# OpenChem: A Deep Learning Toolkit for Computational Chemistry and Drug Design

Maria Korshunova,* Boris Ginsburg, Alexander Tropsha, and Olexandr Isayev*

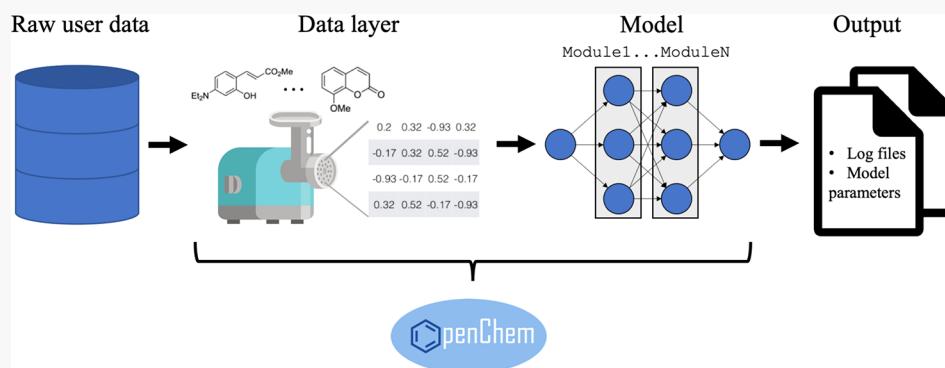Cite This: *J. Chem. Inf. Model.* 2021, 61, 7−13

Read Online

ACCESS | 📊 Metrics & More | 📰 Article Recommendations | Ⓢ Supporting Information



**ABSTRACT:** Deep learning models have demonstrated outstanding results in many data-rich areas of research, such as computer vision and natural language processing. Currently, there is a rise of deep learning in computational chemistry and materials informatics, where deep learning could be effectively applied in modeling the relationship between chemical structures and their properties. With the immense growth of chemical and materials data, deep learning models can begin to outperform conventional machine learning techniques such as random forest, support vector machines, and nearest neighbor. Herein, we introduce OpenChem, a PyTorch-based deep learning toolkit for computational chemistry and drug design. OpenChem offers easy and fast model development, modular software design, and several data preprocessing modules. It is freely available via the GitHub repository.

## ■ INTRODUCTION

Deep Learning is undergoing a rise in various fields of computational chemistry, including chemical reaction design, drug discovery, and material science. Machine learning has been a widely used technique in computational chemistry for the past 70 years but mainly for building models for small molecule activity/property prediction from their chemical descriptors. Known as Quantitative Structure−Activity Relationship/ Quantitative Structure−Property Relationship (QSAR/ QSPR),[1] machine learning approaches have been applied to solving activity or property classification and regression problems. Both types of problems can be solved with deep learning models; however, until recently, chemical data were not big enough to train robust and reliable neural networks. Currently, computational chemistry is entering the age of big data. For instance, a repository of chemical bioactivity data known as PubChem[2] comprises more than 100 million chemical structures, and more than 250 million experimentally measured bioactivities (https://pubchemdocs.ncbi.nlm.nih.gov/ statistics). Thus, the use of deep neural networks to analyze big bioactivity data becomes increasingly justified. Moreover, there are many interesting classical problems in computational chemistry that have never been tackled with machine learning

before the deep learning era. An excellent example of such a problem is the *de novo* generation of molecules with optimized properties. Previously, this problem was attacked with combinatorial methods.[3,4] However, such an approach is not efficient since chemical space is big (with estimates up to $10^{60}$ molecules[5]), and an efficient sampling technique should be "smart", which makes deep learning models a good fit for this problem. Models for *de novo* molecular design require a lot of unlabeled data, which is available at much less cost than labeled data. For example, Enamine REAL database (https://enamine. net/hit-finding/compound-collections/real-database) contains 3.7 billion real organic molecules and can be used to train a deep generative neural network as to how to generate new realistic molecules. There have been various flavors of deep generative models for molecules in multiple representations, with the most
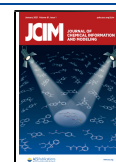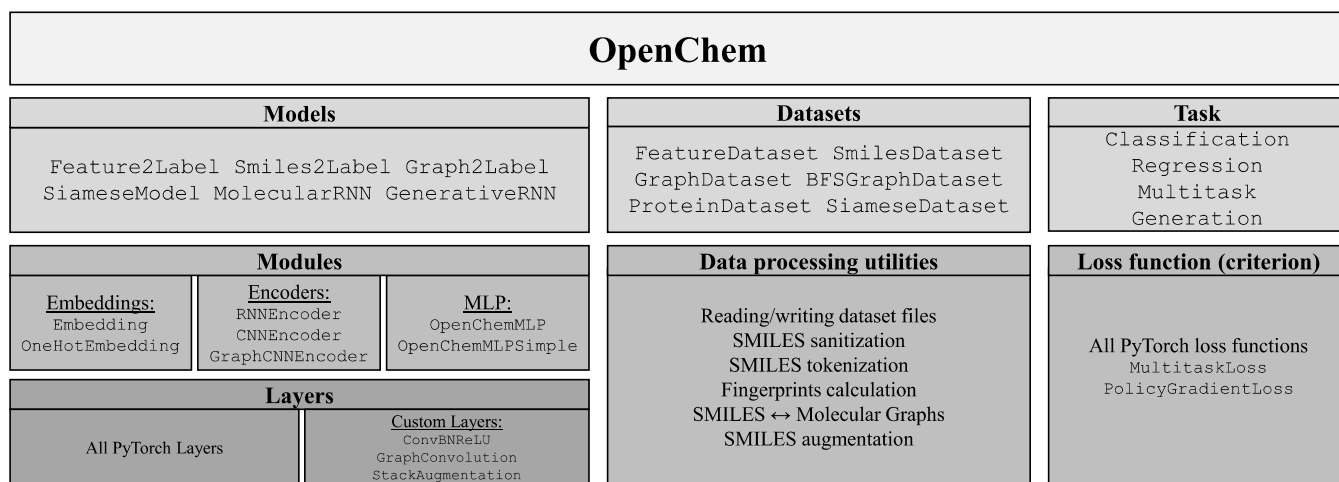
**Figure 1.** Main OpenChem objects.

common ones being SMILES[6] and molecular graphs.[7] There are also multiple property optimization strategies in the literature, such as reinforcement learning, Bayesian optimization, and optimization in the latent space.[8−12]

From a practical perspective, there are two main frameworks used for developing deep learning neural networks, which are PyTorch[13] and Tensorflow.[14] We decided to use PyTorch since it is more suitable for easy experimentation and fast prototyping. In other words, we wanted to build a tool that would let scientists quickly try an idea with as little engineering efforts as possible. Here comes another pitfall for computational chemists, who, in many cases, are not computer scientists or software engineers. Even with focused deep learning libraries such as PyTorch and Tensorflow, building a deep learning network does require an extensive software engineering background. There exist several community-maintained libraries for computational chemistry, such as RDKit,[15] DeepChem (https://deepchem.io), and ATOM Modeling Pipeline[16] which extends DeepChem. RDKit offers functionality for manipulating chemical objects such as atoms, bonds, and molecules. While this functionality is extremely useful for data processing, it is not designed for building machine learning models. Another library we mentioned, DeepChem, is aimed at building deep neural networks for chemistry and built upon Tensorflow. Developing a new model with DeepChem requires writing a lot of Tensorflow code; furthermore, DeepChem does not enable modular design features such as encapsulation and reusability of standard deep neural network blocks, such as encoders, decoders, and embedding layers.

Another critical question is the reproducibility of computational experiments. Frequently, results reported in papers cannot be reproduced by independent researchers. This could happen due to various factors, including the absence of standardized package environments, well-tracked log files, and protocols for reproducing the results, to name a few.

To address the issues discussed above, we developed OpenChem, a deep learning library for computational chemistry built upon the PyTorch framework. OpenChem offers modular design, where building blocks can be combined; ease of use by letting the users define a model with a single config file; and advanced deep learning features such as built-in multi-GPU support. In this application note, we introduce OpenChem design and present three case studies. All data and models to reproduce these examples are available from https://github.com/Mariewelt/OpenChem.

## ■ OPENCHEM DESIGN

OpenChem is a deep learning toolkit for computational chemistry and drug design with a PyTorch backend. The primary purpose of OpenChem is to provide computational chemists with a tool for easy experimentation with deep learning models, i.e., quick implementation of architectures, fast training, debugging, result interpretation, and visualization. The main idea is implementing a toolkit as a set of building blocks with a unified API that will be combined into a single custom architecture by a user.

OpenChem introduces several model types: Feature2Label, Smiles2Label, Graph2Label, SiameseModel, GenerativeRNN, and MolecularRNN. These high-level model types consist of lower level modules, such as embeddings, encoders, and multilayer perceptron. Modules are built from layers, which could be PyTorch or custom layers. Examples of custom layers implemented in OpenChem are graph convolutions, convolutions combined with batch normalization and ReLU, and stack augmentation. Another OpenChem object type is a data set. The OpenChem data set inherits from the PyTorch data set and additionally provides features for converting inputs into tensors for the OpenChem model. OpenChem has a data set for converting SMILES into feature vectors, tokens, and molecular graphs and for converting protein sequences into tokens.

Overall, OpenChem is implemented to offer users a modular design; i.e., blocks with the same input and output formats can be used interchangeably by adjusting the settings in the configuration file. For example, there are several different options for the encoder block, such as the RNN encoder, CNN encoder, or Graph CNN encoder, that could be used to calculate representation vectors for molecules. OpenChem allows choosing these options from the configuration file. OpenChem also supports built-in multi-GPU training and offers several features for logging and debugging. Figure 1 summarizes the types of models, modules, and tasks that are currently implemented in OpenChem. Users can train predictive models for classification, regression, and multitask problems, as well as develop generative models for producing novel molecules with optimized properties. OpenChem can work both with SMILES strings and molecular graphs. Data layers offer utilities for data preprocessing, such as converting SMILES strings to molecular
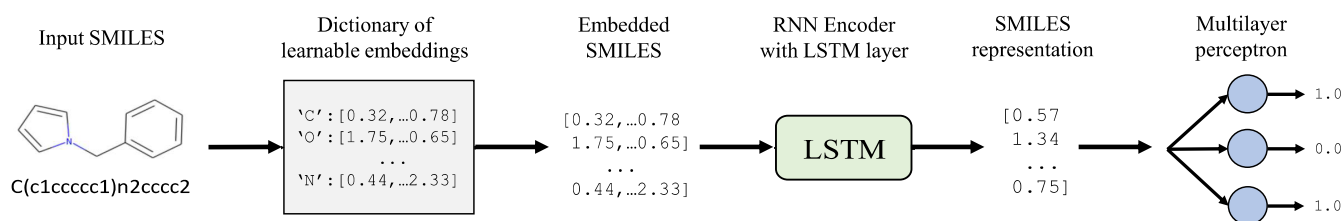
**Figure 2.** Scheme of multitask SMILES2Label model. Input SMILES string is converted to a matrix of embeddings by the dictionary of learnable embeddings. Next, the matrix of embeddings is passed to the RNN encoder with a LSTM layer. The RNN encoder converts the matrix of embedding to a representation vector, which is used by the multilayer perceptron to make predictions for the input SMILES.

graphs and calculating standard structural features for such graphs.

Models in OpenChem are defined in the Python configuration file as a dictionary of parameters. The dictionary must contain parameters that define how to run/train/evaluate a model, as well as parameters defining model architecture. OpenChem provides scripts for model training, and it also natively supports multi-GPU training. After the training process is finished, OpenChem saves model parameters, as well as log files, to a designated folder so that the experiment can be reproduced later.

**Configuration File.** The configuration file must define a model, which should be any class derived from `Open-ChemModel` and dictionary `model_params`, which specifies model hyperparameters. A detailed list of standard training parameters is provided in Table S1.

We consider three use cases, which illustrate how models are defined and used in OpenChem.

**Case Study 1: Graph Convolution Neural Network for Predicting logP.** *Data and Model Description.* In this case study, we built a Graph Convolution Neural Network (GraphCNN)[17] for predicting the n-octanol/water partition coefficient (logP) from molecular graphs. This task is an example of a regression problem, as logP covers a range of continuous values. A detailed model description is provided in the Supporting Information. We used five atomic properties as node attributes: atom type, valence, charge, hybridization, and aromaticity. OpenChem provides a module for declaring an attribute, where a user can specify attribute type (node or edge) and attribute label (e.g., categorical) list all possible values for categorical attributes, and more. The GraphCNN model also requires a user-defined function for calculating node attributes. It is a Python function that receives the RDKit atom object as an input and returns a dictionary of atomic attributes for the input atom.

First, we loaded the logP data set (described below) and split it into train and test subsets. OpenChem provides function `read_smiles_property_file` that accepts the path to the file and indices of the columns to be read from the file as arguments and returns a list where each element is a column from the file. In this example, we loaded the logP_labels.csv file, read columns number 1 and 2, split the data into training and test subsets, and saved the subsets into new files using the `save_smiles_property_file` utility from OpenChem. Next, we created graph data layers from the saved files with train and test subsets. OpenChem provides a `GraphDataset` class, which can convert SMILES strings to molecular graphs and calculate node and edge attributes. The `GraphDataset` also accepts a user-defined dictionary of atomic (node) attributes, such as valency, hybridization, and aromaticity, and functions for computing these attributes. Other parameters to

the `GraphDataset` include a path to the text file with data, a list of columns that will be read from the file, and a column delimiter.

Our model consists of five layers of Graph Convolutions with a hidden size of 128, followed by two layers of multilayer perceptron (MLP) with ReLU nonlinearity and hidden dimensionalities of 128 and 1. We trained the model for 100 epochs with an Adam[18] optimizer with an initial learning rate of 0.01, and a MultiStepLR learning scheduler with step size 15, and gamma factor of 0.5, meaning that the learning rate is decreased by half every 15 epochs of training starting from the initial learning rate of 0.01. For external evaluation, we used $R^2$ score. We also printed the intermediate progress report on the training and evaluation metrics every 10 epochs and saved the model checkpoint every 5 epochs. We specified all these parameters in the dictionary below.

```
model = Graph2Label

model_params = {
    'task': 'regression',
    'random_seed': 42,
    'use_clip_grad': False,
    'batch_size': 256,
    'num_epochs': 101,
    'logdir': 'logs/logp_gcnn_logs',
    'print_every': 10,
    'save_every': 5,
    'train_data_layer': train_dataset,
    'val_data_layer': test_dataset,
    'eval_metrics': r2_score,
    'criterion': nn.MSELoss(),
    'optimizer': Adam,
    'optimizer_params': {
        'lr': 0.0005,
    },
    'lr_scheduler': StepLR,
    'lr_scheduler_params': {
        'step_size': 15,
        'gamma': 0.8
    },
    'encoder': GraphCNNEncoder,
    'encoder_params': {
        'input_size': train_dataset[0]["node_feature_matrix"].shape[1],
        'encoder_dim': 128,
        'n_layers': 5,
        'hidden_size': [128]*5,
    },
    'mlp': OpenChemMLP,
    'mlp_params': {
        'input_size': 128,
        'n_layers': 2,
        'hidden_size': [128, 1],
        'activation': [F.relu, identity]
    }
}
```

*Results.* We trained a GraphCNN for predicting the n-octanol/water partition coefficient (logP) directly from the molecular graph. The modeling data set of 14,500 molecules was obtained based on the public version of the PHYSPROP database.[19] The data set was curated according to our well-established protocol.[20] Structural standardization, cleaning of salts, and removal of mixtures, inorganics, and organometallics were performed using ChemAxon. In the case of replicate compounds, InChI Keys were generated to resolve duplicates. In the case of conflicting property values, the entries were discarded. Using 5-fold cross-validation, we obtained the model accuracy expressed as $R^2 = 0.90$ and root-mean-square error RMSE = 0.56 This is significantly better than traditional QSPR models ($R^2 = 0.86$ and RMSE = 0.78) obtained on the same data set using physicochemical descriptors and E-state indices.[21]

**Case Study 2: Tox21 Challenge.** *Data and Model Description.* In this case study, we built a Recurrent Neural Network (RNN) (Figure 2) for multitask prediction of bioactivity for 12 receptors using data from the Tox21 challenge.[22] This model receives a SMILES string[6] for a ligand as an input and returns a vector of size 12, where each component is interpreted as a probability that the input ligand binds to a corresponding receptor. In other words, multitask allows solving 12 independent binary classification problems with a single model. Since any SMILES string is a sequence of characters, we can use Recurrent Neural Networks to calculate a representation vector for a given molecule.[10] Each symbol of the SMILES string $s_t$ is processed sequentially. At each time step, the model takes a single character $s_t$ from the SMILES string and converts it to a numerical embedding vector $x_t$ with a learnable embedding dictionary layer. Then, $x_t$ is passed to the LSTM layer

$$h_{t+1} = W_x x_t + b_x + W_h h_t + b_h$$

where $h_t$ is the intermediate hidden state for the SMILES prefix of length $t$. When the whole SMILES string is processed, hidden state $h_T$ from the final time step is used as a representation vector for the next feed-forward layer, which outputs the vector of prediction for the input SMILES.

*Defining Model in OpenChem.* The Tox21 data set is available as a benchmark data set, and it can be loaded from the OpenChem GitHub with the `read_smiles_proper-ty_file` function. As Tox21 is a multitarget data set, some of the labels are not available and, therefore, left empty. To account for dummy labels, we used `MultitaskLoss` from Open-Chem, which is a binary cross-entropy loss, averaged across multiple classes. `MultitaskLoss` also does not accumulate losses for dummy labels to the final loss in backpropagation. We filled them with a dummy index that was ignored during training. We also extracted unique tokens from the whole data set before splitting it into training and test subsets to avoid the situation where some of the tokens are not present in one of the pieces of the data set. After this step, we split data randomly into training and test subsets and saved these subsets to new files with the OpenChem `save_smiles_property_file` utility.

Next, we created the SMILES data layer from input files and added data augmentation by SMILES enumeration[23] for the training data set. The idea behind it is to include noncanonical notation for SMILES. Augmentation is enabled by setting the argument `augment=True` when creating an object of class `SmilesDataset`. Since this task is multitarget, we needed to implement a custom evaluation function for calculating

classification accuracy separately for each task. As for accuracy metrics, we used the AUC-ROC averaged across all classes. Next, we defined the model architecture with Smiles2Label modality. This model consists of an Embedding block, a Recurrent Encoder with four LSTM layers, and MLP. We also used dropout with a high probability to enable regularization to avoid model overfitting.

```
model = Smiles2Label

model_params = {
    'task': 'multitask',
    'criterion': MultitaskLoss(ignore_index=9, n_tasks=12).cuda(),
        ...
    'embedding': Embedding,
    'embedding_params': {
        'num_embeddings': train_dataset.num_tokens,
        'embedding_dim': 128,
        'padding_idx': train_dataset.tokens.index(' ')
    },
    'encoder': RNNEncoder,
    'encoder_params': {
        'input_size': 128,
        'layer': "LSTM",
        'encoder_dim': 128,
        'n_layers': 4,
        'dropout': 0.8,
        'is_bidirectional': False
    },
    'mlp': OpenChemMLP,
    'mlp_params': {
        'input_size': 128,
        'n_layers': 2,
        'hidden_size': [128, 12],
        'activation': [F.relu, torch.sigmoid],
        'dropout': 0.0
    }
}
```

*Results.* In this example, we trained a multitask model for predicting biological activity for 12 assays from the Tox21 challenge. The similarity (and dissimilarity) between the tasks is exploited to enrich a model.[24] We obtained a mean AUC of ∼0.84 with the following per target AUC values on the test set:

- NR-AR 0.85
- NR-AR-LBD 0.90
- NR-AhR 0.87
- NR-Aromatase 0.84
- NR-ER 0.76
- NR-ER-LBD 0.82
- NR-PPAR-gamma 0.80
- SR-ARE 0.78
- SR-ATAD5 0.85
- SR-HSE 0.84
- SR-MMP 0.87
- SR-p53 0.86

These results are comparable to the results reported in the literature previously.[22] Our single multitask model approached the accuracy of the winning model,[25] which was a complex ensemble combination of different models and descriptor schemes.

**Case Study 3: Generation of Molecular Graphs with Maximized Melting Temperature.** *Data and Model Description.* In the final case study, we built a MolecularRNN[26] model for the generation of molecular graphs and further optimization of the specific property of the generated molecules. This model generates molecular graphs in an autoregressive manner, i.e., atom by atom. At each time step, the model predicts
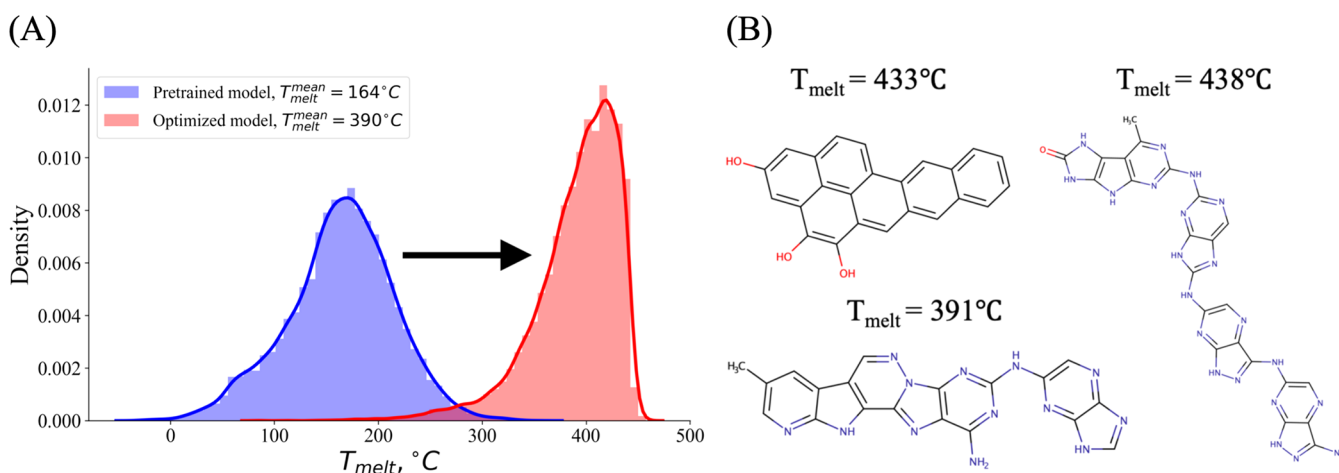
(A)



(B)



**Figure 3.** (A) Distribution shift in melting temperature. (B) Examples of generated molecules with high predicted melting temperatures.

the chemical type of a new atom and the edge connections (including edge types) between the new atom and previously generated ones. Here, we only briefly mention the overall training pipeline. Please see the full example at https://github.com/Mariewelt/OpenChem.

*Unsupervised Pretraining Stage.* At this stage, the model is pretrained on a vast unlabeled data set of real molecules from the ChEMBL database to produce valid and realistic molecules. In other words, during this stage, MolecularRNN learns the distribution over molecular graphs from the training data set. Thus, the generated molecular graphs are similar in structure and properties to known bioactive molecules.

*Property Optimization Stage.* At this stage, we use a policy gradient algorithm[27] to shift the distribution of the generated samples for the desired numerical property. Examples of such properties are solubility, biological activity for the target protein, and toxicity. In this setting, the generative model is used as a policy network. We use an external predictive model to estimate the values of the desired property. This can be a machine learning model (such as neural network built with OpenChem), or it can be any other black box function that accepts molecule as the input and outputs the numerical value of the desired property. Following the policy gradient algorithm, the objective function to be maximized is defined as the expected reward

$$L(\theta) = -\sum_{i=1}^{N} r(s_N) \cdot \gamma^i \cdot \log p(s_i|s_{i-1}; \theta)$$

where $s_N$ is the generated molecular graph, $s_i$, $i = 1, \cdots, N$ is the subgraph of $s_N$ with $0 < i < N$ nodes, $\gamma$ is the discount factor, $p(s_i|s_{i-1}; \theta)$ is the transition probability obtained from the generative model, and $r(s_N)$ is the value of the reward function for the generated molecular graph on the output of the predictive model for the desired property.

*Defining the Model in OpenChem.* The MolecularRNN model was pretrained on the curated ChEMBL24 data set of 1.5 million molecules. We performed the optimization of model parameters to maximize the melting temperature of the generated molecular graphs. The MolecularRNN model had four GRU[28] layers with 256 hidden activations in both NodeRNN and EdgeRNN. The model learns a dictionary of embeddings for nine atoms types (C, N, O, F, P, S, Cl, Br, I) and three bond types (single, double, and triple). We used the kekulized form of molecules according to RDKit[15] which eliminated the need to specify aromatic bonds explicitly.

```
model = GraphRNNModel
model_params = {
    'task': graph_generation,
        ....
    'num_node_classes': num_node_classes,
    'num_edge_classes': num_edge_classes,
    'max_num_nodes': max_num_nodes,
    'start_node_label': start_node_label,
    'max_prev_nodes': max_pred_nodes,
    'label2atom': label2atom,
    'edge2type': edge2type,
    'restrict_min_atoms': restrict_min_atoms,
    'restrict_max_atoms': restrict_max_atoms,
    'max_atom_bonds': max_atom_bonds,
    'EdgeEmbedding': Embedding,
    'edge_embedding_params': {
        'num_embeddings': num_edge_classes,
        'embedding_dim': edge_embedding_dim,
    }
    'NodeEmbedding': Embedding,
    'node_embedding_params': {
        'num_embeddings': num_node_classes,
        'embedding_dim': node_embedding_dim,
    }
    'NodeMLP': OpemChemMLPSimple,
    'node_mlp_params': {
        'input_size': 128,
        'n_layers': 2,
        'hidden_size': [128, num_node_classes],
        'activation': [nn.ReLU(inplace=True),
                identity],
        'init': "xavier_uniform",
    }
    'NodeRNN': GRUPlain,
    'node_rnn_params': {
        'input_size': node_rnn_input_size,
        'embedding_size': 128,
        'hidden_size': 256,
        'num_layers': 4,
        'has_input': True,
        'has_output': True,
        'output_size': 128,
        'has_output_nonlin': False
    }
    'EdgeRNN': GRUPlain,
    'edge_rnn_params': {
        'input_size': edge_embedding_dim,
        'embedding_size': 64,
        'hidden_size': 128,
        'num_layers': 4,
        'has_input': True,
        'has_output': True,
        'output_size': num_edge_classes,
    }
  }
}
```

We also defined structural parameters of the molecular graphs above, such as minimum and the maximum number of nodes and valency constraints for each atom type.

*Results.* Currently, common modeling frameworks do not allow simultaneous ML model building, new compound generation, and property optimization. OpenChem library bridges this gap. Briefly, to optimize melting temperature ($T_{melt}$), we started with our pretrained model and used the policy gradient algorithm. We trained a GraphCNN regression model to predict $T_{melt}$. The model has a RMS error of 39.5 °C that is comparable to the state-of-the-art performance for the same data set of 54,000 molecules.[29] See ref 26 for complete technical details. Figure 3 shows the shift in the distribution of predicted melting temperature before and after optimization with the policy gradient algorithm, as well as examples of the generated molecules with high predicted melted temperature values.

## CONCLUSIONS

Deep learning methods have emerged as a powerful approach for a variety of different tasks, including predictive, discriminative, and generative models. The OpenChem library was created to enable high-performance implementations of deep learning algorithms for drug discovery and molecular modeling applications. Built upon the PyTorch framework, OpenChem is optimized for execution on GPUs and large data sets. One could quickly train ML models from data sets with hundreds of thousands or even millions of data points. OpenChem's modular API allows easy experimentation and fast model prototyping without substantial programming effort. Calculations with OpenChem could be scaled in the Cloud and HPC clusters. It provides well-tracked log files and sharable protocols and models for reproducible results. In this application note, we described just three examples of practical tasks that can be solved with OpenChem. However, the functionality of the proposed framework includes a wide variety of tasks, covering binary, multiclass, and multitask classification; regression and generative modeling; and property optimization. In all three demonstrated examples, we quickly obtained a state-of-the-art performance of models without extensive programming of each model. Our plans include extending the model list and adding new tasks such as message passing neural networks and multiproperty optimization with reinforcement learning.

## ASSOCIATED CONTENT

### Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acs.jcim.0c00971.

    Information as mentioned in the text (PDF)

## AUTHOR INFORMATION

### Corresponding Authors

**Maria Korshunova** − *Computational Biology Department, School of Computer Science and Department of Chemistry, Mellon College of Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, United States;* ◉ orcid.org/ 0000-0003-4391-8228; Email: mariewelt@cmu.edu

**Olexandr Isayev** − *Computational Biology Department, School of Computer Science and Department of Chemistry, Mellon College of Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, United States;* ◉ orcid.org/0000-0001-7581-8497; Email: olexandr@olexandrisayev.com

### Authors

**Boris Ginsburg** − *NVIDIA Corporation, Santa Clara, California 95050, United States*

**Alexander Tropsha** − *UNC Eshelman School of Pharmacy, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina 27599, United States;* ◉ orcid.org/0000-0003-3802-8896

Complete contact information is available at:
https://pubs.acs.org/10.1021/acs.jcim.0c00971

### Notes

The authors declare no competing financial interest.

## REFERENCES

(1) Muratov, E. N.; Bajorath, J.; Sheridan, R. P.; Tetko, I. V.; Filimonov, D.; Poroikov, V.; Oprea, T. I.; Baskin, I. I.; Varnek, A.; Roitberg, A.; Isayev, O.; Curtalolo, S.; Fourches, D.; Cohen, Y.; Aspuru-Guzik, A.; Winkler, D. A.; Agrafiotis, D.; Cherkasov, A.; Tropsha, A. QSAR without Borders. *Chem. Soc. Rev.* **2020**, *49*, 3525.

(2) Wang, Y.; Bryant, S. H.; Cheng, T.; Wang, J.; Gindulyte, A.; Shoemaker, B. A.; Thiessen, P. A.; He, S.; Zhang, J. PubChem BioAssay: 2017 Update. *Nucleic Acids Res.* **2017**, *45* (D1), D955−D963.

(3) Cho, S. J.; Zheng, W.; Tropsha, A. Rational Combinatorial Library Design. 2. Rational Design of Targeted Combinatorial Peptide Libraries Using Chemical Similarity Probe and the Inverse QSAR Approaches. *J. Chem. Inf. Comput. Sci.* **1998**, *38* (2), 259−268.

(4) Gillet, V. J.; Khatib, W.; Willett, P.; Fleming, P. J.; Green, D. V. S. Combinatorial Library Design Using a Multiobjective Genetic Algorithm. *J. Chem. Inf. Comput. Sci.* **2002**, *42*, 375.

(5) Bohacek, R. S.; McMartin, C.; Guida, W. C. The Art and Practice of Structure-Based Drug Design: A Molecular Modeling Perspective. *Med. Res. Rev.* **1996**, *16*, 3.

(6) Weininger, D. SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules. *J. Chem. Inf. Model.* **1988**, *28* (1), 31−36.

(7) Bonchev, D.; Rouvray, D. H. *Chemical Graph Theory: Introduction and Fundamentals*; Abacus Press, 1991.

(8) Zhou, Z.; Kearnes, S.; Li, L.; Zare, R. N.; Riley, P. Optimization of Molecules via Deep Reinforcement Learning. *Sci. Rep.* **2019**, 10752.

(9) Ikebata, H.; Hongo, K.; Isomura, T.; Maezono, R.; Yoshida, R. Bayesian Molecular Design with a Chemical Language Model. *J. Comput.-Aided Mol. Des.* **2017**, *31* (4), 379−391.

(10) Popova, M.; Isayev, O.; Tropsha, A. Deep Reinforcement Learning for de Novo Drug Design. *Sci. Adv.* **2018**, *4*, eaap7885.

(11) Putin, E.; Asadulaev, A.; Vanhaelen, Q.; Ivanenkov, Y.; Aladinskaya, A. V.; Aliper, A.; Zhavoronkov, A. Adversarial Threshold Neural Computer for Molecular *de Novo* Design. *Mol. Pharmaceutics* **2018**, *15*, 4386.

(12) Zhavoronkov, A.; Ivanenkov, Y. A.; Aliper, A.; Veselov, M. S.; Aladinskiy, V. A.; Aladinskaya, A. V.; Terentiev, V. A.; Polykovskiy, D. A.; Kuznetsov, M. D.; Asadulaev, A.; Volkov, Y.; Zholus, A.; Shayakhmetov, R. R.; Zhebrak, A.; Minaeva, L. I.; Zagribelnyy, B. A.; Lee, L. H.; Soll, R.; Madge, D.; Xing, L.; Guo, T.; Aspuru-Guzik, A. Deep Learning Enables Rapid Identification of Potent DDR1 Kinase Inhibitors. *Nat. Biotechnol.* **2019**, *37* (9), 1038−1040.

(13) Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; Facebook, Z. D.; Research, A. I.; Lin, Z.; Desmaison, A.; Antiga, L.; Srl, O.; Lerer, A. Automatic Differentiation in PyTorch. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2019.

(14) Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; Kudlur, M.; Levenberg, J.; Monga, R.; Moore, S.; Murray, D. G.; Steiner, B.; Tucker, P.; Vasudevan, V.; Warden, P.; Wicke, M.; Yu, Y.; Zheng, X. Tensor Flow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, 2016.

(15) Landrum, G. *RDKit: Open-source Cheminformatics*, 2006.

(16) Minnich, A. J.; McLoughlin, K.; Tse, M.; Deng, J.; Weber, A.; Murad, N.; Madej, B. D.; Ramsundar, B.; Rush, T.; Calad-Thomson, S.; Brase, J.; Allen, J. E. AMPL: A Data-Driven Modeling Pipeline for Drug Discovery. *J. Chem. Inf. Model.* **2020**, *60* (4), 1955−1968.

(17) Kipf, T. N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2019.

(18) Kingma, D. P.; Ba, J. L. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

(19) Beauman, J. A.; Howard, P. H. Physprop Database. In *Syracuse Res.*, Syracuse, NY, USA, 1995.

(20) Fourches, D.; Muratov, E.; Tropsha, A. Trust, but Verify: On the Importance of Chemical Structure Curation in Cheminformatics and QSAR Modeling Research. *J. Chem. Inf. Model.* **2010**, *50* (7), 1189−1204.

(21) Tetko, I. V.; Tanchuk, V. Y.; Villa, A. E. P. Prediction of N-Octanol/Water Partition Coefficients from PHYSPROP Database Using Artificial Neural Networks and E-State Indices. *J. Chem. Inf. Comput. Sci.* **2001**, *41*, 1407.

(22) Capuzzi, S. J.; Politi, R.; Isayev, O.; Farag, S.; Tropsha, A. QSAR Modeling of Tox21 Challenge Stress Response and Nuclear Receptor Signaling Toxicity Assays. *Front. Environ. Sci.* **2016**, na DOI: 10.3389/fenvs.2016.00003.

(23) Bjerrum, E. J. SMILES Enumeration as Data Augmentation for Neural Network Modeling of Molecules. *arXiv1703.07076*, 2017.

(24) Caruana, R. Multitask Learning. *Mach. Learn.* **1997**, *28*, 41.

(25) Mayr, A.; Klambauer, G.; Unterthiner, T.; Hochreiter, S. Deep Tox: Toxicity Prediction Using Deep Learning. *Front. Environ. Sci.* **2016**, na DOI: 10.3389/fenvs.2015.00080.

(26) Popova, M.; Shvets, M.; Junier, O.; Isayev, O. Molecular RNN: Generating Realistic Molecular Graphs with Optimized Properties. *arXiv:1905.13372*, 2019.

(27) Williams, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.* **1992**, *8* (3-4), 229−256.

(28) Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Gated Feedback Recurrent Neural Networks. In *32nd International Conference on Machine Learning, ICML 2015*, 2015.

(29) Tetko, I. V.; Sushko, Y.; Novotarskyi, S.; Patiny, L.; Kondratov, I.; Petrenko, A. E.; Charochkina, L.; Asiri, A. M. How Accurately Can We Predict the Melting Points of Drug-like Compounds? *J. Chem. Inf. Model.* **2014**, *54* (12), 3320−3329.