OAK RIDGE National Laboratory

HydraGNN: a Scalable Graph Neural Network Architecture for Material Property Predictions

Massimiliano (Max) Lupo Pasini

Jong Youl Choi

Pei Zhang

Kshitij Mehta

Pablo Seleson

Oak Ridge National Laboratory

ORNL is managed by UT-Battelle LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

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

Kshitij Mehta
Computer Scientist
mehtakv@ornl.gov

Computer Science and Mathematics Division (CSMD)

Oak Ridge National Laboratory

Pablo Seleson
Research Scientist
selesonpd@ornl.gov

Computer Science and Mathematics Division (CSMD)

Oak Ridge National Laboratory

Outline

- Introduction [Max ~ 20 mins]
  - Motivations
  - HydraGNN overview
  - Software ecosystem
  - Highlights of past scientific results
- Mathematical background on graph neural networks (GNNs) [Pablo, ~20 mins]
- Instructions to download docker container with HydraGNN [Jong, ~10 mins]
- Example demonstrations [Pei and Max, ~30 mins]
- Scalable data management and training on DOE supercomputers [Jong and Kshitij, ~20 mins]
- Conclusions [Max, 2 mins]
- Back-up slides with instructions to install and run HydraGNN on OLCF-Frontier

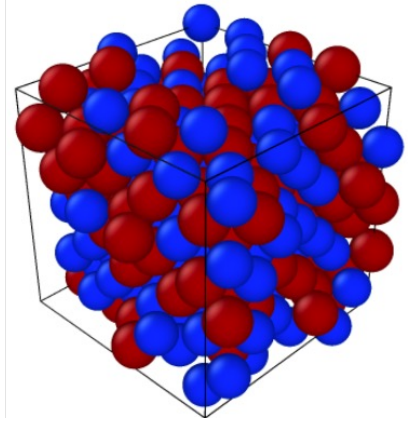
OAK RIDGE National Laboratory

Introduction

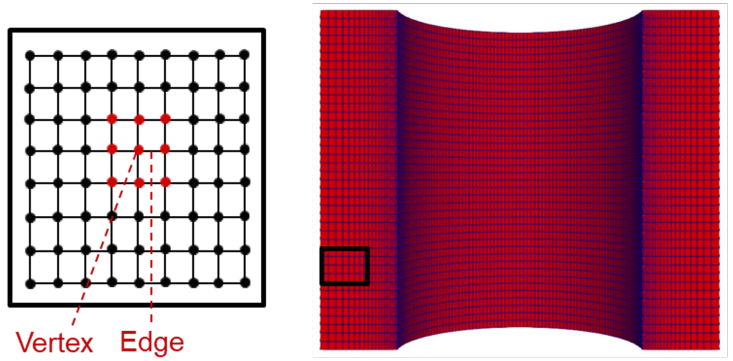
Massimiliano Lupo Pasini (CSED)

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

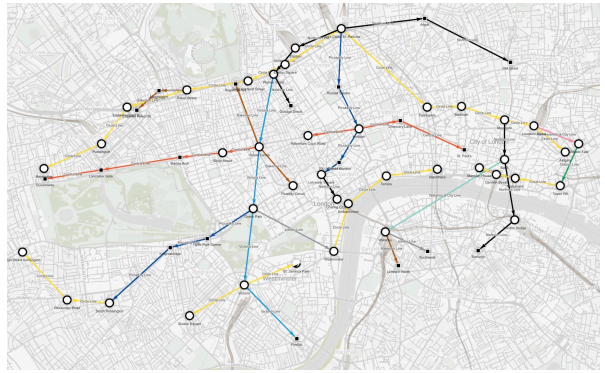


Image from https://memgraph.com/blog/modeling-visualizing-navigating-a-transportation-network-with-memgraph

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

Why use GNNs?

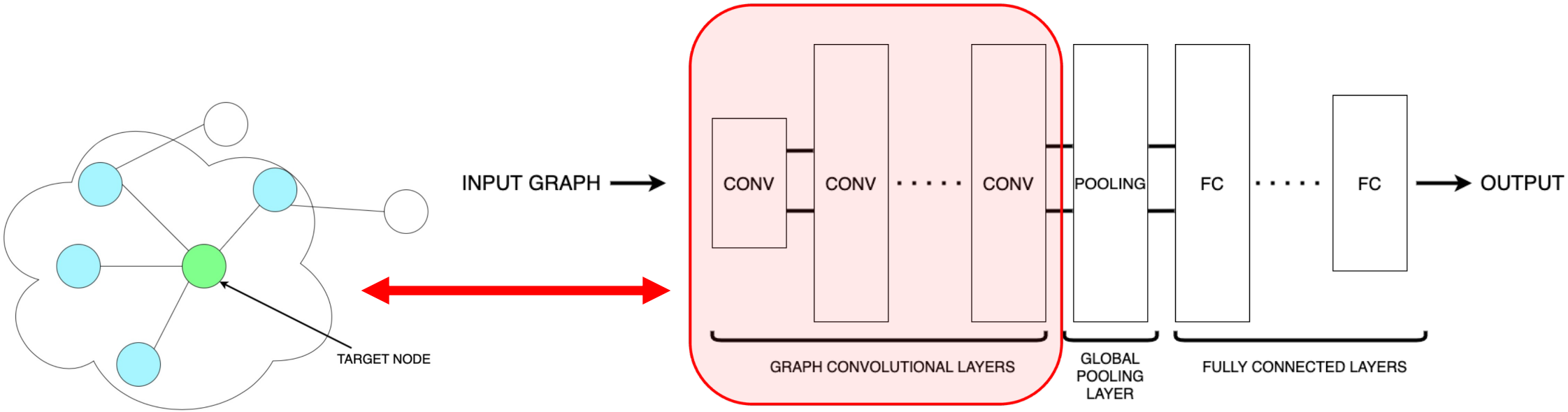
	Scalable training w.r.t. graph sizes	Capability to process unstructured graphs
Multi-layer perceptron (MLP)	✗	✓
Convolutional neural network (CNN)	✓	✗
GNN	✓	✓

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 synthesize 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;

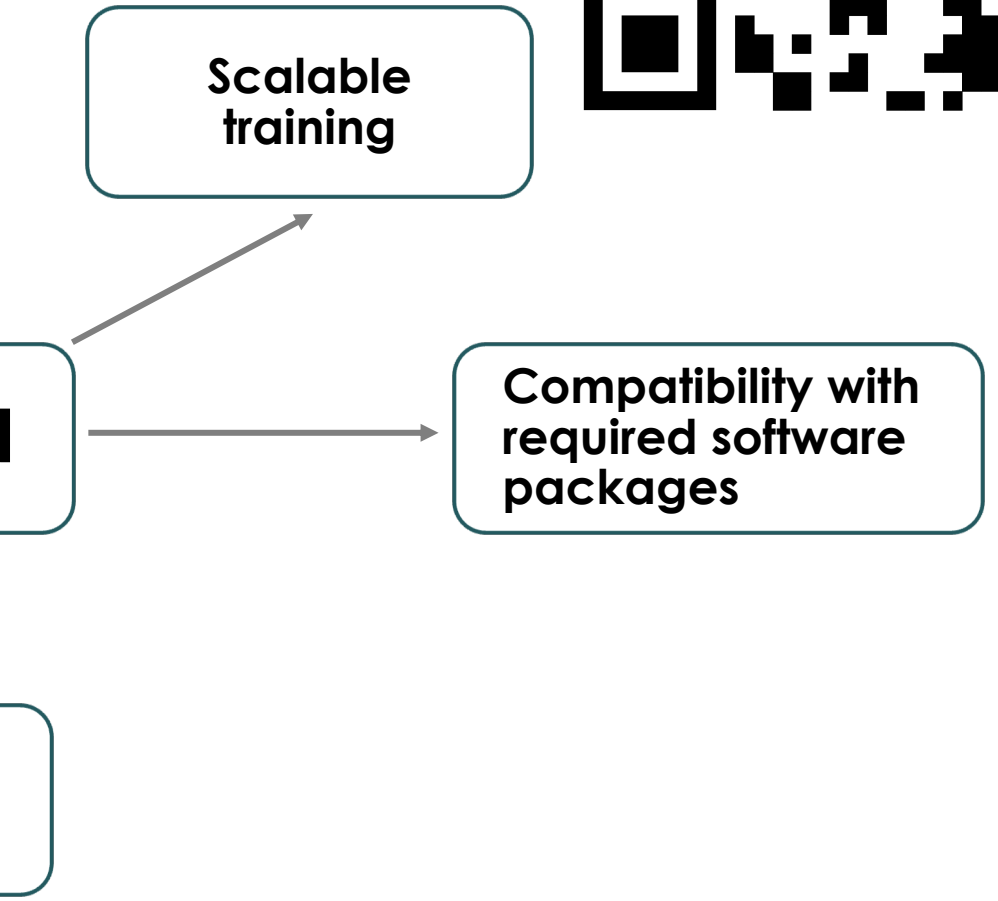
(3) distributed data parallelism (DDP) effectively implemented to address scaling challenges on large-scale supercomputing facilities;

(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

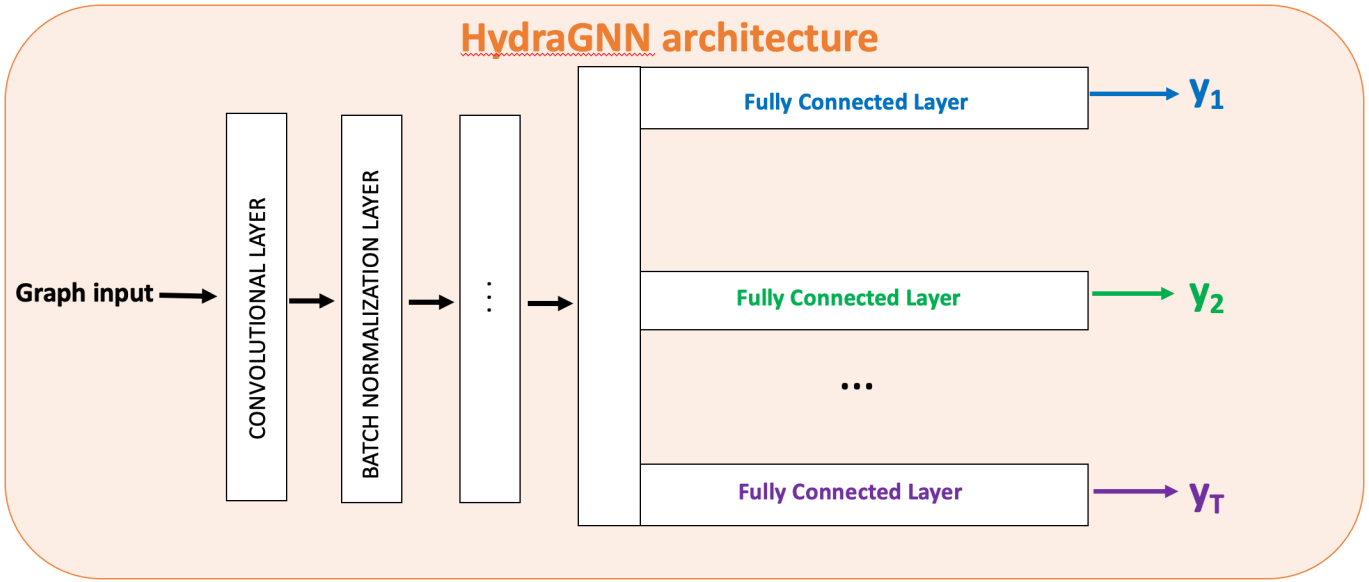


HydraGNN: Multi-task learning (MTL) for stabilization by extracting physics correlations between multiple target properties of interest

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**



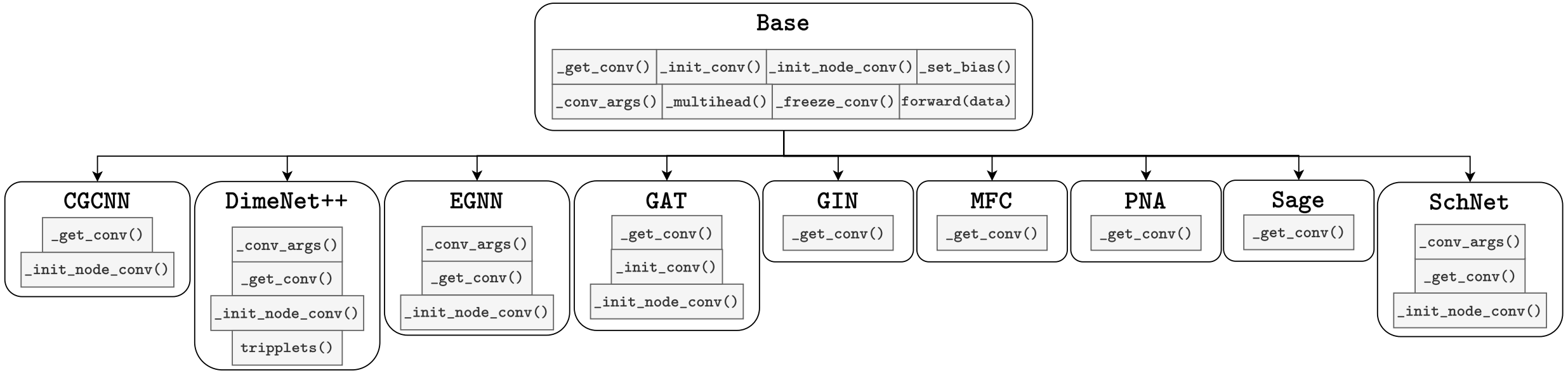
$\mathbf{W}$ = parameters of the neural network to optimize during the training

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left\|\mathbf{y}_{\text{predict},1}(\mathbf{w}) - \mathbf{y}_1\right\|_2^2 + \left\|\mathbf{y}_{\text{predict},2}(\mathbf{w}) - \mathbf{y}_2\right\|_2^2 + \ldots + \left\|\mathbf{y}_{\text{predict},T}(\mathbf{w}) - \mathbf{y}_T\right\|_2^2$$

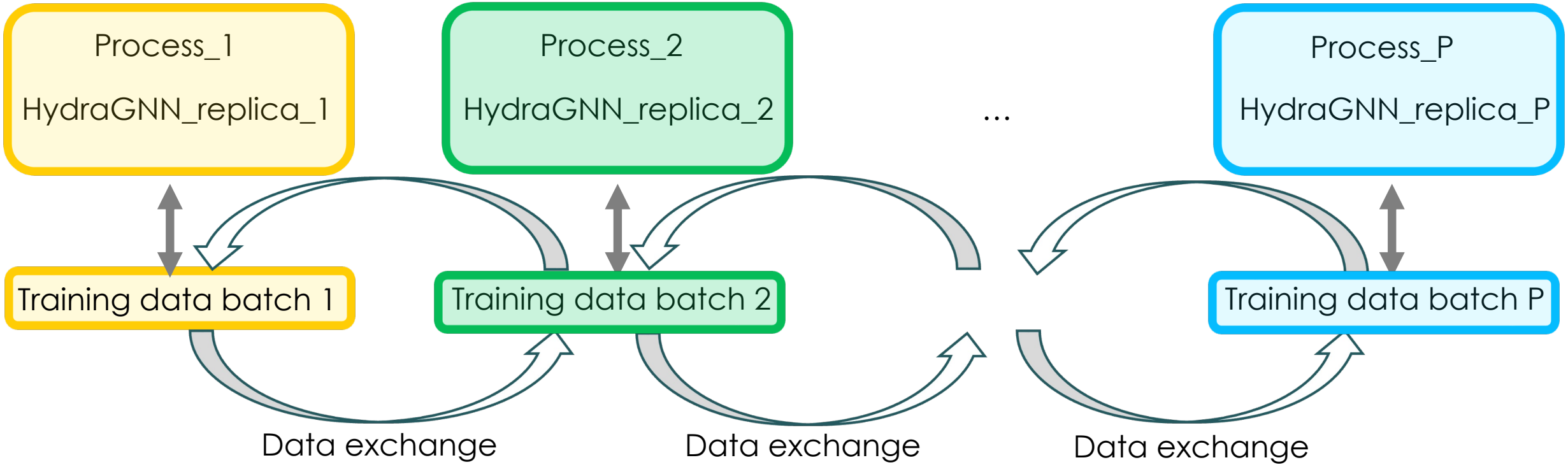
Global Multi-Task Training Loss Function

HydraGNN: Message passing layer treated as hyperparameter

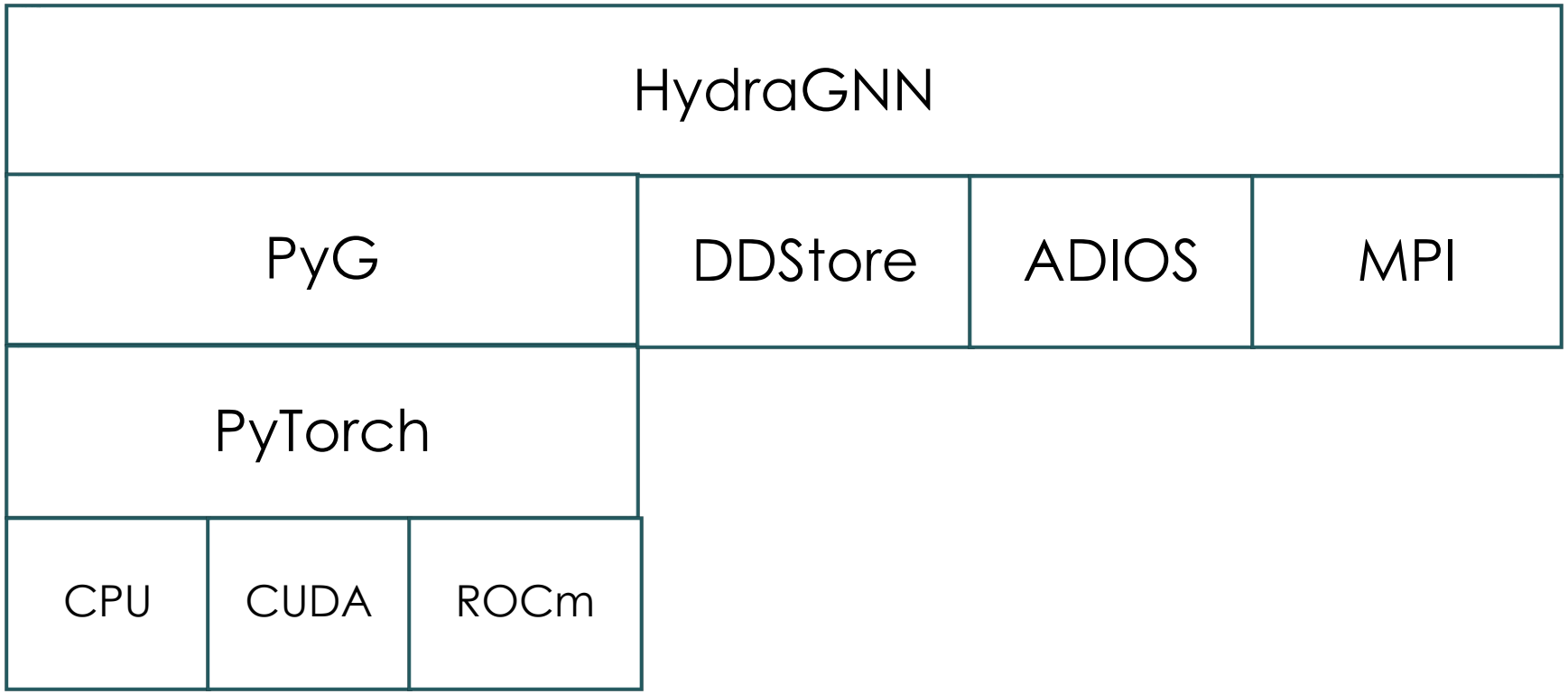
Object-oriented programming enables seamless switch between different MPNN layers that can be treated as hyperparameters



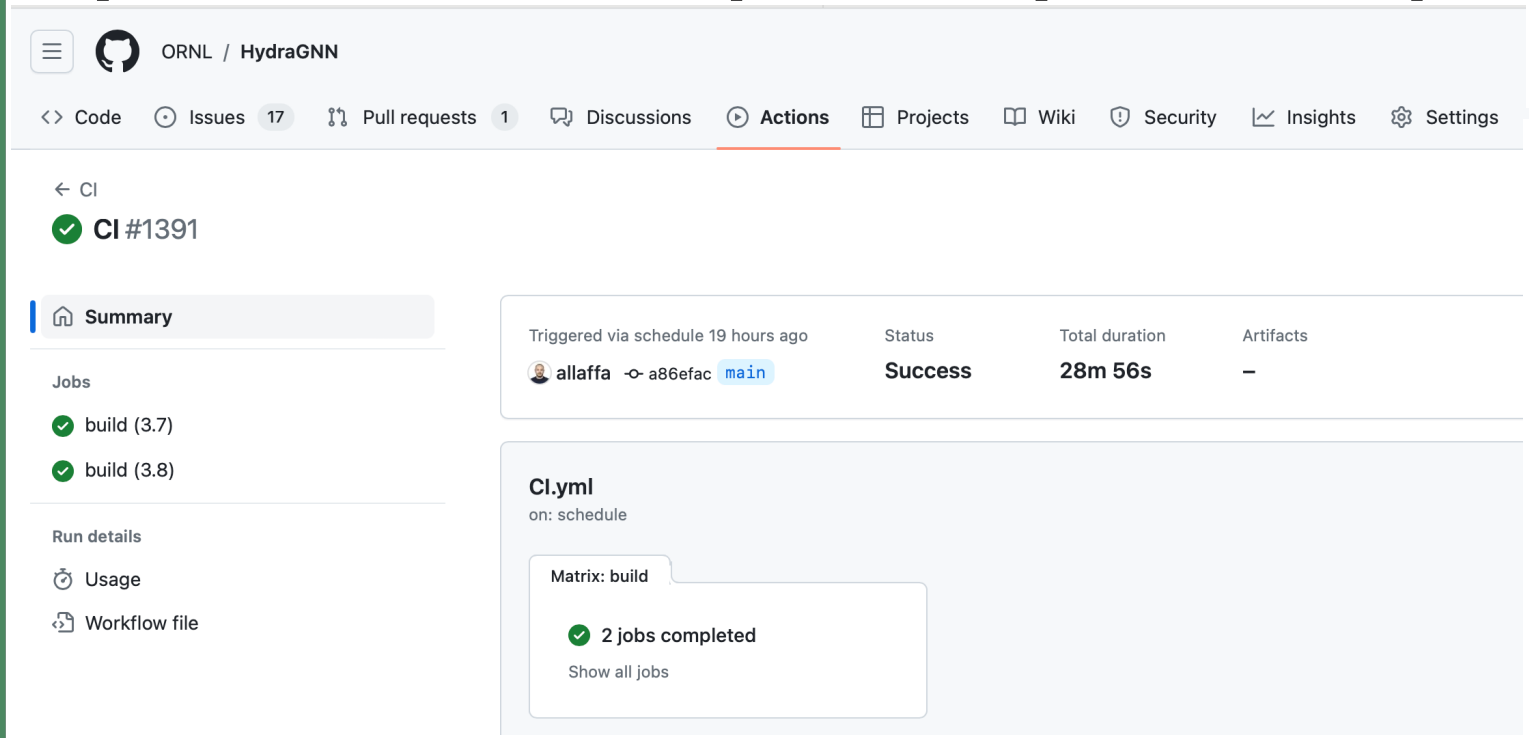
HydraGNN: Scalable training with Distribute Data Parallelism (DDP)

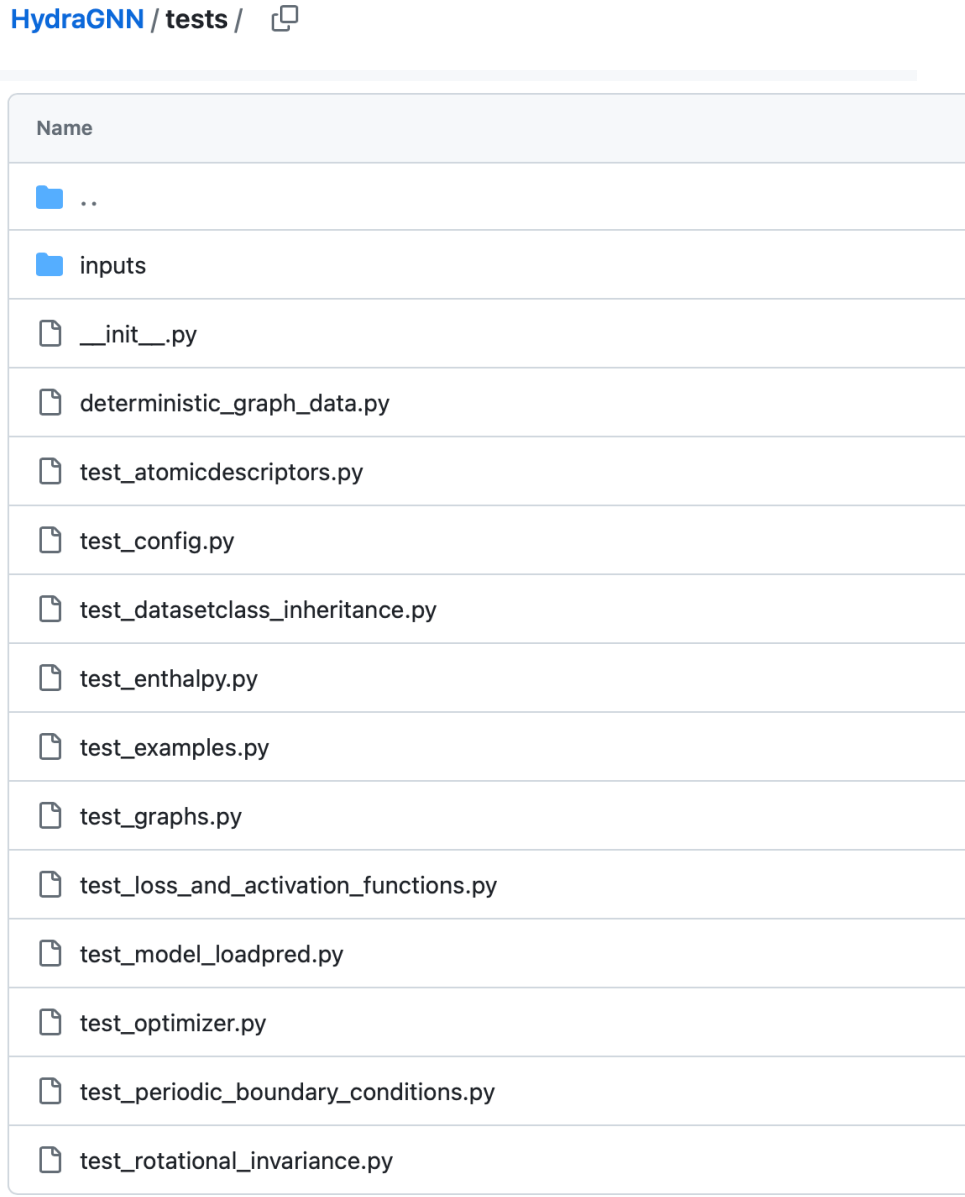


Software ecosystem



HydraGNN: Compatibility with required software packages





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:

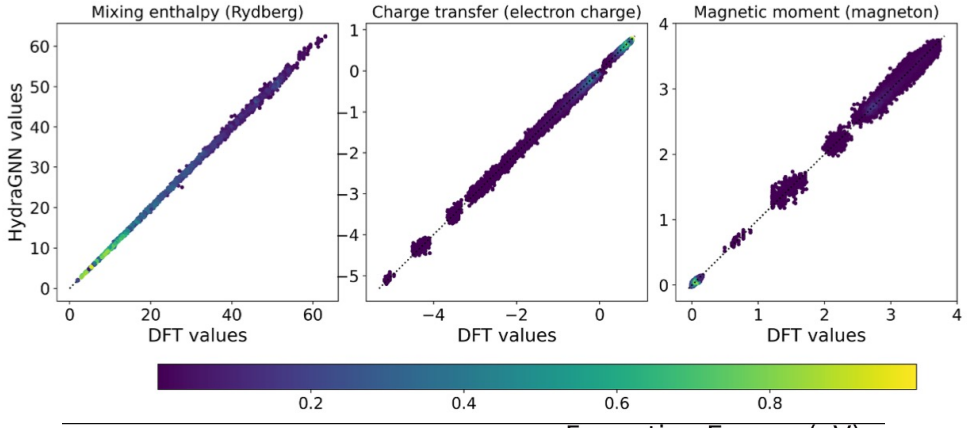
- Personal laptops for small scale training
- ORNL Edge Computing DGX boxes using docker containers
- OLCF CADES clusters using conda environments: https://www.olcf.ornl.gov/tag/cades/
- OLCF supercomputer Summit (NVIDIA V100 GPUs): https://www.olcf.ornl.gov/summit/
- NERSC supercomputer Perlmutter (NVIDIA 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

Highlights of past scientific results with HydraGNN

Solid State Physics

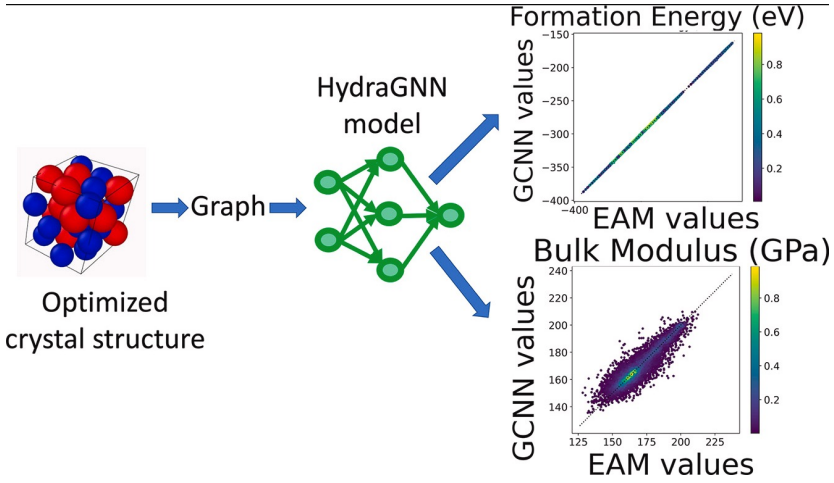
Simultaneous prediction of enthalpy, charge transfer, and magnetic moment for single-phase solid solution alloys

M. Lupo Pasini. P. Zhang, S. T, Reeve, and J. Y. Choi.
Multi-task graph neural networks for simultaneous prediction of global and atomic properties in ferromagnetic systems
Mach. Learn.: Sci. Technol. 3 025007
DOI 10.1088/2632-2153/ac6a51



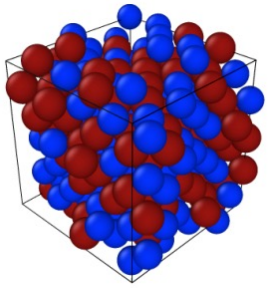
Prediction of enthalpy and mechanical properties for multi-phase solid solution alloys

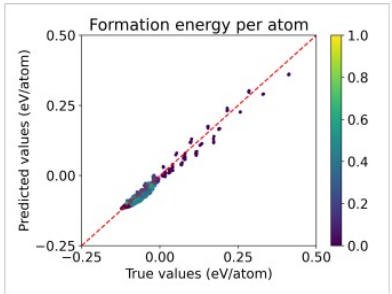
M. Lupo Pasini., GS Jung, and S. Irle. *Graph neural networks predict energetic and mechanical properties for models of solid solution metal alloy phases.*
Computational Materials Science, Volume 224, May 2023, 112141
DOI 10.1088/2632-2153/ac6a51



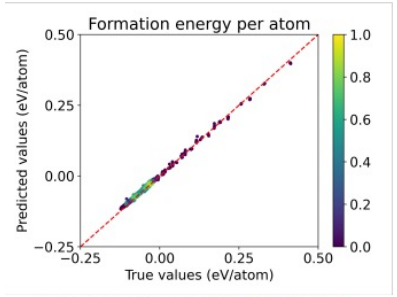
Transferable predictions of formation enegy across lattice of increasing size

M. Lupo Pasini., M. Karabin, and M. Eisnebach, Transferable prediction energy across lattices of increasing size
Mach. Learn.: Sci. Technol (accepted)
Preprint 10.26434/chemrxiv-2023-c14r3-v2





Train on 256 atoms - Test on 2,048 atoms



Train on 256 atoms + 20% 864 atoms
+ 20% 2,048 atoms
Test on 2,048 atoms

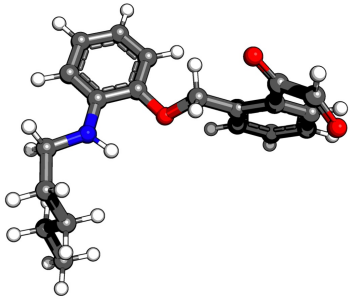
OAK RIDGE National Laboratory

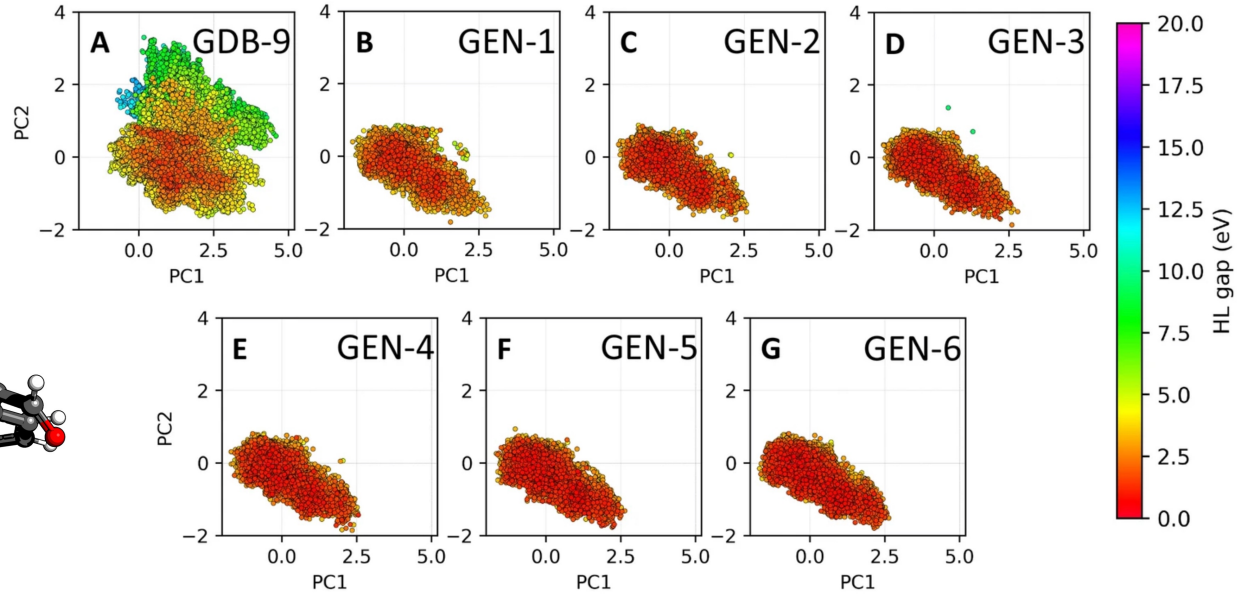
Highlights of past scientific results with HydraGNN

Organic Chemistry

AI-accelerated workflow for design of organic molecules with improved opto-electronic properties

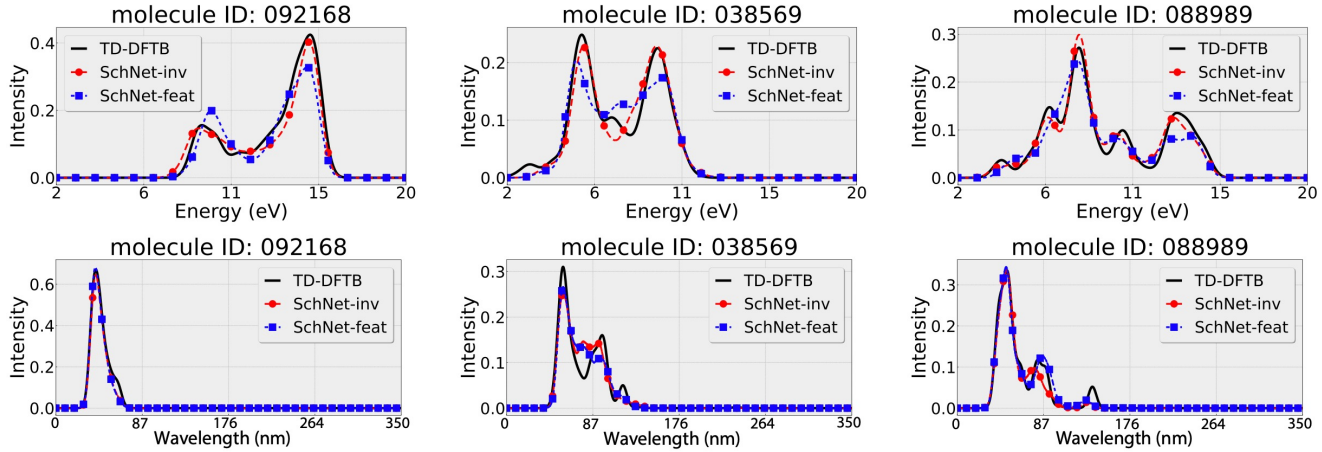
Pilsun Yoo, Debsindhu Bhowmik, Kshitij Mehta, Pei Zhang, Frank Liu, Massimiliano Lupo Pasini, and Stephan Irle. *Deep learning workflow for the inverse design of molecules with specific optoelectronic properties*.
Sci Rep 13, 20031 (2023).
https://doi.org/10.1038/s41598-023-45385-9





Invariant and Equivariant message passing for improved predictions of ultraviolet-visible spectra

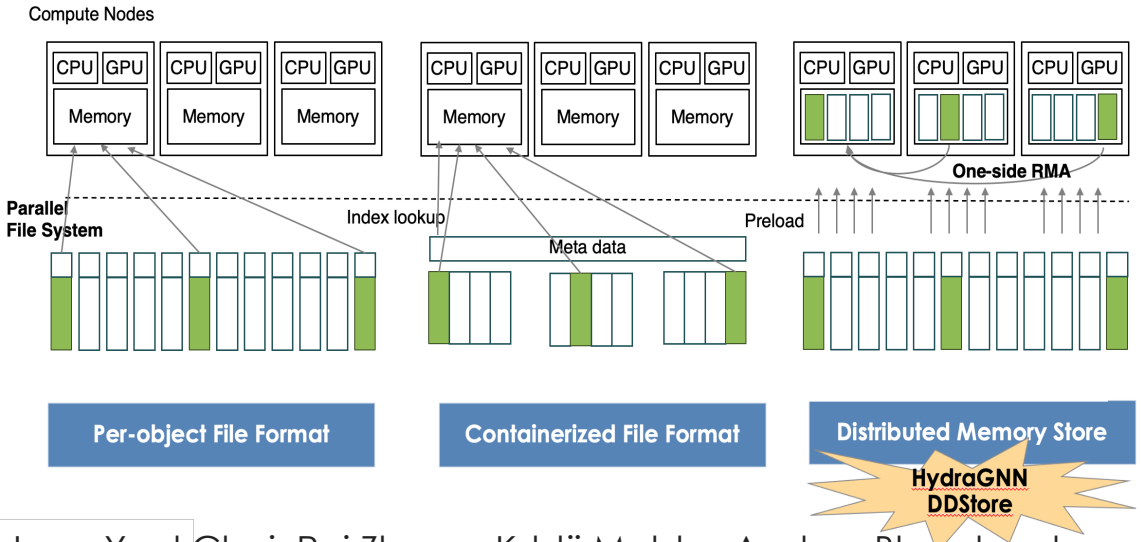
Justin Baker, Massimiliano Lupo Pasini, and Cory Hauck. *Invariant Features for Accurate Predictions of Quantum Chemical UV-vis Spectra of Organic Molecules*. IEEE Southeast Conference 2024 (accepted). Preprint 10.26434/chemrxiv-2023-9n306

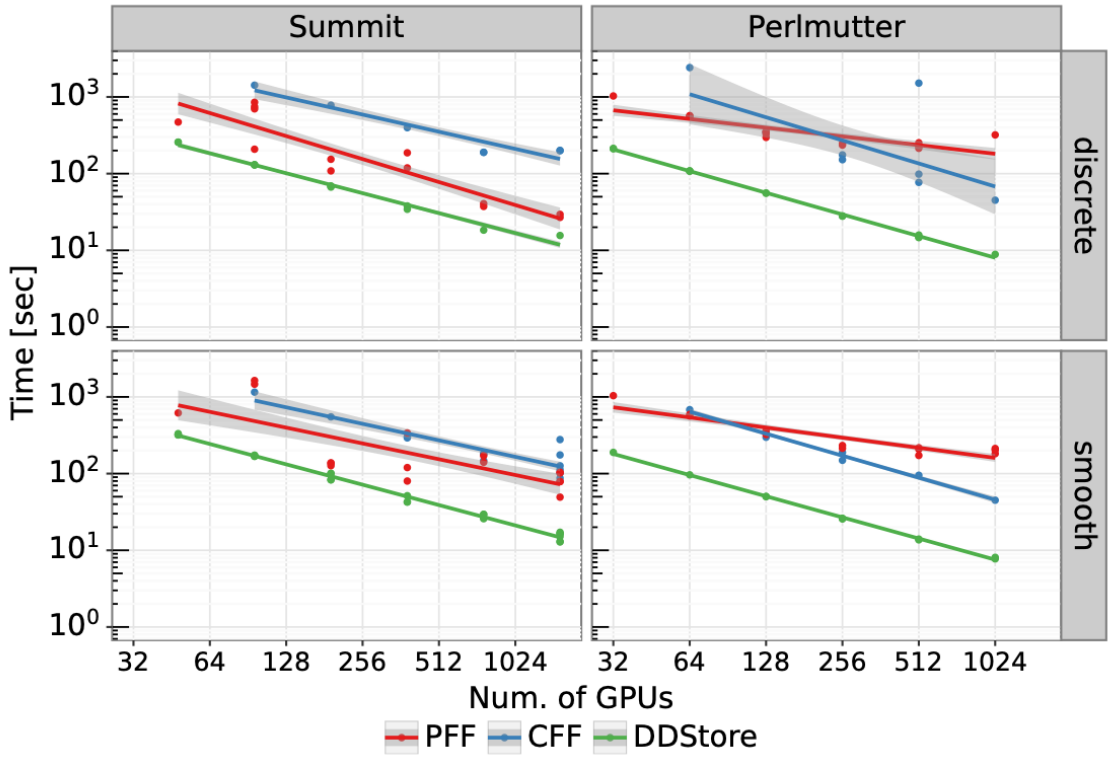


OAK RIDGE National Laboratory

Highlights of past scientific results with HydraGNN

High-Performance Computing



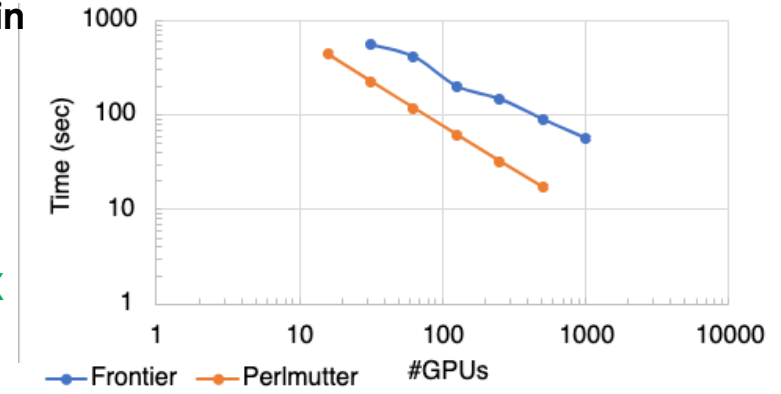


Jong Youl Choi, Pei Zhang, Kshtij Mehta, Andew Blanchard, and Massimiliano Lupo Pasini. *Scalable training of graph convolutional neural networks for fast and accurate predictions of HOMO-LUMO gap in molecules*. *J Cheminform* **14**, 70 (2022). https://doi.org/10.1186/s13321-022-00652-1

Jong Youl Choi, Massimiliano Lupo Pasini, Pei Zhang, Kshitij Mehta, Frank Liu, Jonghyun Bae, and Khaled Ibrahim. 2023. *DDStore: Distributed Data Store for Scalable Training of Graph Neural Networks on Large Atomistic Modeling Datasets*. In Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W '23). Association for Computing Machinery, New York, NY, USA, 941–950. https://doi.org/10.1145/3624062.3624171

OLCF Hackathon in March 2024

Optimized HydraGNN on OLCF-Frontier by speeding up the training time of 3X



OAK RIDGE National Laboratory

OAK RIDGE National Laboratory

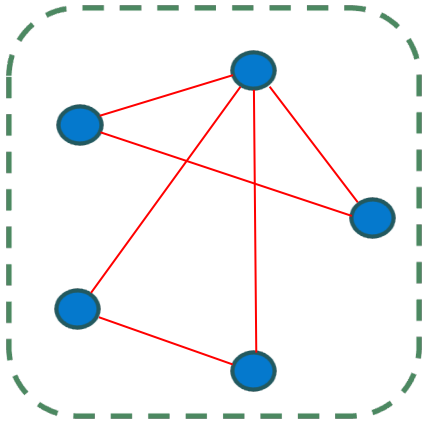
Mathematical Background on Graph Neural Networks

Pablo Seleson (CSMD)

Graph neural networks (GNNs)

GNNs are neural networks that operate on graph data.

Graph: a representation of the relations (edges) between a collection of entities (nodes)



V: Vertex (or node) attributes
e.g., node identity, number of neighbors

E: Edge (or link) attributes and directions
e.g., edge identity, edge weight

U: Global (or master node) attributes
e.g., number of nodes, longest path

https://distill.pub/2021/gnn-intro/

Graph examples: structured vs. unstructured data

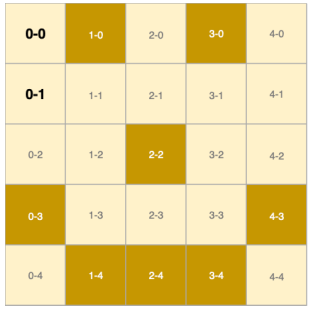
Example: images as graphs (structured data)

Nodes: pixels

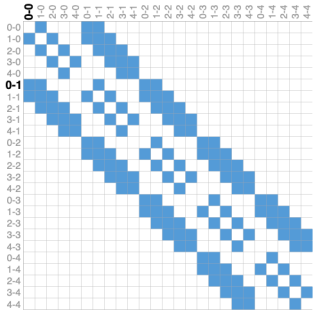
Edges: links to adjacent pixels

Another example: text

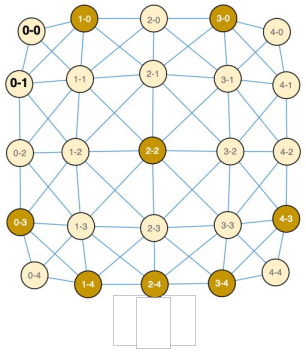
Image pixels



Adjacency matrix



Graph



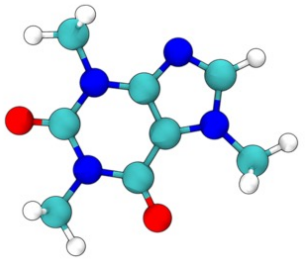
Example: molecules as graphs (unstructured data)

Nodes: atoms

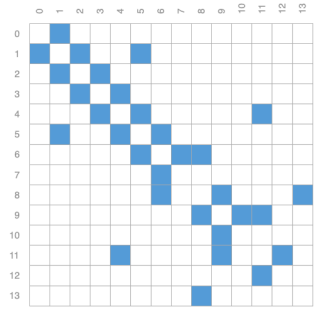
Edges: covalent bonds

Another example: social networks

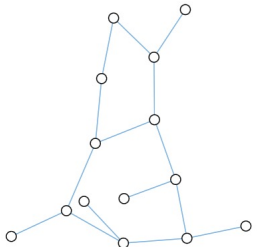
Molecule



Adjacency matrix



Graph



Unstructured data don't have identical adjacency matrices.

https://distill.pub/2021/gnn-intro/

Graph prediction task types

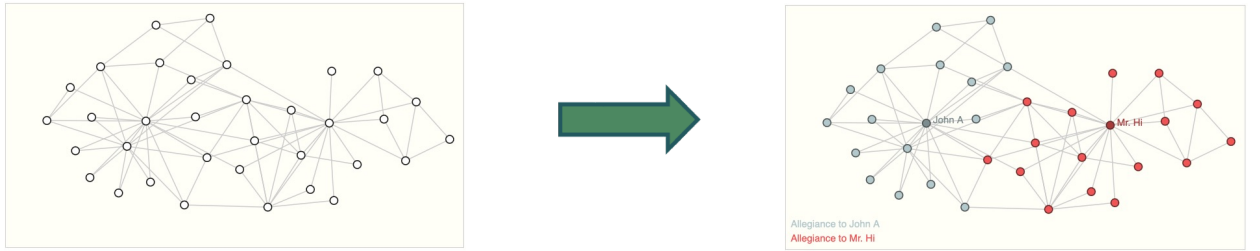
Graph-level task: predict property of entire graph

Input:
graphs

Output:
labels for each graph

Node-level task: predict property of each node in a graph

Input:
graph with
unlabeled nodes



Output:
graph node labels

Edge-level task: predict property (or presence) of edges in a graph

Input:
graph with
unlabeled edges

Output:
labels for edges

OAK RIDGE National Laboratory

https://distill.pub/2021/gnn-intro/

Solving graph tasks with neural networks

Represent graphs to be compatible with neural networks.
(ML models typically take grid-like arrays as input)

Graphs have up to four types of information useful for predictions:

- nodes
- edges
- global-context
- connectivity

} Relatively simple

Connectivity:

Option 1: *Adjacency matrix*

- Costly: graph may have too many nodes (~1M)
- Space-inefficient: sparse representation
- Not permutation invariant

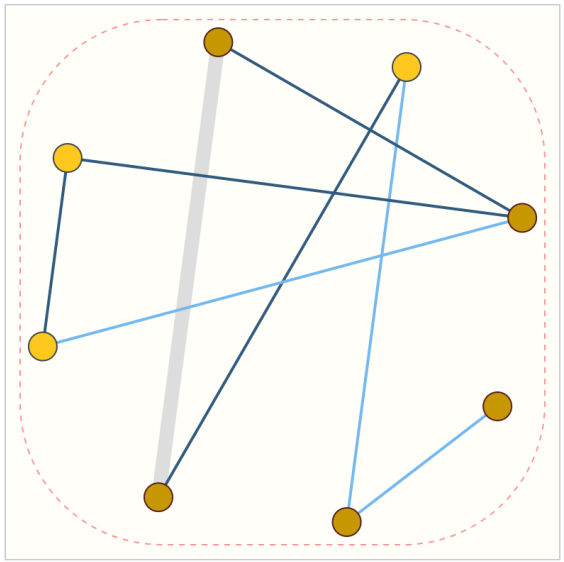
Option 2: *Adjacency list*

- Memory-efficient
- permutation invariant

https://distill.pub/2021/gnn-intro/

Solving graph tasks with neural networks

Adjacency list example (scalar values for node, edge, and global properties)



Nodes

```
[1, 1, 1, 0, 0, 1, 0, 1]
```

→ Node properties: 0 or 1

Edges

```
[2, 1, 2, 1, 2, 1, 1]
```

→ Edge properties: 1 or 2

Adjacency List

```
[[1, 0], [4, 3], [6, 1], [6, 2],
 [7, 3], [7, 4], [7, 5]]
```

→ **Connectivity**

Global

```
0
```

→ Global properties: 0 or 1

https://distill.pub/2021/gnn-intro/

Graph neural networks (GNNs)

A GNN is an **optimizable transformation** on all attributes of a graph (nodes, edges, global-context) that **preserves graph symmetries** (permutation invariances).

GNNs use a **"graph-in, graph-out" architecture:** they accept a graph as input, with information loaded into its nodes, edges, and global-context, and **transform** these embeddings/features **without changing the connectivity** of the input graph.

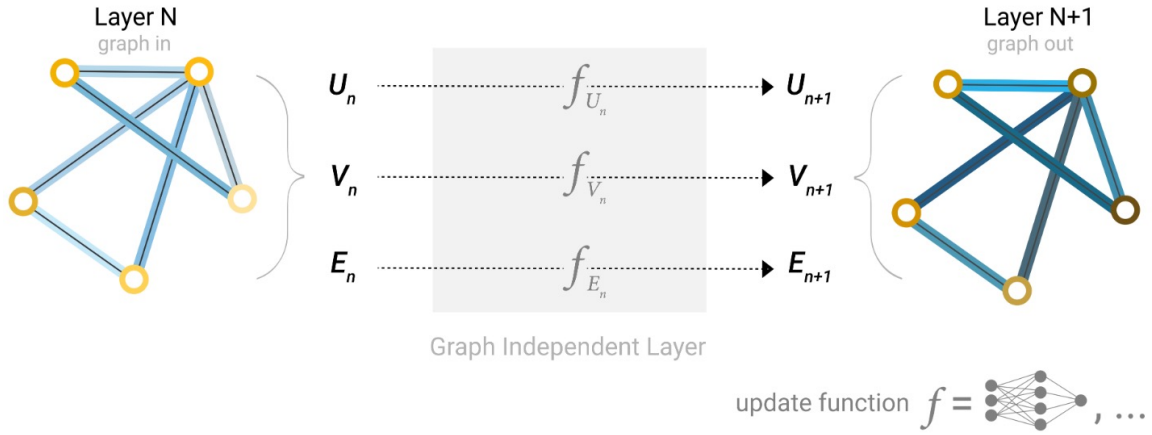
https://distill.pub/2021/gnn-intro/

The simplest GNN

GNN with a **separate multilayer perceptron** (MLP) **on each component** of a graph.

We apply the MLP to each node vector, edge, and global-context vector and get a learned node-vector, per-edge embedding/feature, and single embedding/feature for the entire graph.

A single layer of a simple GNN

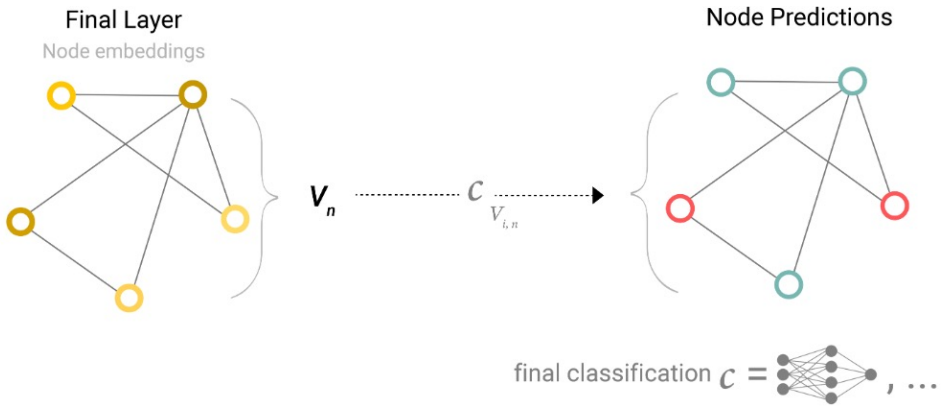


https://distill.pub/2021/gnn-intro/

GNN predictions by pooling information

Example: binary classification

If task is making binary predictions on nodes and graph already contains node information: approach is straightforward (linear classifier)



What if information is only stored in edges but we need node predictions?

Need to collect edge information and give it to nodes for prediction: ***Pooling***.

OAK RIDGE National Laboratory

https://distill.pub/2021/gnn-intro/

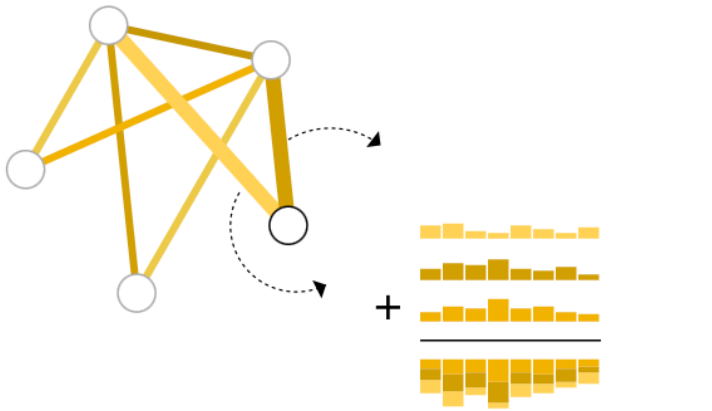
GNN predictions by pooling information

Example: binary classification

Pooling steps:

1. For each item to be pooled, gather each of their embeddings and concatenate them into a matrix.

2. Gathered embeddings are then aggregated, usually via a sum operation.



Aggregate information
from adjacent edges

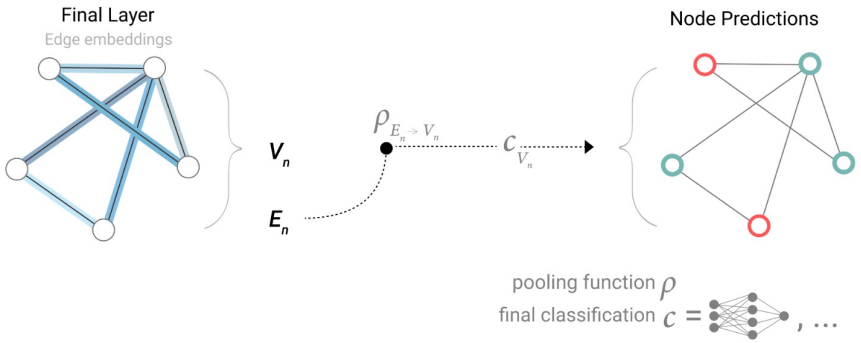
$p_{E_n \to V_n}$: gathering information **from edges to nodes**

https://distill.pub/2021/gnn-intro/

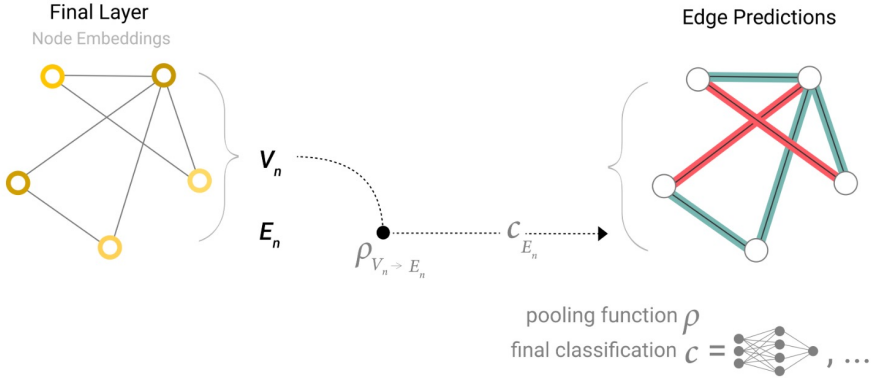
GNN predictions by pooling information

Example: binary classification

Only **edge-level features**:
predict binary **node-level information**



Only **node-level features**:
predict binary **edge-level information**



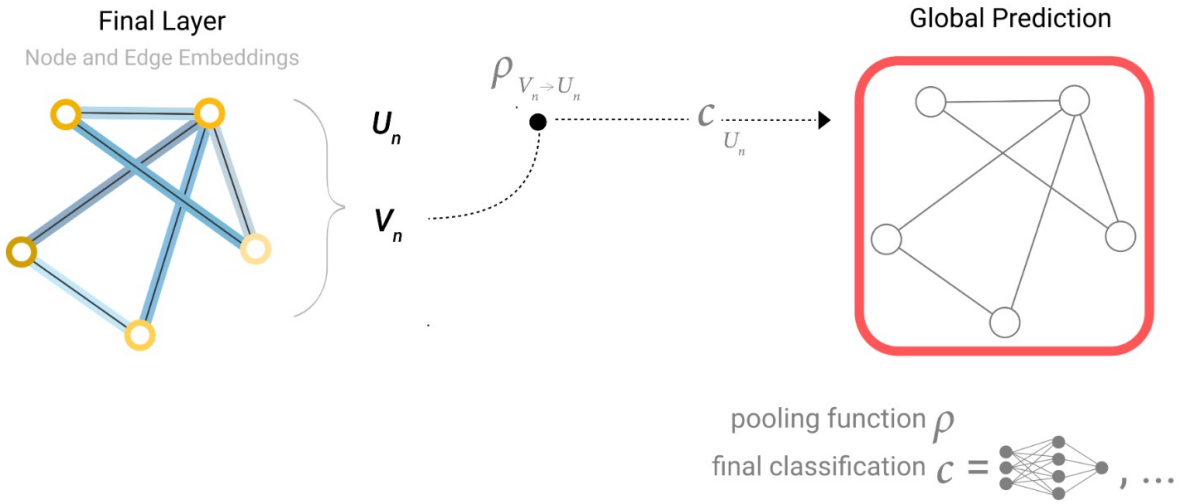
OAK RIDGE National Laboratory

https://distill.pub/2021/gnn-intro/

GNN predictions by pooling information

Example: binary classification

node-level features:
predict binary **global information**



This is a common scenario for **predicting molecular properties**: Given atomic information, predict toxicity of a molecule (toxic/not toxic) or if it has a particular odor (rose/not rose).

https://distill.pub/2021/gnn-intro/

GNN predictions by pooling information

In this simplest GNN, **graph connectivity is not used inside the GNN layer:** nodes, edges, and global context are processed independently.

Connectivity is only **used when pooling** information for prediction.

More sophisticated predictions can be made by **using pooling within the GNN layer** to make learned embeddings aware of graph connectivity.

https://distill.pub/2021/gnn-intro/

Passing messages between parts of the graph

Message passing:
neighboring nodes/edges exchange information and influence each other's updated embeddings.

Steps:

1. For each node, gather all the neighboring node embeddings (messages)

2. Aggregate all messages via an aggregate function (like sum)

3. All pooled messages are passed through an update function, usually a learned neural network

Message passing can occur between either nodes or edges.

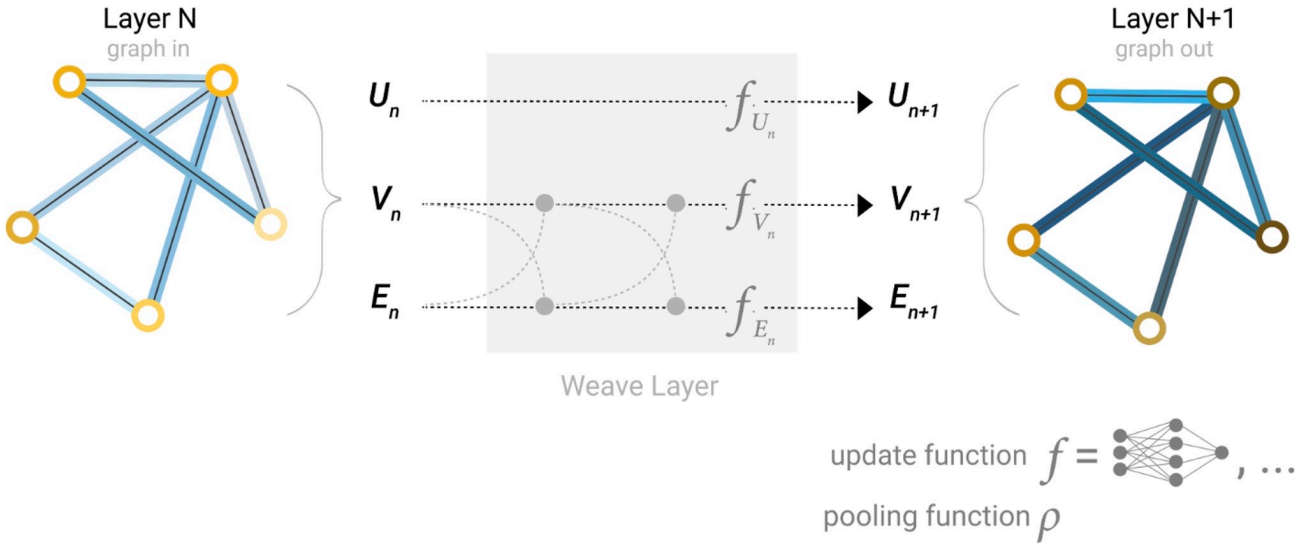
Different **message passing** can yield GNN models of **increasing expressiveness** and **power**.

https://distill.pub/2021/gnn-intro/

J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, Neural message passing for quantum chemistry, Proceedings of the 34th International Conference on Machine Learning, Vol 70, pp. 1263-1272. PMLR. 2017.

Passing messages between parts of the graph

Which graph attributes are updated and in **which order** is a **design decision** when constructing GNNs.



https://distill.pub/2021/gnn-intro/

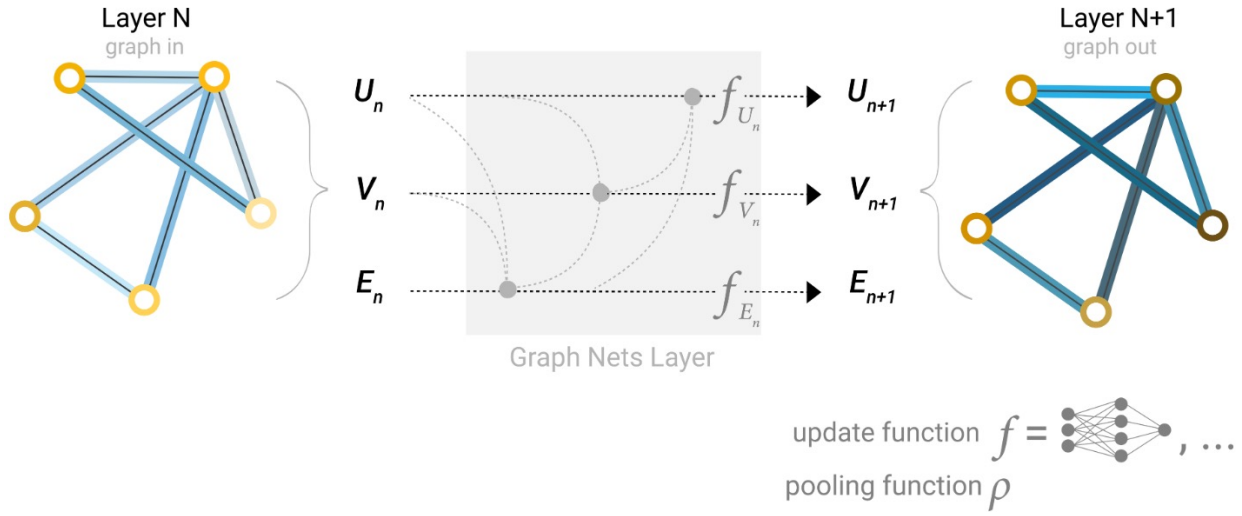
Passing messages between parts of the graph

Adding global representations: nodes far away from each other may never be able to efficiently transfer information to one another, even if we apply message passing several times.

Solution 1: all nodes are able to pass information to each other '**virtual edges**'

- Used for small graphs such as molecules
- Computationally expensive for large graphs

Solution 2: use global representation of graph (U): **master node**



https://distill.pub/2021/gnn-intro/

Aggregation operations

Pooling information from neighboring nodes and edges is a **critical step** in any reasonably powerful GNN architecture.

Simple candidates for aggregation operation:

1. **Sum** : provides balance between mean and max (commonly used)
2. **Mean** : useful when nodes have highly-variable number of neighbors or one needs normalized view of features of local neighborhood
3. **Max** : useful to highlight single salient features in local neighborhoods

No operation is uniformly the best choice!

Message passing neural networks (MPNNs)

Mathematical formulation:

G: graph x_v: node feature e_{vw}: edge feature

Message passing phase: hidden states at each node in graph updated based on messages

$$m_v^{t+1} = \sum_{w \in N(v)} M_t\left(h_v^t, h_w^t, e_{vw}\right)$$

$$h_v^{t+1} = U_t\left(h_v^t, m_v^{t+1}\right)$$

← message function

← vertex update function

h_v^t: hidden states

e_{vw}: edge features

m_v^{t+1}: messages

M_t: message functions (differentiable)

$N(v)$: neighbors of v

U_t: vertex update functions (differentiable)

Function differentiability is important for automatic differentiation.

J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, Neural message passing for quantum chemistry, Proceedings of the 34th International Conference on Machine Learning, Vol 70, pp. 1263-1272. PMLR. 2017
https://link.springer.com/chapter/10.1007/978-3-030-40245-7_10

Message passing neural networks (MPNNs)

Mathematical formulation:

G: graph x_v: node feature e_{vw}: edge feature

Readout phase: computes feature vector for whole graph using some readout function

$$\hat{y} = R(\{h_v^T \mid v \in G\})$$

R : readout function (differentiable), invariant to permutations of node states

J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, Neural message passing for quantum chemistry, Proceedings of the 34th International Conference on Machine Learning, Vol 70, pp. 1263-1272. PMLR. 2017
https://link.springer.com/chapter/10.1007/978-3-030-40245-7_10

Message passing neural networks (MPNNs)

Example: Convolutional networks for learning molecular fingerprints

$M(h_v, h_w, e_{vw}) = (h_w, e_{vw})$: concatenation ← message function

$U_t(h_v^t, m_v^{t+1}) = \sigma\left(H_t^{\deg(v)} m_v^{t+1}\right)$ ← vertex update function

σ: sigmoid function

$\deg(v)$: degree of vertex v

H_t^N: learned matrix for each time step *t* and vertex degree *N*

Message passing phase:

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

$R = f\left(\sum_{v,t} \text{softmax}(W_t h_v^t)\right)$: *f* is a neural network

W_t: learned readout matrices (one for each time step *t*)

Readout phase:

$$\hat{y} = R(\{h_v^T \mid v \in G\})$$

https://arxiv.org/pdf/1704.01212.pdf
D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, R. P. Adams, Convolutional networks on graphs for learning molecular fingerprints. In Advances in neural information processing systems, pp. 2224–2232, 2015.

Message passing neural networks (MPNNs)

Example: Deep tensor neural networks

$$M_t = \tanh\left(W^{fc}((W^{cf} h_w^t + b_1) \odot (W^{df} e_{vw} + b_2))\right)$$

$$U_t(h_v^t, m_v^{t+1}) = h_v^t + m_v^{t+1}$$

message function

vertex update function

W^{fc}, W^{cf}, W^{df}: matrices

b_1, b_2: biases

$\odot$: elementwise multiplication

Message passing phase:

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

$$R = \sum_v \mathrm{NN}(h_v^T)$$

NN is a single hidden layer neural network.

Readout phase:

$$\hat{y} = R(\{h_v^T \mid v \in G\})$$

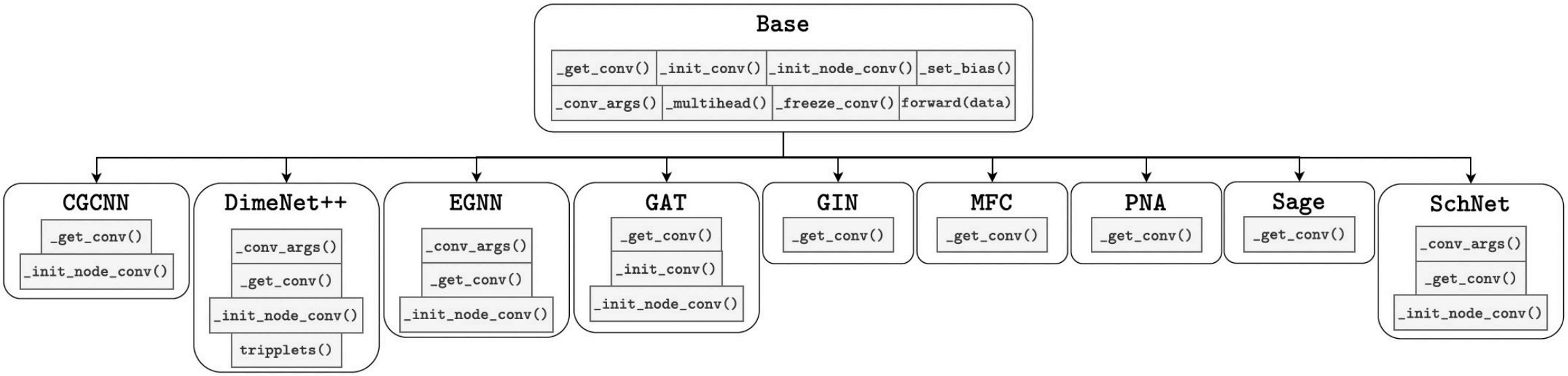
https://arxiv.org/pdf/1704.01212.pdf

K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, A. Tkatchenko, Quantumchemical insights from deep tensor neural networks, *Nature Communications* 8 2017.

Message passing in HydraGNN

Different choices of **message function** & **vertex update function** give different **message passing policies**.

Message passing policies in HydraGNN



New models can be implemented by the user.

M. L. Pasini, J. Y. Choi, P. Zhang, J. Baker, User Manual - HydraGNN: Distributed PyTorch Implementation of Multi-Headed Graph Convolutional Neural Networks, Tech, Rep. ORNL/TM-2023/3203, November 2023.

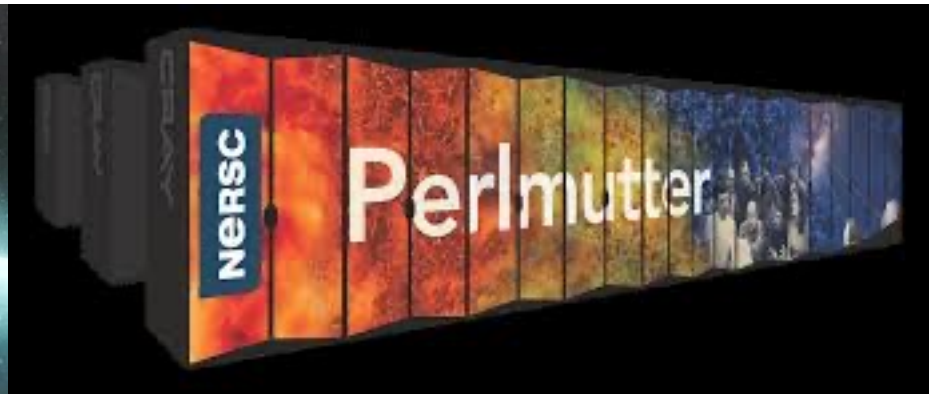
OAK RIDGE National Laboratory

Docker

Jong Youl Choi (CSMD)

HydraGNN environment

- Performant packages
  - Python
  - PyTorch with CUDA/ROCM
  - **PyG**
  - **DDStore**
  - **mpi4py**
  - **Adios**



HydraGNN Docker Image

- Docker is a virtualization tool
- First, need to install Docker desktop locally: https://docs.docker.com/desktop/
- Download our HydraGNN docker image
- HydraGNN installed inside the docker image: /workspace

Quick getting started commands

```
$ docker pull
ghcr.io/ornl/hydragnn-tutorial
$ docker run -it --rm -v
"$HOME/project:/mnt"
ghcr.io/ornl/hydragnn-tutorial


docker$ cd /HydraGNN
docker$ python examples/qm9/qm9.py
```

PyTorch

PyG

Docker run

```
$ docker run -it --rm -v "$HOME/project:/mnt" ghcr.io/ornl/hydragnn-tutorial
```

- Command line options:
  - -it: interactive mode
  - --rm: delete the working container when exit. No data will be saved.
  - -v "$HOME/project:/mnt": mount your working directory ($HOME/project) into "/mnt" on the Docker container.

Related information

- Install and run instructions:
  - https://github.com/ORNL/HydraGNN/wiki
- ORNL github
  - https://github.com/ORNL/HydraGNN
  - https://github.com/orgs/ORNL/packages

OAK RIDGE National Laboratory

Example Demonstrations

Pei Zhang (CSED)

Massimiliano Lupo Pasini (CSED)

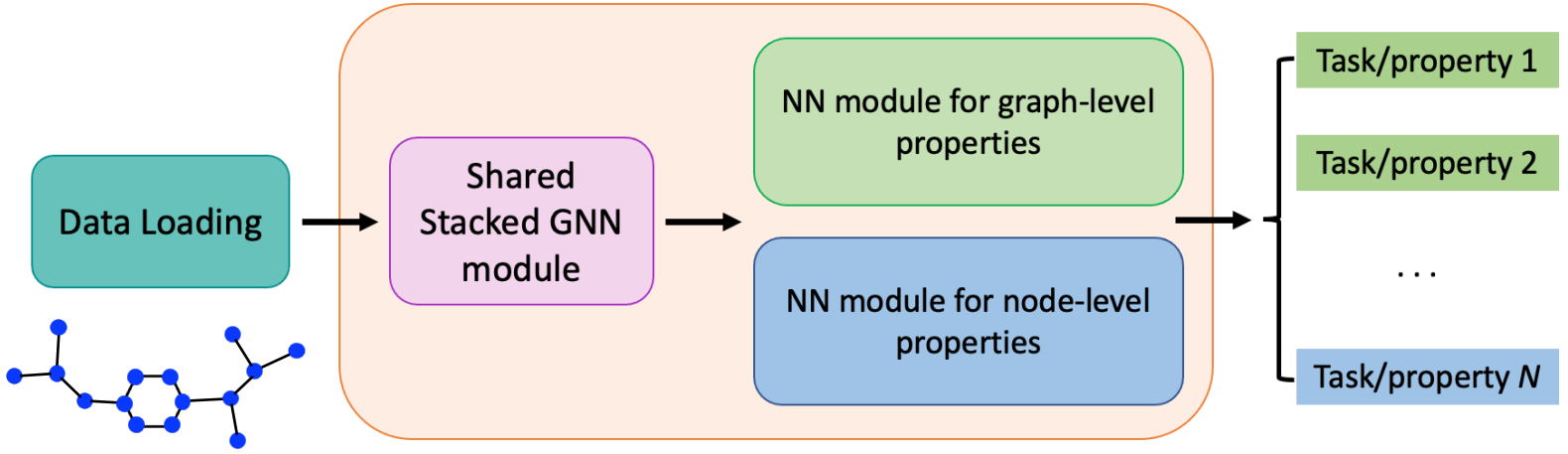
Two small-scale demonstrations

https://github.com/ORNL/HydraGNN/tree/LoG2023_tutorial

- Prerequisites
  - Setting up virtual environment
    - Activate your virtual environment
  - Downloading code (https://github.com/ORNL/HydraGNN)
    - "*git clone* -b LoG2023_tutorial *https://github.com/ORNL/HydraGNN*"
- QM9 (*inherent from pytorch geometric [**PyG**]*)
  - Single tasking for a graph-level property
  - Load a pre-trained model for inference
    - Multitasking: predicting multiple properties simultaneously
- LSMS-3
  - Users building graph objects from data files

User set up HydraGNN case via a configuration json file

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

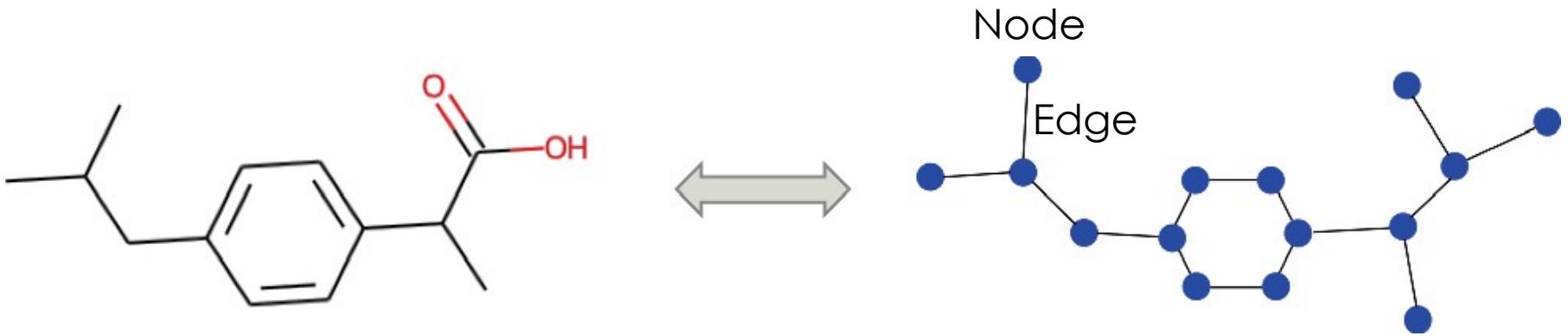


HydraGNN

QM9 dataset

QM9 (Ramakrishnan et al., 2014)

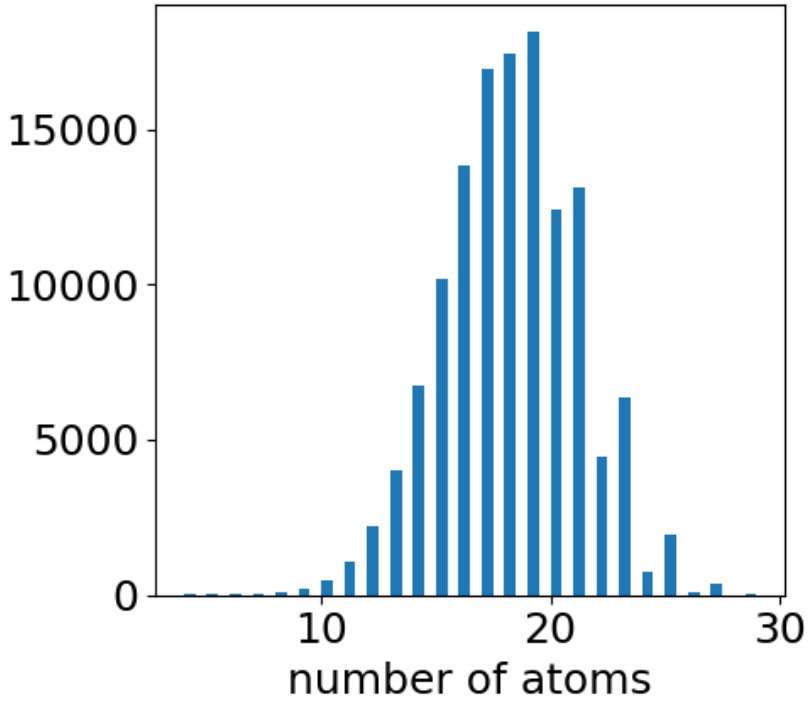
- 130k molecules with up to nine heavy atoms
- Molecule ←→ Graph representation
  - Atom→ Node: ***Graph.x***
    - e.g., atom type, atomic number, aromatic, hybridization types)
  - Chemical bond → Edge: ***Graph.edge_attr***
    - e.g., type of the covalent bond (e.g., single, double, triple, or aromatic)
  - Molecular properties → regression targets/outputs of HydraGNN: **Graph.y**
    - e.g., free energy, partial charge



Ibuprofen

QM9 (Ramakrishnan et al., 2014)

- **torch_geometric.datasets.QM9**
  - 19 properties (**graph-level**)
    - geometric, energetic, electronic, and thermodynamic properties
- 20 properties → regression targets
  - 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 capavity at 298.15K	$\frac{\text{cal}}{\text{mol K}}$
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

QM9

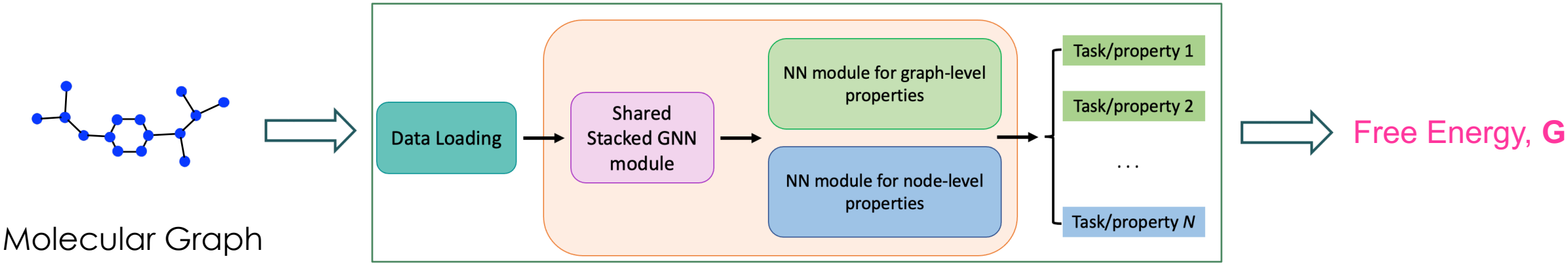
1. **Training**: single tasking for a graph-level property
   - Free Energy, **G**
2. Load a pre-trained model for **inference**
   - Trained on all 20 properties (both graph-level and node-level)

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	$\frac{\text{cal}}{\text{mol K}}$
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

1. **Training**: single-tasking on free energy, G

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

- Files required: **qm9.py** and **qm9.json**
- *"python examples/qm9/qm9.py"*



1. **Training**: single-tasking on free energy, G

- Loading **data** from PyG
  - 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"]
)
```

1. **Training**: 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,
)
```

1. **Training**: single-tasking on free energy, G

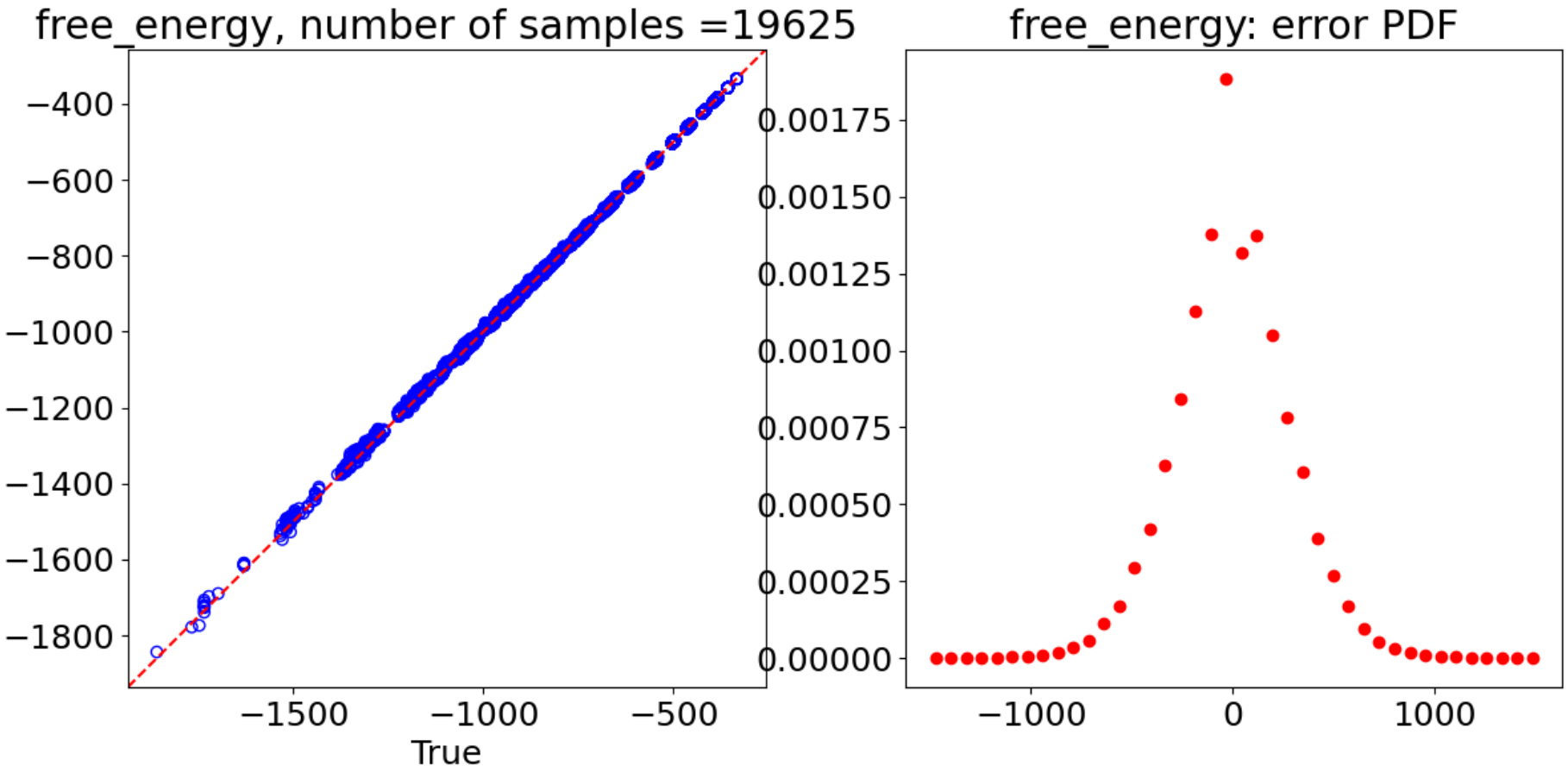
- Create model with **config** from qm9.json
- Set up optimizer
- Train the model
  - hydragnn.train.train_validate_test

```
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"],
)
```

1. **Training**: single-tasking on free energy, G



Results of test set

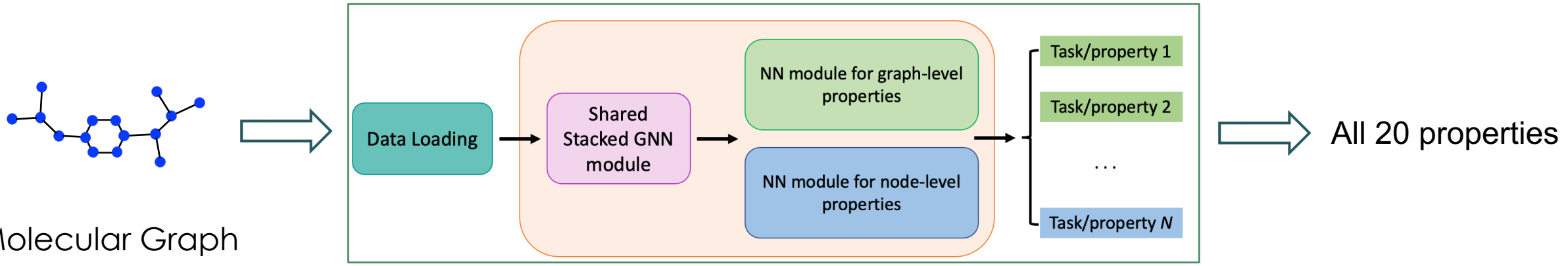
"python examples/qm9/qm9.py"

2. **Inference** of a pretrained model for multitasking on all 20 (=19+1) properties

- Files required
  - **qm9_custom20_inference.py**
  - Values used for normalization in training
    - **train_minmax_output.pt**
  - Graph structure: **data.pt**
  - Pre-trained model
    - ./logs/qm9_all20/**qm9_all20.pk**
    - config.json

- "*python examples/qm9/qm9_custom20_inference.py --pretraineddir=./logs/ --pretrainedmodelname=qm9_all20 --graphfile=data.pt*"

2. **Inference** of a pretrained model for multitasking on all 20 (=19+1) properties



Run: *python examples/qm9/qm9_custom20_inference.py --pretraineddir=./logs/ --pretrainedmodelname=qm9_all20 --graphfile=data.pt*

2. **Inference** of a pretrained model for multitasking on all 20 (=19+1) properties

- Create model from saved config
- Load saved model parameters
- Run inference

```
filename = os.path.join(modeldir,model_name, "config.json")
with open(filename, "r") as f:
    config = json.load(f)
```

```
#load pretrained model
model = hydragnn.models.create_model_config(
    config=config["NeuralNetwork"],
    verbosity=verbosity,
)
model = hydragnn.utils.get_distributed_model(model, verbosity)
hydragnn.utils.model.load_existing_model(model, model_name, path=modeldir)
model.eval()
```

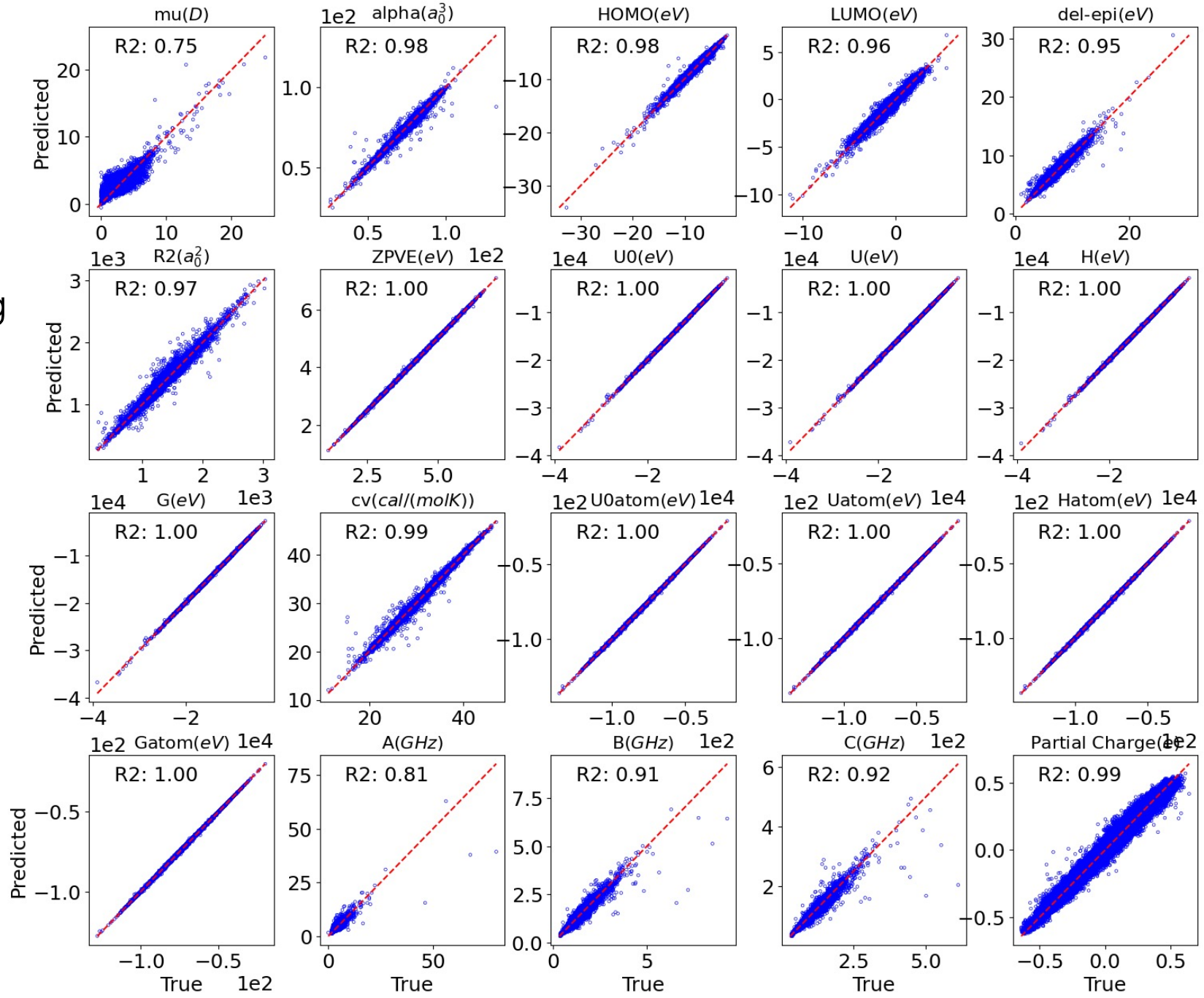
```
if os.path.exists(graph_file):
    graphdata = torch.load(graph_file)
```

```
print(graphdata)
pred = model(graphdata)
ytrue = graphdata.y
pred = model(graphdata)
print("Head, Normalized properties, True  VS   Pred")
for ival in range(len(ytrue)):
    if ival<19:
        ihead = ival
        print("%4d, %21s, %.3f VS  %.3f"%(ival, var_config["output_names"][ihead], pred[ival].item(), ytrue[ival].item()))
    else:
        ihead = 19
        print("%4d, %21s, atom %2d,  %.3f VS %.3f"%(ihead, var_config["output_names"][ihead], ival-ihead, pred[ihead][ival-ihead].item(), ytrue[ival].item()))
```

2. **Inference** of a pretrained model for multitasking on all 20 (=19+1) properties

```
Data(x=[18, 11], edge_index=[2, 38], edge_attr=[38, 4], y=[37, 1], pos=[18, 3], z=[18], smiles='[H]C1=C(N([H])[H])[C@]2([H])C([H])([H])[C@]([H])(C1=O)C2([H])[H]', name='gdb_74100', idx=[1],
 y_loc=[1, 21])
Head, Normalized properties, True  VS   Pred
  0,                    mu, 0.170 VS  0.196
  1,                 alpha, 0.350 VS  0.358
  2,                  HOMO, 0.920 VS  0.920
  3,                  LUMO, 0.381 VS  0.389
  4,               del-epi, 0.079 VS  0.080
  5,                    R2, 0.269 VS  0.266
  6,                  ZPVE, 0.815 VS  0.814
  7,                    U0, 0.852 VS  0.850
  8,                     U, 0.852 VS  0.850
  9,                     H, 0.853 VS  0.850
 10,                     G, 0.853 VS  0.850
 11,                    cv, 0.538 VS  0.543
 12,                U0atom, 0.636 VS  0.641
 13,                 Uatom, 0.635 VS  0.638
 14,                 Hatom, 0.632 VS  0.637
 15,                 Gatom, 0.653 VS  0.659
 16,                     A, 0.035 VS  0.032
 17,                     B, 0.153 VS  0.142
 18,                     C, 0.113 VS  0.113
 19,         chargedensity, atom  0,  0.091 VS 0.079
 19,         chargedensity, atom  1,  0.753 VS 0.784
 19,         chargedensity, atom  2,  0.269 VS 0.271
 19,         chargedensity, atom  3,  0.757 VS 0.775
 19,         chargedensity, atom  4,  0.269 VS 0.230
 19,         chargedensity, atom  5,  0.376 VS 0.412
 19,         chargedensity, atom  6,  0.349 VS 0.333
 19,         chargedensity, atom  7,  0.397 VS 0.417
 19,         chargedensity, atom  8,  0.349 VS 0.332
 19,         chargedensity, atom  9,  0.667 VS 0.673
 19,         chargedensity, atom 10,  0.667 VS 0.675
 19,         chargedensity, atom 11,  0.569 VS 0.548
 19,         chargedensity, atom 12,  0.558 VS 0.547
 19,         chargedensity, atom 13,  0.571 VS 0.571
 19,         chargedensity, atom 14,  0.571 VS 0.564
 19,         chargedensity, atom 15,  0.545 VS 0.533
 19,         chargedensity, atom 16,  0.571 VS 0.564
 19,         chargedensity, atom 17,  0.571 VS 0.568
```

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



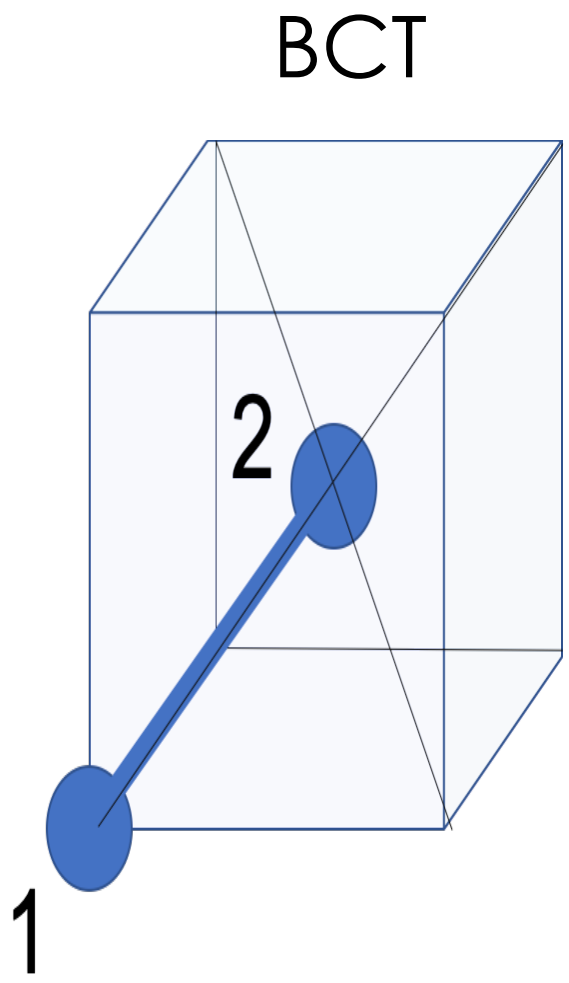
Iron-platinum (FePt) binary alloy with 32 atoms LSMS-3 data

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



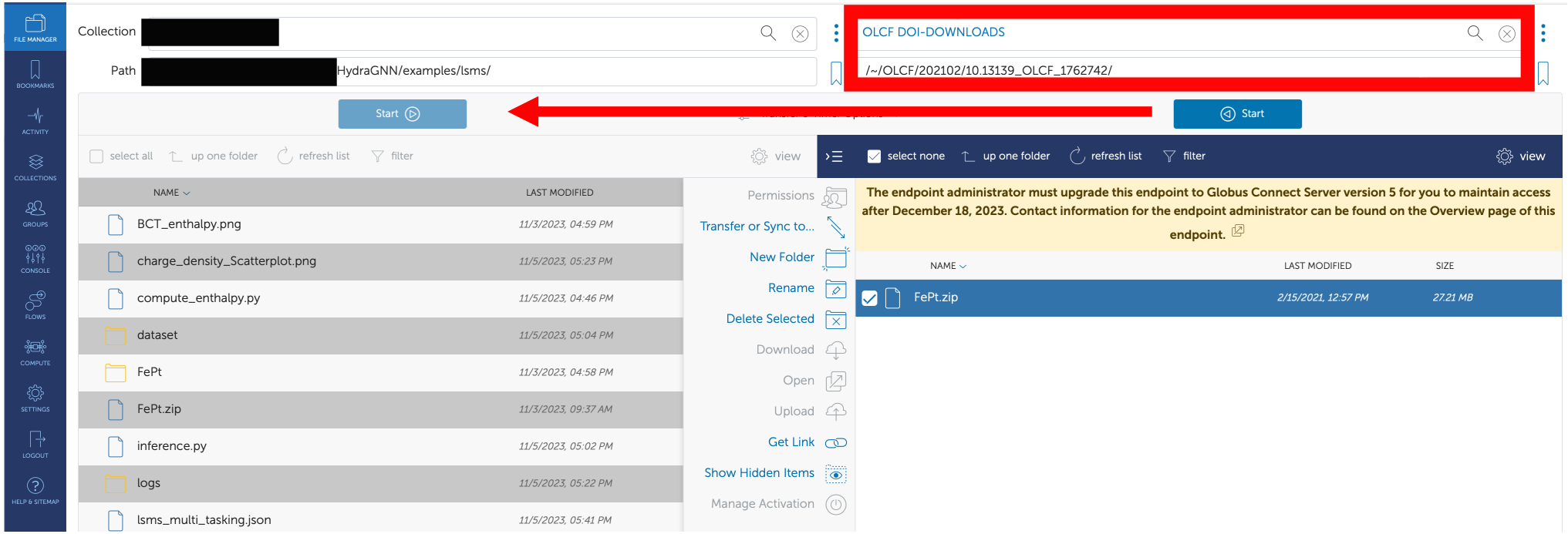
FePt binary alloy with 32 atoms - LSMS-3 data

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

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/



OAK RIDGE National Laboratory

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:**

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

If you have not installed hydragnn as a package, you need to perform these steps before executing the python scripts above:

- From the **HydraGNN** main directory: $ **export $PYTHONPATH=$PWD:PYTHONPATH**
- $ **mv examples/lsms**

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

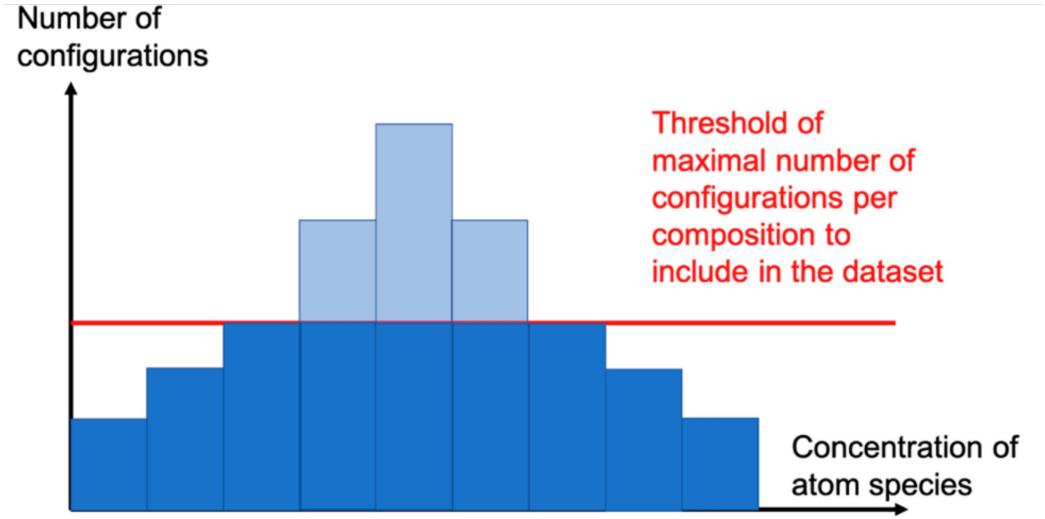
compute_enthalpy.py

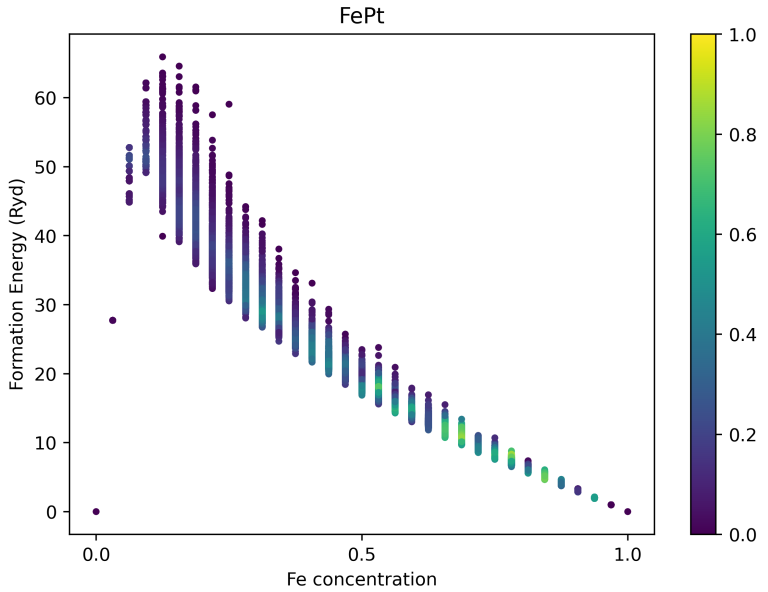
1. Performs histogram cutoff to ensure that the atomic configurations are balanced across all chemical compositions.

   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 mixing terms from the total energy of each DFT calculation





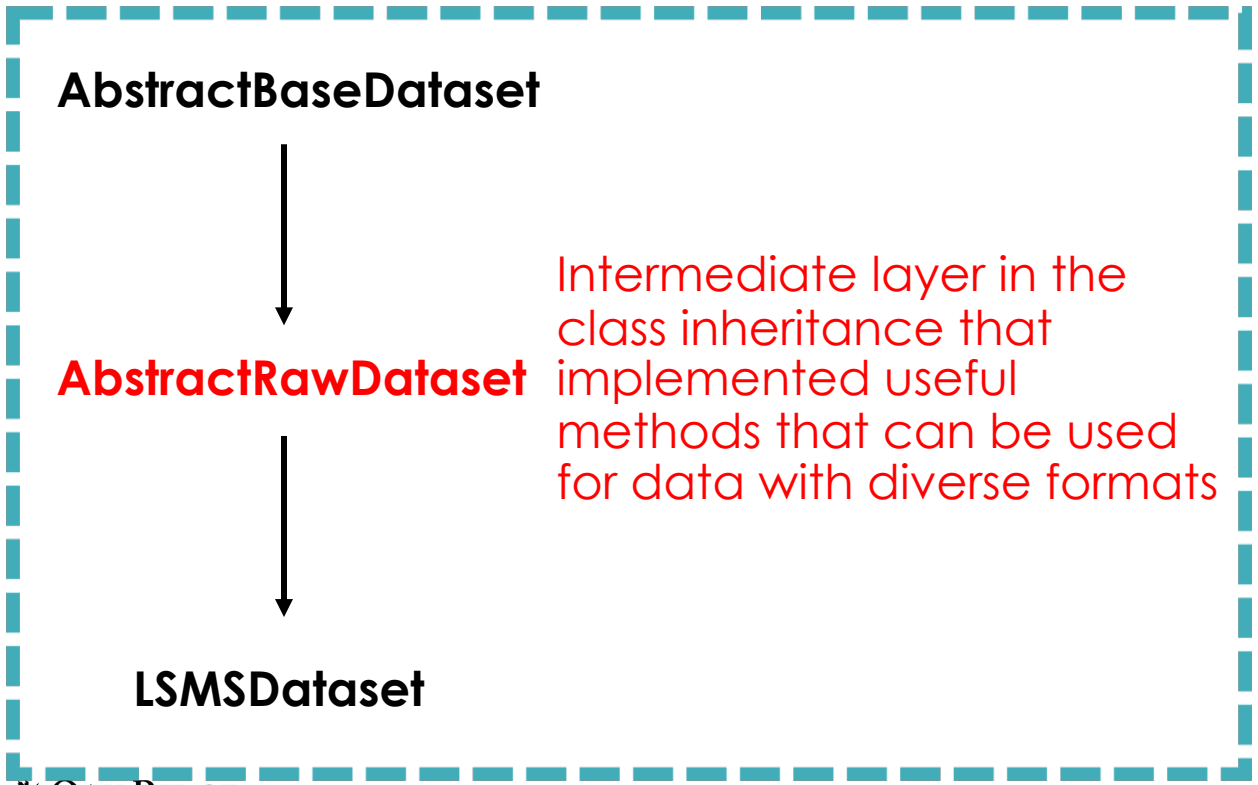
FePt binary alloy with 32 atoms - LSMS-3 data

$ python lsms.py --preonly

1. Dataset reading and pre-loading

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

Class inheritance for dataset classes



Remark:

Your customized dataset does not need to inherit from AbstractRawDataset.

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

$ mkdir dataset

$ mv FePt_enthalpy dataset

$ python lsms.py --preonly

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

$ python lsms.py --pickle

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
```

FePt binary alloy with 32 atoms - LSMS-3 data

lsms.py

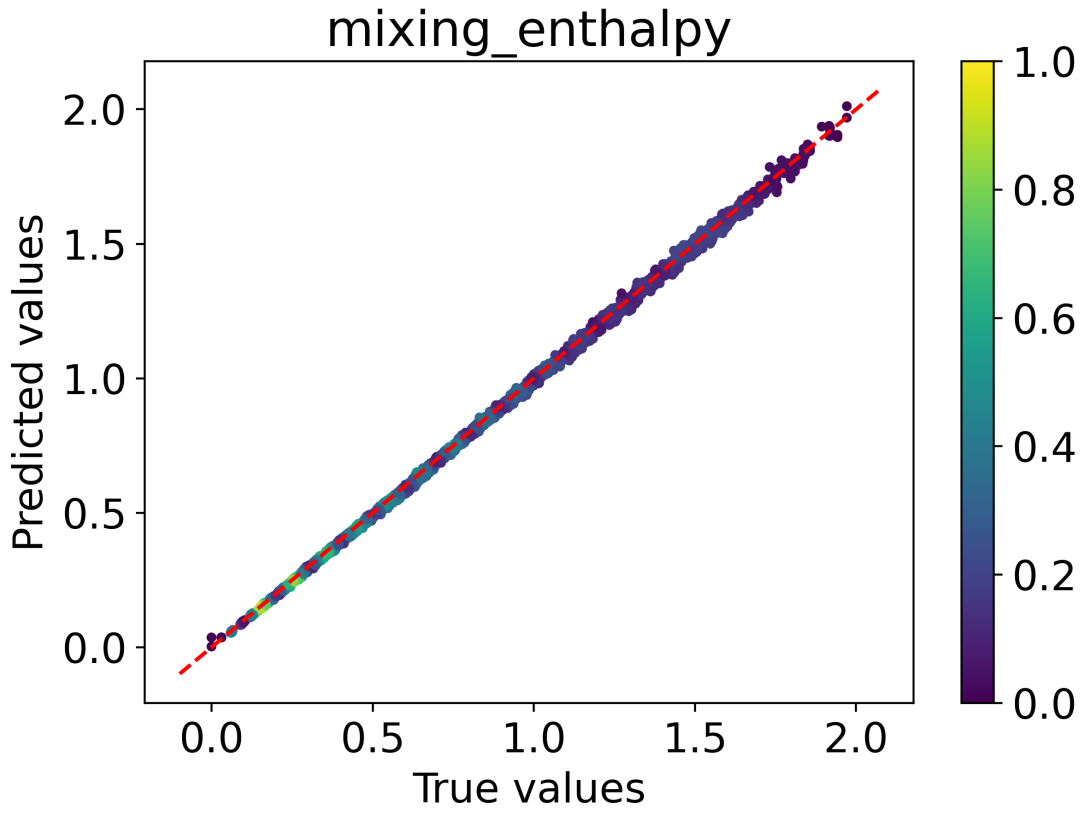
Single-task training for predictions of mixing enthalpy

inference.py

Load pre-trained model and run inference on testing data

Test MAE:

0.010 Rydberg

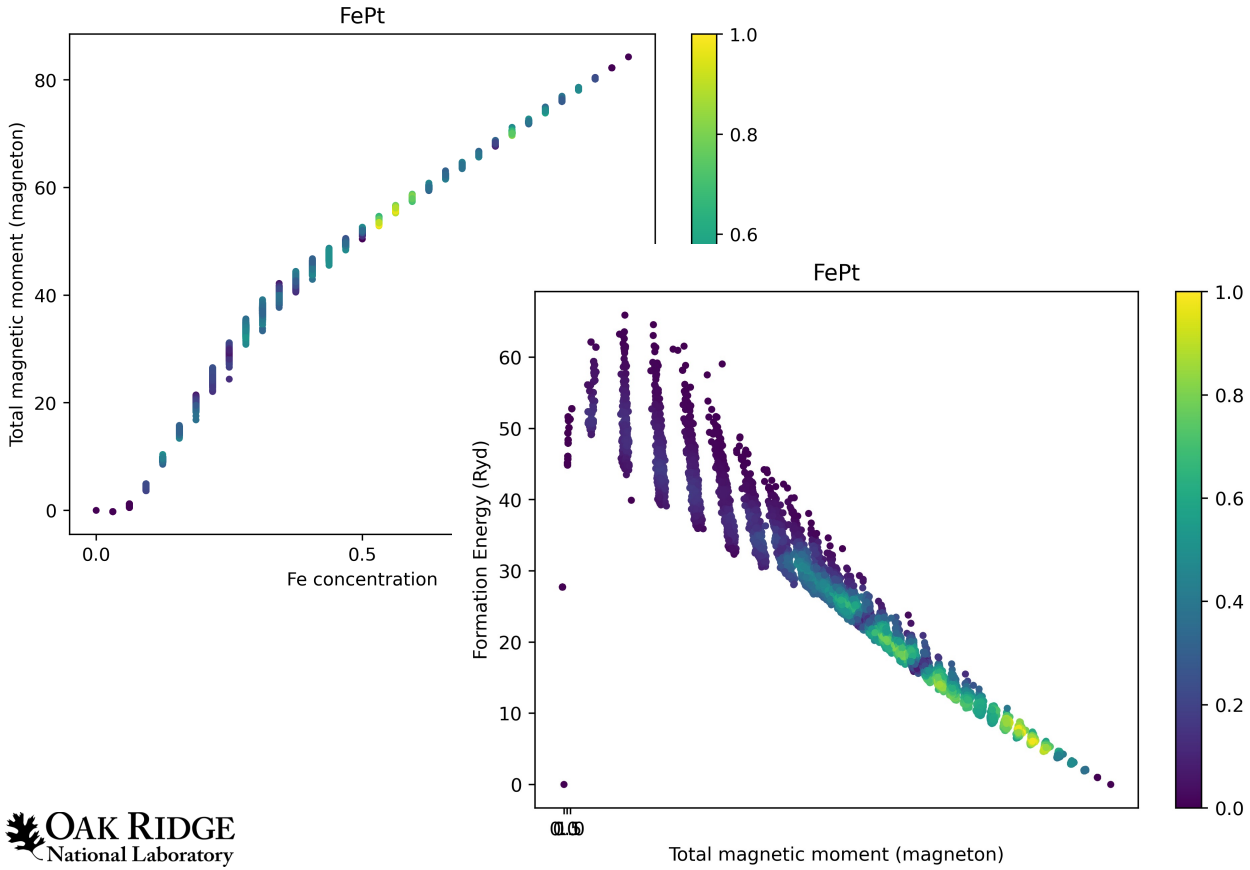



```
"NeuralNetwork": {
    "Architecture": {
        "model_type": "PNA",
        "radius": 7,
        "max_neighbours": 100,
        "periodic_boundary_conditions": false,
        "hidden_dim": 100,
        "num_conv_layers": 6,
        "output_heads": {
            "graph":{
                "num_sharedlayers": 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
    },
```


OAK RIDGE National Laboratory

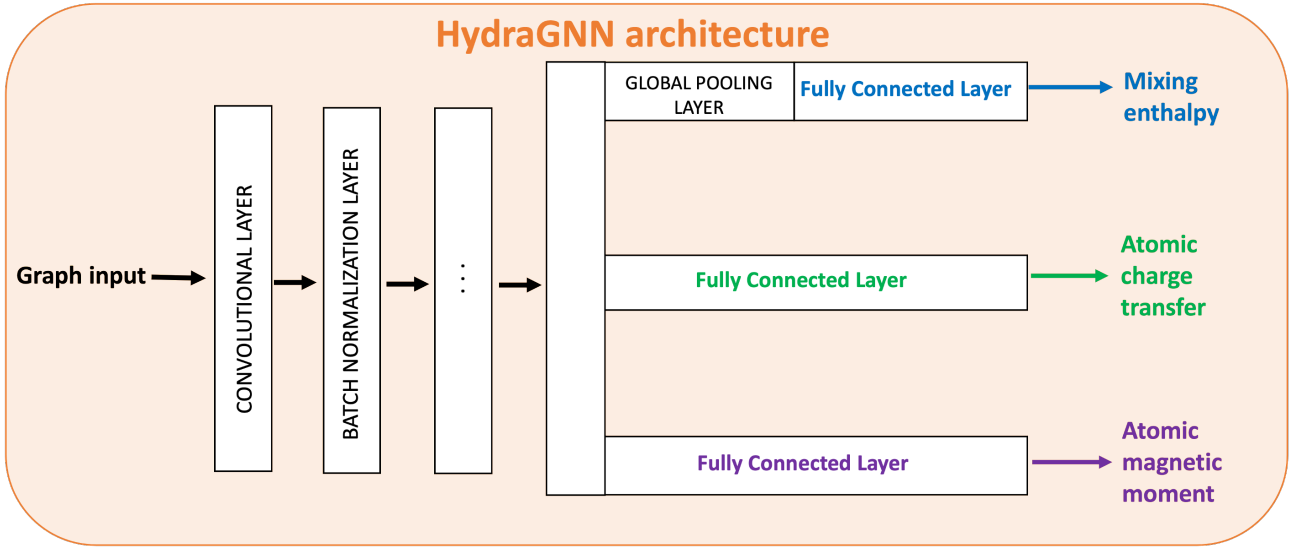
FePt binary alloy with 32 atoms - LSMS-3 data

lsms.py

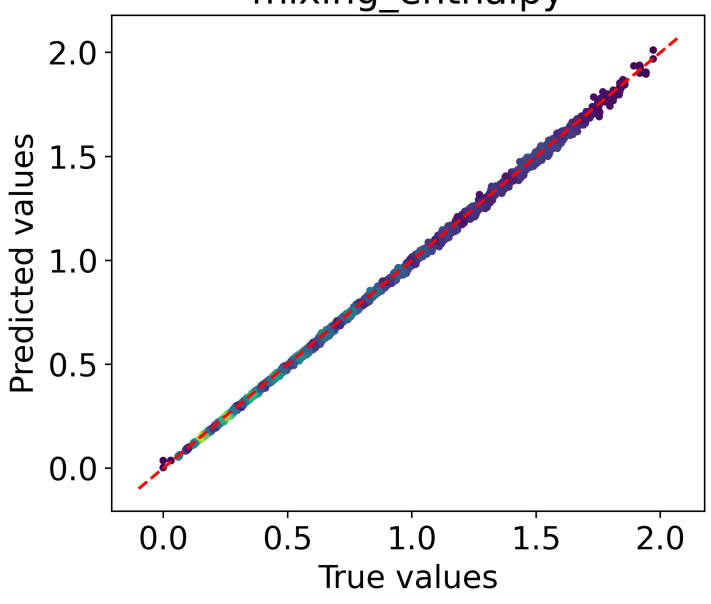
Multi-task training for predictions of mixing enthalpy, atomic charge density, and atomic magnetic moment

inference.py

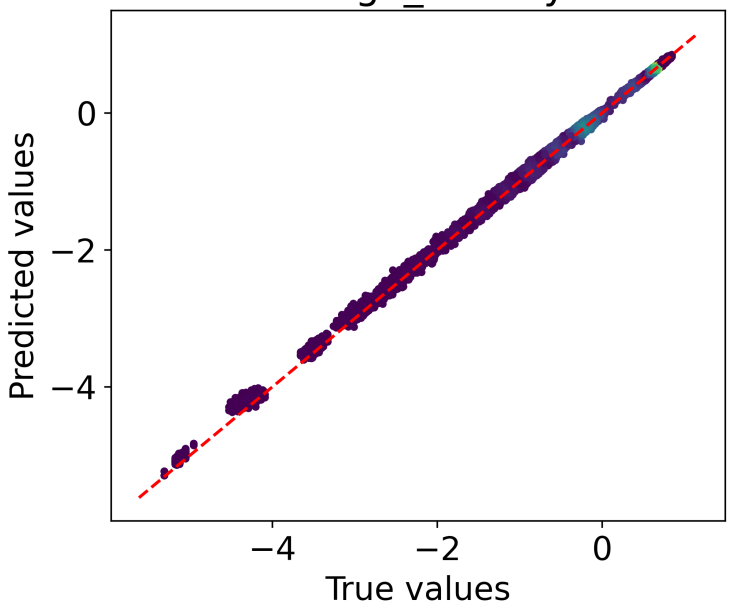
Load pre-trained model and run inference on testing data



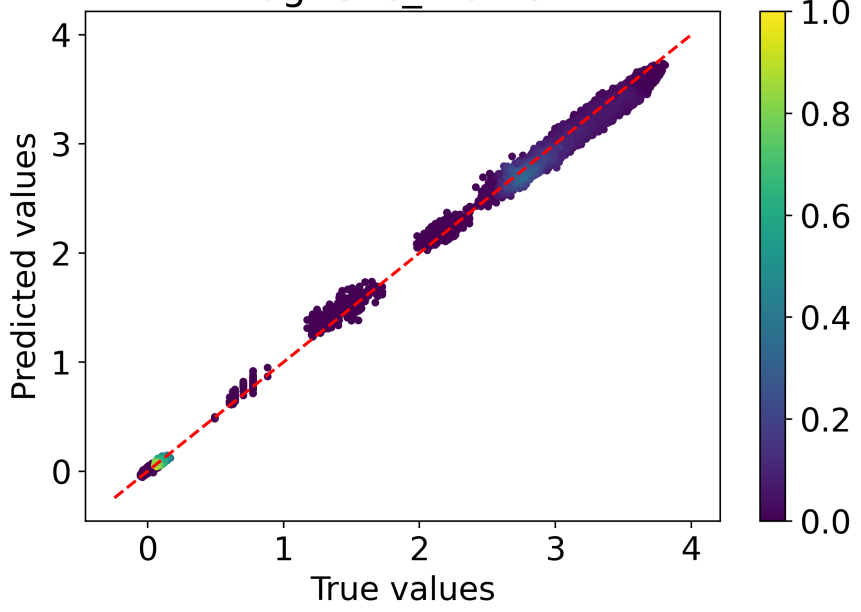
Test MAE: 0.010 Rydberg



Test MAE: 0.66 electron charges



Test MAE: 0.98 magnetons



OAK RIDGE National Laboratory

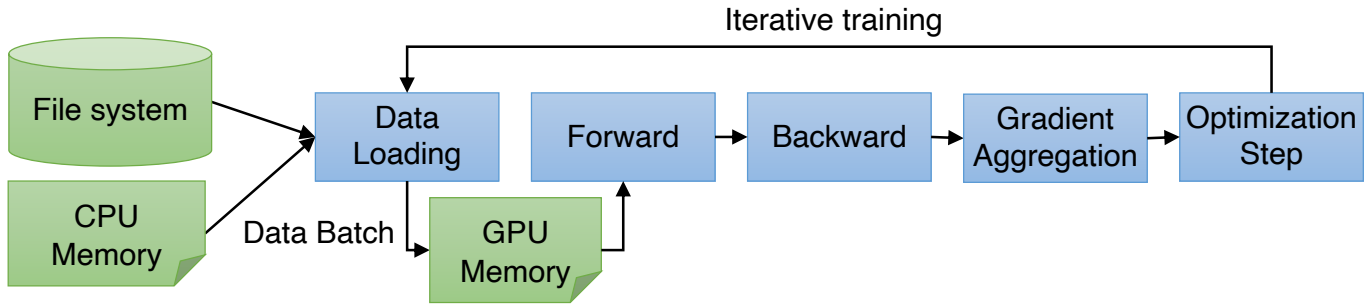
Scalable Data Management and Training on DOE Supercomputers

Jong Youl Choi (CSMD)

Kshitij Mehta (CSMD)

Data Challenges In Deep Learning

- Conventional HPC I/O
  - Write-oriented
  - Bulk
- DL I/O characteristics
  - **Read-oriented**
  - **Frequent** access
  - **Shuffled** access

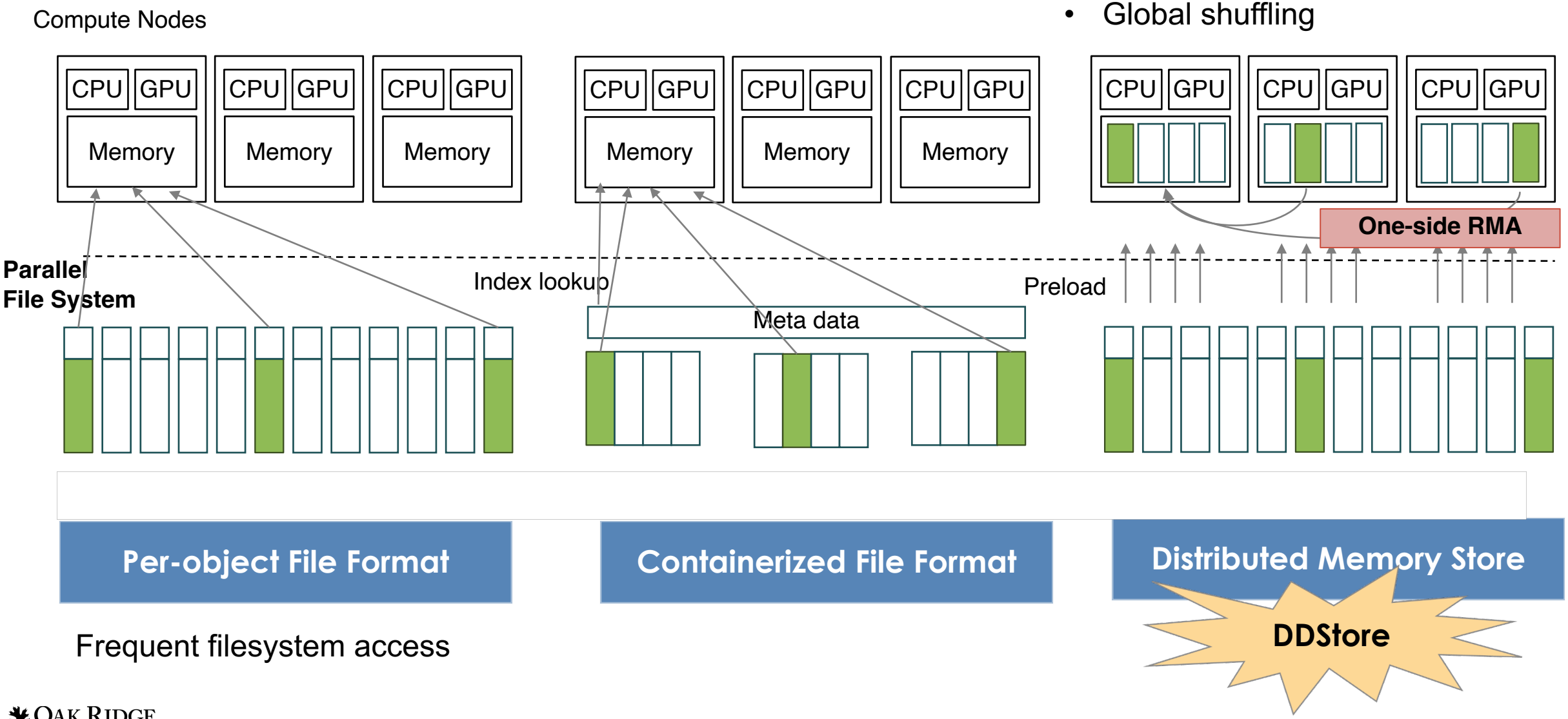


Iterative GNN training: Data loading is one of the key steps for performance

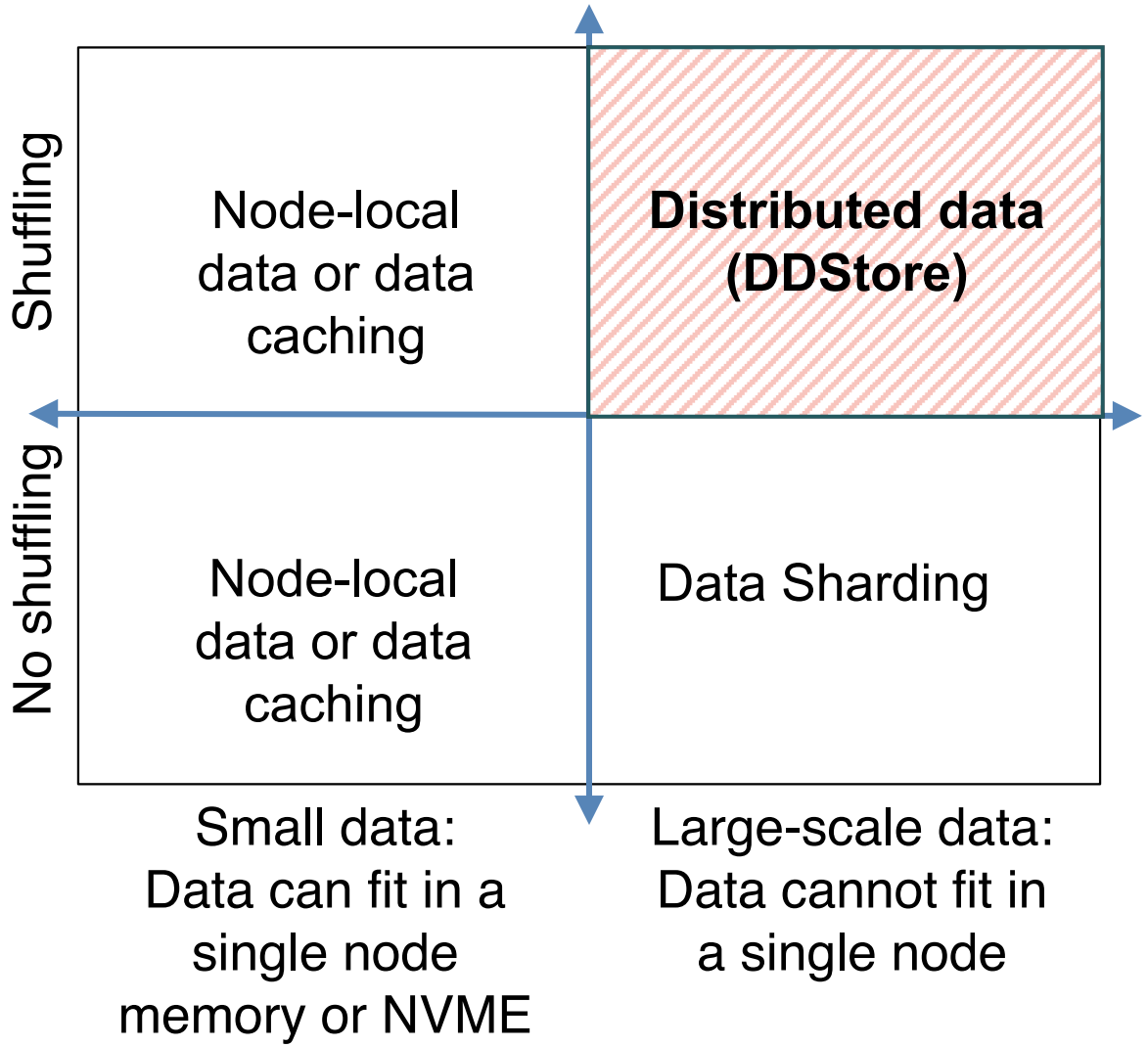
File per molecule	NVME/Node-local SSD	Sharding
• 10s of millions of files • Large metadata • Huge stress on filesystem • Multiple requests to increase space/inode quotas	• Non-negligible setup time • Total (N nodes x data size) byte transfer	• Flexibility issue • May limit the quality of training

DDStore: In-memory Data Store

- Load entire dataset in distributed memory
- Transparent data access through memory-to-memory
- Global shuffling



DDStore Mileage



DDStore Setup

1. Create DDStore object (`PyDDStore`)
2. Register your graph objects

DDStore object is `DataSet`.

Combine with `DataLoader` and `DistributedSampler`

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

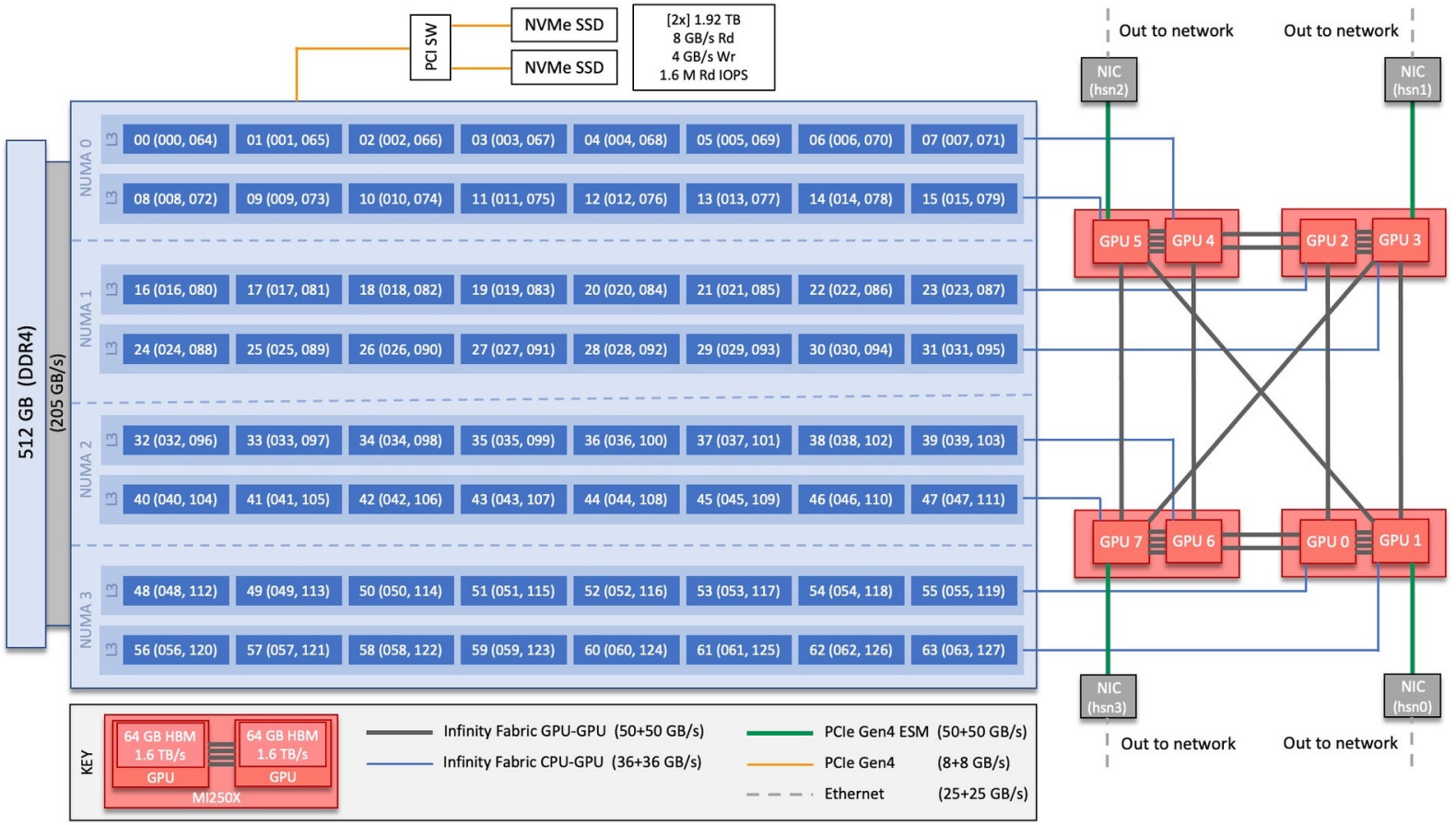
```python
class DistDataset(AbstractBaseDataset):
    """Distributed dataset class"""

    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,
```

Training

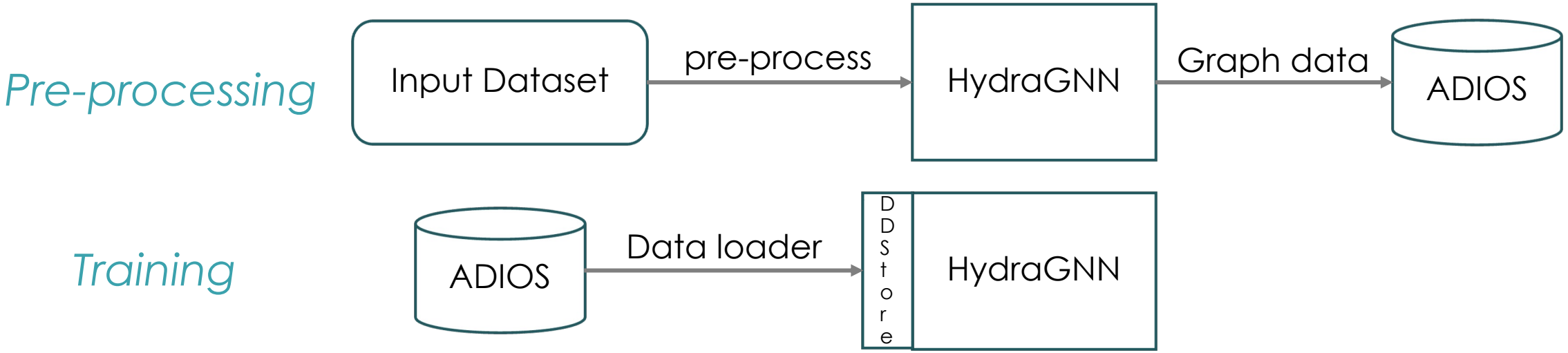
```
#!/bin/bash
#SBATCH -A LRN026
#SBATCH -J HydraGNN
#SBATCH -t 00:30:00
#SBATCH -p batch
#SBATCH -N 32

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

Managing Large Training Data using ADIOS

- ADIOS is a high-performance data management library
- State-of-the-art software developed at ORNL
- Supports file I/O, streaming, WAN, data reduction
- Significant improvement in I/O performance for HydraGNN at scale

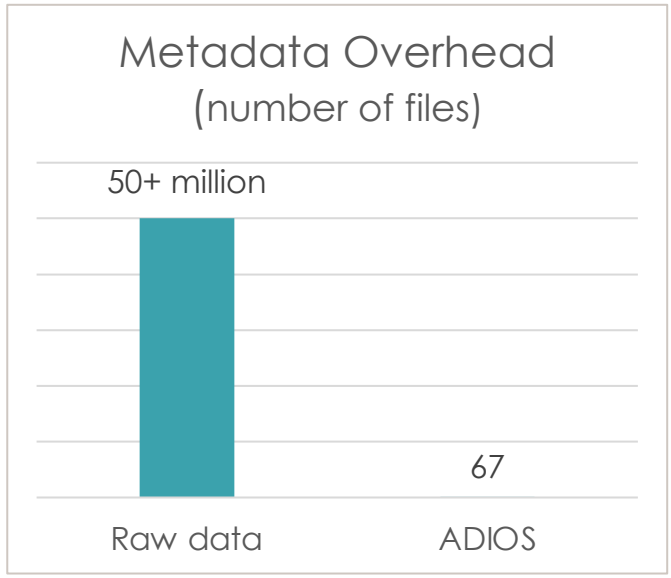


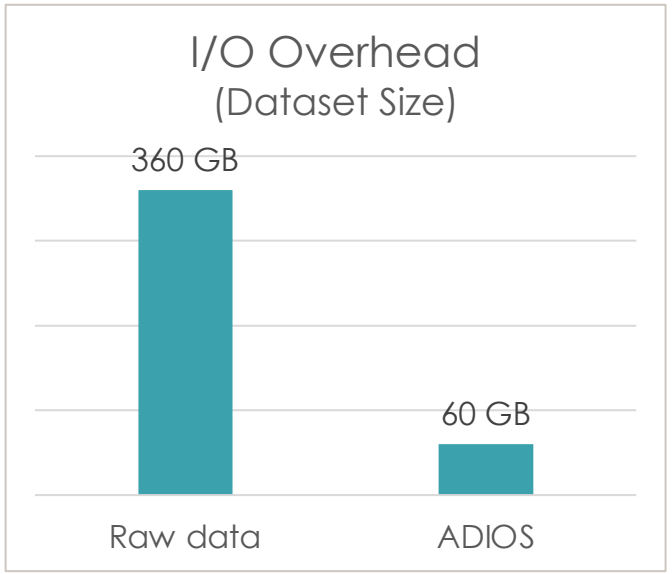
http://adios2.readthedocs.io

Managing Large Training Data using ADIOS

- Plethora of options to tune various I/O parameters
- Obtains high I/O bandwidth and alleviates metadata overhead

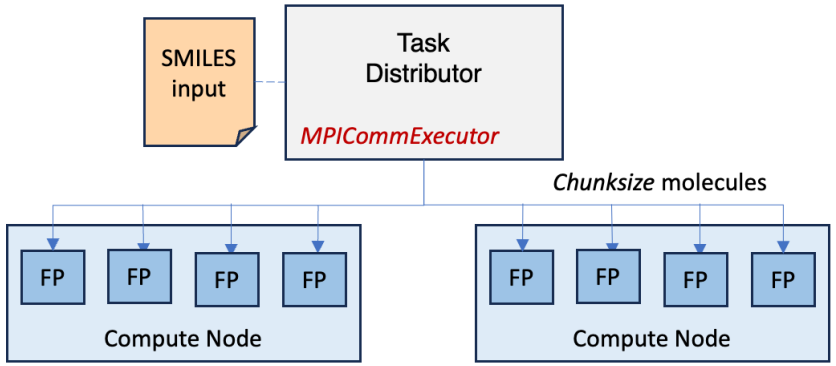
ORNL_AISD-Ex dataset with 10+ million molecules

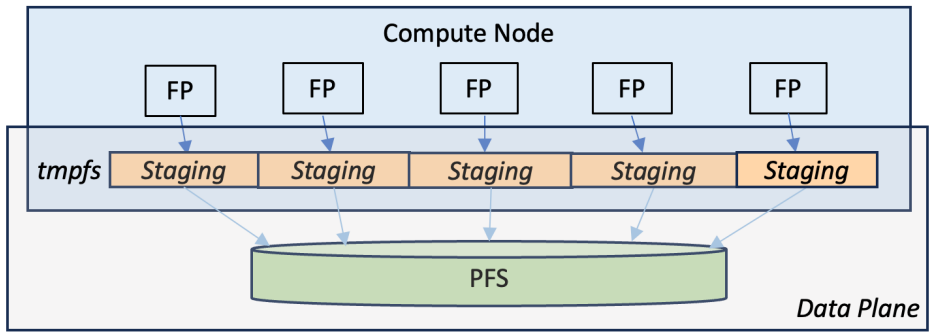




Helper Tools: Framework for Executing First-Principle Calculations at Large-Scale

- MPI-based framework for running ensembles of calculations
- Ranks internally organize into a manager-worker pattern
- Manager dynamically assigns tasks to worker processes
- Uses node-local storage as transient scratch space
- https://github.com/kshitij-v-mehta/molecular-ensemble-FP
  https://github.com/ORNL/Analysis-of-Large-Scale-Molecular-Datasets-with-Python



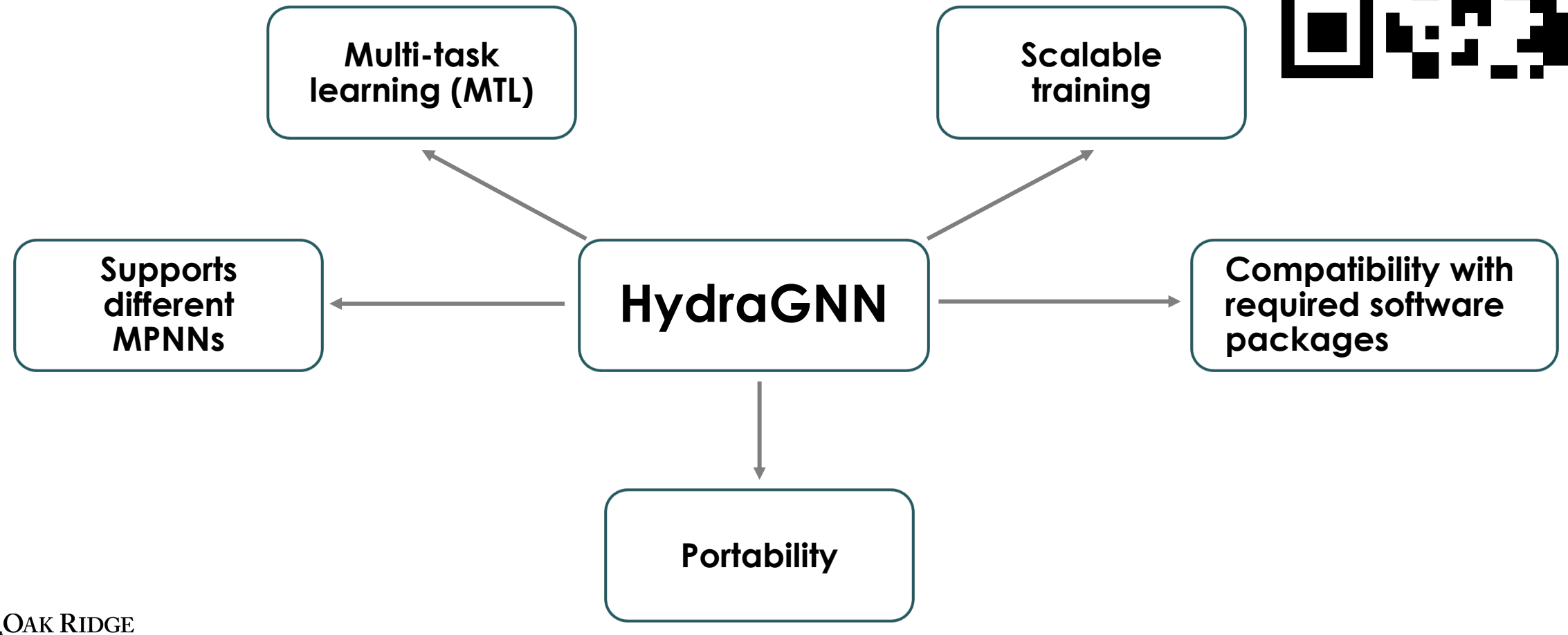


OAK RIDGE National Laboratory

Conclusions

HydraGNN: enabling large-scale GNN training on HPC

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



Past and present contributors

- Marko Burčul (master thesis at Politecnico di Milano, Italy)
- Samuel Temple Reeve (Oak Ridge National Laboratory) reevest@ornl.gov
- David Rogers (Oak Ridge National Laboratory) rogersdm@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 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. Computer time was provided by the INCITE program using the OLCF award CPH161 (AY2024) and OLCF Director's Discretion Project program using the OLCF awards MAT250 (AY2022) and LRN026 (AY2023).

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 awards ASCR-ERCAP0022058 (AY2022), ASCR-ERCAP0025216 (AY2023) and ASCR-ERCAP0027259 (AY2024).

Thank you!

Questions?

OAK RIDGE National Laboratory

Back-up Slides with Instructions to Install and Run HydraGNN on OLCF-Frontier

Jong Youl Choi (CSMD)

Massimiliano Lupo Pasini (CSED)

Modules for Frontier

```
module reset
ml PrgEnv-gnu
ml rocm/5.4.3
ml cmake/3.23.2
ml craype-accel-amd-gfx90a
ml amd-mixed/5.4.3
ml cray-mpich/8.1.26
```

Prepare Python base environment

- Use OCLF system module

```
module load miniforge3/23.11.0
```

- Use own Anaconda

```
wget https://repo.anaconda.com/archive/Anaconda3-2023.09-0-Linux-x86_64.sh

sh Anaconda3-2023.09-0-Linux-x86_64.sh –p /dir/to/install

export PATH=/dir/to/install/bin:$PATH
```

Create conda ‘hydragnn’ environment within base environment

```
conda create -n hydragnn python=3.8

source activate hydragnn
```

(Un)Installation of preliminary requirements

```
conda uninstall -y mpi # Conda's MPI cannot be used with ADIOS

conda install -y ninja

conda install -y astunparse expecttest hypothesis numpy psutil pyyaml
requests setuptools typing-extensions sympy filelock networkx jinja2
tqdm

conda install -y -c conda-forge types-dataclasses

conda install -y pyparsing build

conda install -y cython

conda install -y tensorboard

#sometimes libmkl libraries are not linked correctly by environment
conda install nomkl
```

Install stable version of PyTorch

```
pip3 install torch torchvision torchaudio torchdata --
index-url https://download.pytorch.org/whl/rocm5.4.2

Check:

python -c "import torch; print(torch.__version__)"

python -c "import torch; print(torch.__file__)"
```

Install Pytorch Geometric and its dependencies

```
## Checkout
git clone --recursive git@github.com:rusty1s/pytorch_scatter.git \
    && pushd pytorch_scatter \
    && git checkout 2.1.2-2-gc095c62 \
    && popd ## This includes Ashwin's fix
git clone --recursive git@github.com:rusty1s/pytorch_sparse.git \
    && pushd pytorch_sparse \
    && git checkout 0.6.18-3-g1577470 \
    && popd ## This includes PyTorch 2.2 support
git clone --recursive git@github.com:rusty1s/pytorch_cluster.git \
    && pushd pytorch_cluster \
    && git checkout e0eb0c1143de632786e074dee185f120afe7b852 \
    && popd ## This includes PyTorch 2.2 support
git clone --recursive git@github.com:rusty1s/pytorch_spline_conv.git \
    && pushd pytorch_spline_conv \
    && git checkout 1.2.2-5-gecf8a4a \
    && popd ## This includes PyTorch 2.2 support
git clone --recursive git@github.com:rusty1s/pytorch_geometric.git \
    && pushd pytorch_geometric \
    && git checkout 2.5.2 \
    && popd ## This includes PyTorch 2.2 support

## Use the following command to install per each directory
pip install . --verbose
```

Build and install ‘mpi4py’ from source code

```
git clone -b 3.1.5 https://github.com/mpi4py/mpi4py.git
CC=cc MPICC=cc pip install . --verbose
```

Extra packages that are needed for some examples

```
conda install -y scipy scikit-learn tensorboard
conda install -y -c conda-forge ase
conda install -y -c conda-forge rdkit
pip install mendeleev
pip install pymatgen # conda-forge was spinning all time
pip install jarvis-tools # needed for MPTrj dataset
pip install Mendeleev # atomic descriptors
pip install optuna # hyperparameter optimization
pip install deephyper # hyperparameter optimization
pip install h5py # utils to read hdf5 files
```

Build and install Adios2

```
1. Checkout
git clone -b v2.8.3 https://github.com/ornladios/ADIOS2.git

2. Install with cmake

mkdir install

mkdir build && cd build

CC=cc CXX=CC FC=ftn \
cmake -DCMAKE_INSTALL_PREFIX=/dir/to/install \
  -DCMAKE_BUILD_TYPE=Release \
  -DBUILD_TESTING=OFF \
  -DADIOS2_USE_MPI=ON \
  -DADIOS2_USE_Fortran=ON \
  -DADIOS2_BUILD_EXAMPLES_EXPERIMENTAL=OFF \
  -DADIOS2_BUILD_TESTING=OFF \
  -DADIOS2_USE_HDF5=OFF \
  -DADIOS2_USE_SST=OFF \
  -DADIOS2_USE_BZip2=OFF \
  -DADIOS2_USE_PNG=OFF \
  -DADIOS2_USE_DataSpaces=OFF \
  -DADIOS2_USE_Python=ON \
  -DPython_EXECUTABLE=`which python` \
  ..

make -j8 && make install

3. Export env:
export PYTHONPATH=/dir/to/install/lib/python3.8/site-packages:$PYTHONPATH
```

Build and install 'gptl4py' (Performance counter)

1. Need gptl: https://jmrosinski.github.io/GPTL/

```
wget https://github.com/jmrosinski/GPTL/releases/download/v8.1.1/gptl-8.1.1.tar.gz

tar xvf gptl-8.1.1.tar.gz

cd gptl-8.1.1

./configure --prefix=/dir/to/install CC=cc CXX=CC FC=ftn

make install
```

2. Checkout

```
git clone git@github.com:jychoi-hpc/gptl4py.git
```

3. Install

```
GPTL_DIR=/dir/to/gptl CC=cc CXX=CC python setup.py install
```

Build and install DDStore (data loader) for efficient data management

1. Checkout source

```
git clone https://github.com/ORNL/DDStore.git
```

2. Install

```
CC=cc CXX=CC python setup.py install
```

Example of job submission script on OLCF-Frontier

```
#!/bin/bash
#SBATCH -A CPH161
#SBATCH -J HydraGNN
#SBATCH -o job-%j.out
#SBATCH -e job-%j.out
#SBATCH -t 00:30:00
#SBATCH -p batch
#SBATCH -N 10
#SBATCH -q debug

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 HYDRAGNN_AGGR_BACKEND=mpi

source $DIRPATH/module-to-load-frontier.sh
source $DIRPATH/max_conda_envs_frontier/bin/activate
conda activate hydragnn

export PYTHONPATH=$DIRPATH/ADIOS_frontier/install/lib/python3.8/site-packages/:$PYTHONPATH

export PYTHONPATH=$PWD:$PYTHONPATH
cd  examples/mptrj/

srun -n$((SLURM_JOB_NUM_NODES*8)) --gpus-per-task=1 --gpu-bind=closest python -u train.py --preonly
```

Load modules
Activate base conda environment
Activate ‘hydragnn’ environment

Export path where ADIOS2 is installed

Export path for main directory of HydraGNN