Open MatSci ML Toolkit: A Flexible Framework for Machine Learning in Materials Science

Santiago Miret *† Intel Labs
Kin Long Kelvin Lee * Intel AXG
Carmelo Gonzales Intel Labs
Marcel Nassar Intel Labs
Krysztof Sadowski ‡ IP Rally Technologies

Abstract

The Open MatSci ML Toolkit is a flexible, self-contained and scalable Python-based framework to apply deep learning models and methods on scientific data with a specific focus on materials science and the OpenCatalyst Dataset. Our toolkit provides: 1. Scalable MLOps of materials science machine learning experiments leveraging PyTorch Lightning across different computation capabilities (laptop, server, cluster) and hardware platforms (CPU, GPU, XPU) without sacrificing performance in compute and modeling; 2. DGL support for rapid graph neural network prototyping and development. By sharing this toolkit with the research community via open-source release, we hope to: 1. Lower the entry barrier for new machine learning researchers and practitioners that want get started on interacting with the OpenCatalyst dataset, which currently makes up the largest computational materials science dataset. 2. Enable the scientific community to apply advanced machine learning tools to high-impact scientific challenges, such as modeling of materials behavior for climate change applications. Experiments applying our framework on OpenCatalyst tasks show promising results in compute scaling and model performance.

1 Introduction

Catalysts are essential components in chemical systems that help accelerate the speed of chemical reactions. Catalytic materials design, especially low-cost catalysts, remain an ongoing challenge that will continue to become more important for a variety of applications, including renewable energy and sustainable agriculture. The OpenCatalyst Project, jointly developed by Fundamental AI Research (FAIR) at Meta AI and Carnegie Mellon University's Department of Chemical Engineering, encompasses one of the first large-scale datasets to enable the application of machine learning (ML) techniques. The full dataset contains over 1.3 million molecular relaxations of 82 adsorbates on 55 different catalytic surfaces. The original release from 2019 has also been supplemented by subsequent updates in 2020 and 2022 with the researchers also maintaining an active leaderboard and annual competition [Chanussot* et al., 2021]. The significant effort of providing high-quality data for catalytic materials is a major step forward in enabling ML researchers and practitioner to innovate on materials design challenges as shown by the large variety of deep learning and high performance computing features. This collection of data and software capabilities has already enabled

*Equal Contribution
†Correspondence to: <santiago.miret@intel.com>
‡Work performed while at Intel Poland

36th Conference on Neural Information Processing Systems (NeurIPS 2022) AI4Mat Workshop.

the development of new geometric deep learning architectures ([Klicpera et al., 2020] [Gasteiger et al., 2021]) trained with nearly billions of parameters [Sriram et al., 2022].

While the software framework of the original OpenCatalyst repository is very powerful, it contains a significant amount of complexity due to various interacting pieces of software: model definitions, functions for distributed training, and task abstraction are not always self-contained. This can make it very challenging for new ML researchers to navigate and interact with the repository, create new architectures or modeling methods, and run experiments on the dataset. To address the challenges of usability and ease of use, we introduce the Open MatSci ML Toolkit, a flexible and easy-to-scale framework for deep learning on materials science with focus on the Open Catalyst dataset.

2 Software Framework

The Open MatSci ML Toolkit software framework is designed with great emphasis on abstraction and inheritance in order to maximize reusability and agility for machine learning researchers. These ideas are achieved in part by present-day best practices in Python as a language, and through modern, specialized frameworks such as PyTorch Lightning and DGL. We believe these design choices make it significantly easier to apply novel model architectures and training techniques to scientific data, in particular the OpenCatalyst dataset. In the following sections, we will discuss reabstractions and refactors from the original OpenCatalyst implementation.

2.1 PyTorch Lightning Refactor

In modern AI/ML workflows, the concept of "MLOps" comprises the lifecycle from model conception and implementation, to training and testing in a variety of software/hardware environments, to drawing inferences on new data, and all of the iterative cycles in between. Thus, a non-negligible amount of time spent by researchers for new workloads is typically in engineering: interfacing data with new architectures, metric logging, performance profiling, and ensuring consistent functionality when developing on a laptop to distributed training on multiple computing nodes, across multiple accelerators. Because of the grand scale that OpenCatalyst aims and successfully achieves, a large amount of the original codebase corresponds to performance and functionality; this goes to say that complexity is necessary to be able to take advantage of data parallelism, to perform hyperparameter optimization, and to support the various catalyst prediction tasks. This lays a significant amount of responsibility on both developers and users: the former must create a comprehensive suite of tests and rely heavily on CI/CD to ensure functionality, and the latter must navigate a maze of software dependencies and documentation, which are also maintained by the developer.

One half of the conceptual changes in Open MatSci ML Toolkit—the other half being the primary graph framework—is to offload MLOps related components to a well designed and maintained framework, PyTorch Lightning [Falcon and The PyTorch Lightning Team, 2019]. By reusing certain components in OpenCatalyst—both dataset and framework—and relying on PyTorch Lightning for pipeline abstraction, we are required to maintain less of the codebase while providing more flexibility/extendibility, transparency, and functionality. Figure 1 illustrates the end-to-end pipeline/directed acyclic graph for the Open MatSci ML Toolkit, whose elements should be somewhat familiar to those who have used OpenCatalyst and/or PyTorch Lightning.

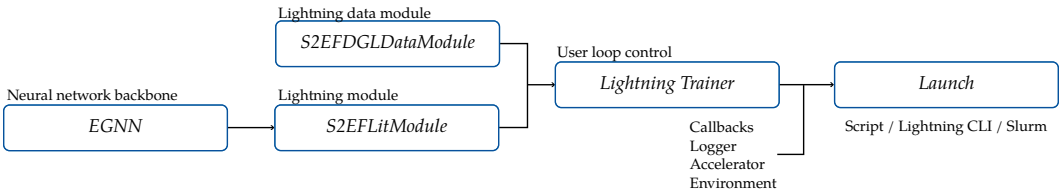

Figure 1: Illustration of the Open MatSci ML Toolkit pipeline with concrete components using the "S2EF" task. Dataset/task splits and configurations are specified through `LightningDataModules`. Task specific `LightningModules` encode the logic for training, metric logging, and how data is passed from dataset to an underlying abstract deep learning model. The `Trainer` interface provides an the ability to control feedback (e.g. logging, progress bars), training flow (`Callbacks`), and XPU usage without the need to modify the pipeline source code.

2.1.1 Data abstraction

In order to support future neural network research, we expanded the scope of the original OpenCatalyst dataset to support graph and non-graph data structures, as well as implemented a number of quality of life improvements to the general developer workflow. We refer the reader to the Appendix (i.e. Figure 4) for more details pertaining to the changes, and here we only briefly highlight the core differences in user experience.

One of the core principles in the Open MatSci ML Toolkit is to have continuity from developing and testing on local environments such as laptops, to using the pipeline in high performance computing environments. In terms of data pipeline abstraction, on the one end the Open MatSci ML Toolkit provides preprocessed, miniature ($\sim$100 graphs) development or "devset"s: this circumvents the need to download, extract, and preprocess the data on personal computers constrained by storage and by computational power, while allowing researchers to prototype on the full pipeline. The development sets are created by taking random subsplits of the 200K data splits from the OpenCatalyst dataset, and the mechanism for creating other splits are provided with the Open MatSci ML Toolkit, facilitating further research into data efficiency. To use the devsets for development, there is a convenient mechanism for retrieving the DGL version of each task:

```
from ocpmodels.lightning.data_utils import S2EFDGLDataModule,
    IS2REDGLDataModule
# default settings optimized for local development; small batch, no
    parallel loaders
devset_module = S2EFDGLDataModule.from_devset()
```

On the other end of the spectrum, where one wishes to distribute the dataset across multiple workers on multiple compute nodes, users can leverage the same data modules as the miniature case: the `DistributedDataParallel` data sampling and loading is offloaded to PyTorch Lightning internals as shown in Section 2.1.3. Moreover, the Open MatSci ML Toolkit data pipeline abstraction is designed to facilitate exploration of other data representations of materials systems: an example of this includes the use of geometric algebra on point clouds (see, for example, Spellings [2021]), which do not use graph structures, but retain the advantages of model equivariance and invariance.

2.1.2 Model abstraction

The model abstraction, as seen in the bottom left nodes in Figure 1, comprises a neural network backbone and a task-specific `LightningModule`. In the concrete example described in Section 3, the `EGNN` model represents a subclass of an `AbstractEnergyModel`: a model that takes arbitrary input, and predicts the energy. For instance, a graph-based model will process nodes and perform some readout operation to regress to a scalar value for the energy. At a higher level, the task specific `S2EFLitModule` is instantiated by passing an instance of `EGNN`, and implements the logic for training (i.e. forward-backward passes), validation and testing, and logging. By conceptually separating model (i.e. the neural network itself) from training mechanism, researchers only need to focus on architecture development by subclassing `AbstractEnergyModel`, as the rest of the pipeline stays the same barring changes in *what* data is required by the model.

2.1.3 Training loop

The primary component relevant to the training process is the PyTorch Lightning `Trainer` class, which orchestrates the components mentioned above and executes training, validation, testing, and inference loops. The `Trainer` interface also configures performance oriented settings such as accelerator usage, distributed compute, and mixed precision as shown in the example below:

```
trainer = pl.Trainer(
    max_epochs=5,
    callbacks=[...],     # configure callbacks
    accelerator="gpu",   # move between XPUs
    precision="bf16",    # use new data types
    strategy="ddp",      # 8 workers across 4 nodes
    num_nodes=4,
    devices=2
)
```

The main advantage is being able to seamlessly navigate between development and training cycles: the core pipeline remains unchanged, however with a simple change in configuration, the user is able to take advantage of computational resources as they become available. Under the hood, the Lightning abstractions handle data movement to devices, autocasting in correct contexts, and orchestrate workers.

3 Experiments & Testing

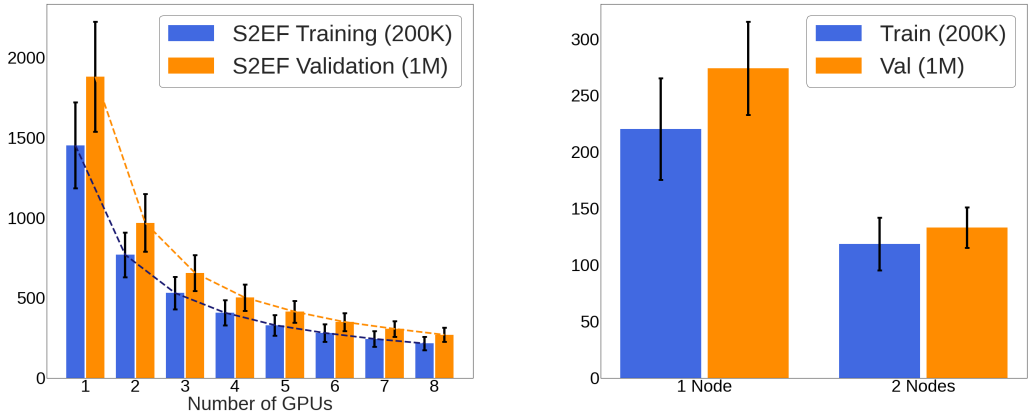

Figure 2: Time per Epoch (s) for Multiple Devices on Single Node and 8-Device Multi-Node Setting for the S2EF Task with various dataset sample splits

We applied the Open MatSci ML Toolkit to the OCP-20 [Chanussot* et al., 2021] S2EF task with 200K training samples and 1M validation samples, which is a common task amendable to studying both the compute and task performance one can achieve using our framework. Single node scaling to multiple GPUs shows a decreasing benefit as more GPUs get added, likely due to increasing communication cost between the different devices outweighing the benefits of parallel computing of batches. Epoch training throughput in the multi-node suggests close to linear scaling. While the benefits of compute scaling increase with a more intensive task, the overall compute time, both in core-time and in wall-clock time, also increases making the overall experiment more costly.

3.1 S2EF Task Performance

We perform a training experiment using the Open MatSci ML Toolkit framework for the S2EF 200K/1M task on a single node with 8 GPUs on E(n)-GNN proposed by Satorras et al. [2021] for 8 epochs recording training loss at every step and validation loss at every epoch. E(n)-GNN has the inductive bias of equivariance with respect to positions, where general equivariance for functions $f(I)$, $g(I)$ for an entity I is defined as: $f(g(I) = g(f(I))$. Intuitively this means that the features of an entity transform equally with a given manipulation, such as a rotation. This is particularly useful for property prediction in material compounds, such as the S2EF task, where rotation of the entire compound itself does not affect the properties of the compound.

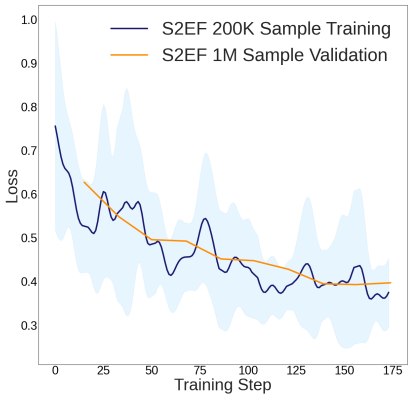

Figure 3: Training and Validation Loss for S2EF Task (200K Training Samples & 1M Validation Samples) on E(n)-GNN

The results in Figure 3 show that the validation loss generally tracks the training loss. The trend of the training loss also indicates a downwards slope with recurring deviation pattern, suggesting that the network may find particular data more challenging than others.

References

Lowik Chanussot*, Abhishek Das*, Siddharth Goyal*, Thibaut Lavril*, Muhammed Shuaibi*, Morgane Riviere, Kevin Tran, Javier Heras-Domingo, Caleb Ho, Weihua Hu, Aini Palizhati, Anuroop Sriram, Brandon Wood, Junwoong Yoon, Devi Parikh, C. Lawrence Zitnick, and Zachary Ulissi. Open catalyst 2020 (oc20) dataset and community challenges. *ACS Catalysis*, 2021. doi: 10.1021/acscatal.0c04525.

William Falcon and The PyTorch Lightning Team. PyTorch Lightning, 3 2019. URL `https://github.com/Lightning-AI/lightning`.

Johannes Gasteiger, Florian Becker, and Stephan Günnemann. Gemnet: Universal directional graph neural networks for molecules. *Advances in Neural Information Processing Systems*, 34: 6790–6802, 2021.

Johannes Klicpera, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs. *arXiv preprint arXiv:2003.03123*, 2020.

Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

Matthew Spellings. Geometric algebra attention networks for small point clouds. *arXiv preprint arXiv:2110.02393*, 2021.

Anuroop Sriram, Abhishek Das, Brandon M Wood, Siddharth Goyal, and C Lawrence Zitnick. Towards training billion parameter graph neural networks for atomic simulations. *arXiv preprint arXiv:2203.09697*, 2022.

A Hyperparameters

Example hyperparameters for E(n)-GNN

Table 1: Hyperparameters for E(n)-GNN

Hyperparameter	Value
MLP hidden dim	32
MLP output dim	32
# of EGNN layers	3
Node MLP dim	$[48, 48]$
Edge MLP dim	$[16, 16]$
Atom position MLP dim	$[64, 64]$
MLP activation	ReLU
Graph read out	Sum
Node projection block depth	2
Node projection hidden dim	128
Node projection activation	ReLU
Output block depth	3
Output hiddem dim	64
Output activation	ReLU
Optimizer parameters	
Learning Rate	0.003626
Gamma	0.6878
Batch Size	8

Example hyperparameters for MegNet

Table 2: Hyperparameters for MegNet

Hyperparameter	Value
Edge MLP dim	2
Node MLP dim	5
Graph variable MLP dim	9
MLP projection dim	11
MegNet blocks	4
MLP hidden dims	$[128, 64]$
MegNet convolution dims	$[128, 128, 64]$
# of S2S layers	5
# of S2S iterations	4
Output projection dims	$[64, 16]$
Dropout	0.1
Optimizer parameters	
Learning Rate	0.0001
Gamma	0.2
Batch Size	8

Example hyperparameters for Gala

B Development Example

A self-contained python script running the full pipeline on one of our dev-sets is shown below:

```
""" Sample Python Script Without Imports """
```

Table 3: Hyperparameters for Gala

Hyperparameter	Value
Input dimension	200
Hidden dimension	100
Merge function	concat
Join function	concat
Rotation-invariant mode	full
Rotation-covariant mode	full
Rotation-invariant value norm	momentum
Rotation-equivariant value norm	momentum layer
Value function normalization	layer
Score function normalization	layer
Block-level normalization	layer
Optimizer parameters	
Learning Rate	0.001
Gamma	0.8
Batch Size	1

```python

# Define Parameters
BATCH_SIZE = 8
NUM_WORKERS = 4
REGRESS_FORCES = False
epochs = 5


# Model configuration for MegNet
model_config = {
    "edge_feat_dim": 2,
    "node_feat_dim": 5,
    "graph_attr_dim ": 9,
    "dim": 1,
    "num_blocks": 4,
    "hiddens": [128, 64]
    "conv_hiddens": [128, 128, 64]
    "s2s_num_layers": 5,
    "s2s_num_iters": 4,
    "output_hiddens": [64, 16],
    "is_classification": False,
    "dropout": 0.1,
}

# use default settings for MegNet
megnet = MegNet(**model_config)


# use the GNN in the LitModule for all the logging, loss computation, etc.
model = S2EFLitModule(megnet, regress_forces=REGRESS_FORCES, lr=1e-3, gamma=0.1)
data_module = S2EFDGLDataModule.from_devset(
    batch_size=BATCH_SIZE, num_workers=NUM_WORKERS
)

# alternatively, if you don't want to run with validation, just do S2EFDGLDataModule.from_devset
data_module = S2EFDGLDataModule(
    train_path=s2ef_devset,
```

```
    val_path=s2ef_devset,
    batch_size=BATCH_SIZE,
    num_workers=NUM_WORKERS,
)

trainer = pl.Trainer(accelerator="gpu", strategy="ddp", devices=2,
    max_epochs=epochs)
trainer.fit(model, datamodule=data_module)
```

Listing 1: Self-Contained Example Script With Scalable Devices

As can be seen in the definition of `trainer`, this short script already performs training on two GPUs with users being able to change the `devices` variable to adjust the numbers of GPUs they want to leverage for distributed training on a single node.

C Data pipeline abstraction

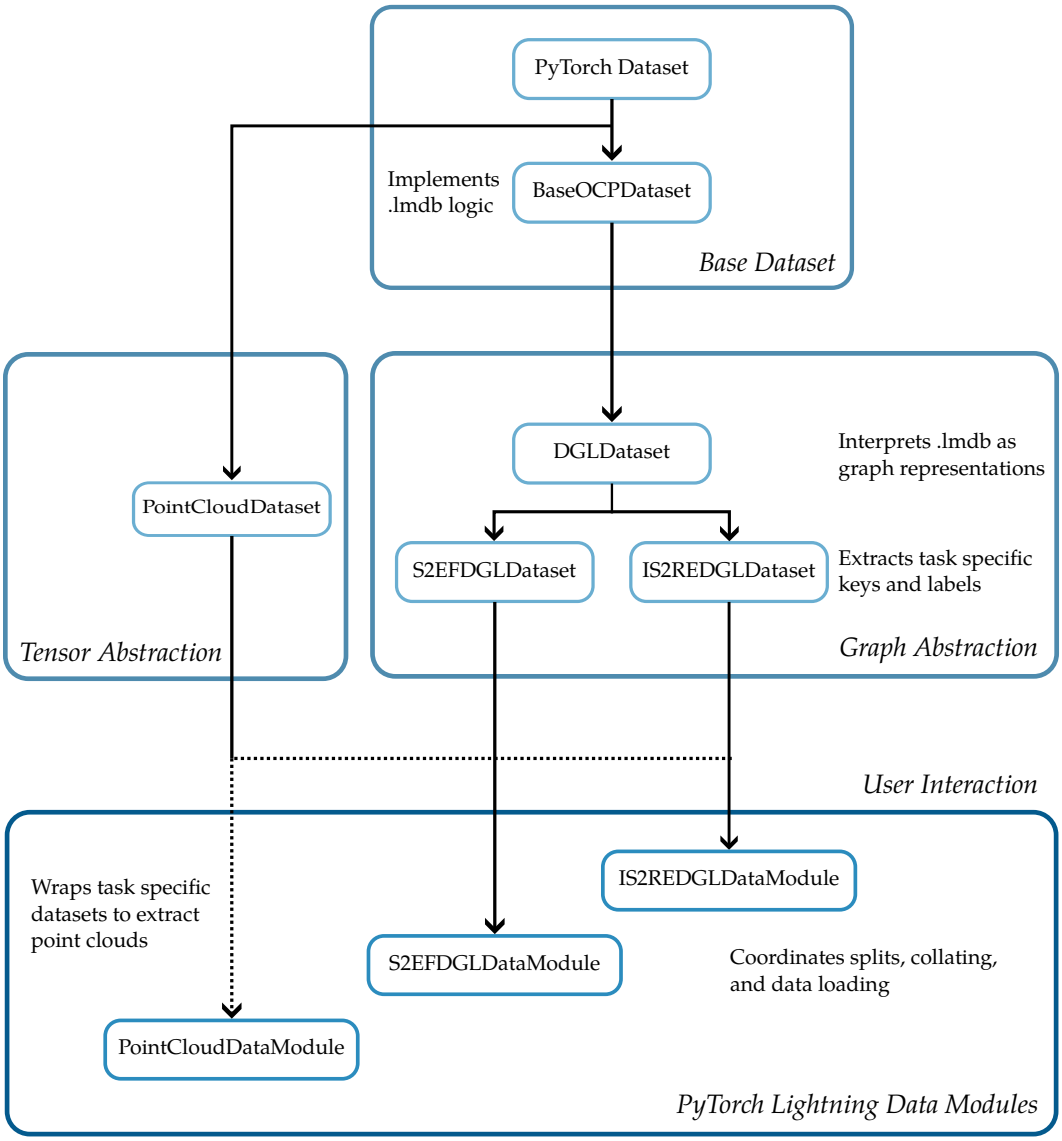

Figure 4: Inheritance diagram for the data abstraction in Open MatSci ML Toolkit. The main user interaction layer is presented at the bottom, corresponding to subclasses of `LightningDataModule`. Arrows denote directional relationship between the classes; the dashed line indicates that the `PointCloudDataset` wraps the task specific datasets, whereby the user is provided with a sampled point cloud representation of the original graphs.