel Temple Reeve (Oak Ridge National Laboratory) reevest@ornl.gov
- Kshitij Mehta (Oak Ridge National Laboratory) mehtakv@ornl.gov
- Justin Baker (University of Utah, Salt Lake City) baker@math.utah.edu
- Jonghyun Bae (Lawrence Berkeley National Laboratory) jbae2@lbl.gov
- Khaled Ibrahim (Lawrence Berkeley National Laboratory). kzibrahim@lbl.gov

Acknowledgments

This research is sponsored by the Artificial Intelligence Initiative as part of the Laboratory Directed Research and Development (LDRD) Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the US Department of Energy under contract DE-AC05-00OR22725. This work was supported in part by the US-DOE Advanced Scientific Computing Research (ASCR) under contract DE-SC0023490.

This research has been supported by the SciDAC Institute for Computer Science, Data, and Artificial Intelligence (RAPIDS), Lawrence Berkeley National Laboratory, which is operated by the University of California for the U.S. Department of Energy under contract DE-AC02-05CH11231.

This research used resources of the Oak Ridge Leadership Computing Facility and of the Edge Computing program at the Oak Ridge National Laboratory. An award of computer time was provided by the OLCF Director's Discretion Project program using the OLCF awards MAT250 and LRN026.

This research also used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725 and No. DE-AC02-05CH11231 using NERSC award ASCR-ERCAP0025216.

Thank you!

Questions?

Massimiliano (Max) Lupo Pasini
Data Scientist
lupopasinim@ornl.gov

Computational Sciences and Engineering Division (CSED)

Oak Ridge National Laboratory

Pei Zhang
Computational Scientist
zhangp1@ornl.gov

Computational Sciences and Engineering Division (CSED)

Oak Ridge National Laboratory

Jong Youl Choi
Computer Scientist
choij@ornl.gov

Computer Science and Mathematics Division (CSMD)

Oak Ridge National Laboratory