# End-to-end AI framework for interpretable prediction of molecular and crystal properties

IOP Publishing Mach. Learn.: Sci. Technol. 4 (2023) 025036 https://doi.org/10.1088/2632-2153/acd434

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
21 December 2022

REVISED
11 April 2023

ACCEPTED FOR PUBLICATION
10 May 2023

PUBLISHED
29 June 2023



PAPER

# End-to-end AI framework for interpretable prediction of molecular and crystal properties

**Hyun Park**[1,3,8], **Ruijie Zhu**[2,3], **E A Huerta**[3,4,5,*], **Santanu Chaudhuri**[3,6], **Emad Tajkhorshid**[1,7,8] and **Donny Cooper**[9]

[1] Theoretical and Computational Biophysics Group, Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States of America
[2] Department of Materials Science and Engineering, Northwestern University, Evanston, IL 60208, United States of America
[3] Data Science and Learning Division, Argonne National Laboratory, Lemont, IL 60439, United States of America
[4] Department of Computer Science, The University of Chicago, Chicago, IL 60637, United States of America
[5] Department of Physics, University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States of America
[6] Multiscale Materials and Manufacturing Lab, University of Illinois Chicago, Chicago, IL 60607, United States of America
[7] Department of Biochemistry, University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States of America
[8] Center for Biophysics and Quantitative Biology, University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States of America
[9] Computational Science and Engineering, Data Science and AI Department, TotalEnergies EP Research & Technology USA, LLC, Houston, TX 77002, United States of America
[*] Author to whom any correspondence should be addressed.

**E-mail:** elihu@anl.gov

**Keywords:** AI, inorganic crystals, small molecules, metal-organic frameworks, interpretable AI

## Abstract

We introduce an end-to-end computational framework that allows for hyperparameter optimization using the `DeepHyper` library, accelerated model training, and interpretable AI inference. The framework is based on state-of-the-art AI models including `CGCNN`, `PhysNet`, `SchNet`, `MPNN`, `MPNN-transformer`, and `TorchMD-NET`. We employ these AI models along with the benchmark `QM9`, `hMOF`, and `MD17` datasets to showcase how the models can predict user-specified material properties within modern computing environments. We demonstrate transferable applications in the modeling of small molecules, inorganic crystals and nanoporous metal organic frameworks with a unified, standalone framework. We have deployed and tested this framework in the ThetaGPU supercomputer at the Argonne Leadership Computing Facility, and in the Delta supercomputer at the National Center for Supercomputing Applications to provide researchers with modern tools to conduct accelerated AI-driven discovery in leadership-class computing environments. We release these digital assets as open source scientific software in GitLab, and ready-to-use Jupyter notebooks in Google Colab.

## 1. Introduction

With the explosion of AI models [1–5] developed to predict various material properties over the recent years, it has become difficult to keep track of the available AI models and the datasets that are used for training and inference. Numerous efforts [6, 7] have been made toward the integration of AI models and their associated datasets in one place to streamline their use for a wide range of applications and a broad community of users [8–10]. AI models and datasets are often available through open repositories, in the best scenario, so a user can download, deploy and reproduce their putative capabilities. Unfortunately, this is a time-consuming and laborious process, which can be further complicated when tools and libraries used to develop the AI models are not available, deprecated, or non-backwards compatible in computing environments of new users. Furthermore, most of the existing packages are specialized in predicting quantum mechanical properties of small molecules, few of them support crystals.

In order to address these shortcomings, here we report the construction of a computational framework that consolidates libraries, AI models and AI interpretability tools to study molecules, crystals, and

metal-organic frameworks. The framework enables hyperparameter tuning through the open source library DeepHyper [11], model training, and interpretable inference of small-molecule quantum mechanics (QM) properties from public datasets such as QM9 [12], and crystal properties from datasets such as hMOF [13]. Key aspects of this computational framework include:

**Novel features of AI models.** The node and edge embedding schemes of two graph neural network models, PhysNet [2] and CGCNN [5], were modified from the original adjacency matrix format to an adjacency list format to reduce redundant information and enable faster training. We also adapted small-molecule property prediction models to take in crystal structures as input such as the crystal version of SchNet [1].

**Transferable AI applications.** We demonstrate the transferability of the learned force fields by training TorchMD-NET [3] model using selected molecular dynamics (MD) trajectory data of a given set of molecules in the MD17 dataset [14] to perform MD simulations of similar molecules. In particular, we show that a model trained based on ethanol is transferable to both *n*-propanol and iso-propanol, and a model trained based on uracil is transferable to pyrimidine and naphthalene. All of the results are automatically logged to weights and biases (WandB) [15], a machine learning tracking tool, for simple access.

**Interpretable AI inference.** Interpretation methods provide a novel pathway to deepen the understanding of the structure-property relationships of materials. Previous work [16] has shown great success in applying interpretation methods to identify key functional groups in molecules that contribute to toxicity, including Excitation Backpropagation [17], CAM [18] and Grad-CAM [19] and Contrastive gradient [20]. Moreover, The uniform manifold approximation and projection (UMAP) method has also been applied to effectively visualize the distribution shift of sampled molecules on a 2D plane with and without transfer learning [21]. To gain a better understanding of model predictions, we provide two interpretation methods to explain the learned features. The first method, Grad-CAM, highlights selected atoms of molecules that are significant for model predictions. The second method, uniform manifold approximation and projection [22], or UMAP, maps the last hidden layer of the model onto a 3D plane. In this way, we can make more sense of the molecular clusters with similar properties.

This AI suite and scientific software are released with this manuscript, and may be found in GitLab [23]. To facilitate the use of these resources, we have prepared Jupyter notebooks in Google Colab [24], which have been tested independently by researchers not involved in this work to ensure that these resources are easy to follow and use. To ensure that all the resources used in this article are self-contained, we have also published the datasets used for these studies in Zenodo [25]. We expect that this collection of state-of-the-art graph neural networks, transformer models, and analysis methods for small molecules and crystals will empower AI practitioners to seamlessly perform hyperparameter optimization, accelerated training, and interpretable AI inference in modern computing environments with a unified, standalone computational framework.

## 2. Related work

Graph neural networks have shown great success for modeling molecular and crystal structures. For small molecules, a suite of models have been proposed, including DimeNet [4], GemNet [26], SphereNet [27], ComENet [28], SchNet [1] and PhysNet [2]. These models take in atomic coordinates and atomic numbers as input, and represent atoms as nodes and bonds as edges. Typical target properties for these models are QM properties of molecules such as internal energy, heat capacity and zero point vibrational energy (ZPVE). For crystal structures, periodic boundary conditions need to be considered, therefore crystal graph representations are typically used. Example graph neural networks that take in crystal structures as input include ALIGNN [29], CGCNN [5], and MEGNet [30]. These models first extract crystal graphs from the structures, then generate atom and edge embeddings for the center atoms and their neighbors. The bond and edge information is then updated via message passing. The target properties for these models are typically QM properties of crystals, e.g. formation energy and band gap.

The growing number of the graph neural networks available for this purpose pushes the need for an end-to-end AI framework. Previous efforts toward such a goal typically missed one or more important aspects. For example, MatDeepLearn [7] integrates a suite of graph neural networks, including CGCNN, MEGNet, MPNN [31], GCN [32] and SchNet. Although it can be used for hyperparameter tuning, model training, and inference, it lacks the explainability feature, which limits the amount of chemical insights that could be extracted from the results. Another example is dive in graphs [6], which enables model training and explanation. However, it does not allow for hyperparameter tuning, therefore only models with preset hyperparameters can be used. A complete package offering all of the aforementioned functionalities is therefore needed.

Our AI framework also offers the functionality to perform MD simulations for small molecules, enabled by TorchMD-NET, an SE3-equivariant transformer interatomic potential model that establishes a relationship between atomic configurations and potential energies and forces. The MD trajectories of selected molecules taken from the MD17 dataset were used for training the TorchMD-NET models.

## 3. Methods

Here we describe the key building blocks of our general-purpose AI framework:

(i) It provides built-in datasets and neural networks that we modified to take in adjacency list format node and edge embeddings, a more memory efficient embedding scheme than adjacency matrix format
(ii) It enables distributed hyperparameter tuning of neural networks via the scalable and computationally efficient library DeepHyper
(iii) Model training and interpretable inference are performed by specifying a few command line arguments
(iv) Results are auto-logged to WandB, a machine learning tool for easy tracking and visualization
(v) MD simulations can be performed for small molecules using TorchMD-NET if trained with MD trajectories from the MD17 dataset, enabled by the atomic simulation environment (ASE) library [33].

This framework has been deployed and tested in leadership computing platforms to reduce the overhead for researchers that require access to hyperparameter tuning, model training and explainable inference tools in a single, unified framework. Below we describe each of these components in further detail.

### 3.1. Hyperparameter tuning

This feature was done using the DeepHyper [11] library. In this method, hyperparameters of interest are given prior distributions and their posterior distributions are adjusted based on the centralized Bayesian optimization (CBO) algorithm with a given acquisition function and a surrogate model. The graph neural networks in this framework are coupled with DeepHyper to enable faster hyperparameter tuning.

### 3.2. Datasets

QM9 and MD17 datasets were used as input to graph neural networks. The QM9 dataset consists of molecular structures and QM properties of 133 885 molecules with up to nine atoms of type H, C, O, N and F. For demonstration purposes, the selected QM properties in this work include the highest occupied molecular orbital (HOMO), and ZPVE. The MD17 dataset consists of *ab-initio* MD trajectories of ten molecules at different levels of theory. Both datasets are available in the PyTorch Geometric library.

### 3.3. Node and edge embedding schemes

Instead of using the original adjacency matrix format for node and edge embeddings, we modified it to adjacency list format. The term embedding, for both molecular and crystal graphs, refers to the information attached to a node (an atom) or an edge (a bond). Both node and edge embeddings can be scalars, vectors or higher order tensors. Node embeddings encode information such as mass, charge and orbital hybridization, whereas edge embeddings encode information such as interatomic distance and bond order. Depending on the model architecture, some embeddings are physics- or chemistry-based while others are learned. For physics- or chemistry-based embeddings, fixed information such as hybridization, mass, atomic radius, and whether the fragment is a part of an aromatic ring is encoded. On the other hand, learned embeddings refer to embeddings that are iteratively optimized by a neural network model via stochastic gradient descent.

In adjacency matrix format edge embeddings, the adjacency matrix is encoded into a fixed-size matrix, whose size is determined by the largest molecule in the dataset. For other molecules, their vectors are padded to be the same dimension as the largest one. Each element in the adjacency matrix indicates whether the two corresponding nodes are connected, as determined via some distance-based criteria. Since padding is applied to smaller molecules in the matrix, users need to know *a priori* the largest molecule size, then perform masking to obtain the padding values, which can be burdensome for graphics processing units (GPU) memories. By using the adjacency list format, however, only the information for connected atoms is preserved, thereby avoiding the need for padding and taking less memory to load. In this case, faster loss convergence and higher prediction accuracy are expected. The adjacency list format has been implemented in a number of Python libraries such as Deep Graph Library (DGL) [34] and PyTorch Geometric [35]. In this work, we use CGCNN model as an example to demonstrate a boost in model training performance when an adjacency list format is used in place of an adjacency matrix format.

Our AI framework allows users to perform hyperparameter tuning, model training and interpretable inference for pre-trained models or train new models with a few arguments passed. The main improvements

over previously proposed general-purpose machine learning model libraries is the explainability feature, which consists of two parts. First, by extracting high dimensional hidden layer information from the learned models and projecting it onto low dimensions via the UMAP method, we can effectively visualize the clustering of molecules, with similar practice as in [36–38]. Second, Saliency Map [39], CAM and Grad-CAM methods are used to highlight important atoms in molecular graphs, as described in [40].

## 4. Results

Below we present a comprehensive analysis of our results, from hyperparameter optimization to interpretable AI inference.

### 4.1. Hyperparameter optimization

The `DeepHyper` library is used for hyperparameter optimization of graph neural networks. `DeepHyper` is easy to use and can be readily deployed on GPU-based high-performance computing platforms. central processing units (CPUs) can be used if GPUs are not available. However, if the user has access to multiple GPUs, then the GPU option will be automatically chosen, with each core performing hyperparameter search using the CBO algorithm, given an acquisition function such as the upper confidence bound, and a surrogate model, e.g. random forest.

The list of hyperparameters considered in this work along with their ranges are summarized in table 1. Hyperparameter optimization results for `PhysNet` with `ZPVE` as target property are shown in tables 2 and 3. It is worth mentioning that since `DeepHyper` tries to maximize the objective of search, the opposite number of validation error was used as the objective, therefore a larger absolute value of objective corresponds to a better combination of hyperparameters. The hyperparameter tuning results for `PhysNet` with `HOMO` as the target property are shown in tables A1 and A2. For hyperparameters with integer or floating number values, the ranges represent the lower and upper bounds. For hyperparameters `amp` and `optimizer`, the ranges represent all available options.

Among the hyperparameters, `agb`, or accumulated grad batches, helps overcome memory constraints; `amp`, or automatic mixed precision, speeds up neural network training; and `gradient clip`, a machine learning technique where the gradients of neural network parameters are rescaled to between 0 and 1, is known to stabilize neural network training by avoiding sudden changes in parameter values (also known as the exploding gradient problem) [41].

The optimal hyperparameter combinations found by `DeepHyper` are listed in the top rows of tables 2 and 3, with the optimal objective being −0.9226. We notice that this set of hyperparameters include f32 precision (amp = 'false'), a standard learning rate (0.002 96), and a low gradient norm clipping value (0.002 45). These result in small gradient accumulation, which may help mitigate sudden gradient updates.

We have tested multiple sets of hyperparameters with varying ranges and prior distributions. Our hyperparameter tuning configuration input file is prepared in `YAML` format. Discrete hyperparameter values such as the number of epochs and the batch size are sampled from uniform distributions whereas continuous hyperparameters such as the learning rate and the gradient clip are sampled from normal distributions with/without log scale. The ranges of hyperparameters along with the prior distributions for sampling are both user-customizable.

Once the hyperparameter configuration and the prior distributions are in place, `DeepHyper` can use multiple GPUs to perform hyperparameter tuning, taking full advantage of GPU parallelization. Next, all the optimization results will be saved and automatically logged to Weights and Biases. If the tuning step is interrupted, it can be resumed from the last saved checkpoint by specifying the `----resume` tag.

For the `PhysNet` model with `HOMO` and `ZPVE` as target properties, we compared hyperparameter tuning performance of `DeepHyper` with a naive algorithm that performs random selection of hyperparameters. Since `DeepHyper` utilizes the CBO algorithm to optimize hyperparameters, the target property values are used for decision making. For the naive algorithm, however, hyperparameters were randomly selected from the hyperparameter grid in table 1. A total of 20 models were trained for 30 epochs with hyperparameters given by the two methods. The distributions of the losses (mean squared error) are compared in figure 1, and the metrics are summarized in table 4.

*Key findings:* For the prediction of `HOMO` and `ZPVE`, `DeepHyper` yields better hyperparameter combinations, which accelerate convergence and provide optimal performance.

### 4.2. AI model training

We trained `PhysNet`, `SchNet`, `MPNN` and `MPNN-transformer` (with attention mechanism) with `HOMO` and `ZPVE` as target properties from the `QM9` dataset. The models were trained for 1500 epochs to ensure convergence of validation loss. A new model is saved when the validation loss drops. The model training

**Table 1.** List of hyperparameters and their ranges.

| Hyperparameter | Log scale | Range |
|---|---|---|
| `agb` | True | [1, 20] |
| `amp` | False | [True, False] |
| `batch size` | / | [128, 512] |
| `epochs` | True | [10, 100] |
| `gradient clip` | True | $[1 \times 10^{-05}, 2]$ |
| `learning rate` | True | $[1 \times 10^{-3}, 1]$ |
| `optimizer` | / | [SGD, TorchAdam, Adam, LAMB] |
| `weight decay` | True | $[2 \times 10^{-6}, 0.02]$ |

**Table 2.** Top ten `DeepHyper` hyperparameter combinations for `PhysNet` with `ZPVE` as target property.

| `agb` | `amp` | `batch_size` | `gradient_clip` |
|---|---|---|---|
| 4 | TRUE | 190 | 0.002 45 |
| 3 | TRUE | 190 | 0.000 22 |
| 1 | TRUE | 397 | 0.000 32 |
| 4 | TRUE | 196 | 0.001 22 |
| 3 | TRUE | 174 | $1.60 \times 10^{-05}$ |
| 4 | TRUE | 154 | $3.25 \times 10^{-05}$ |
| 4 | TRUE | 300 | 0.732 |
| 4 | TRUE | 228 | 0.003 89 |
| 2 | FALSE | 359 | 0.725 37 |
| 11 | TRUE | 168 | 0.002 43 |

**Table 3.** As table 2 for the rest of parameters optimized through `DeepHyper`.

| `learning_rate` | `optimizer` | `weight_decay` | `objective` |
|---|---|---|---|
| 0.002 96 | torch_adam | $1.03 \times 10^{-05}$ | −0.9226 |
| 0.751 69 | torch_adam | $5.14 \times 10^{-06}$ | −6.7526 |
| 0.000 15 | lamb | $2.69 \times 10^{-06}$ | −6.7925 |
| 0.322 74 | lamb | $1.45 \times 10^{-05}$ | −7.1168 |
| 0.176 73 | lamb | $7.80 \times 10^{-06}$ | −12.31 |
| 0.001 44 | torch_adam | $1.13 \times 10^{-05}$ | −15.394 |
| 0.029 86 | lamb | $3.57 \times 10^{-06}$ | −26.13 |
| 0.009 66 | torch_adam | $7.16 \times 10^{-06}$ | −27.627 |
| 0.024 91 | sgd | $1.04 \times 10^{-05}$ | −29.335 |
| 0.001 24 | torch_adamw | 0.000 11 | −30.203 |

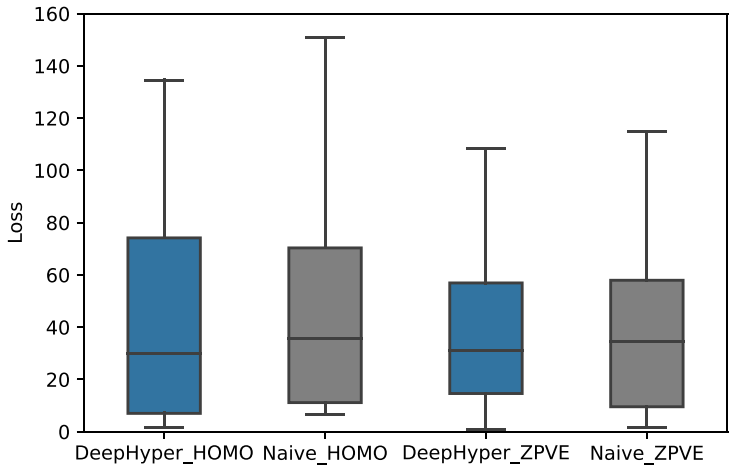


**Figure 1.** Comparison of the loss distributions of `PhysNet` with hyperparameters found by `DeepHyper` (blue) and a naive random selection algorithm (gray). Two outliers for the `DeepHyper_HOMO` box were neglected to retain details.

**Table 4.** Performance of 20 models with hyperparameters found by `DeepHyper` and a naive random selection algorithm with `HOMO` and `ZPVE` as target properties. For both properties, the minimum loss and the standard deviation of loss are reported.

| | HOMO | | ZPVE | |
|---|---|---|---|---|
| | min_loss | std_loss | min_loss | std_loss |
| DeepHyper | 1.604 | 74.871 | 0.923 | 31.771 |
| Naive Algorithm | 6.632 | 52.43 | 1.751 | 65.117 |

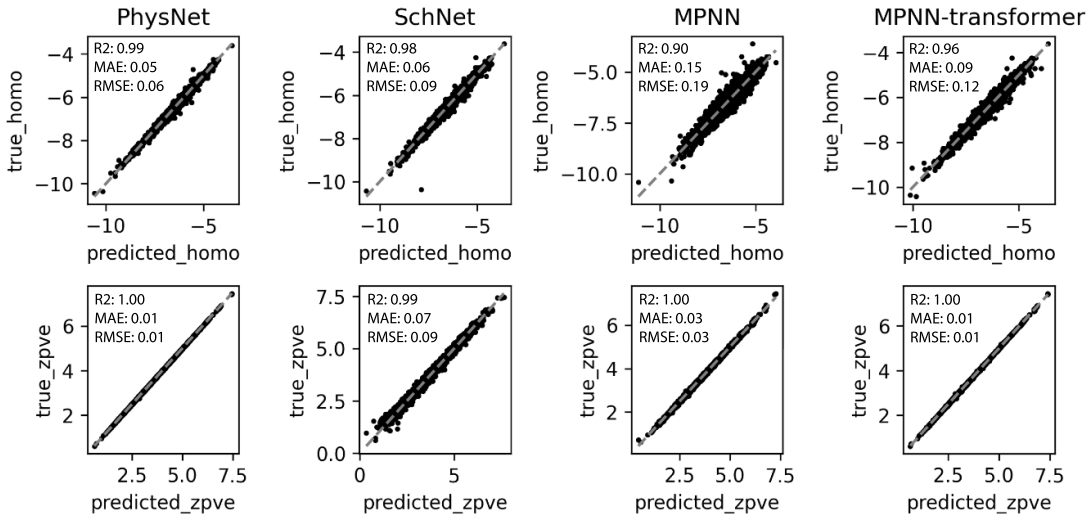


**Figure 2.** From left to right, model inference performance of `PhysNet`, `SchNet`, `MPNN` and `MPNN-transformer` with `HOMO` (top row) or `ZPVE` (bottom row) as the target property.

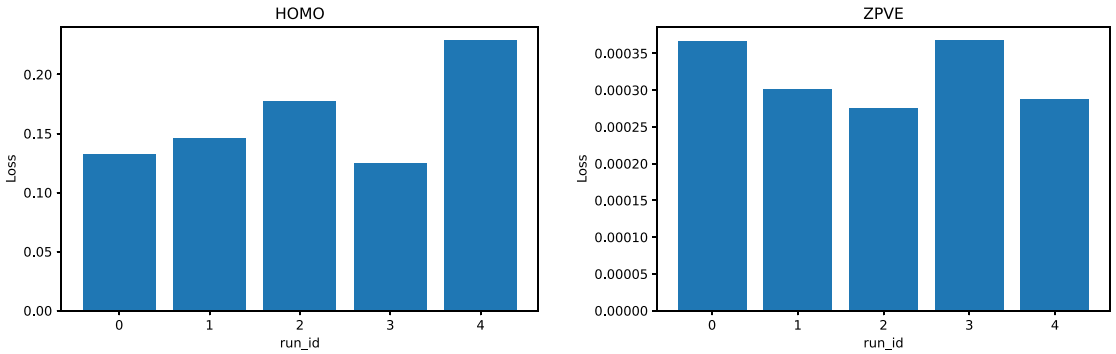


**Figure 3.** Model training performance of `PhysNet` with `HOMO` (left) and `ZPVE` (right) as target properties, initialized with five random seeds.

results are summarized in figure 2. We found that `ZPVE` is an easier property to learn compared to `HOMO` for all four models, as indicated by significantly lower losses. Moreover, the addition of attention layer in the `MPNN` model (`MPNN-transformer`) further lowers the mean absolute error (0.09 eV for `HOMO` and 0.01 eV for `ZPVE`) compared to the original `MPNN` model (0.15 eV for `HOMO` and 0.03 eV for `ZPVE`).

Model uncertainty quantification was performed for `PhysNet` model with `HOMO` and `ZPVE` as target properties. Five `PhysNet` models with randomly initialized weights were generated using the random seeds method. The optimal hyperparameter combinations found by `DeepHyper` in section 4.1 were used. The models were trained for 100 epochs to achieve convergence of loss function. Figure 3 shows that `PhysNet` makes consistent predictions regardless of the random initial weights. The standard deviations of losses for the five models with `HOMO` and `ZPVE` as target properties are 0.0379 eV and $3.9646 \times 10^{-05}$ eV, respectively, and the mean absolute errors are comparable with those reported in the literature [42].

*Key findings:* Our suite of AI models provide state-of-the-art results. Novel features that we added to the models, such as attention in `MPNN-transformer`, further improve model performance. We have also demonstrated that hyperparameter optimization leads to stable, statistically robust AI predictions.

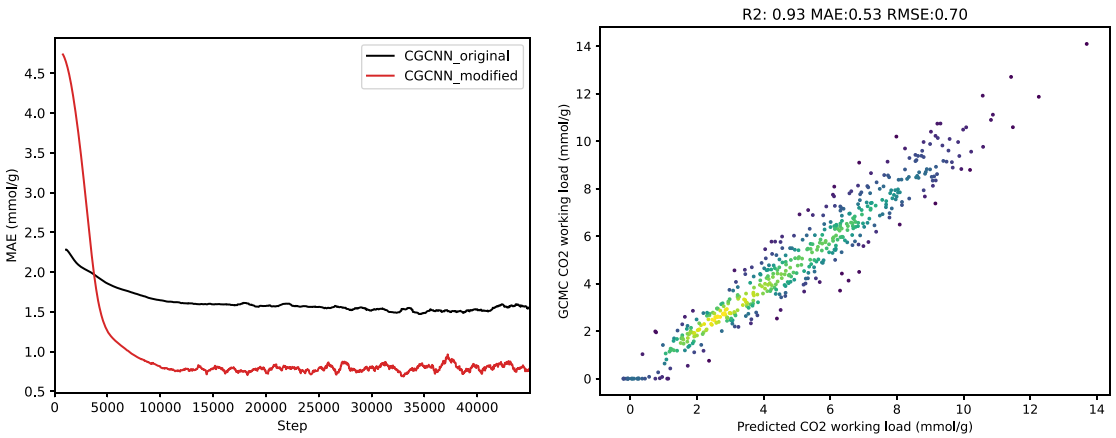


**Figure 4.** Left panel, comparison of the original CGCNN model (black) with adjacency matrix format node and edge embedding schemes and the modified CGCNN model (red) with adjacency list format node and edge embedding schemes. Right panel, inference performance of the modified CGCNN model on a test set of 500 MOFs of the hMOF dataset.

### 4.3. Model improvement via modified node and edge embedding schemes

We modified the node and edge embedding schemes of CGCNN model from the original adjacency matrix format to an adjacency list format. There are two main advantages in using the adjacency list format. First, compared to the adjacency matrix format, it takes up less memory for loading, which speeds up model training. Second, the redundant information (zero paddings) in the representation is removed, resulting in higher training accuracy and stability. As an example, CGCNN models with the two embedding schemes were trained on a subset of the hMOF database [43], which contains 5000 randomly selected MOF structures along with their $CO_2$ working capacities at 2.5 bar. The 5000 MOFs are splitted into 80% training set, 10% validation set and 10% test set. The mean absolute error on the test set as a function of training steps for both models are shown in left panel of figure 4. To smooth out local fluctuations, thirty point moving averaging was performed on both curves. We notice that the modified CGCNN model achieved faster convergence speed, higher training stability, and a lower mean absolute error compared to the original model.

From the right panel of figure 4, we show that the modified CGCNN model predicts $CO_2$ working capacity with an $R^2$ score of 0.93 and a mean absolute error of 0.53 mmol $g^{-1}$. To better understand the predictive performance of the modified CGCNN model, we benchmarked it against two recently proposed machine learning models for predicting $CO_2$ working capacity of MOFs, namely ALIGNN [44] and random forest regressor [45].

When trained on the entire hMOF dataset, ALIGNN predicts $CO_2$ working capacity at 2.5 bar with a mean absolute error of 0.48 mmol $g^{-1}$ [44]. ALIGNN uses both normal graph and line-graph embedding schemes for training and inference of working capacity prediction. For a normal graph, it uses physical and chemical features for node embedding, and distances between atoms as edge embedding; for a line graph, distances correspond to nodes whereas bond angles correspond to edges. This scheme, however, can cause occasional CUDA memory issues and training batch size may have to be reduced (64 in [44]; 32 in our independent experiment), hence slower training. On the other hand, our modified CGCNN model only takes in crystal structures as input (i.e. atomic species and Cartesian coordinates) with larger batch size (256 in our model).

The random forest regression model takes in topological, structural, and word embedding features as input. When trained on the entire hMOF dataset, it achieves an $R^2$ and root mean squared error (RMSE) score of roughly 0.95 and 0.65, respectively, for the prediction of $CO_2$ working capacity at 2.5 bar. We show some of the predictions of our modified CGCNN model, and compare them to ground truth carbon dioxide adsorption values, in figure 5.

In other papers, extensive physical and chemical featurization schemes were used [46, 47] to feature the MOF structures, whereas our modified CGCNN model captures MOF information solely from atom species and coordinates.

It is worth noting that we trained the modified CGCNN model on 5000 randomly selected MOFs instead of all of the structures, therefore lower prediction error is expected if the model is trained on the entire hMOF dataset. Overall, the modified CGCNN model achieves competitive predictive performance compared to state-of-the-art machine learning models.

*Key findings:* Adopting adjacency list format node and edge embedding scheme improves the predictive capabilities of our modified CGCNN model. When making inference on a test set of 500 MOFs from the hMOF

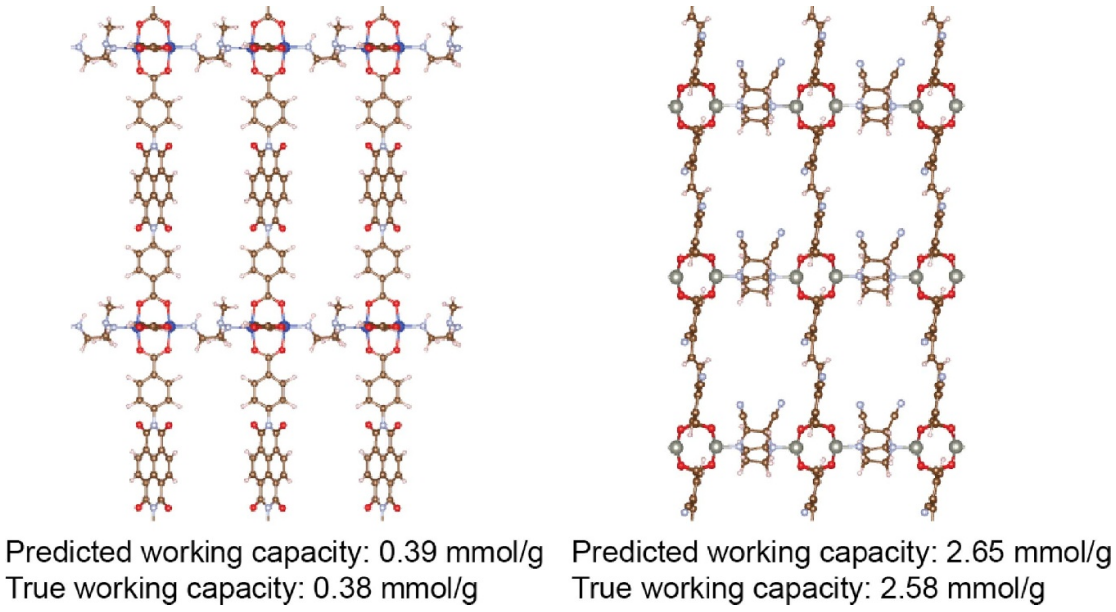


**Figure 5.** Sample MOF structures in the `hMOF` database along with the AI predicted and ground truth $CO_2$ working capacities at 2.5 bar.

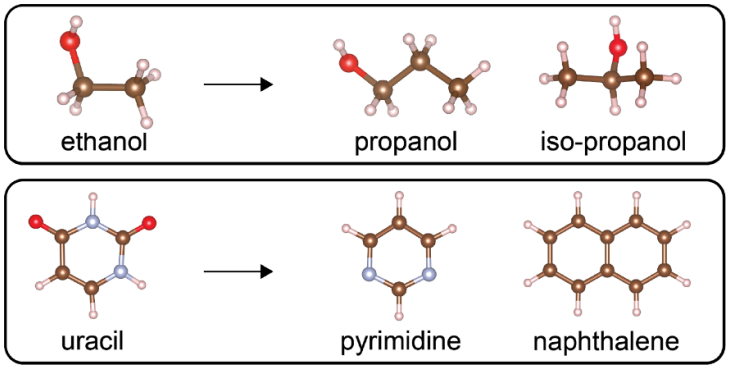


**Figure 6.** Example molecules used to demonstrate the transferability of `TorchMD-NET`. MD trajectories of molecules on the left are used to train the `TorchMD-NET` models, which are then used to perform MD simulations for the molecules on the right.

dataset, we have found that our modified `CGCNN` model provides state-of-the-art predictions for $CO_2$ working capacities at 2.5 bar.

### 4.4. Transferable AI applications

MD simulations of two sets of small molecules were performed to demonstrate the transferability of `TorchMD-NET`: from ethanol to *n*-propanol/*iso*-propanol, and from uracil to pyrimidine/naphthalene.

In figure 6, within each set, the `TorchMD-NET` model trained with MD trajectories of the molecule on the left was used to perform MD simulations of the molecules on the right. The NVE ensemble was used, where the total number of particles in the simulation box and the box volume are fixed, and the total energy is conserved. For all molecules, the timestep and the total simulation time were chosen to be 0.1 fs and 10 ps (100 000 timesteps), respectively. Figure 7 shows that the C–C and C–O bond length distributions of ethanol, propanol and *iso*-propanol have similar means, whereas the latter two have a larger spread. It is worth noting that for propanol, the C–C bond length closer to the oxygen atom has a similar distribution to that of ethanol, which is expected because their local environments are similar. For bond angles, The C–C–O bond angle distribution of ethanol exhibits two peaks, whereas the other two only have one peak. For uracil, pyrimidine and naphthalene, the C–C bond length distribution of naphthalene is shifted to a higher range compared to the other two, which may be due to the absence of N atoms in its ring structure. The C–N bond length distributions of uracil and pyrimidine have similar means, whereas the latter has a larger spread. Similarly, we observe comparable C–C–C bond angle distributions in uracil and pyrimidine, whereas the same

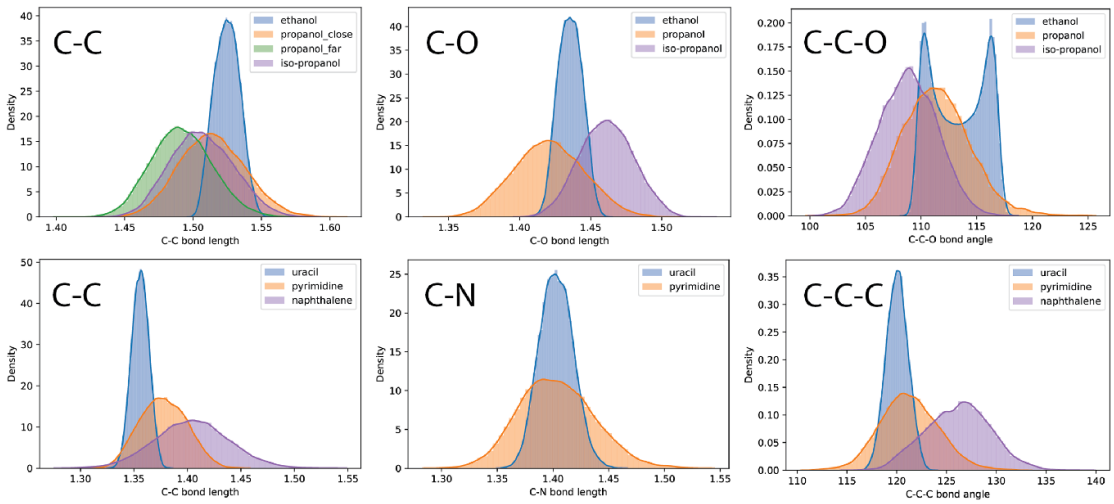


**Figure 7.** Top row, distributions of select bond lengths and angles in ethanol, *n*-propanol, and *iso*-propanol. Bottom row, distributions of select bond lengths and angles in uracil, pyrimidine, and naphthalene. Propanol_close and propanol_far of C–C bond correspond to bonds that are close and far away from the oxygen atom in propanol.

distribution for naphthalene is shifted to a higher range, again an effect which may be due to the absence of N atoms in naphthalene's ring structure. The similarity and differences of bond length and angle distributions demonstrate that `TorchMD-NET` trained on one type of molecule is transferable to other similar molecules.

*Key findings:* We present a novel application of `TorchMD-NET`, in which this AI model was fine-tuned to describe a given small molecule by accurately predicting its potential energy and forces and perform NVE MD simulations, and then seamlessly used to describe other molecules with different structures, while still capturing physically realistic bond length and angle distributions.

### 4.5. Interpretable inference

Here we explore the use of explainable AI and dimension reduction techniques. This is motivated by recent studies in which explainable AI approaches, such as Excitation Backpropagation [17], CAM [18] and Grad-CAM [19] and Contrastive gradient [20], were used to identify key functional groups in molecular structures which contribute to toxicity [16]. Similarly, dimension reduction has been used to visualize the distribution shift of sampled molecules in the feature space with or without transfer learning [21]. We explore the use of these tools to gain new insights into the information that our AI suite extracts from input data to produce reliable predictions.

**Model performance attribution.** By projecting the second last layer's high dimensional vector representation of graph neural network onto molecular structure and visualizing the projection, we can better understand the physical and chemical properties of the input data that affect predictions of our AI models. The top panels of figure 8 present model interpretation results of how `PhysNet` model predicts `HOMO` based on molecular structures via the Grad-CAM method. The N atoms and the H atoms connected to them are highlighted, possibly indicating that for `PhysNet`, these atoms carry more weight in the prediction of `HOMO`. We do not claim that explanations found by our deep learning model are the definite reasons for accurate prediction of QM properties such as `HOMO` or `ZPVE`, since these QM properties may not be simply determined by atomic species and coordinates. However, AI-explained visualization can help us better make sense of the patterns of complex molecular property predictions.

**Dimension reduction.** To reveal the correlation between model features and target properties, we applied the UMAP dimension reduction technique to find the distributions of small molecules by projecting their high-dimensional structural and chemical data onto low-dimension spaces. UMAP has been shown to achieve comparable or even better performance than other dimension reduction techniques such as principal component analysis [48] and t-distributed stochastic neighbor embedding [49] on non-linear datasets [50]. Dimension reduction results for `PhysNet` model with `HOMO` as target property is shown in the bottom panel of figure 8, where each dot in the plot represents a molecule, color-coded based on its `HOMO` value. A 10% randomly selected subset (13.4 k molecules) of the `QM9` dataset was used to produce these results, which

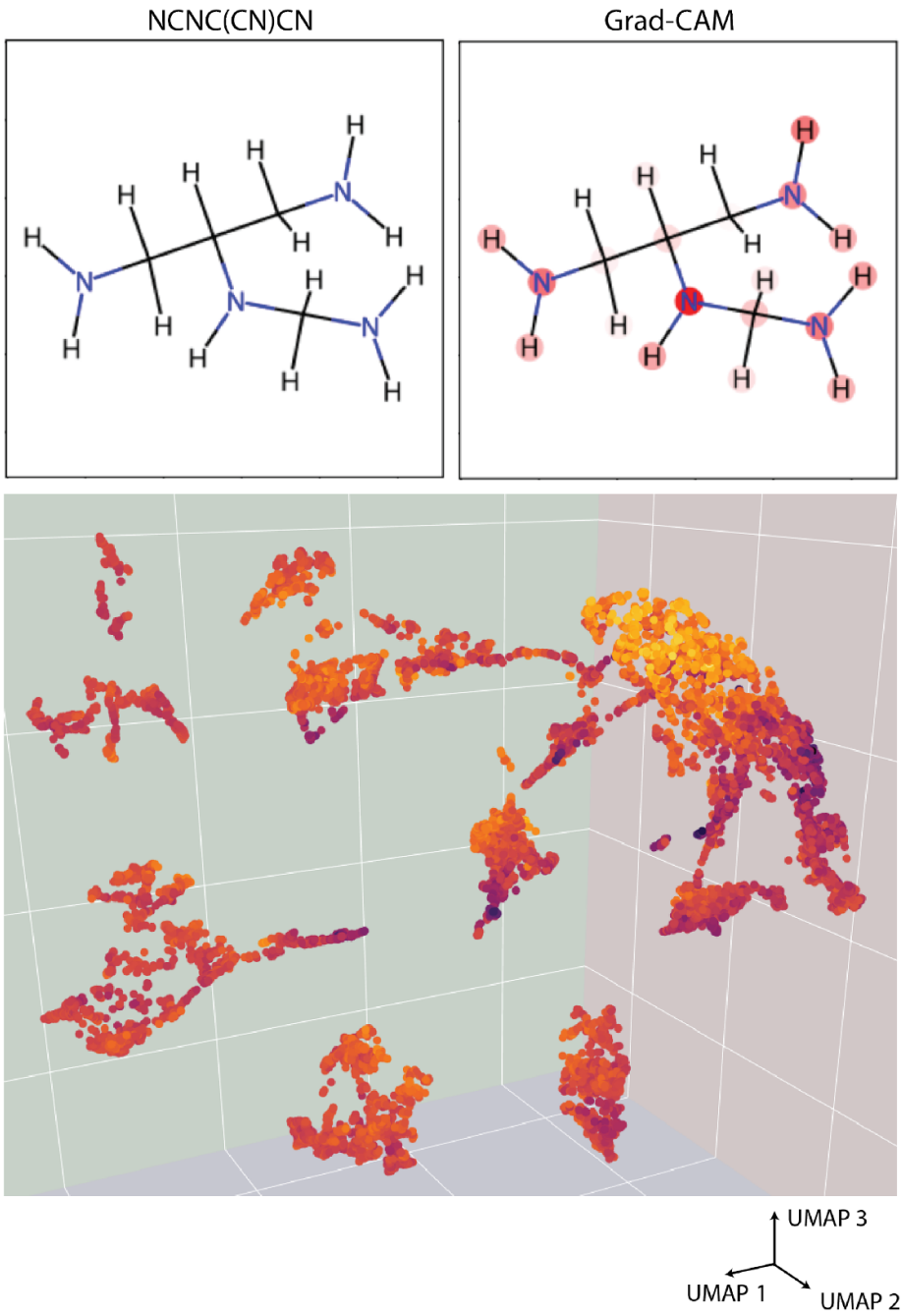


**Figure 8.** Top panels: exploration of the `PhysNet` model prediction using Grad-CAM method. The N atoms and the H atoms connected to them are highlighted in red, indicating that they carry more weight in `HOMO` prediction. Bottom panel: UMAP dimension reduction results for the `PhysNet` model with `HOMO` as target property. Gold and purple dots represent molecules with high and low `HOMO` values, respectively.

consists of stable small molecules composed of CHONF. From the scatter plot we know that molecules with similar `HOMO` values are clustered together and there is clear separation of molecules with low and high `HOMO` values. We present additional illustrative results in appendix B.

**Roughness of molecular property landscape.** For molecular property prediction, the predictive performance of graph neural networks has been shown to correlate to the roughness of molecular property landscape [51–53]. We adopted the recently proposed state-of-the-art roughness index (ROGI) [54] to measure how rough the `HOMO` and `ZPVE` landscapes are for four graph neural network models: `PhysNet`, `SchNet`, `MPNN`, `MPNN-transformer`. The calculation of molecular landscape roughness involves specifying a molecular representation and a distance metric. Molecular representations can be either learned by the graph neural network, with values extracted from the second last layer, or calculated based on molecular structures,

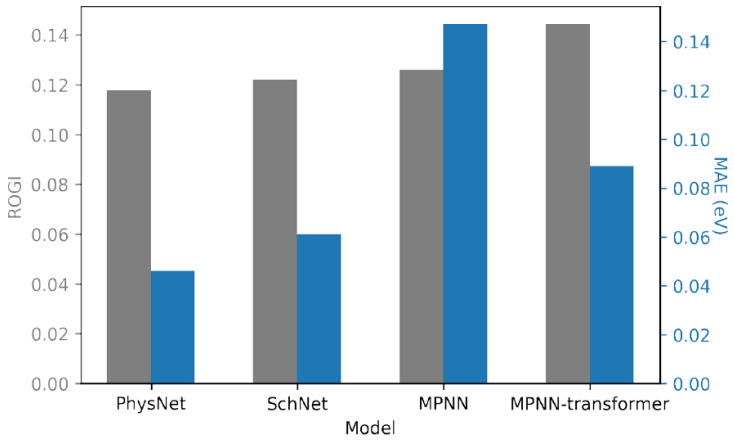


**Figure 9.** Roughness of the `HOMO` landscape (gray) and mean absolute error of model predictions (blue) for `PhysNet`, `SchNet`, `MPNN` and `MPNN-transformer`.

as represented by SMILES strings or 3D Cartesian coordinates. The distance metrics are used to measure how different two molecular representations are. Example distance metrics include Tanimoto similarity, Euclidean distance, cityblock distance and cosine distance. To calculate the ROGI values, learned molecular representation and one of the aforementioned distance metrics are used. Using Euclidean distance as the distance metric and `HOMO` as the target property, we show the ROGI values and mean absolute errors of four graph neural networks in figure 9. We observe that a lower ROGI value in general corresponds to a lower mean absolute error, that is, higher predictive performance. We attribute this trend to the direct relation of a higher performing model to the smoothness of the resulting molecular property landscape. The exception is `MPNN`, which corresponds to a lower ROGI value despite a higher MAE as compared to the `MPNN-transformer`, which may be because the addition of transformer layers roughens the molecular property landscape while facilitating model training.

*Key findings:* Our proposed approach brings together disparate interpretability AI tools to explore and make sense of AI model predictions, encompassing model performance attribution and scientific visualization; dimension reduction with UMAP to explore clustering of molecules with similar properties; and metrics such as the roughness index to quantify the predictive performance of our AI models for QM properties. These complementary tools provide valuable insights into the features and patterns of input data that are relevant for AI inference.

## 5. Conclusion

The rise of AI in the early 2010s was possible by a combination of elements, including disruptive technologies and computing approaches, as well as the desire to advance state-of-the-art practice through collaborative and friendly competitions in which high-quality datasets and AI models were freely shared. Similar approaches have been mirrored in science and engineering in recent years. These efforts are now being formalized through FAIR (findable, accessible, interoperable and reusable) initiatives [55, 56] in the context of scientific datasets [57], research software [58] and AI models [8, 59]. This study represents yet another significant step in this direction. We have assembled benchmark datasets, added novel features to state-of-the-art graph neural networks and transformer models, coupled them with robust libraries for hyperparameter tuning to improve their capabilities for scientific discovery, and developed and adapted a set of visualization and interpretability tools to make sense of the AI predictions. All these elements are unified within a single computational framework that has been deployed and extensively tested on leadership-class, high-performance computing platforms. Researchers using this computational framework will be able to conduct scientific discovery combining state-of-the-art AI models with datasets that are coupled with advanced supercomputing platforms. We expect that this approach will catalyze the sharing of AI knowledge and tools in the context of molecular and crystal property prediction applications.

## Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://doi.org/10.5281/zenodo.7758490.

## Acknowledgments

This work was supported by the FAIR Data program and the Braid project of the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract Number DE-AC02-06CH11357. It used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This work was supported by Laboratory Directed Research and Development (LDRD) funding from Argonne National Laboratory, provided by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-06CH11357. This research used the Delta advanced computing and data resource which is supported by the National Science Foundation (Award OAC 2005572) and the State of Illinois. Delta is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. We thank Prasanna Balaprakash and the `DeepHyper` team for their expert support and guidance as we coupled their library into our computational AI framework.

## Code availability

The AI models presented in this article are open source in GitLab [23]. We also provide a stand alone Colab tutorial [24] that shows users how to use our framework, i.e. loading data and AI models, doing AI inference, and interpreting AI model predictions. Following best practices, the datasets used in these studies are published in Zenodo [25]. We provide an exemplary case of our published AI suite in appendix B.3.

## Appendix A. Hyperparameter optimization results of PhysNet with HOMO as target property

**Table A1.** Top ten `DeepHyper` hyperparameter combinations for `PhysNet` with `HOMO` as target property.

| `agb` | `amp` | `batch_size` | `gradient_clip` |
|---|---|---|---|
| 3 | TRUE | 235 | $1.36 \times 10^{-01}$ |
| 1 | TRUE | 349 | $1.10 \times 10^{+00}$ |
| 14 | FALSE | 130 | $5.40 \times 10^{-05}$ |
| 3 | FALSE | 159 | $1.79 \times 10^{-02}$ |
| 5 | TRUE | 404 | $8.24 \times 10^{-02}$ |
| 4 | TRUE | 460 | $7.21 \times 10^{-02}$ |
| 13 | TRUE | 160 | $3.42 \times 10^{-05}$ |
| 4 | FALSE | 147 | $6.16 \times 10^{-02}$ |
| 7 | TRUE | 163 | $1.90 \times 10^{-01}$ |
| 1 | TRUE | 258 | $3.58 \times 10^{-02}$ |

**Table A2.** As table A1 for the rest of parameters optimized through `DeepHyper`.

| `learning_rate` | `optimizer` | `weight_decay` | `objective` |
|---|---|---|---|
| $1.15 \times 10^{-03}$ | sgd | $2.94 \times 10^{-05}$ | −1.604 |
| $8.69 \times 10^{-04}$ | sgd | $2.30 \times 10^{-06}$ | −2.454 |
| $1.49 \times 10^{-04}$ | lamb | $1.17 \times 10^{-03}$ | −2.976 |
| $5.64 \times 10^{-01}$ | lamb | $2.09 \times 10^{-04}$ | −3.323 |
| $1.38 \times 10^{-03}$ | sgd | $4.83 \times 10^{-05}$ | −6.474 |
| $2.33 \times 10^{-02}$ | sgd | $2.84 \times 10^{-04}$ | −7.214 |
| $4.61 \times 10^{-04}$ | lamb | $1.74 \times 10^{-04}$ | −12.494 |
| $5.99 \times 10^{-04}$ | torch adamw | $1.59 \times 10^{-03}$ | −14.5111 |
| $5.26 \times 10^{-01}$ | lamb | $1.44 \times 10^{-04}$ | −18.0449 |
| $6.95 \times 10^{-01}$ | lamb | $8.62 \times 10^{-05}$ | −29.505 |

## Appendix B. Examples of model explainability features

We present results to complement the interpretable AI analysis presented in section 4.5. Figure B1 illustrates what information AI models may extract from input data to make predictions that are consistent with state-of-the-art knowledge on QM properties.

### B.1. Grad-CAM interpretation

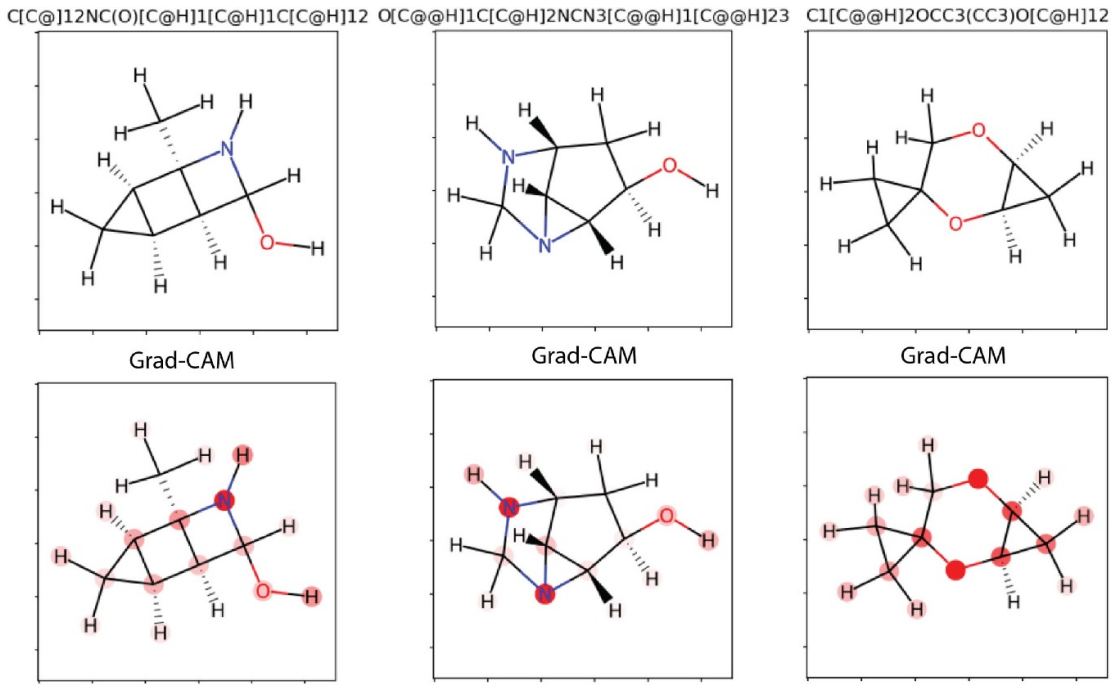


**Figure B1.** Molecular graphs (top row) and Grad-CAM interpretations (bottom row) of `PhysNet` with `HOMO` as target property for three example molecules. The N atoms and the H atoms attached to them are highlighted in red, indicating that they carry more weight in model predictions.

### B.2. UMAP interpretation

Figure B2 shows that we can turn our AI predictors into feature extractors to explore clustering of molecules with similar properties.

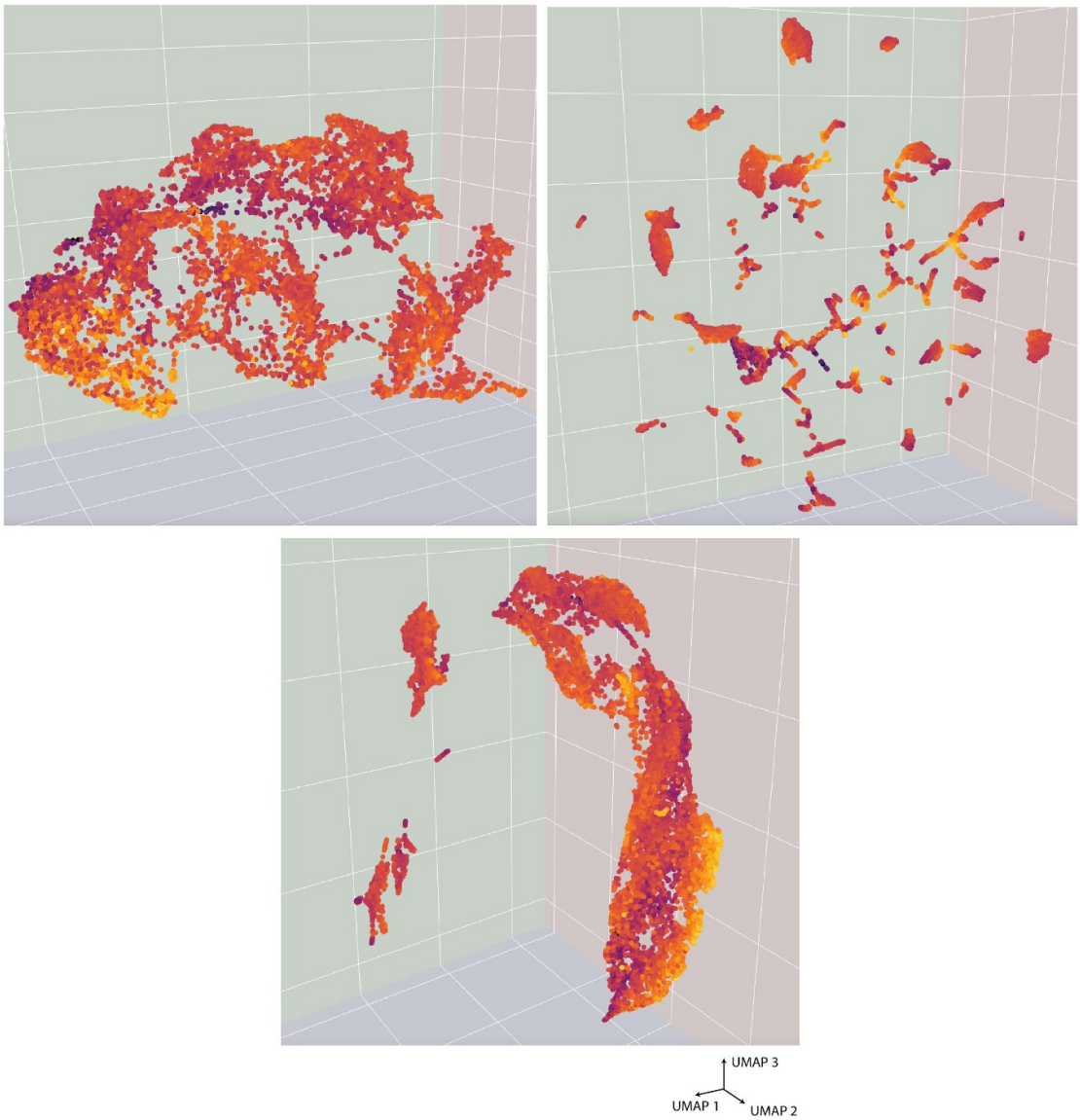


**Figure B2.** UMAP dimension reduction results for `SchNet` (top left panel), `MPNN` (top right panel) and `MPNN-transformer` (bottom panel) with `HOMO` as target property. A randomly selected 10% subset (13.4 k molecules) of the `QM9` dataset was used for analysis, which consists of stable small organic molecules composed of CHONF.

### B.3. Example code

Here we provide an exemplar of our released AI suite [24]. The scientific software below shows, step-by-step, how to download the `hMOF` dataset that we published in Zenodo [25], and then how to use our `CGCNN` model to infer $CO_2$ adsorption for a variety of MOF structures.

**Listing 1.** Exemplar of our step-by-step tutorials, published in [24], which shows how to combine the `hMOF` dataset and our `CGCNN` model to compute $CO_2$ adsorption for a variety of MOF structures.

```
1
2 #Below pip install all necessary packages
3 !pip install torch torchvision torchaudio --extra-index-url https://download.
     pytorch.org/whl/cu116
4 !pip install pyg_lib torch_scatter torch_sparse torch_cluster torch_spline_conv
     torch_geometric -f https://data.pyg.org/whl/torch-1.13.0+cu116.html
5 !pip install pytorch-lightning
6 !pip install dgl -f https://data.dgl.ai/wheels/cu116/repo.html
7 !pip install dglgo -f https://data.dgl.ai/wheels-test/repo.html
8 !pip install rdkit
```

```
!pip install einops curtsies p_tqdm transformers pathlib scikit-image argparse wandb cairosvg h5py pynvml jupyter
!pip install persim ripser ray PyCifRW fairscale ase fast-ml
!pip install pymatgen timm captum
!pip install MDAnalysis
!pip install scikit-tda umap-learn plotly dash

#Below import relevant packages
import torch, torch_geometric, torch_cluster, pytorch_lightning, dgl, ray, rdkit, ripser, sklearn, transformers, timm, h5py, cairosvg, ase, os, sys, pymatgen, os

#Below set variable names
ROOT="/content"
os.chdir(ROOT)
DATA_ROOT=os.path.join(ROOT,"data")
CRYSTAL_DATA_ROOT=os.path.join(ROOT, "hMOF/cifs")
SAVE_ROOT=os.path.join(ROOT, "pretrained_models")
os.makedirs(SAVE_ROOT, exist_ok=True)
os.makedirs(DATA_ROOT, exist_ok=True)

#Below get tar-zipped gitlab repository
!wget https://zenodo.org/record/7758490/files/ai4molcryst_argonne-0.0.1.tar.gz?download=1 -O ai4molcryst_argonne-0.0.1.tar.gz
!tar -xvf ai4molcryst_argonne-0.0.1.tar.gz
!ls ai4molcryst_argonne-0.0.1/

#Below move some necessary files
os.chdir("ai4molcryst_argonne-0.0.1")
!mv main/main_pub.py .
!mv main/config_pub.py .
!mv main/dlhub.py .
os.system("mv dlhub.py dlhub.yaml")

#Below import modules and define dictionary
import main_pub
import config_pub
from train.dist_utils import *
from main_pub import *
import yaml

with open("dlhub.yaml","r") as f:
    d = yaml.safe_load(f)
class obj(object):
    """dict to object"""
    def __init__(self, d):
        for k, v in d.items():
            if isinstance(k, (list, tuple)):
                setattr(self, k, [obj(x)if isinstance(x, dict) else x for x in v])
            else:
                setattr(self, k, obj(v) if isinstance(v, dict) else v)
opt = obj(d)

#Below download hMOF dataset
os.chdir(ROOT)
!wget https://zenodo.org/record/7758490/files/cifs.zip?download=1 -O cifs.zip
!unzip cifs.zip
os.chdir("ai4molcryst_argonne-0.0.1")
```

```
63
64 #Below download pretrained CGCNN model from Zenono
65 !mkdir pretrained_models
66 !wget https://zenodo.org/record/7758490/files/cgcnn_pub_hmof_5000.pth?
     download=1 -O $SAVE_ROOT/cgcnn_pub_hmof_5000.pth
67 ckpt = torch.load(os.path.join(SAVE_ROOT, 'cgcnn_pub_hmof_5000.pth'),
     map_location=torch.device('cpu'))
68 model = config_pub.BACKBONES["cgcnn"](**config_pub.BACKBONE_KWARGS["cgcnn"])
69 print(model.__class__.__name__)
70 model.load_state_dict(ckpt['model'])
71
72 #Below define dataloader
73 opt.log = False
74 opt.backbone = "cgcnn"
75 opt.gpu = True
76 opt.name = "cgcnn_pub_hmof_5000"
77 opt.epoches = 1000
78 opt.batch_size = 16
79 opt.optimizer = "torch_adam"
80 opt.data_dir = DATA_ROOT
81 opt.data_dir_crystal = CRYSTAL_DATA_ROOT
82 opt.use_tensors = True
83 opt.load_ckpt_path = SAVE_ROOT
84 opt.dataset = "cifdata"
85 opt.task = "homo"
86 opt.which_mode = "infer"
87 opt.smiles_list = None
88 opt.crystal = True
89 opt.num_oversample = 0
90 train_loader, val_loader, test_loader, mean, std = call_loader(opt)
     #Loader and data sets
91
92 #Below define inference
93 import tqdm
94 def one_inference(dataloader: torch.utils.data.DataLoader):
95     device="cpu" #torch.cuda.current_device()
96     model.to(device)
97     model.eval()
98     return_preds = collections.defaultdict(list)
99     props, diffs, targs = [], [], []
100
101     for pack, _in tqdm.tqdm(dataloader, total=len(dataloader)):
102         atom_fea, nbr_fea, nbr_fea_idx, crystal_atom_idx, batch, dists,
      targetE = pack.x, pack.edge_attr, pack.edge_index, pack.cif_id,
      pack.batch, pack.edge_weight, pack.y
103         pack = atom_fea, nbr_fea, nbr_fea_idx, batch, dists, targetE
104
105         atom_fea, nbr_fea, nbr_fea_idx, batch, dists, targetE = to_
     cuda(pack)if device !="cpu"else pack
106         pred=model.forward(atom_fea.to(device), nbr_fea.to(device),
     nbr_fea_idx.to(device), dists.to(device), crystal_atom_idx.to(device),
     batch.to(device))
107         props.append(pred.view(-1).detach().cpu().numpy()) targs.
     append(targetE.view(-1).detach().cpu().numpy()) diffs.append(pred.view(-1).
     detach().cpu().numpy() - targetE.view(-1).detach().cpu().numpy())
108
109     return_preds["property"] = np.concatenate(props, axis=0)
110     return_preds["targets"] = np.concatenate(targs, axis=0)
111     return_preds["property_difference"] = np.concatenate(diffs, axis=0)
```

```
112     return return_preds
113
114 #Below run inference
115 def run_inference(input_data):
116     predictions: dict = one_inference(input_data)
117     return predictions
118 preds = run_inference(test_loader)
```

## ORCID iDs

Hyun Park https://orcid.org/0000-0001-5550-5610
Ruijie Zhu https://orcid.org/0000-0001-9316-7245
E A Huerta https://orcid.org/0000-0002-9682-3604
Santanu Chaudhuri https://orcid.org/0000-0002-4328-2947
Emad Tajkhorshid https://orcid.org/0000-0001-8434-1010
Donny Cooper https://orcid.org/0000-0002-2432-972X

## References

[1] Schütt K T, Sauceda H E, Kindermans P J, Tkatchenko A and Müller K R 2018 *J. Chem. Phys.* **148** 241722
[2] Unke O T and Meuwly M 2019 *J. Chem. Theory Comput.* **15** 3678–93
[3] Thölke P and De Fabritiis G 2022 arXiv:2202.02541
[4] Klicpera J, Groß J and Günnemann S 2020 arXiv:2003.03123
[5] Xie T and Grossman J C 2018 *Phys. Rev. Lett.* **120** 145301
[6] Liu M *et al* 2021 *J. Mach. Learn. Res.* **22** 1–9
[7] Fung V, Zhang J, Juarez E and Sumpter B G 2021 *npj Comput. Mater.* **7** 1–8
[8] Ravi N, Chaturvedi P, Huerta E A, Liu Z, Chard R, Scourtas A, Schmidt K J, Chard K, Blaiszik B and Foster I 2022 *Sci. Data* **9** 657
[9] Huerta E A *et al* 2020 *J. Big Data* **7** 88
[10] Huerta E A *et al* 2021 *Nat. Astron.* **5** 1062–8
[11] Balaprakash P, Salim M, Uram T D, Vishwanath V and Wild S M 2018 DeepHyper: asynchronous hyperparameter search for deep neural networks *2018 IEEE 25th Int. Conf. on High Performance Computing (HiPC)* (IEEE) pp 42–51
[12] Ruddigkeit L, Van Deursen R, Blum L C and Reymond J L 2012 *J. Chem. Inf. Model.* **52** 2864–75
[13] Wilmer C E, Farha O K, Bae Y S, Hupp J T and Snurr R Q 2012 *Energy Environ. Sci.* **5** 9849–56
[14] Chmiela S, Tkatchenko A, Sauceda H E, Poltavsky I, Schütt K T and Müller K R 2017 *Sci. Adv.* **3** e1603015
[15] Biewald L 2020 Experiment tracking with weights and biases software available from wandb.com (available at: www.wandb.com/)
[16] Pope P, Kolouri S, Rostrami M, Martin C and Hoffmann H 2018 arXiv:1812.00265
[17] Zhang J, Bargal S A, Lin Z, Brandt J, Shen X and Sclaroff S 2018 *Int. J. Comput. Vis.* **126** 1084–102
[18] Zhou B, Khosla A, Lapedriza A, Oliva A and Torralba A 2016 Learning deep features for discriminative localization *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* pp 2921–9
[19] Selvaraju R R, Cogswell M, Das A, Vedantam R, Parikh D and Batra D 2017 Grad-CAM: visual explanations from deep networks via gradient-based localization *Proc. IEEE Int. Conf. on Computer Vision* pp 618–26
[20] Tieleman T 2008 Training restricted Boltzmann machines using approximations to the likelihood gradient *Proc. 25th Int. Conf. on Machine Learning* pp 1064–71
[21] Moret M, Friedrich L, Grisoni F, Merk D and Schneider G 2020 *Nat. Mach. Intell.* **2** 171–80
[22] McInnes L, Healy J and Melville J 2018 arXiv:1802.03426
[23] Park H, Zhu R, Huerta E A, Chaudhuri S, Tajkhorshid E and Cooper D 2023 AI Suite for small molecules and inorganic crystals (available at: https://gitlab.com/hyunp2/ai4molcryst_argonne)
[24] Park H, Zhu R, Huerta E A, Chaudhuri S, Tajkhorshid E and Cooper D 2023 Colab AI tutorial for small molecules and inorganic crystals (available at: https://tinyurl.com/49reb4su)
[25] Park H, Zhu R, Huerta E A, Chaudhuri S, Tajkhorshid E and Cooper D 2023 End-to-end AI framework for interpretable prediction of molecular and crystal properties (available at: https://doi.org/10.5281/zenodo.7758490)
[26] Gasteiger J, Becker F and Günnemann S 2021 *Advances in Neural Information Processing Systems* vol 34 pp 6790–802
[27] Liu Y, Wang L, Liu M, Lin Y, Zhang X, Oztekin B and Ji S 2022 Spherical message passing for 3D molecular graphs *Int. Conf. on Learning Representations*
[28] Wang L, Liu Y, Lin Y, Liu H and Ji S 2022 ComENet: towards complete and efficient message passing for 3D molecular graphs *Advances in Neural Information Processing Systems* ed A H Oh, A Agarwal, D Belgrave and K Cho
[29] Choudhary K and DeCost B 2021 *npj Comput. Mater.* **7** 1–8
[30] Chen C, Ye W, Zuo Y, Zheng C and Ong S P 2019 *Chem. Mater.* **31** 3564–72
[31] Gilmer J, Schoenholz S S, Riley P F, Vinyals O and Dahl G E 2020 *Machine Learning Meets Quantum Physics* (Berlin: Springer) pp 199–214
[32] Schlichtkrull M, Kipf T N, Bloem P, Van Den Berg R, Titov I and Welling M 2018 Modeling relational data with graph convolutional networks *The Semantic Web: 15th Int. Conf., ESWC 2018* (*Heraklion, Crete, Greece, 3–7 June 2018*) vol 15 (Springer) pp 593–607
[33] Larsen A H *et al* 2017 *J. Phys.: Condens. Matter* **29** 273002
[34] Wang M *et al* 2019 arXiv:1909.01315
[35] Fey M and Lenssen J E 2019 Fast graph representation learning with PyTorch geometric *ICLR Workshop on Representation Learning on Graphs and Manifolds*
[36] Leow Y Y, Laurent T and Bresson X 2019 GraphTSNE: a visualization technique for graph-structured data *ICLR Workshop on Representation Learning on Graphs and Manifolds*
[37] Gelman S, Fahlberg S A, Heinzelman P, Romero P A and Gitter A 2021 *Proc. Natl Acad. Sci.* **118** e2104878118

[38] Mnih V et al 2015 Nature 518 529–33
[39] Simonyan K, Vedaldi A and Zisserman A 2013 arXiv:1312.6034
[40] Pope P E, Kolouri S, Rostami M, Martin C E and Hoffmann H 2019 Explainability methods for graph convolutional neural networks Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition pp 10772–81
[41] Meta Platforms, Inc. 2021 Papers with code—gradient clipping explained (available at: https://paperswithcode.com/method/gradient-clipping)
[42] Glavatskikh M, Leguy J, Hunault G, Cauchy T and Da Mota B 2019 J. Cheminform. 11 1–15
[43] Bucior B J, Rosen A S, Haranczyk M, Yao Z, Ziebel M E, Farha O K, Hupp J T, Siepmann J I, Aspuru-Guzik A and Snurr R Q 2019 Cryst. Growth Des. 19 6682–97
[44] Choudhary K, Yildirim T, Siderius D W, Kusne A G, McDannald A and Ortiz-Montalvo D L 2022 Comput. Mater. Sci. 210 111388
[45] Krishnapriyan A S, Montoya J, Haranczyk M, Hummelshøj J and Morozov D 2021 Sci. Rep. 11 1–11
[46] Burner J, Schwiedrzik L, Krykunov M, Luo J, Boyd P G and Woo T K 2020 J. Phys. Chem. C 124 27996–8005
[47] Moosavi S M et al 2022 Nat. Mater. 21 1419–25
[48] Jolliffe I T and Cadima J 2016 Phil. Trans. R. Soc. A 374 20150202
[49] Van der Maaten L and Hinton G 2008 J. Mach. Learn. Res. 9 2579–605
[50] Wang Y, Huang H, Rudin C and Shaposhnik Y 2021 J. Mach. Learn. Res. 22 1–73
[51] Peltason L and Bajorath J 2007 J. Med. Chem. 50 5571–8
[52] Guha R and Van Drie J H 2008 J. Chem. Inf. Model. 48 646–58
[53] Golbraikh A, Muratov E, Fourches D and Tropsha A 2014 J. Chem. Inf. Model. 54 1–4
[54] Aldeghi M, Graff D E, Frey N, Morrone J A, Pyzer-Knapp E O, Jordan K E and Coley C W 2022 J. Chem. Inf. Model. 62 4660–71
[55] Wilkinson M D, Sansone S A, Schultes E, Doorn P, da Silva Santos L O B and Dumontier M 2018 Sci. Data 5 180118
[56] Wilkinson M D et al 2016 Sci. Data 3 160018
[57] Chen Y et al 2022 Sci. Data 9 31
[58] Barker M et al 2022 Sci. Data 9 622
[59] Duarte J et al 2022 arXiv:2212.05081