# A GPU-Accelerated Machine Learning Framework for Molecular Simulation: Hoomd-blue with TensorFlow

Rainier Barrett[1], Maghesree Chakraborty[1], Dilnoza B. Amirkulova[1], Heta A. Gandhi[1], and Andrew D. White[1*]

[1]Department of Chemical Engineering, University of Rochester, Rochester, NY 14627

**Abstract**

We have designed and implemented software that enables integration of a scalable GPU-accelerated molecular mechanics engine, HOOMD-blue, with the machine learning (ML) TensorFlow package. TensorFlow is a GPU-accelerated, scalable, graph-based tensor computation model building package that has been the implementation of many recent innovations in deep learning and other ML tasks. Tensor computation graphs allow for designation of robust, flexible, and easily replicated computational models for a variety of tasks. Our plugin leverages the generality and speed of computational tensor graphs in TensorFlow to enable four previously challenging tasks in molecular dynamics: (1) the calculation of arbitrary force-fields including neural-network-based, stochastic, and/or automatically-generated force-fields which are differentiated from potential functions; (2) the efficient computation of arbitrary collective variables; (3) the biasing of simulations via automatic differentiation of collective variables and consequently the implementation of many free energy biasing methods; (4) ML on any of the above tasks, including coarse grain force fields, on-the-fly learned biases, and collective variable calculations. The TensorFlow models are constructed in Python and can be visualized or debugged using the rich set of tools implemented in the TensorFlow package. In this article, we present examples of the four major tasks this method can accomplish, benchmark data, and describe the architecture of our implementation. This method should lead to both the design of new models in computational chemistry research and reproducible model specification without requiring recompiling or writing low-level code.

## 1 Introduction

HOOMD-blue[1,2] is a GPU-accelerated engine for molecular dynamics (MD) and hard particle Monte Carlo simulations, and has simulated clathrate crystal colloids,[3] coarse grained solar cell polymers,[4] intrinsically disordered proteins,[5] and many other systems.[6–24] HOOMD-blue can simulate large systems with high speed due to the scalable nature of GPU processing. Another advantage of HOOMD-blue is that it has a Python interface that can be incorporated into a larger Python workflow.

TensorFlow is a ML library created and maintained by Google.[25] TensorFlow uses a graph-based computation framework with tensor operations to represent its underlying mathematical operations. This tensor graph abstraction allows for flexible model design, where one can easily add, remove, or alter operations (nodes) of a model (graph) while preserving the overall

structure and flow of the tensor data (edges). TensorFlow has a built-in visualization tool called TensorBoard to aid in this process. Another consequence of this tensor computation graph model designation is a major advantage of TensorFlow: most of the tensor operations defined in its library are analytically differentiable, and these derivatives can be automatically propagated throughout any TensorFlow model upon request. Thus, as long as we can express a model as a composition of tensor operations, we automatically have access to its derivatives at every step to compute forces or perform learning. Additionally, TensorFlow can optimize model evaluation by caching values of tensors as they are evaluated so that branches in the graph need not have these values explicitly stored in memory.

The HTF package described in this work gives TensorFlow access to the per-particle positions, neighbor lists, and forces generated by HOOMD-blue at each timestep of a simulation. Since both HOOMD-blue and TensorFlow can execute entirely on the GPU, there is minimal loss of speed due to communication. With HTF, a user may specify any tensor operation on the neighbor list and automatically calculate its derivatives and thus, its forces. This enables use of arbitrary force fields (e.g., from neural networks), calculation of arbitrary collective variables, and biasing of arbitrary collective variables. In addition, since TensorFlow contains a suite of ML algorithms implementation, a user can also perform learning on any of the items calculated using HTF.

ML has been defined a number of different ways in the past,[26–28] but for the purpose of this work, it is taken to mean a computer "learning" how to best perform some task under a given performance metric via optimization of a set of vector operations. For example, common ML applications are regression problems and classification problems, where the performance metric is usually obvious – e.g. how many items the program correctly sorts in a classification problem, or the mean squared error from a target function or distribution in a regression problem. Specifying the correct performance metric and model structure to accomplish a given task is neither deterministic nor necessarily simple,[29,30] but TensorFlow eases the process with its library of ML algorithms.

Recently, ML methods have been used to improve the accuracy of MD simulations, and some have even achieved configurational accuracy on par with ab-initio methods.[31–33] However, the way this process is typically performed can be difficult or cumbersome to reproduce.[34] This is because implementing a given ML model often requires custom low-level code to achieve learning in the context of MD simulation engines, or because it necessitates an iterative process of generating data, training, and validating, rather than a single ongoing process.[35,36] HTF can bridge this gap and make the process of connecting ML models with MD simulations easy, transparent, and reproducible via the model specification interface of TensorFlow.

The rest of this paper is broken up into five sections plus some concluding remarks. The first section describes the architecture of the HTF package and how it works. The remaining four sections each describe an application of HTF to a particular task in molecular dynamics, demonstrating online ML in MD, arbitrary collective variable calculations, a use-case of TensorFlow's automatic differentiation, and learning of coarse-grain forces.

## 2 Methods and Implementation

A more complete description of TensorFlow may be found in Abadi et al.[25] TensorFlow models are built and run in separate steps, similar to how a computer program is compiled and then executed. The models in TensorFlow are expressed as tensor computation graphs where nodes

are tensor operations and edges are tensors. The HTF plugin follows the same approach, whereby there is a graph building step and then an execution step.

During the graph building step, placeholder tensors are accessible for neighbor lists, positions, and forces. During the execution step (i.e., running the simulation), they will be populated with their respective values in HOOMD-blue at the current timestep. A neighbor list is an $N \times M \times 4$ tensor, where $N$ is the number of particles, $M$ is a pre-set maximum number of neighbors, and the last dimension is $x, y, z, w$. Here, $w_m$ denotes the $m^{\text{th}}$ neighbor's particle type, and $x_{n,m}, y_{n,m}, z_{n,m}$ are the components of the distance vector to the $m^{\text{th}}$ neighbor of the $n^{\text{th}}$ particle. The positions, neighbor lists, and forces are provided as possible inputs at each step. The tensor computation graph can also output forces and a virial at each step. During the graph building step, these may be computed in the tensor computation graph or automatically calculated from a per-particle or total potential energy tensor. The neighbor lists are taken directly from HOOMD-blue and are not reconstructed. The automatic differentiation computes the forces on particles as:

$$\vec{F}_i = -\frac{\partial U(\mathbf{r})}{\partial \vec{r}_i} - \sum_{i<j} \frac{\partial U(\mathbf{r})}{\partial \vec{r}_{ij}} \tag{1}$$

where $U(\mathbf{r})$ is the potential energy as a function of both positions and/or pairwise distances, $\vec{F}_i$ is the net force on particle $i$, and $\vec{r}_{ij}$ is the distance vector from particle $i$ to $j$. HTF computes the per-particle virial contribution from pairwise interactions only as:

$$\tau_i = \sum_{i<j} \frac{F_{ij}}{r_{ij}} (\vec{r}_{ij} \otimes \vec{r}_{ij}) \tag{2}$$

where $\tau_i$ is the virial stress tensor of particle $i$, $F_{ij}$ is the magnitude of the force on particle $i$ from particle $j$, and $\otimes$ indicates an outer product. If a biasing force in HTF is not pairwise additive, then the virial contribution can be computed by the user to override this default contribution.

During the execution step, while the simulation is running, the TensorFlow graph is executed with the current neighbor lists, positions and forces. Forces can be an input or output. If the TensorFlow graph outputs forces, those will be set in HOOMD-blue to be forces on the particles. Sometimes there is no output, for example if computing a collective variable. Variables in the tensor computation graph allow values to be saved and updated at each step. This allows computed quantities at all stages of the graph to be output. Further, this allows accumulation of values and thus computing quantities like running averages or time-dependent biases. The variables can also be checkpointed and restarted within HTF.

Constructing tensor computation graphs may be confusing, especially because the dimensionalities of the positions and neighbor list tensors are unknown until runtime. However, TensorFlow has a rich set of debugging tools, including TensorBoard,[25] which allows visualization and step-by-step execution of the graph. These are accessible with the HTF plugin. TensorBoard also allows printing, histogramming, and plotting of variables in real time.

HTF is implemented as a multithreading Python module with most of the code written in C++ linked via pybind11 .[37] Data is transferred from HOOMD-blue to TensorFlow simply via GPU-GPU `memcpy` calls. The implementation of HTF in TensorFlow is done by creating custom node types called `hoomd2tf` and `tf2hoomd` which enable reading/writing via the `memcpy` calls to HOOMD-blue data structures. The multithreading is necessary to isolate

the TensorFlow and HOOMD-blue GPU contexts, which allows them to run simultaneously on one GPU. The main computation done by HTF is the reshaping of neighbor lists from the ragged-array format of HOOMD-blue into an $N \times M \times 4$ tensor usable by TensorFlow. In addition to GPU-GPU communication, HTF supports MPI parallelization and has a CPU reference implementation.

The MPI parallelization is done via HOOMD-blue's domain decomposition. Thus, there are $N$ independent TensorFlow instances running, one for each HOOMD-blue domain. This is desirable for pairwise force computation. TensorFlow has the capability of communicating between instances via TCP/IP, which has high latency relative to MPI. This could be used, however, for communicating training data done on batches of frames every so many timesteps, or at the end of a simulation for accumulating/reducing computed values. There exist community-supported MPI implementations of TensorFlow but these have not been explored. The source code of HTF is available as specified in the Additional Information section.

TensorFlow has a variety of extensions, including TensorFlow Fold [38] which is specifically designed for representing molecular graphs in TensorFlow. Another is TensorFlow Probability, [39] which allows stochastic terms in the TensorFlow graph. This may be used for MC-MD coupled simulations, where the force-field depends on sampled stochastic terms. Inference can also be done in TensorFlow Probability, so that these probability distributions can be trained for methods like Ultra coarse-graining [40].

# 3 Case Studies with HTF

## 3.1 Neural Network Force Field (Arbitrary Force Fields)

Neural networks are a powerful class of ML tools whose original theory dates back to the 1950s and 60s. [41–43] These highly flexible tools have been applied to a variety of diverse tasks in the past, including guiding surveillance technology, [44,45] tracking visual targets, [46–48] predicting cancer occurrence in patients, [49,50] processing medical imaging for enhanced diagnostic accuracy, [51] controlling the beam of a particle accelerator, [52] and aiding in finance applications. [53] They have also seen recent use in molecular dynamics simulations, allowing for trained force fields that accurately reproduce experimental conformations [32] and dynamical properties in coarse-grained simulations. [31]

The HTF plugin allows users to designate the structure of neural networks and achieve online training based on HOOMD-blue data during the HOOMD-blue simulation. Any collective variable that can be expressed as a tensor operation on the HOOMD-blue neighbor list or per-particle positions or forces can be used as training data or neural network input. As a proof of concept, we demonstrate here a neural network trained to reproduce the Lennard-Jones forces on a 2D simulation of 64 particles. Training is done online, i.e. during the simulation. The neural network takes as input the HOOMD-blue neighbor list and forces, and is trained to output the forces on each particle with an $l2$ loss function (mean squared error) and the TensorFlow built-in ADAM optimizer [54] with a learning rate $\eta = 0.001$, and with TensorFlow's default parameters of $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-8}$. The simulation used reduced LJ units with a timestep of 0.005, and Langevin integration with $kT = 1.0$.

The neural network structure was a simple multilayer perceptron with three layers: one input layer, one hidden layer, and one output layer. Each layer is size 1 and uses a linear activation function. This is the simplest possible neural network with one hidden layer. A

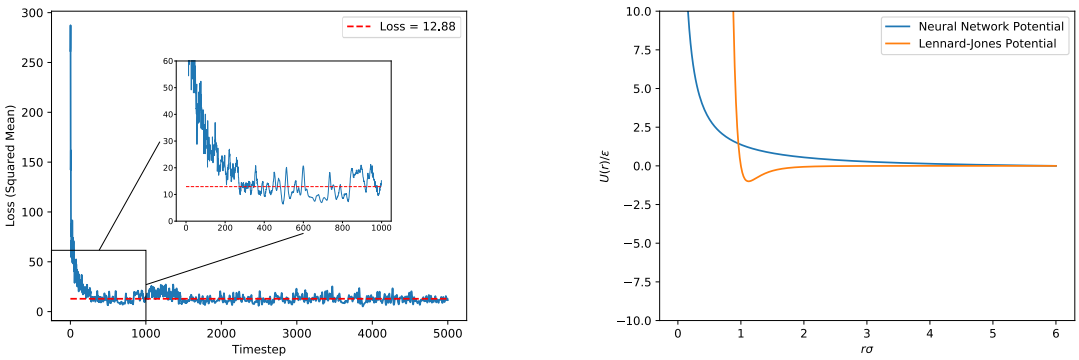
multilayer perceptron can be thought of as repeated application of operations $\{\mathcal{F}_i\}$ defined in eq 3, where $\vec{w}_i$ is a weight vector and $\vec{b}_i$ is a bias vector, both of which are unique to each layer, and $\vec{x}_i$ is the vector of input values to layer $i$.

$$\mathcal{F}_i\left(\vec{x}_i\right) = \vec{w}_i \cdot \vec{x}_i + \vec{b}_i \tag{3}$$

In this case, the neural network's output is then $\mathcal{F}_2\left(\mathcal{F}_1\left(\mathcal{F}_0\left(\vec{x}\right)\right)\right)$, and the trained quantities are the weight and bias vectors, with $\vec{x} = \vec{r}^{-1}$, the vector of inverse inter-particle distances.

After a 4000 step equilibration period, the model was trained for 5000 steps. The validation run likewise had a 4000 step equilibration period before loading the model and running for 5000 steps to gather statistics. This was to ensure the model was trained and tested on systems of roughly equal energy. The $l2$ loss of the model over time is shown in Figure 1. We can see that the model converges quickly (within 1000 timesteps) and averages a loss value of 12.88. A source that may account for this loss is a possible lack of data for particles that are close together due to repulsion. By the same reasoning, the model may also be over-fitting for the long distance region where particle-particle forces are roughly zero. Another possibility is that the input of $\vec{r}^{-1}$ was too simple to capture the important features of the Lennard-Jones potential with a linear model.

To benchmark HTF, we repeated this process for a range of particle numbers from 100 to 90000 and compared the run times of CPU-only and GPU-enabled HTF to assess its scaling. In Figure 2, we show this benchmarking data, illustrating how low the additional computational cost of training these neural networks can be. Exact values for timesteps per second and $N$ for these trials can be found in Table 1 in the Additional Information section.




**(a)** **(b)**

**Figure 1** Performance of a simple neural network in a 64 particle Lennard-Jones simulation. **(a):** The $l2$ loss over time of the neural network force field during training. The $l2$ loss is computed as squared difference between the Lennard-Jones forces calculated by HOOMD-blue and those generated by the neural network. The dashed red line is the average loss across all timesteps after loss fell below one standard deviation from the overall mean. **(b):** Comparison of the learned potential and the true Lennard-Jones potential.

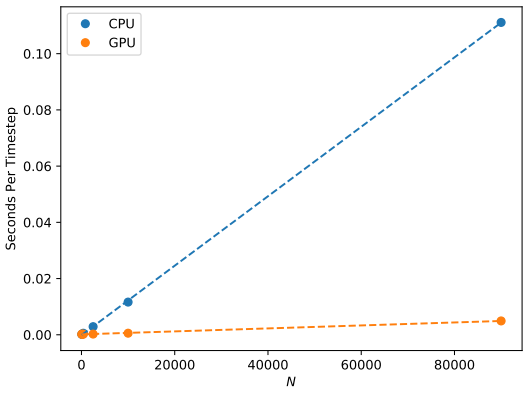



**Figure 2** Median seconds per timestep over five trials of 5000 Training Steps of a HTF Neural Network with $N$ Particles on CPU and GPU. This demonstrates the gain in speed from GPU execution. The linear scaling in particle number, $N$, shows communication of HOOMD-blue and TensorFlow minimally affects performance. Exact timing values may be found in Table 1.

### 3.1.1 Example Code

To demonstrate the ease and readability of using HTF, we include here some sample code for building a neural network model such as the one used to obtain these results. In this excerpt, $NN$ is the size of the neighbor list to be used in HOOMD-blue and `N_hidden_nodes` is the number of nodes in each hidden layer. This code produces a TensorFlow tensor computation graph model for training a neural network force-field. To use this model for inference, the `output_forces` argument would be set to `True`, and it would load the trained model by using the `bootstrap_dir` optional argument to `graph_builder`. This code uses the Keras [55] python module to initialize the neural network according to eq 3, with the dimension of $\vec{w}$ set by `N_hidden_nodes` and the operation $\mathcal{F}$ repeated three times.

```
import tensorflow as tf
import keras
import hoomd.tensorflow_plugin as htf

NN = 64
N_hidden_nodes = 5
graph = htf.graph_builder(NN, output_forces=False)
r_inv = graph.nlist_rinv
input_tensor = tf.reshape(r_inv, shape=(-1,1), name='r_inv')
input_layer = keras.layers.Input(tensor=input_tensor)
hidden_layer = keras.layers.Dense(N_hidden_nodes)(input_layer)
output_layer = keras.layers.Dense(1, input_shape=(N_hidden_nodes,))(hidden_layer)
nn_energies = tf.reshape(output_layer, [-1, NN])
calculated_energies = tf.reduce_sum(nn_energies, axis=1, name='calculated_energies
    ')
calculated_forces = graph.compute_forces(calculated_energies)
cost = tf.losses.mean_squared_error(calculated_forces, graph.forces)
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)
graph.save(model_directory='/tmp/keras_model/', out_nodes=[ optimizer])
```

## 3.2 Force Matching (Learning)

Bottom-up coarse graining (CG) [56–58] has been used to model various systems like alkanes [58], polymers [59], and biomolecules [56]. In the bottom-up approach, the CG potentials are derived from the underlying all-atom (AA) simulation. Force-matching (FM) [60–62] is one of the bottom-up CG approaches which aims to match the forces on CG particles as closely as possible to the cumulative forces on their constituent atoms in the reference AA simulation [56]. Besides obtaining the CG potential, another vital step in defining the CG system is to determine the mapping operator which indicates how atoms in the AA system are grouped into CG particles. The general practice to define a CG mapping operators has been driven by chemical intuition [63]. However, recently there have been efforts towards choosing mapping operators more systemically [64–66].

The reference mapped force is calculated by eq 4, where $F_I^{ref}$ is the force on the $I^{th}$ CG particle, $S_I$ refers to the subset of particles which are mapped into the $I^{th}$ CG particle and $\vec{F}_i$ is the net force on the $i^{th}$ particle in the AA model.

$$\mathbf{F}_I^{ref}(\mathbf{r}) = \sum_{j \in S_I} \vec{F}_i(\mathbf{r}) \tag{4}$$

The FM method finds $\mathbf{F}^{CG}(\mathbf{R})$ that minimizes the objective function given by eq 5,

$$\chi^2 = \left\langle \frac{1}{3N} \sum_{I=1}^{N} \left| \mathbf{F}_I^{CG}(\mathbf{R}) - \mathbf{F}_I^{ref}(\mathbf{r}) \right|^2 \right\rangle \tag{5}$$

where $N$ is the number of CG particles and the angular brackets denote an ensemble average. In the most common implementation, $\mathbf{F}^{CG}(\mathbf{R})$ is approximated using cubic splines as the basis set [61,67].

Previous studies have reported the use of various ML techniques for CG models [68,69]. Here we model the CG potential using a basis set ($\mathcal{B}$) of 48 Gaussian functions and a repulsive term, $U_{rep}(R) = u(R - r_0)(R - r_0)^{12}$, where $u()$ is the unit step function. The variance and the height of each of the Gaussian functions in the basis set were trained on-the-fly using mapped CG positions from an atomistic NVT simulation of 1000 methanol molecules in HOOMD-blue at 298.5 K with a density of 787.727 kg/m$^3$. Each methanol molecule was mapped to one CG bead at its center of mass (COM). Corresponding mapped pairwise distances were calculated considering 128 nearest neighbors. The optimization was done at a learning rate of $10^{-3}$ using the ADAM optimizer [54] by minimizing the objective function given by eq 6

$$\chi^2 = \left\langle \frac{1}{3N} \sum_{I=1}^{N} \left| \mathbf{F}_I^{\mathcal{B}}(\mathbf{R}) - \mathbf{F}_I^{ref}(\mathbf{r}) \right|^2 \right\rangle - r_0 \tag{6}$$

where $\mathbf{F}_I^{\mathcal{B}}$ refers to the forces calculated using the basis set and the mapped pairwise distances and $r_0$ is a fitting parameter for the repulsive function $U_{rep}$. In eq 6, the first term is the mean squared error between the mapped CG forces from the atomistic simulation and the forces calculated using the the basis set, and the last term pushes the short range repulsive term to the right. A cutoff radius of 12 Å was used for the simulation. The CG simulation was run for 20,000 timesteps. Figure 3 shows the learned CG potential along with the locations of the Gaussian basis set functions of $\mathcal{B}$. Figure 4 compares the COM radial distribution function (RDF) obtained from the AA simulation and that obtained from running a CG simulation using the learned potentials. The RDF shows good agreement.

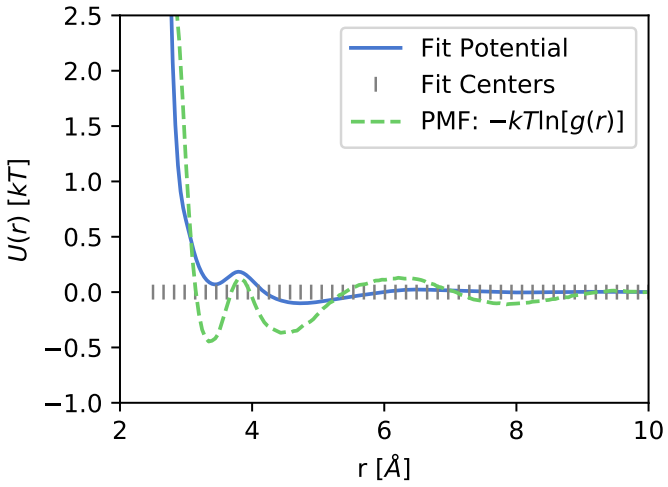



**Figure 3** The learned CG potential along with the fit centers of the Gaussian basis set functions for a one bead methanol simulation. A reference potential of mean force (PMF) computed as $KT\ln[g(r)]$ plotted. Agreement between CG potential and PMF is not expected, but provides a reference for location of potential energy minima. The repulsion term provides stronger repulsion than is possible with the Gaussian basis set alone.

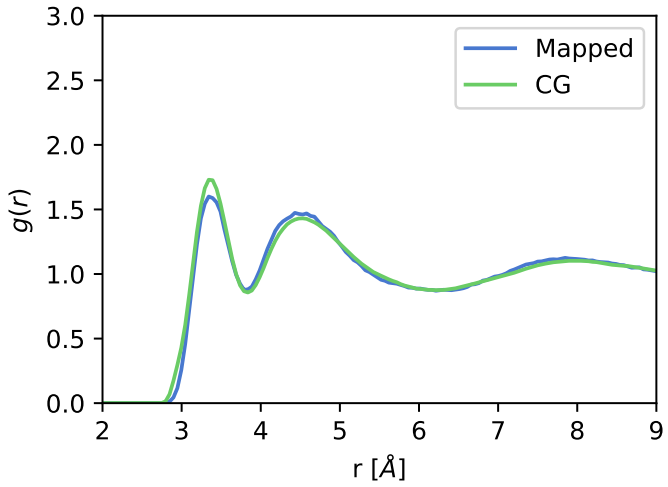



**Figure 4** The center of mass radial distribution function from the AA (Mapped) and the CG simulation of methanol. The one bead CG simulation shows good agreement with a mapped AA simulation.

### 3.3 EDS in HTF (Biasing/Automatic differentiation)

Experiment Directed Simulation (EDS) is a maximum-entropy biasing technique that matches certain reference values such as experimental data during simulations.[70] EDS minimally mod-

ifies the free energy surface such that average values of simulated collective variables (CVs) match certain average reference values.[70] Instead of a single average value, a probability distribution of the free energy surface along a CV needs to match to a another free energy distribution, EDS can be applied.[71] EDS can match multiple experimental data in a single iteration of the simulation.[70] EDS has been previously applied in various systems such as Li ion battery and bead-spring polymer models,[70] a water model,[72] a peptide model,[73] and in coarse graining techniques.[74] Since EDS minimally modifies the free energy surface, the dynamical studies of water with EDS showed improved prediction of both biased and unbiased dynamic properties of water, which are known to be hard to reproduce even with quantum mechanics.[72] A recent review can be found in Amirkulova and White.[75] In this work EDS was implemented in the TensorFlow framework.

In EDS, the potential energy, $U(\vec{r})$ is modified minimally with addition of multiple coupling constants, $\alpha_j$ as shown in eq 7, where the index $j$ is over the number of CVs to be biased. We desire that the ensemble average of a simulated CV, $f(\vec{r})$, satisfy $\langle f(\vec{r})\rangle = \hat{f}$, where $\langle f(\vec{r})\rangle$ is the ensemble average of the CV and $\hat{f}$ is the reference value of that CV. The force of the $i^{\text{th}}$ particle, is calculated from the potential given by eq 7 at each time step.

$$U'(\vec{r},\alpha) = U(\vec{r}) + \sum_j \alpha_j \frac{f_j(\vec{r})}{\bar{f}_j} \tag{7}$$

$$R = \frac{1}{N}\sum_i^N |\vec{r}_i - \vec{r}_{COM}| \tag{8}$$

A Lennard-Jones particle simulation was performed to bias the simulation with EDS. Mean radius (R), which is defined ineq 8 as an average of absolute differences between each particle position and COM of all particles, N. By using an average radius as a CV for EDS simulation, we can enforce the simulation on average to form a circle around the COM of with radius of a bias value. We ran EDS and control simulations of Lennard-Jones particles.

For EDS and control simulations, the simulations were in identical conditions except for the presence of a bias. A square box of evenly spaced 64 (N=64) Lennard-Jones particles was simulated using HTF. The size of the box was 16 by 16, with periodic boundary conditions. The simulations were conducted at NVE ensemble for 200,000 timesteps. Lennard-Jones potential had $\sigma$=1 and $\epsilon$=1. Lennard- Jones pair potential had a cutoff radius of 3. At every timestep EDS added a minimal bias such that the criterion, $\langle f(r)\rangle = \hat{f}$, is achieved.

We biased the EDS simulation such that an ensemble average of $R$ matches 5.8, as shown in fig 5. In fig 5, the average R in the EDS simulation, shown in green, oscillates and finally matches the reference value, shown in red. In comparison, the control simulation has an average radius of 6 and is different from the reference value.

In order to see how particles populate the simulation box during EDS and control simulations, the distribution of particles in the box from all time steps is shown as a heat-map in fig 6. EDS tries to form a circle around the center of the box, as demonstrated by the fact that particles are mostly inside the circle with radius=5.8. EDS bias (A) pushes the particles to be inside of the white circle which corresponds to the bias. The control simulation tries to distribute the particles around the square simulation box more uniformly as shown in fig 6(B). We have shown that EDS can modify the simulation such that there is an agreement between simulations and our observables. This technique can be applied to larger and more complex

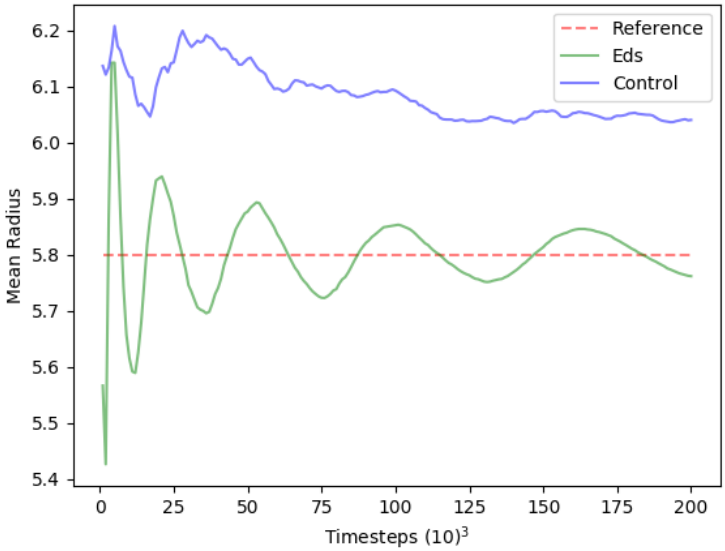



**Figure 5 EDS matches a reference value that was used as a bias.** At the end of the EDS simulation, the average radius (green) matches a the reference value of 5.8 (red). The control simulation converges to an average radius around 6.

systems in HOOMD-blue where there may be a major discrepancy between simulations and experiments.

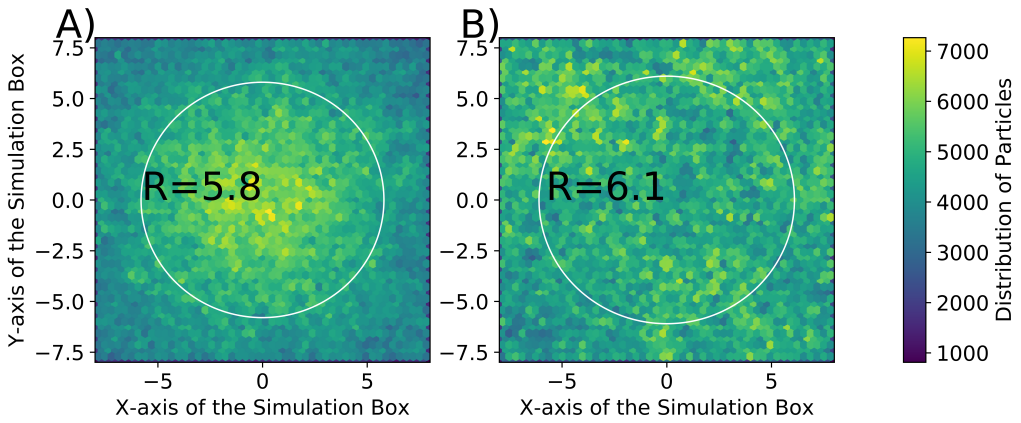



**Figure 6** Comparison of particle positions from EDS (A) and control (B) simulation trajectories. In expectation, the center of mass of particles in both EDS and control simulations should be at the center of the box. Hence, circles with radii corresponding to computed average radii in EDS and control simulations are plotted for visual aid. On average more particles can be found inside the white circle in the EDS simulation (A) due to a bias that makes the radius to be 5.8. The control simulation (B) does not have a bias, and hence it randomly samples the square box.

### 3.4 Calculating Scattering Profiles (Arbitrary Collective Variables)

Scattering techniques are used extensively for material characterization and structure determination [76]. Neutron scattering in particular has applications in the field of condensed matter physics, [77,78] nuclear physics and nuclear materials research, [79] biology [80], crystallography, and catalysis. [78] For this work, we demonstrate that HTF can be used to approximate the neutron scattering profile of a simulation box at every timestep.

The neutron scattering profile for particles can be analytically estimated using Debye's formula given by eq (9). [81]

$$I(q) = \sum_i \sum_j b_i b_j \frac{\sin(q\ r_{ij})}{q\ r_{ij}} \tag{9}$$

where $b_i$ and $b_j$ are the coherent scattering lengths for the particles $i$ and $j$ respectively, $r_{ij}$ is the inter-particle distance between particles $i$ and $j$, and $q$ is the magnitude of $\vec{q}$, the scattering vector or the momentum transfer vector, which is a function of the scattering angle $\theta$ and wavelength $\lambda$. The data for scattering lengths of different particles is available in literature. [82]

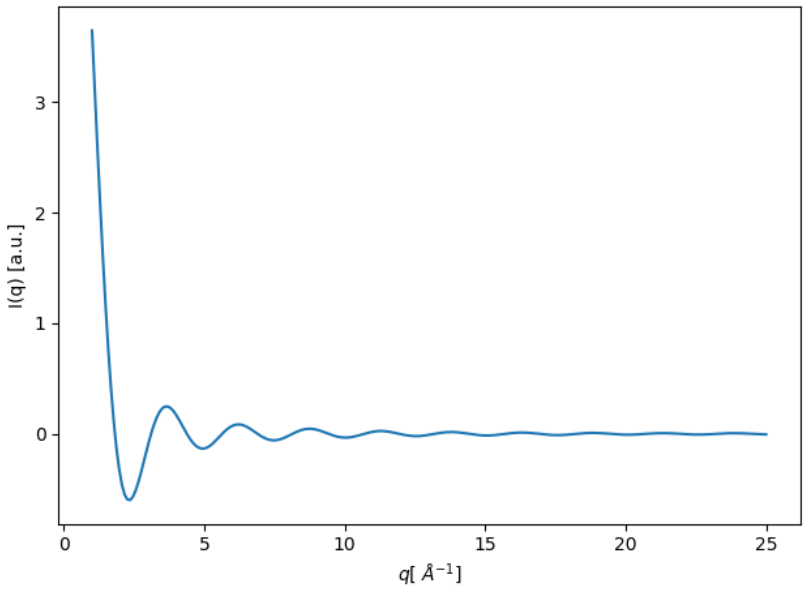



**Figure 7** The neutron scattering profile for 5000 $H_2O$ molecules calculated using HTF.

A box of 5000 water molecules was simulated in order to calculate their neutron scattering profile. Figure 7 shows the resultant scattering profile, which can be calculated at any timestep during a simulation. Due to the automatic differentiation included in TensorFlow, a more complex system could be biased toward an experimentally-obtained neutron scattering profile on-the-fly.

## 4 Concluding Remarks

We have presented a method for utilizing tensor computation graphs in tandem with the MD and MC simulation engine HOOMD-blue. These tensor computation graphs are expressive

enough to allow force-fields, biasing methods, learning, and collective variable computation within a single framework. HTF enables GPU-accelerated and low-latency use of tensor computation graphs in HOOMD-blue and thus makes online learning in a simulation possible. The online nature of learning in HTF simplifies the "traditional" workflow of ML in simulation by removing the need for post-processing of trajectories or custom implementations of common ML algorithms. The tensor computation graphs allow for transparent and simple model designation with a high degree of customizability, replicability, and efficiency. The HTF source code is available and each of the systems presented in this article are available as examples.

# 5 Additional Information

Source code for our implementation of the methods discussed in the main text can be found at the following GitHub repository: `https://github.com/ur-whitelab/hoomd-tf`.

| $N$ | CPU TPS | GPU TPS |
|---|---|---|
| 64 | 6113 | 4843 |
| 144 | 4042 | 5119 |
| 400 | 1831 | 4977 |
| 2500 | 346 | 3974 |
| 10000 | 86 | 1781 |
| 90000 | 9 | 203 |

**Table 1** Median timesteps per second (TPS) over five trials of 5000 training steps of a HOOMD-TF neural network with $N$ Lennard-Jones particles on CPU and GPU.

# 6 Acknowledgements

The authors would like to thank Joshua Anderson and Jens Glaser for instructive conversations about the HOOMD-blue architecture. The authors thank the Center for Integrated Research Computing (CIRC) at the University of Rochester for providing computational resources and technical support. This work was supported by the National Science Foundation (CBET-1751471 and CHE-1764415).

## References


1. Joshua A. Anderson, Chris D. Lorenz, and A. Travesset. General purpose molecular dynamics simulations fully implemented on graphics processing units. *J. Comput. Phys.*, 227(10):5342–5359, may 2008.

2. Jens Glaser, Trung Dac Nguyen, Joshua A. Anderson, Pak Lui, Filippo Spiga, Jaime A. Millan, David C. Morse, and Sharon C. Glotzer. Strong scaling of general-purpose molecular dynamics simulations on GPUs. *Comput. Phys. Commun.*, 192:97–107, jul 2015.

3. Haixin Lin, Sangmin Lee, Lin Sun, Matthew Spellings, Michael Engel, Sharon C Glotzer, and Chad A Mirkin. Clathrate colloidal crystals. *Science*, 355(6328):931–935, mar 2017.

4. Nanjia Zhou, Alexander S Dudnik, Ting I N G Li, Eric F Manley, Thomas J Aldrich, Peijun Guo, Hsueh-Chung Liao, Zhihua Chen, Lin X Chen, Robert P H Chang, Antonio Facchetti, Monica Olvera De La Cruz, and Tobin J Marks. All-Polymer Solar Cell Performance Optimized via Systematic Molecular Weight Tuning of Both Donor and Acceptor Polymers. *J. Am. Chem. Soc.*, 138(4):1240–1251, 2015.

5. Gregory L Dignon, Wenwei Zheng, Robert B Best, Young C Kim, and Jeetain Mittal. Relation between single-molecule properties and phase behavior of intrinsically disordered proteins. *Proc. Natl. Acad. Sci. U. S. A.*, 115(40):9929–9934, oct 2018.

6. Rodrigo E. Guerra, Colm P. Kelleher, Andrew D. Hollingsworth, and Paul M. Chaikin. Freezing on a sphere. *Nature*, 554(7692):346–350, feb 2018.

7. Emmanuel Millán Kujtiuk, Eduardo M Bringa, Andrew Higginbotham, and Carlos Garcia Garino. GP-GPU Processing of Molecular Dynamics Simulations. In *HPC - LATAM*, pages 3234–3248, Buenos Aires, 2010.

8. Lin Yang, Feng Zhang, Cai-Zhuang Wang, Kai-Ming Ho, and Alex Travesset. Implementation of metal-friendly EAM/FS-type semi-empirical potentials in HOOMD-blue: A GPU-accelerated molecular dynamics software. *J. Comput. Phys.*, 359:352–360, apr 2018.

9. Trung Dac Nguyen, Carolyn L. Phillips, Joshua A. Anderson, and Sharon C. Glotzer. Rigid body constraints realized in massively-parallel molecular dynamics on graphics processing units. *Comput. Phys. Commun.*, 182(11):2307–2313, nov 2011.

10. Andrew W. Long, Carolyn L. Phillips, Eric Jankowksi, and Andrew L. Ferguson. Nonlinear machine learning and design of reconfigurable digital colloids. *Soft Matter*, 12(34):7119–7135, aug 2016.

11. David N. LeBard, Benjamin G. Levine, Russell DeVane, Wataru Shinoda, and Michael L. Klein. Premicelles and monomer exchange in aqueous surfactant solutions above and below the critical micelle concentration. *Chem. Phys. Lett.*, 522:38–42, jan 2012.

12. I.V. Morozov, A.M. Kazennov, R.G. Bystryi, G.E. Norman, V.V. Pisarev, and V.V. Stegailov. Molecular dynamics simulations of the relaxation processes in the condensed matter on GPUs. *Comput. Phys. Commun.*, 182(9):1974–1978, sep 2011.

13. Lisa Teich and Christian Schroder. Numerical Investigation of the Magnetodynamics of Self-Organizing Nanoparticle Ensembles: A Hybrid Molecular and Spin Dynamics Approach. *IEEE Trans. Magn.*, 51(11):1–4, nov 2015.

14. Caleb L. Breaux, Jakin B. Delony, Peter J. Ludovice, and Clifford L. Henderson. Effect of homopolymer concentration on LER and LWR in block copolymer/homopolymer blends. In Eric M. Panning and Martha I. Sanchez, editors, *Nov. Patterning Technol. 2018*, volume 10584, page 52. SPIE, apr 2018.

15. Fabrizio Benedetti, Dusan Racko, Julien Dorier, Yannis Burnier, and Andrzej Stasiak. Transcription-induced supercoiling explains formation of self-interacting chromatin domains in S. pombe. *Nucleic Acids Res.*, 45(17):9850–9859, sep 2017.

16. Fabrizio Benedetti, Aleksandre Japaridze, Julien Dorier, Dusan Racko, Robert Kwapich, Yannis Burnier, Giovanni Dietler, and Andrzej Stasiak. Effects of physiological self-crowding of DNA on shape and biological properties of DNA molecules with various levels of supercoiling. *Nucleic Acids Res.*, 43(4):2390–2399, feb 2015.

17. Carolyn L. Phillips, Joshua A. Anderson, and Sharon C. Glotzer. Molecular Dynamics Models of Shaped Particles Using Filling Solutions. *Phys. Procedia*, 53:75–81, jan 2014.

18. Benjamin Trefz and Peter Virnau. Scaling behavior of topologically constrained polymer rings in a melt. *J. Phys. Condens. Matter*, 27(35):354110, sep 2015.

19. Michael P. Howard, Athanassios Z. Panagiotopoulos, and Arash Nikoubashman. Efficient mesoscale hydrodynamics: Multiparticle collision dynamics with massively parallel GPU acceleration. *Comput. Phys. Commun.*, 230:10–20, sep 2018.

20. Benjamin G. Levine, David N. LeBard, Russell DeVane, Wataru Shinoda, Axel Kohlmeyer, and Michael L. Klein. Micellization Studied by GPU-Accelerated Coarse-Grained Molecular Dynamics. *J. Chem. Theory Comput.*, 7(12):4135–4145, dec 2011.

21. Carolyn L. Phillips, Joshua A. Anderson, and Sharon C. Glotzer. Pseudo-random number generation for Brownian Dynamics and Dissipative Particle Dynamics simulations on GPU devices. *J. Comput. Phys.*, 230(19):7191–7201, aug 2011.

22. Michael P. Howard, Joshua A. Anderson, Arash Nikoubashman, Sharon C. Glotzer, and Athanassios Z. Panagiotopoulos. Efficient neighbor list calculation for molecular simulation of colloidal systems using graphics processing units. *Comput. Phys. Commun.*, 203:45–52, jun 2016.

23. Joshua A. Anderson, M. Eric Irrgang, and Sharon C. Glotzer. Scalable Metropolis Monte Carlo for simulation of hard shapes. *Comput. Phys. Commun.*, 204:21–30, jul 2016.

24. Matthew Spellings, Ryan L. Marson, Joshua A. Anderson, and Sharon C. Glotzer. GPU accelerated Discrete Element Method (DEM) molecular dynamics for conservative, faceted particle simulations. *J. Comput. Phys.*, 334:460–467, apr 2017.

25. Martín Abadi, Ashish˜Agarwal, Paul˜Barham, Eugene˜Brevdo, Zhifeng˜Chen, Craig˜Citro, Greg˜S.˜Corrado, Andy˜Davis, Jeffrey˜Dean, Matthieu˜Devin, Sanjay˜Ghemawat,

Ian˜Goodfellow, Andrew˜Harp, Geoffrey˜Irving, Michael˜Isard, Yangqing Jia, Rafal˜Jozefowicz, Lukasz˜Kaiser, Manjunath˜Kudlur, Josh˜Levenberg, Dandelion˜Mané, Rajat˜-Monga, Sherry˜Moore, Derek˜Murray, Chris˜Olah, Mike˜Schuster, Jonathon˜Shlens, Benoit˜Steiner, Ilya˜Sutskever, Kunal˜Talwar, Paul˜Tucker, Vincent˜Vanhoucke, Vijay˜-Vasudevan, Fernanda˜Viégas, Oriol˜Vinyals, Pete˜Warden, Martin˜Wattenberg, Martin˜Wicke, Yuan˜Yu, and Xiaoqiang˜Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015.

26. Hua Wang, Cuiqin Ma, and Lijuan Zhou. A Brief Review of Machine Learning and Its Application. In *2009 Int. Conf. Inf. Eng. Comput. Sci.*, pages 1–4. IEEE, dec 2009.

27. Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, 2012.

28. Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2009.

29. Avrim L. Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artif. Intell.*, 97(1-2):245–271, dec 1997.

30. Chris. McDonald and Lloyd A. Smith. *Computer science '98 : proceedings of the 21st Australasian Computer Science Conference, ACSC'98, Perth, 4-6 February 1998*, volume Volume 20. Springer, 1998.

31. Linfeng Zhang, Jiequn Han, Han Wang, and Roberto Car. DeePCG: constructing coarse-grained models via deep neural networks. Technical report, 2018.

32. Kun Yao, John E. Herr, David W. Toth, Ryker Mckintyre, and John Parkhill. The TensorMol-0.1 model chemistry: a neural network augmented with long-range physics. *Chem. Sci.*, 9(8):2261–2269, feb 2018.

33. Han Wang, Linfeng Zhang, Jiequn Han, and Weinan E. DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics. *Comput. Phys. Commun.*, 228:178–184, jul 2018.

34. Katja Hansen, Grégoire Montavon, Franziska Biegler, Siamac Fazli, Matthias Rupp, Matthias Scheffler, O. Anatole von Lilienfeld, Alexandre Tkatchenko, and Klaus-Robert Müller. Assessment and Validation of Machine Learning Methods for Predicting Molecular Atomization Energies. *J. Chem. Theory Comput.*, 9(8):3404–3419, aug 2013.

35. Matthias Rupp, Alexandre Tkatchenko, Klaus-Robert Müller, and O Anatole Von Lilienfeld. Fast and Accurate Modeling of Molecular Atomization Energies with Machine Learning. *Phys. Rev. Lett.*, 108:058301, 2012.

36. Venkatesh Botu and Rampi Ramprasad. Adaptive Machine Learning Framework to Accelerate Ab Initio Molecular Dynamics. *Int. J. Quantum Chem.*, (115):1074–1083, 2015.

37. Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 – seamless operability between c++11 and python, 2017. https://github.com/pybind/pybind11.

38. Moshe Looks, Marcello Herreshoff, DeLesley Hutchins, and Peter Norvig. Deep learning with dynamic computation graphs. *arXiv preprint arXiv:1702.02181*, 2017.

39. Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A. Saurous. TensorFlow Distributions. *ArXiv*, nov 2017.

40. James F. Dama, Anton V. Sinitskiy, Martin McCullagh, Jonathan Weare, Benoît Roux, Aaron R. Dinner, and Gregory A. Voth. The theory of ultra-coarse-graining. 1. general principles. *Journal of Chemical Theory and Computation*, 9(5):2466–2480, 2013. PMID: 26583735.

41. Bernard Widrow. *Adaptive "adaline" neuron using chemical "memistors"*. Stanford University, Stanford, 1960.

42. J A Anderson and E Rosenfeld. *Talking Nets: An Oral History of Neural Networks*. Bradford Books. MIT Press, 2000.

43. John Von Neumann and Others. *The general and logical theory of automata*, volume 5. New York: John Wiley& Sons, 1951.

44. T. Jan. Neural network based threat assessment for automated visual surveillance. In *2004 IEEE Int. Jt. Conf. Neural Networks (IEEE Cat. No.04CH37541)*, volume 2, pages 1309–1312. IEEE, 2004.

45. Gaige Wang, Lihong Guo, and Hong Duan. Wavelet neural network using multiple wavelet functions in target threat assessment. *ScientificWorldJournal.*, 2013:632437, feb 2013.

46. Chin-Der Wann and S.C.A. Thomopoulos. Unsupervised learning neural networks with applications to data fusion. In *Proc. 1994 Am. Control Conf. - ACC '94*, volume 2, pages 1361–1365. IEEE, 1994.

47. S. Shams. Neural network optimization for multi-target multi-sensor passive tracking. *Proc. IEEE*, 84(10):1442–1457, 1996.

48. Seunghoon Hong, Tackgeun You, Suha Kwak, and Bohyung Han. Online Tracking by Learning Discriminative Saliency Map with Convolutional Neural Network. In *32 nd Int. Conf. Mach. Learn.*, pages 1–10, 2015.

49. Murat Karabatak and M. Cevdet Ince. An expert system for detection of breast cancer based on association rules and neural network. *Expert Syst. Appl.*, 36(2):3465–3469, mar 2009.

50. Zhi-Hua Zhou, Yuan Jiang, Yu-Bin Yang, and Shi-Fu Chen. Lung cancer cell identification based on artificial neural network ensembles. *Artif. Intell. Med.*, 24(1):25–36, jan 2002.

51. A. S. Miller, B. H. Blott, and T. K. Hames. Review of neural network applications in medical imaging and signal processing. *Med. Biol. Eng. Comput.*, 30(5):449–464, sep 1992.

52. Bernard Widrow, David E. Rumelhart, and Michael A. Lehr. Neural networks: applications in industry, business and science. *Commun. ACM*, 37(3):93–106, mar 1994.

53. Bo K Wong, Thomas A Bodnovich, and Yakup Selvi. Neural network applications in business: A review and analysis of the literature (1988–1995). *Decis. Support Syst.*, 19(4):301–320, apr 1997.

54. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

55. François Chollet et al. Keras. `https://keras.io`, 2015.

56. Helgi I. Ingólfsson, Cesar A. Lopez, Jaakko J. Uusitalo, Djurre H. de Jong, Srinivasa M. Gopal, Xavier Periole, and Siewert J. Marrink. The Power of Coarse Graining in Biomolecular Simulations. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 4(3):225–248, 2014.

57. Patrick G. Lafond and Sergei Izvekov. Multiscale Coarse-Graining with Effective Polarizabilities: A Fully Bottom-Up Approach. *Journal of Chemical Theory and Computation*, 14(4):1873–1886, 2018.

58. Nicholas J.H. Dunn and W. G. Noid. Bottom-up coarse-grained models that accurately describe the structure, pressure, and compressibility of molecular liquids. *Journal of Chemical Physics*, 143(24), 2015.

59. Vaidyanathan Sethuraman, Santosh Mogurampelly, and Venkat Ganesan. Multiscale Simulations of Lamellar PS-PEO Block Copolymers Doped with LiPF6 Ions. *Macromolecules*, 50(11):4542–4554, 2017.

60. F Ercolessi and J B Adams. Interatomic Potentials from First-Principles Calculations: The Force-Matching Method. *EPL (Europhysics Letters)*, 26(8):583, 1994.

61. Sergei Izvekov and Gregory A Voth. Multiscale coarse graining of liquid-state systems. *Journal of Chemical Physics*, 123(13), oct 2005.

62. Avisek Das, Lanyuan Lu, Hans C. Andersen, and Gregory A. Voth. The multiscale coarse-graining method. X. Improved algorithms for constructing coarse-grained potentials for molecular systems. *Journal of Chemical Physics*, 136(19), 2012.

63. Joseph F. Rudzinski and William G. Noid. Investigation of coarse-grained mappings via an iterative generalized Yvon-Born-Green method. *Journal of Physical Chemistry B*, 118(28):8295–8312, 2014.

64. Maghesree Chakraborty, Chenliang Xu, and Andrew D. White. Encoding and Selecting Coarse-Grain Mapping Operators with Hierarchical Graphs. *J. Chem. Phys.*, 149(13):134106, apr 2018.

65. Emiliano Brini, Elena a. Algaer, Pritam Ganguly, Chunli Li, Francisco Rodríguez-Ropero, and Nico F. a. Van Der Vegt. Systematic Coarse-Graining Methods for Soft Matter Simulations – A Review. *Soft Matter*, 9(7):2108–2119, 2013.

66. Yi Ling Chen and Michael Habeck. Data-driven coarse graining of large biomolecular structures. *PLoS ONE*, 12(8):1–17, 2017.

67. C. Scherer and D. Andrienko. Comparison of systematic coarse-graining strategies for soluble conjugated polymers. *European Physical Journal: Special Topics*, 225(8-9):1441–1461, 2016.

68. Karteek K. Bejagam, Samrendra Singh, Yaxin An, and Sanket A. Deshmukh. Machine-Learned Coarse-Grained Models. *The Journal of Physical Chemistry Letters*, 9(16):4667–4672, 2018.

69. Linfeng Zhang, Jiequn Han, Han Wang, Roberto Car, and Weinan E. Weinan. DeePCG: Constructing coarse-grained models via deep neural networks. *Journal of Chemical Physics*, 149(3), 2018.

70. Andrew D White and Gregory A Voth. Efficient and Minimal Method to Bias Molecular Simulations with Experimental Data. *J. Chem. Theory Comput.*, 2014.

71. Andrew D. White, James F. Dama, and Gregory A. Voth. Designing Free Energy Surfaces That Match Experimental Data with Metadynamics. *Journal of Chemical Theory and Computation*, 11(6):2451–2460, 2015.

72. Andrew D. White, Chris Knight, Glen M. Hocky, and Gregory A. Voth. Communication: Improved ab initio molecular dynamics by minimally biasing with experimental data. *J. Chem. Phys.*, 146(4):041102, 2017.

73. Dilnoza B Amirkulova and Andrew D White. Combining Enhanced Sampling with Experiment Directed Simulation of the GYG peptide. *J. Theor. Comput. Chem.*, 17(3):1840007, 2018.

74. Thomas Dannenhoffer-Lafage, Andrew D. White, and Gregory A. Voth. A Direct Method for Incorporating Experimental Data into Multiscale Coarse-Grained Models. *J. Chem. Theory Comput.*, 12(5):2144–2153, 2016.

75. Dilnoza B Amirkulova and Andrew D White. Recent Advances in Maximum Entropy Biasing Techniques for Molecular Dynamics. *Submitted*, feb 2019. https://arxiv.org/abs/1902.02252.

76. Benjamin Chu and Tianbo Liu. Characterization of nanoparticles by scattering techniques. *Journal of Nanoparticle Research*, 2(1):29–41, Mar 2000.

77. Stewart F Parker and Paul Collier. Applications of Neutron Scattering in Catalysis Where atoms are and how they move Neutron Properties and their Applications. *Johnson Matthey Technol. Rev*, 60(2):132–144, 2016.

78. Felix Fernandez-Alonso and David L. (David Long) Price, editors. *Neutron scattering : applications in biology, chemistry, and materials science*. Academic Press, 2017.

79. Sven Vogel. A review of neutron scattering applications to nuclear materials. *ISRN Mat. Sci.*, 2013, 08 2013.

80. Jeremy C Smith, Pan Tan, Loukas Petridis, and Liang Hong. Dynamic Neutron Scattering by Biological Systems. *Annual Review of Biophysics*, 47:335 – 354, 2018.

81. Christopher L. Farrow and Simon J. L. Billinge. Relationship between the atomic pair distribution function and small-angle scattering: implications for modeling of nanoparticles. *Acta Crystallographica Section A*, 65(3):232–239, May 2009.

82. Varley F Sears. Special Feature Neutron scattering lengths and cross sectioirn. *Neutron News*, 3(3):26–37, 1992.