JCTC Journal of Chemical Theory and Computation

pubs.acs.org/JCTC Review

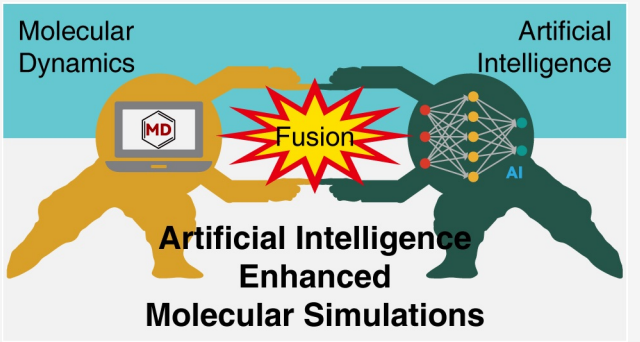
Artificial Intelligence Enhanced Molecular Simulations

Jun Zhang, Dechin Chen, Yijie Xia, Yu-Peng Huang, Xiaohan Lin, Xu Han, Ningxi Ni, Zidong Wang, Fan Yu, Lijiang Yang, Yi Isaac Yang,* and Yi Qin Gao*

Cite This: J. Chem. Theory Comput. 2023, 19, 4338−4350 Read Online

ACCESS | Metrics & More | Article Recommendations

ABSTRACT: Molecular simulations, which simulate the motions of particles according to fundamental laws of physics, have been applied to a wide range of fields from physics and materials science to biochemistry and drug discovery. Developed for computationally intensive applications, most molecular simulation software involves significant use of hard-coded derivatives and code reuse across various programming languages. In this Review, we first align the relationship between molecular simulations and artificial intelligence (AI) and reveal the coherence between the two. We then discuss how the AI platform can create new possibilities and deliver new solutions to molecular simulations, from the perspective of algorithms, programming paradigms, and even hardware. Rather than focusing solely on increasingly complex neural network models, we introduce various concepts and techniques brought about by modern AI and explore how they can be transacted to molecular simulations. To this end, we summarized several representative applications of molecular simulations enhanced by AI, including from differentiable programming and high-throughput simulations. Finally, we look ahead to promising directions that may help address existing issues in the current framework of AI-enhanced molecular simulations.



I. INTRODUCTION

Since their inception, molecular simulations (MS), particularly molecular dynamics (MD), have become a crucial research tool in chemistry, biology, physics, and materials science, among other fields. After decades of development, many molecular simulation packages are available: to name only a few, CHARMM,[1] AMBER,[2] GROMACS,[3] NAMD,[4] LAMMPS,[5] OpenMM,[6] and SPONGE,[7] etc. These high-quality simulation programs are stable, functional, and usually fast, thus gaining a wide user base. However, these well-established simulation packages contain large specialized codebases written in C/C++ or FORTRAN, along with custom CUDA kernels for GPU acceleration,[8] and almost all of them include significant amounts of code duplication and hard-coded gradients. These restrictions make it hard for MS packages to keep pace with the ever-evolving scientific computing techniques that are being propelled by other research areas such as artificial intelligence.

During the development of molecular simulation software, one of the most pressing issues has been synchronizing with the ever-evolving computational infrastructures. Particularly, with the advent of accelerated computational devices, such as GPUs, programs which are not compatible with these accelerators are doomed to fade away.[9] Meanwhile, the popularization of high-performance supercomputer clusters further requires MS software to support large-scale parallel computing. As common practice, codes of simulation programs would have to be substantially changed in order to follow such updates in computational hardware.

Furthermore, a useful molecular simulation package should not be designed merely to increase the speed of programs (more specifically, time per integration step). In fact, for brute-force molecular dynamics simulations, it is unlikely to outperform highly optimized or specially coded MD software or hardware such as AMBER[10,11] and ANTON.[12−14] However, brute-force MD simulations are usually inadequate to cover the time or length scale of interest, and many "rare events" require acceleration, giving rise to a range of enhanced sampling techniques and multiscale modeling strategies (such as coarse graining) based on machine learning and artificial intelligence. Unfortunately, these techniques are often difficult to integrate efficiently with highly optimized MD programs. This difficulty is partly due to the fact that most popular MD simulation software is programmed in C/C++ and FORTRAN, whereas the current popular AI frameworks are Pythonic.

Special Issue: Machine Learning for Molecular Simulation

Received: February 21, 2023
Published: June 26, 2023

https://doi.org/10.1021/acs.jctc.3c00214
J. Chem. Theory Comput. 2023, 19, 4338−4350

Finally, with the growing trend of AI for science, we have witnessed the huge success of AI in solving some scientific challenges like protein folding. However, with rapid advancements in AI, traditional MD tools are no longer able to handle innovative concepts that incorporate deep learning such as AlphaFold,[15] Boltzmann Generator,[16] and machine-learned potentials.[17−20] Consequently, numerous researchers have had to modify existing MD packages or even create new MD engines on AI platforms tailored to their specific needs (e.g., HOOMD-TF,[21] JAX-MD[22] and Torch-MD[23]). This trend highlights the necessity, in numerous scientific fields, to embrace a revolution in infrastructure (including software and hardware) and to align with AI frameworks in order to fully benefit from AI technology.

Even though they serve different purposes, the developments of MD and AI software share significant similarities in their logics and even face similar challenges such as hand-coded derivatives, cumbersome parallelization, incompatibility with state-of-the-art hardware, and a lack of support for user-plugin functions. Despite the expertise of leading MD packages such as AMBER[2] or OpenMM,[6] these issues remain largely unresolved, impeding the development of MD infrastructure to a great extent. Conversely, AI research has made noteworthy strides in addressing these challenges, with many viable solutions already proposed. In Section II, we will first explore the potential of aligning MS with AI and assess the feasibility of unifying AI and MS programming. In Section III, we will present several intriguing applications empowered by AI-enhanced MS. Finally, we will look ahead to the emerging trends that are transforming our research landscape.

II. UNIFYING PROGRAMMING FRAMEWORKS OF AI AND MS

II.1. Coherence between AI and MS. Most MS packages include significant amounts of software-specific codes but written in various programming languages, making it hard to transfer functionalities from one package to another, and many functions or modules are duplicated across MS programs. This state of affairs is reminiscent of machine learning in the early stage before the popularization of relevant techniques like automatic differentiation (AD) and automatic parallelization (AP). Fortunately, the AI community has delivered elegant solutions to these issues, thus driving the evolution of modern scientific computing infrastructure.

The design of any modern AI platform, including TensorFlow,[24] PyTorch,[25] and MindSpore,[26] etc., is mainly oriented by the efficiency and convenience for developing, training, and deploying deep neural network models. As Figure 1 illustrates, during training of a neural network $f(\mathbf{x}; \boldsymbol{\theta})$, where $\mathbf{x}$ and $\boldsymbol{\theta}$ denote the input and parameters of the model, respectively, a scalar loss function $L(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y})$ is specified for optimization. Then, an optimizer updates the model parameters according to the gradient of this loss function $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$. While less apparent, MD simulations indeed share many similarities to AI training. In classical MD simulation, the potential energy $U(\mathbf{R})$ is a scalar function of the particle coordinates $\mathbf{R}$ of the molecular system. An integrator is employed to update the particle coordinates $\mathbf{R}$ according to the forces $\mathbf{F}(\mathbf{R}) = -\nabla_{\mathbf{R}} U(\mathbf{R})$, which is the negative gradient of the potential energy function. From this perspective, MD simulation can be considered as a special case of AI training: The particle coordinates $\mathbf{R}$ are in analogy to the neural network parameters $\boldsymbol{\theta}$; the potential energy $U(\mathbf{R})$ is parallel to the loss function $L(\boldsymbol{\theta})$; and the MD integrator is equivalent to the AI optimizer.

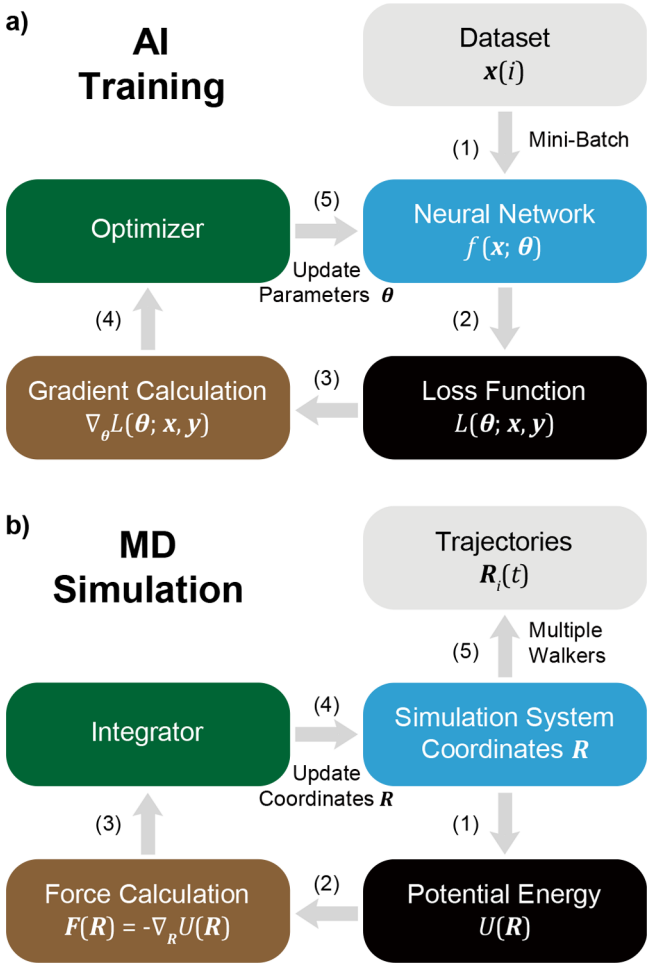


Figure 1. Comparison between AI training and MD simulation. (A) The process of AI training. (B) The process of MD simulation.

Given the similar logic between the two, molecular simulation programs can be naturally reinterpreted and wrapped within an AI framework. Several teams have attempted to merge AI with MS (e.g., HOOMD-TF[21]) or transplant MS into an AI platform (e.g., JAX-MD[22] and Torch-MD[23]). Particularly, the authors have initiated an effort toward a conceptually new AI-enhanced simulation package MindSPONGE[27] based on a modern AI platform MindSpore. Unlike most MS packages, MindSPONGE is "transparent" to developers given its modular architecture and Python API. Hence, it is effortless for developers to integrate or introduce new functionalities at their own desire. Moreover, by utilizing the advanced techniques equipped by MindSpore, MindSPONGE provides several compelling features including differentiable programming, high-throughput computing, and automatic hardware transferability. In the following we will elaborate on these key features and provide exemplary applications in Section III.

II.2. Automatic Differentiation (AD) and Differentiable MS. By its definition, MD simulation seeks to simulate the dynamics of particles according to fundamental physical laws. In Hamiltonian systems, this process typically involves computing the forces or the coordinate derivatives of an energy function. Consequently, researchers trying a new idea often

https://doi.org/10.1021/acs.jctc.3c00214
J. Chem. Theory Comput. 2023, 19, 4338−4350

have to spend significant effort in computing the derivatives and integrating them into the large and specialized codebases. Worse still, due to the variety of "energy functions", MS packages have diverged for applications or tasks, including protein structure optimization, molecular docking, and MD simulations, largely because the forms of hardcoded forces are different across cases. For goals such as virtual screening of drugs starting from protein structure prediction, such divergence makes it nontrivial to assemble a self-contained pipeline within a single MS package.

Automatic differentiation is particularly relevant in the application of force fields for molecular modeling. Most current machine-learned[28] force fields or potential functions, including those based on descriptors[20,29−31] and deep graph neural networks,[32−36] require calculation of atomic forces through automatic differentiation. This allows for the development of relevant models to be programmed solely for potential energy, eliminating the need for additional code for atomic forces.[37,38] Additionally, the automatic differentiation technique provides a "top-down" approach to optimize force field parameters directly from experimental data,[39−41] as opposed to the traditional "bottom-up" approach of fitting a set of force field models from quantum mechanical data.

Automatic differentiation has a rich history in machine learning as well as the physical sciences.[42] Given AD, backpropagation[43] has become the core algorithm in the recent explosion of machine-learning research, enabled by packages like TensorFlow. While still generally focused on machine learning, more recent packages have made automatic differentiation more generally available as a computational tool, e.g., Autograd,[44] JAX,[45] and Zygote.[46] In MindSPONGE, the gradients of an energy function can be computed via AD, so users may simply specify the forms of the energy function and no longer need to manually code the forces. Such a feature makes it possible to develop and apply complex energy functions such as neural network or machine-learned potential[17] to accelerate or calibrate MS without manually deriving the derivatives of these functions.

Besides, collective variables (CVs) are widely adopted in MS, which are functions of the particle coordinates in a molecular system. The selection of CV is crucial to capture relevant degrees of freedom of the model being simulated.[47] This is especially important when employing enhanced sampling techniques, such as umbrella sampling[48] and metadynamics,[49] which apply biasing forces over CVs to enhance the sampling of rare events. CVs may be simple geometrical observables like distances and torsions, but often they are more complex functions designed to capture structural features, such as experimental observables and crystal symmetries,[50,51] etc. The computation of the gradients or Jacobians of CV with respect to the particle coordinates, which can be formidably complicated, is essential for most of the CV-dependent molecular simulations. In this regard, some specialized programs like the PLUMED library[52,53] and the COLVARS module[54] were developed and provided access to certain predefined sets of differentiable CVs. Nevertheless, such libraries are usually coded in relatively low-level programming language and require explicit compiling, making it error-prone and difficult for users to explore new CVs. Thanks to AD, crafting a complex CV and running a CV-dependent simulation in MindSPONGE are as simple as running a vanilla simulation in a traditional MD package. We will provide an example of running Metadynamics in MindSPONGE in Section III.

II.3. Automatic Parallelization (AP) and High-Throughput MS. Gains in computing power are increasingly attributed to device parallelism rather computing speed. For instance, GPUs are designed to process large amounts of data in parallel, and NPUs or TPUs take this further by offering high-speed interconnects between chips. This parallelism is often used to simulate increasingly larger systems. In deep learning, mini-batch gradient descent is employed to train neural networks, necessitating the parallel computation of loss functions over batches of samples. To meet this need, an automatic parallelization (AP) technique was developed. With an AP-enabled AI framework, users no longer need to manually code or allocate resources for parallelizable computations. Instead, users can simply "vectorize" these processes, for example, by just adding a "batchwise" dimension to the relevant tensors or arrays involved in the computation. This vectorization, together with automatic parallelism, makes the code for parallel computing easier to be programmed, thus facilitating the handling of large-scale systems.

While batchwise operation or "vectorization" is natural to deep learning, other interesting uses of parallelism have received less attention. Many molecular simulation methods (e.g., replica exchange,[55] multiple-walkers,[56] and nudged elastic band[57]) involve simulating an ensemble of states simultaneously. Given that the shape of the particle coordinates $\mathbf{R}$ in a single molecular system is $(A, 3)$ (where A is the number of particles and 3 is the Cartesian coordinates), if B different replicas of the system are stacked into a vectorized array of shape $(B, A, 3)$, it is then possible to update these B sets of coordinates simultaneously via AP, making it just as straightforward to run multiple-replica simulations as it is to run single-replica ones.

For traditional MS packages, the implementation of multiple-replica[55] or multiple-walker[56] simulations is nontrivial and often restricted by hardware. For example, for CPU-hosted simulation packages, the number of replicas cannot exceed the number of CPU cores. For GPU-hosted software, parallel computing is even more challenging and necessitates specialized coding practice. These restrictions make it highly difficult to implement high-throughput computing in traditional MS software. In contrast, with the help of an AP-enabled MS platform, it is possible to carry out multiple trajectories of simulations simultaneously on a single GPU or NPU, which is impossible for most traditional MD simulation software. If deployed on a high-performance computing cluster consisting of multiple GPUs or other AI accelerators, even higher throughput can be achieved for molecular simulations. Therefore, incorporation of AP will substantially democratize high-throughput MS.

II.4. Just-in-Time (JIT) Compilation and Extensible MS. One widely concerned issue of a MS package is its extensibility or its support for user-contributed functionalities. Since most simulation packages are not modularized, users have to dive into the core part of the package and modify the code as needed. This error-prone process frustrates most of the attempts. Such kind of programming convention makes it difficult for code reuse, and consequently, various packages arise for different functionalities and make the community diverge. The reason behind such divergence is rooted in the dilemma between functional flexibility (or extensibility) and computational efficiency. Since most of the traditional

https://doi.org/10.1021/acs.jctc.3c00214
J. Chem. Theory Comput. 2023, 19, 4338−4350

Chart 1. Code Snippet 1: (Left) Training an AI Model in MindSpore; (Right) Running a MD Simulation in MindSPONGE

```
from mindspore.nn import WithLossCell
from mindspore.train import Model
from mindspore.train.callback import LossMonitor
from mindspore.train.callback import
ModelCheckpoint, CheckpointConfig

# Wrap loss function with neural network
net_with_loss = WithLossCell(network, loss_fn)

# Set optimizer
ai = Model(net_with_loss, optimizer=opt)

# Callback functions
monitor_cb = LossMonitor(16)
ckpoint_cb = ModelCheckpoint('network',
config=CheckpointConfig(32))

# Start training
ai.train(1000, dataset, callbacks=[monitor_cb,
ckpoint_cb])
```

```
from mindsponge import WithEnergyCell
from mindsponge import Sponge
from mindsponge.callback import RunInfo
from mindsponge.callback import WriteH5MD

# Wrap potential energy with simulation system
sys_with_ene = WithEnergyCell(system, potential)

# Set integrator
md = Sponge(sys_with_ene, optimizer=integrator)

# Callback functions
run_info = RunInfo(10)
cb_h5md = WriteH5MD(system, 'traj.h5md',
save_freq=10)

# Start running
md.run(1000, callbacks=[run_info, cb_h5md])
```

packages are coded in a highly optimized fashion, modifications to the codes are highly restricted. The entanglement between performance and functions not only hinders the extensibility of MS packages but also limits their compatibility with hardware. For instance, when GPU is upgraded, usually the entire codes need redevelopment to match a different driver version (e.g., CUDA), which is costly and time-consuming. Besides, since MS packages are differently coded for CPU and GPU, it is almost impossible for users to plug-in a new function which can be executed in both CPUs and GPUs, let alone new hardware like NPUs.

To strike a balance between performance and extensibility, MindSPONGE adopts a modular architecture, which is commonly used in modern AI frameworks. This architecture separates the functionality parts from the performance support. The functional modules are exposed to users through Python API, allowing them to easily incorporate available functionalities or develop and plug-in new ones. Thanks to MindSpore and CANN,[26] these Pythonic functional modules can be just-in-time (JIT) compiled to any supported host device (e.g., CPU, GPU, or NPU) and automatically exploit the acceleration of advanced parallel-computing devices. Therefore, JIT liberates users from a hardware-specific coding paradigm and allows them to achieve on-hardware acceleration through convenient Python code, while guaranteeing flexible feature extensions.

III. APPLICATIONS OF AI-ENHANCED MOLECULAR SIMULATIONS (AIMS)

In this section, we will present several compelling applications which integrate AI and MS and provide novel solutions to some long-standing challenges in the fields of chemistry and biophysics. Nevertheless, these applications necessitate certain functions that go beyond the scope of traditional molecular simulation software. For instance, sampling of rare events in MD simulations can be dramatically speeded up by machine-learned potentials, but this was unfeasible unless the simulation engine could access the force or gradients of a neural network potential. Consequently, before delving into the particular applications which leverage the power of AD and AP, we shall first introduce how these capabilities are seamlessly incorporated into an AI-styled MS package such as MindSPONGE.

III.1. Running AI-Styled Molecular Simulations.

We start from a brief introduction to MindSpore,[26] which is a popular framework of training and deploying deep neural network and other AI models. Code Snippet 1 (Chart 1) shows an example of training an AI model in MindSpore. Three modules are required for neural network training using MindSpore: the neural network, the loss function, and the optimizer. In Code Snippet 1 (Chart 1), the "network" object corresponds to a neural network model, whereas the "loss_fn" object specifies how the loss function should be computed, and "opt" object is the optimizer for updating the model parameters. To execute AI training, one first needs to wrap the neural network $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$ with the loss function $L(\mathbf{y})$ using the "WithLossCell" class, which creates a "net_with_loss" object and computes the value of the loss given the output of the neural network: $L(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y})$. The "model" class then computes and passes the gradient of loss yielded by "net_with_loss" object to the "opt" object. Finally, we can perform the training by calling the method (member function) "train()" of the "Model" and monitor the training process with callback functions.

Given the analogy between AI training and molecular simulations, the code of running a simulation in MindSPONGE mirrors the above example as provided in the right panel of Code Snippet 1 (Chart 1). The three objects, "system", "potential", and "integrator", correspond to the three modules for simulation in MindSPONGE: the simulation system, the potential energy, and the integrator, respectively. MindSPONGE wraps the simulation system (including coordinate $\mathbf{R}$) with the potential energy $U(\mathbf{R})$ as a "WithEnergyCell", which we can then package together with the integrator into the main program "Sponge". It is also possible to use callback functions to monitor and record the simulation process and save the simulation trajectory as a structured compressed file.

Since an AI model often requires multiple training instances (e.g., minibatch stochastic gradient descent), it is natural to feed more than one datum simultaneously in MindSpore. Likewise, it is also straightforward to run multiple-replica simulations in MindSPONGE by executing the code with automatic parallelization. This feature allows high-throughput molecular simulations even on a single-computing device like a

https://doi.org/10.1021/acs.jctc.3c00214
J. Chem. Theory Comput. 2023, 19, 4338−4350

GPU, which becomes the cornerstone for the applications of Sections III.2 and III.3 as well as many other AI-enhanced molecular simulations.

As a compelling feature, MindSPONGE utilizes automatic differentiation (AD) to compute derivatives, eliminating the need to program the gradients of the potential energy function with respect to the atomic coordinate, i.e., the atomic forces. It is similar to calculating the gradient of the loss function with respect to the parameters of a neural network via AD. As a result, programming and calling energylike functions in MindSPONGE become very simple. For example, it is convenient to call CV-dependent enhanced sampling methods in MindSPONGE and apply biasing potentials. Using "WithEnergyCell" allows loading of additional functional modules for MD simulations, such as bias potential.

While various enhanced sampling techniques have been developed to accelerate "rare events" in MD simulations in order to cover the time or length scale of interest, unfortunately, these techniques are often difficult to integrate efficiently with highly optimized MD programs. However, an AI-enhanced molecular simulation platform can easily develop and improve them, making molecular simulations considerably more efficient. For example, here is a sample code for loading the metadynamics[49] enhanced sampling method in MindSPONGE.

In the following, we presented a code snippet demonstrating the use of metadynamics,[49,58] a widely used method for enhanced sampling and free energy calculation (Chart 2). In

Chart 2. Code Snippet 2: Running Metadynamics in MindSPONGE

```
from mindsponge.colvar import Torsion
from mindsponge.potential.bias import Metadynamics

# Collective variables
phi = Torsion([4, 6, 8, 14])
psi = Torsion([6, 8, 14, 16])

# Metadynamics
metad = Metadynamics(
    colvar=[phi, psi],
    height=2.5,
    sigma=0.05,
    grid_min=-PI,
    grid_max=PI,
)

sys_with_ene = WithEnergyCell(system, potential,
bias=metad)
md = Sponge(sys_with_ene, optimizer=opt,
metrics={'phi': phi, 'psi': psi})
```

traditional simulation packages, metadynamics relies on specialized plug-in libraries and is not compatible with high-speed host devices like GPUs. Consequently, running metadynamics in programs such as AMBER,[10,11] GROMACS,[59] or LAMMPS[60] is not efficient. MindSPONGE, on the other hand, allows for metadynamics to be executed efficiently on GPUs and NPUs with just a single code block. The AI-enhanced simulation package presented in this study has indeed significantly improved program speed.

Similar to the above example, other energy functions like machine-learned force fields or bias potentials can be easily plugged into molecular simulations via MindSPONGE, without the need of manually coding the derivatives or forces. Such a feature allows calibration and flexible manipulation of the Hamiltonian systems as needed, and it is crucial to many applications including examples in Sections III.3 and III.4, which will be elaborated on later.

III.2. Unsupervised AIMS Zooms-in a Protein-Folding Landscape. Many molecular dynamic processes in chemistry and biology, e.g., chemical reaction, protein folding, and ligand binding, etc., can be described by a clustered free energy landscape (FEL)[61] consisting of many local free energy minima which correspond to the metastable states. This picture of a clustered FEL (or the metastability) is the cornerstone of many kinetic models dealing with diffusive and complex dynamics.[62−64] A simplified and informative visualization of a complex FEL amenable for downstream tasks, such as clustering, is often required by these kinetic models.[65−67] Given the high-throughput simulation capability empowered by AIMS, Zhang et al. proposed a parametric learning approach, called Information Distilling of Metastability (IDM),[68] to perform clustering in the meantime of reducing the dimensionality for molecular systems. IDM is end-to-end differentiable and thus scalable to an ultralarge data set. IDM requires neither a cherry-picked distance metric nor the ground-true number of clusters defined a priori, and it can be used to unroll and zoom-in the hierarchical FEL with respect to different time scales. The code of IDM has been published and is accessible via MindSPONGE.[69]

IDM is based on the idea of information codistillation[70] originating from the machine-learning community. Given two observations $\mathbf{x}$ and $\mathbf{x}'$ belonging to the same metastable state, one can find a function χ capturing what is in common between them by maximizing the mutual information (MI), $I(\chi(\mathbf{x}), \chi(\mathbf{x}'))$. In order to avoid the trivial solution of χ being the identity function, χ is confined to the family of classification functions (or indicator functions) with finite categories. Consequently, the entropy of χ is upper bounded, and clustering can thus be achieved by maximizing the following closed-form mutual information:

$$I_\beta(\chi(\mathbf{x}), \chi(\mathbf{x}')) = \sum_{c,c'=1}^{K} \langle \chi_c(\mathbf{x}), \chi_{c'}(\mathbf{x}') \rangle \times \ln \frac{\langle \chi_c(\mathbf{x}), \chi_{c'}(\mathbf{x}') \rangle}{\langle \chi_c(\mathbf{x}) \rangle^\beta \langle \chi_{c'}(\mathbf{x}') \rangle^\beta} \quad (1)$$

where a controlling hyperparameter β is introduced without loss of generality. $\chi_c(\mathbf{x})$ denotes the cth entry of χ corresponding to the probability of $\mathbf{x}$ belonging to cluster c. To generate samples belonging to the same cluster of $\mathbf{x}$, high-throughput MD simulations are launched to generate short trajectories initiated from $\mathbf{x}$, and the resulting temporal neighbors of $\mathbf{x}$ are collected as $\mathbf{x}'$. From a machine-learning perspective, in IDM, the MD engine serves as high-throughput data augmentation which constructs contrastive samples for the input provided by the user, while a deep neural network is trained to distill the information between the paired data, which is a common strategy in modern contrastive learning. The code of IDM has been integrated into the MindSPONGE repository.

Intuitively, eq 1 means that the clustering mapping of $\mathbf{x}$, namely $\chi(\mathbf{x})$, should be maximally informative with its temporal neighbors. In practice, this is obtained when $\chi(\mathbf{x}) \approx \chi(\mathbf{x}')$ for arbitrary $\mathbf{x}$. Since eq 1 is a variational objective, one is motivated to adopt deep neural networks as the clustering

https://doi.org/10.1021/acs.jctc.3c00214
J. Chem. Theory Comput. 2023, 19, 4338−4350

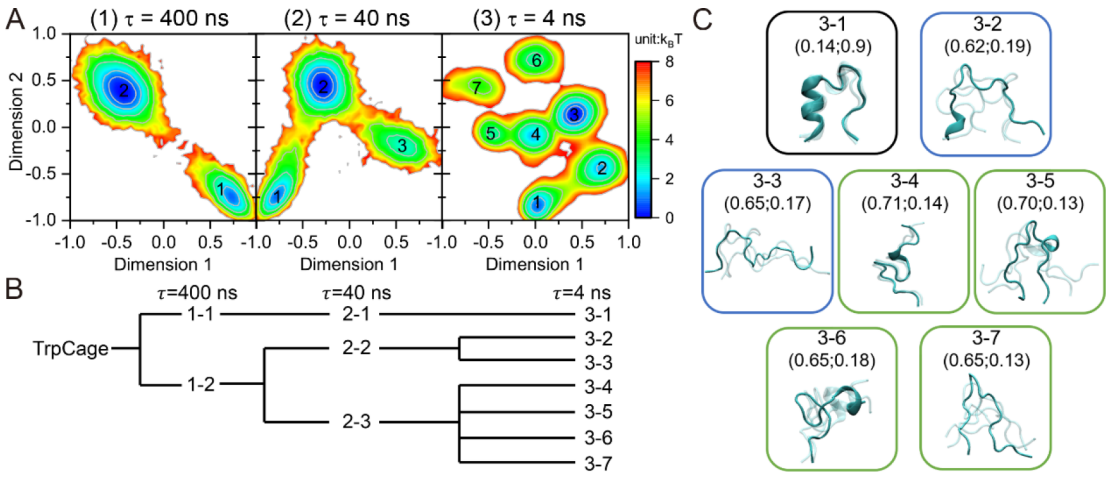


Figure 2. Unrolling the folding landscape of TrpCage. (A) Reduced free energy landscape of TrpCage obtained by IDM on different time scales. From left to right: (1) τ = 400 ns, (2) τ = 40 ns, and (3) τ = 4 ns. Metastable states are indexed accordingly. (B) The dendrogram tracking the hierarchy of different metastable states identified by IDM. (C) Representative structures of different metastable states, including the folded state (3-1 in black box), folding intermediates (3-2 and 3-3 in blue boxes), and denatured structures with non-native local patterns (3-4 to 3-7 in green boxes). For each metastable state, three structures to the nearest of the centroid are superimposed. Numbers in parentheses are the RMSD (unit: nm) and Q-value, respectively. Figures were reproduced with permission from Zhang et al. *J. Phys. Chem. Lett.* **2019**, *10* (18), 5571−5576. Copyright 2019 American Chemical Society.

function χ because they usually offer sufficient expressivity for high-dimensional data. Noteworthy, IDM does not require a reference to the exact number of ground-true clusters. Instead, IDM robustly clusters the data as long as the maximal allowed number of clusters, K in eq 1, is large enough. Besides, IDM yields soft cluster labels $\chi(\mathbf{x})$, which are preferred in many scenarios to hard ones. Finally, trained upon N decreasing temporal neighborhood sizes ($\tau_1 \gg \tau_2 \gg \cdots \gg \tau_N$), IDM allows one to zoom-in the FEL with increasing time resolution; hence, IDM is able to extract the hierarchy of the FEL according to time scales.

For illustration, IDM was applied to a fast-folding protein TrpCage (Figure 2) and revealed molecular details of its hierarchal folding mechanisms.[68] On the longest time scale (i.e., τ = 400 ns), IDM projects the FEL onto a reduced representation consisting of two distinguishable metastable states (Figure 2A, panel 1) and categorizes the protein conformers into two clusters accordingly. For each cluster the averaged root-mean-squared-deviation (RMSD) with respect to the native structure and the fraction of native contacts (Q-value)[71] are calculated, respectively. Cluster 1-1, with a high Q-value and low RMSD, is identified as the native state, whereas Cluster 1-2 corresponds to the denatured state. In line with intuitions, the native state forms a narrow and sharp local minimum on the IDM embedding surface, while the unfolded states spread more widely over the space (Figure 2A, panel 1). This two-state picture agrees well with the observations that the folding/unfolding events of proteins can be viewed and measured as a two-state kinetic process on relatively long time scales. An intriguing finding is that, even if the time resolution is raised up to τ = 4 ns, no further division of the native state is observed (Figure 2B and 2C), indicating no discernible slow relaxations within the native state. This result strongly corroborates the common belief that proteins exhibit a stable and well-defined native structure whose fluctuations (and the conformational entropy) are rather limited. In contrast, some slow relaxation processes within the denatured state can be reconsidered as interstate transitions. Consequently, more unfolded metastable states can be identified. Specifically, at the resolution of τ = 40 ns, the unfolded state (Cluster 1-2) divides into two distinguishable clusters (indexed by 2-2 and 2-3, respectively; Figure 2B), and State 2-2 is more closely connected to the native state while State 2-3 is farther away (Figure 2B, panel 2), implying that State 2-2 is likely to be located on the folding pathway while State 2-3 may be less relevant to folding. This hypothesis is confirmed by the IDM embedding achieved at τ = 4 ns (Figure 2A, panel 3): Cluster 2-2 bifurcates into States 3-2 and 3-3 (Figure 2B), and the former corresponds to a folding bottleneck where some characteristic contacts start to form (e.g., the α-helix), while the latter is composed of extended random coils which represent the fully denatured states (Figure 2C). On the other hand, Cluster 2-3 is divided into four metastable states (indexed from 3-4 to 3-7; Figure 2B): Most of them exhibit some locally formed contacts and structural patterns absent in the native structure (Figure 2C), but these potentially folding traps only constitute a small fraction of the denatured conformation ensemble, as expected for a fast-folding protein. These results echo the well-known funnel landscape theory[72] and demonstrate that we can exploit IDM to unroll and project the funnel energy landscape of protein in a hierarchical manner. Therefore, IDM may be applied to shed more light on the mechanisms of protein folding.

III.3. Reinforced AIMS Reveals Transition States of Chemical Reactions. Being able to explain and predict reaction mechanisms will add insights to the relevant theories and is essential for manipulating reactions and designing catalysts. One widely applied approach in the study of reaction mechanisms is to introduce the reaction coordinate(s) (RC)[73,74] and transition states (TSs),[73,75] which give rise to a simplified description of reduced dimensionality. However, since real-world chemical processes usually take place in a high-dimensional space, which includes redundant coordinates apart from those delimiting reactants and products, extracting the relevant factors that characterize the reaction can be very difficult. Combining reinforcement learning (RL) and MD simulations, Zhang et al. proposed a machine-learning

https://doi.org/10.1021/acs.jctc.3c00214
J. Chem. Theory Comput. 2023, 19, 4338−4350

approach, called RL‡, to automatically unravel chemical reaction mechanisms.[76]

In RL‡, locating the transition state of a chemical reaction is formulated as a game: First a coordinate from the phase space is randomly chosen according to a certain distribution p_{shoot} (eq 2) induced by the potential energy function $U(\mathbf{R})$ and a policy function (or biased potential) $V_\theta(\mathbf{R})$

$$p_{\text{shoot}} \propto \exp(-\beta(U(\mathbf{R}) + V_\theta(\mathbf{R}))) \tag{2}$$

Then, two paired trajectories with opposite momenta from this initial coordinate are shot under a microcanonical ensemble.[75,77]

Though the mathematics behind RL‡ is straightforward, implementing the entire workflow within current AI or MD platforms poses significant challenges. In order to conduct RL‡, a multitude of short MD trajectories must be launched and feedback shall be collected with high throughput, much like a gaming engine in other RL settings. Unfortunately, no AI platforms currently support this process. Additionally, the AI agent's policy (i.e., the sampling probabilities or MD force fields) must be manipulated through neural networks, a feature not natively provided by popular MD programs. These two processes are mutually dependent and therefore must be executed simultaneously, necessitating that both MD and AI models be run on the same GPU device. To effectively implement RL‡, it will be necessary to completely re-engineer both MD and AI software and hardware or utilize an AIMS package such as MindSPONGE.

By virtue of AIMS, these shooting moves can be efficiently done with high throughput. Any of the trajectories ends once it reaches the product (or reactant) region or it exceeds the maximal allowed simulation time. Now consider the outcomes of the shooting move: if one of the paired trajectories ends in the reactant region while the other in the product, we win the game; otherwise, we lose it. Two functions are optimized to iteratively improve the chance of winning the game, one for value estimation and the other for policy making. Both functions can be approximated by deep neural networks.

The value function estimates the winning chance (i.e., the probability of a successful shooting) from an initial configuration $\mathbf{R}$, which can also be interpreted as the "transition path probability", $p(\text{TP}|\mathbf{R})$. Configurations maximizing $p(\text{TP}|\mathbf{R})$ constitute the separatrix (or TS) of the reaction. Instead, we can optimize $p_w(\text{TP}|\mathbf{R})$ by minimizing the following contrastive loss

$$\mathcal{L}(w; \mathbf{R}_i) = -\log \frac{p_w(\mathbf{R}_i)}{p_w(\mathbf{R}_i) + \sum_{j=1,N} p_w(\mathbf{R}_j)} \tag{3}$$

where $\mathbf{R}_i$ is an initial configuration corresponding to a successful shooting while $\mathbf{R}_j$ corresponds to failed ones, and $p_w(\mathbf{R})$ is short for $p_w(\text{TP}|\mathbf{R})$. Equation 3 effectively balances the positive and negative feedbacks with optimization of p_w by shooting merely one or a few trajectories from a given configuration.

The policy function $V_\theta(\mathbf{R})$ in eq 2 is used to manipulate the sampling distribution. With automatic differentiation in AIMS, one can adopt complex functional forms like deep neural networks for $V_\theta(\mathbf{R})$ and variationally optimize $V_\theta(\mathbf{R})$ with respect to a properly defined target distribution p_T, by minimizing the Kullback–Leibler divergence (or relative entropy) between p_T and p_{shoot}. To strike better balance between exploration and exploitation, the target distribution p_T takes the form of eq 4:

$$\log p_T = \alpha \log p_{\text{exploit}} + (1 - \alpha)\log p_{\text{explore}} \tag{4}$$

where $p_{\text{exploit}} \propto p_w(\text{TP}|\mathbf{R})$, and p_{explore} is often a uniform distribution. $0 \leq \alpha \leq 1$ is an inverse-temperature-like hyperparameter trading-off exploration and exploitation: When α is close to zero, the sampling is dominant by exploration, while if α approaches unity, the sampling will focus merely on high-valued (transition-state-like) configurations.

By virtue of RL‡, one can directly interpret the reaction mechanism according to the value function. Meanwhile, the policy function allows efficient sampling of the transition path ensemble, which can be further used to analyze reaction dynamics and kinetics. To benchmark whether RL‡ is able to discover the transition state and unravel the correct reaction mechanism from scratch, RL‡ was applied to a textbook reaction, the substitution between Cl^- and CH_3Cl (Figure 3A),

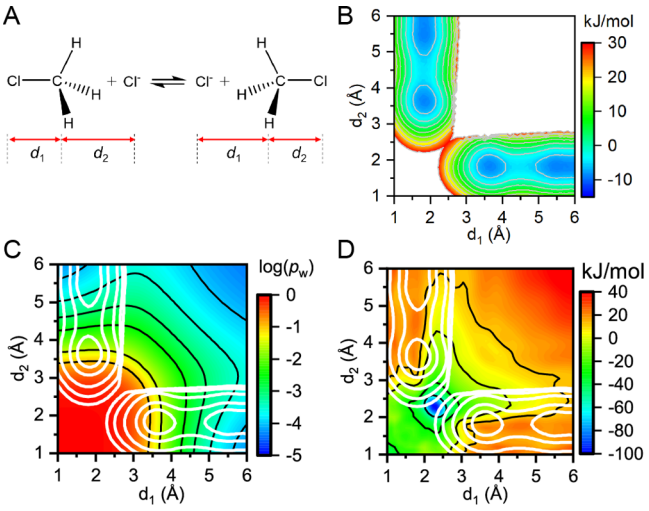


Figure 3. RL‡ for the S_N2 reaction. (A) Scheme of the reaction, where the two C–Cl bond distances are denoted by d_1 and d_2, respectively. (B) A contour plot for the reference free energy surface (FES) of the reaction. (C) A contour plot for the optimized value function $p_w(\text{TP}|\mathbf{s})$ in logarithm scale with $\mathbf{s} = (d_1, d_2)$. The reference FES of the reaction is shown as white contours in the background. (D) A contour plot for the optimized policy function $V_\theta(\mathbf{s})$. The reference FES of the reaction is shown as white contours in the background. Figures were reproduced with permission from Zhang et al. *Phys. Chem. Chem. Phys.* **2021**, *23* (11), 6888–6895. Copyright 2021 Royal Society of Chemistry.

which is known to undergo a typical S_N2 mechanism.[76] The two C–Cl bond distances were selected as order parameters $\mathbf{s} = (d_1, d_2)$ to define the reactant and product (Figure 3A), and a reference free energy surface spanned over $\mathbf{s}$ is shown in Figure 3B. The value function $p_w(\text{TP}|\mathbf{s})$ and policy function $V_\theta(\mathbf{s})$ selectively input $\mathbf{s}$ rather than the coordinates of the whole system. RL‡ was trained to improve the chance of successful shootings to exceed 10% according to the final p_{shoot} (eq 1). After training, the optimized value function $p_w(\text{TP}|\mathbf{s})$ (Figure 3C) sketches the TS regions in the lower-left diagonal regions of the (d_1, d_2) space, indicating a concerted (i.e., S_N2) mechanism which agrees well with textbook knowledge. The bias potential $V_\theta(\mathbf{s})$ obtained by RL‡ (Figure 3D) is

https://doi.org/10.1021/acs.jctc.3c00214
J. Chem. Theory Comput. 2023, 19, 4338–4350

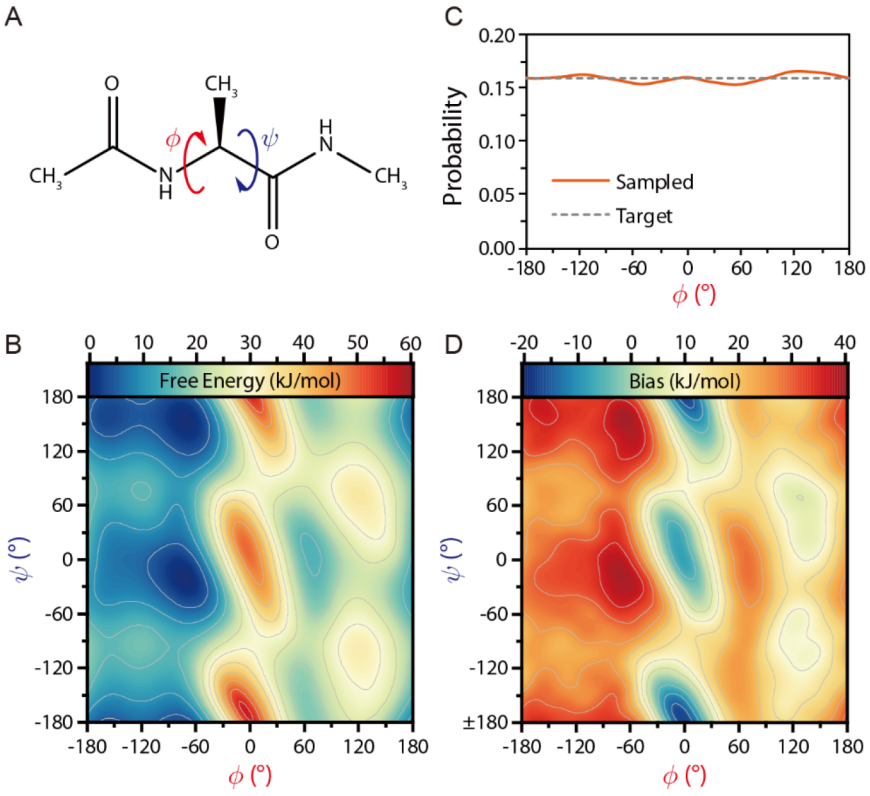


Figure 4. Targeted adversarial learning optimized sampling (TALOS) of Ala2 in explicit solvent. (a) Torsional angles (ϕ, ψ) are slowly changing variables leading to configurational transitions, based on which we built bias potentials. (b) The original 2D FES over (ϕ, ψ) space. (c) For system A, a uniform 1D target distribution is defined along ϕ (gray dashed), and the converged biased distribution projected is shown as a red solid line. (d) For system B, the converged 2D biased distribution learned according to a uniform 2D target distribution defined over (ϕ, ψ). (e) The optimized 2D bias potentials for System A (upper panel) and System B (lower panel), respectively. Figures were reproduced with permission from Zhang et al. *J. Phys. Chem. Lett.* **2019**, *10* (19), 5791−5797. Copyright 2019 American Chemical Society.

reminiscent of the harmonic potential commonly adopted in umbrella sampling, demonstrating that RL‡ is able to automatically create an external potential that effectively biases and concentrates the sampling over the TS regions. This example shows that by combining the strengths of both knowledge-based and physics-based methods, RL‡ can efficiently simulate and extract mechanisms underlying chemical transitions involving high free energy barriers.

III.4. Generative AIMS Explores a Complex Free Energy Landscape. Many dynamic events of interest in molecular systems, e.g., phase transitions, protein folding, and ligand binding, are well out of the reach of brute-force simulations, and enhanced sampling methods are largely needed to this end.[48,49,55,78−81] Zhang et al. recently reformulated the enhanced sampling problem as a distribution learning problem and proposed a *minimax* game to solve it. The proposed approach is called targeted adversarial learning optimized sampling (TALOS).[82]

TALOS simultaneously trains a bias potential, $V_\theta(\mathbf{x})$, which is applied to a simulation engine like MD (which plays a role as the "generator"), and a discriminator $D_w(\mathbf{q}(\mathbf{x}))$ that attempts to differentiate samples generated by the biased sampler from those drawn from a desired target distribution $p(\mathbf{q})$ (where $\mathbf{q}$ is a descriptor as a function of $\mathbf{x}$). Note that θ and w denote the trainable parameters of two separate deep molecular models. The biased sampler receives a reward when it fools the discriminator, whereas the discriminator is rewarded when it successfully classifies a sample as from the "real" target distribution or the "fake" biased simulation. Similar to generative adversarial networks (GANs),[83] the training ends when the *Nash equilibrium* of the game is reached. Inspired by Wasserstein-GAN,[84] the entire training process of TALOS is equivalent to solving the following *minimax* problem:

$$\min_{V_\theta} \max_{D_w \sim \mathbb{L}1} [\langle D_w(\mathbf{q}) \rangle_{p(\mathbf{q})} - \langle D_w(\mathbf{q}) \rangle_{p_\theta(\mathbf{q})}] \tag{5}$$

where $\max_{D_w \sim \mathbb{L}1} [\langle D_w(\mathbf{q}) \rangle_{p(\mathbf{q})} - \langle D_w(\mathbf{q}) \rangle_{p_\theta(\mathbf{q})}]$ is the Wasserstein-1 distance between the target distribution p and the biased distribution $p_\theta \propto \exp(-\beta(U(\mathbf{x}) + V_\theta(\mathbf{x})))$. $D_w(\mathbf{q})$ belongs to the family of 1-Lipschitz functions mapping the descriptor space $\mathbf{q}$ to the real space. However, the analytical form of the optimal D_w is unknown, so a neural network conforming to a Lipschitz restraint, parametrized by w, is employed to approximate it following the gradient

$$\nabla_w L(w) = \langle \nabla_w D_w(\mathbf{q}) \rangle_{p_\theta} - \langle \nabla_w D_w(\mathbf{q}) \rangle_p \tag{6}$$

Once the discriminator D_w is fixed, the bias potential $V_\theta(\mathbf{x})$ is optimized following the gradient

$$\nabla_\theta L(\theta) = \langle (D_w(\mathbf{q}) - \langle D_w(\mathbf{q}) \rangle_{p_\theta}) \beta \nabla_\theta V_\theta(\mathbf{x}) \rangle_{p_\theta} \tag{7}$$

where β is the inverse temperature factor. Equations 6−7 can also be interpreted in terms of RL: eq 6 is parallel to the training of a Critic which is used to approximate the value function of being in a state $\mathbf{q}(\mathbf{x})$, whereas eq 7 is analogous to the famous REINFORCE algorithm[85] which maximizes the expected value $\langle D_w(\mathbf{q}) \rangle_{p_\theta}$ for the Actor who visits state $\mathbf{q}(\mathbf{x})$ with a probability proportional to $\exp(-\beta(U(\mathbf{x}) + V_\theta(\mathbf{x})))$.

https://doi.org/10.1021/acs.jctc.3c00214
J. Chem. Theory Comput. 2023, 19, 4338−4350

Noteworthy, by virtue of the equipped autodifferentiation in AIMS, one can adopt complex functions as V_θ and manipulate the simulations by the biasing forces $\nabla_{\mathbf{x}} V_\theta(\mathbf{x})$ without manually deriving this complex gradient.

As an example, TALOS was applied to the all-atom MD simulation of alanine dipeptide in explicit water (Figure 4A).[82] This is a prototypical biomolecular system consisting of several metastable states, and TALOS is adopted to boost the configurational transitions of (ϕ, ψ) torsional angles (Figure 4B). During simulation, TALOS was trained toward a 1D uniform target distribution of ϕ (Figure 4C), while the system was boosted by a bias potential $V_\theta(\phi, \psi)$ which modifies the 2D (ϕ, ψ) space. The training process is remarkably efficient which leads to convergence less than 1 ns and finally flattens the distribution over ϕ (Figure 4C). Remarkably, although only a 1D target distribution is specified, TALOS is able to construct the bias potential complementary to the 2D (ϕ, ψ) FES (Figure 4D) and achieves the prescribed distribution. Since the target is defined solely along ϕ, the barrier along ψ will manifest as slow diffusion terms that hinder the transitions of ϕ. Therefore, TALOS manages to boost the transitions of ϕ not only through lowering the free energy barrier but also by enhancing the diffusion along it. This behavior is qualitatively different from other established CV-based sampling methods such as metadynamics and rather shares similarities with generalized-ensemble-based methods.

In addition to enhanced sampling, given that the MD force field is differentiably programmed in an AIMS package, it is feasible to conduct MD simulations or sampling without the need for "integration" and instead perform these processes in a parallel manner with great efficiency. We refer readers of this topic to related research such as the Boltzmann Generator[16] and variationally adversarial density estimation (VADE).[86]

IV. OUTLOOK

The integration of MD and AI ecosystem has been of growing interest for both communities. AI-enhanced molecular simulations have already yielded several high-impact research results that are transforming the way MD is approached. Despite the encouraging progress made thus far, as the fusion of AI and molecular modeling is still in its infancy, there are undoubtedly numerous significant issues that require resolution. To conclude this Review, we will enumerate some emerging concerns about the infrastructure, algorithms and even research paradigms in this field and provide possible solutions or trends to address these issues. We envision that with proper incorporation and adaptation of the most advanced AI techniques, the establishment of a new MS infrastructure could yield further exciting opportunities beyond the scope of this Review.

IV.1. Operators for Scientific Computing. One of the most striking distinctions between AI programming and MS programming is the disparity between their "operator" libraries. Operators, which are fundamental computational functions or modules, are routinely invoked during programming. For example, tensor operators like matrix multiplication are frequently called during execution of a deep neural network model and thus are highly optimized in the AI framework. On the other hand, molecular simulations may require some scientific computing functions which are rarely encountered in deep learning: to name only a few, fast Fourier transform, matrix inversion, eigendecomposition, etc. The lack of support for these scientific computing operators would significantly impair the functionality and performance of AIMS programs. With the increasing popularity of AI for science, we expect that the support for scientific computing operators would be strengthened in the near future.

Besides, AI programming is highly optimized toward vectorized operations, so the shape of arrays or tensors is assumed to be static and should not change throughout the computation. Clearly the restriction of static shapes can help fully unleash the power of autoparallelization and enable JIT compiling; however, it causes difficulties for some sparse operators in MS. For instance, neighbor list or space partitioning is commonplace in most MS packages in order to deal with condensed-phase systems, which are designed to sparsify the interactions between particles and save both computational time and memory cost. However, the most efficient algorithms do not assume a fixed shape of the neighbor list. Consequently, AP and JIT are often at odds with such algorithms, and limit the size of systems that can be simulated on a single computing device to a certain extent. We note that this problem may be hard to be completely resolved but can be partially ameliorated by tailoring the particular algorithms. For example, the development of "dynamic shapes" in the newest AI frameworks may help us to break through the limitations of static shapes and strike a balance between computational efficiency and memory cost. Besides, with the increasing memory of parallel computing hardware like GPUs and NPUs, we expect that the benefit earned by AP may overwhelm the sacrifice of sparse operators. As recent research implies,[22] in AIMS, although an individual simulation may be slowed to some extent on a single device, the size of simultaneously executed simulations is enlarged, and the increase of computational time is sublinear to the increase of replicas, thus leading to overall gains of computational throughput.

IV.2. Differentiable Simulations and Metaoptimization. Built upon an AI framework that enables automatic differentiation, in principle, a molecular simulation trajectory in AIMS can also be differentiable. This is termed as "differentiable simulations" in some literature.[22,87,88] To be more general, evolving an initial state with any ordinary differential equations (ODEs) like Newtonian dynamics governed by a neural network field has been an active research topic in machine learning, and efficient approaches like adjoint methods have been developed to backpropagate through such neural ODEs.[89] When the dynamics takes the form of gradient descent, this process is also known as "metaoptimization" in the literature. As opposed to neural ODEs, the evolving trajectory is usually first discretized and then backpropagated during metaoptimization.[90]

Many studies have implemented differentiable simulations to learn how to control or infer movements of dynamical systems, such as those that obey Hamiltonian dynamics[88,91] as well as more general nonlinear dynamics.[92] Differentiable simulations and metaoptimization have also been utilized in constructing models from data in various differential equation settings, including computational fluid dynamics,[93,94] quantum chemistry,[95] and generating protein structures,[87] etc. However, backpropagation through relatively long simulation trajectories is not yet sufficiently efficient for practical use currently. We expect that differentiable simulations and metaoptimization can be made faster and more convenient in the future and possibly reevaluate many aspects in molecular simulations, such as the development of force fields.

https://doi.org/10.1021/acs.jctc.3c00214
J. Chem. Theory Comput. 2023, 19, 4338−4350

IV.3. Data-Driven AI versus Physics-Informed AI. Finally, differentiable programming blurs the boundary between scientific computing and machine learning, allowing AI models to be trained solely based on physical laws, even without any data or labels. For instance, the Boltzmann Generator[16] can generate or sample molecular configurations through a generative neural network which is trained according to the energy function of a force field. Similar examples include learning a neural density functional by end-to-end solving the Kohn–Sham equation,[96] as well as other physics-informed neural networks.[97]

It is thus intriguing to weigh the relative importance of the role played by data and physics in AI for science. On the one hand, the power of big data is the cornerstone of the success of modern deep learning; on the other hand, physics laws could incorporate additional optimization objectives during training of AI models. Data-driven AI models are prone to overfitting and cannot guarantee generalizability outside the training data distribution, while physics-informed AI does not rely on data and is trained to always follow the physics laws. Therefore, from a novel perspective, physics laws can play the role of "regularizer" and help AI models to learn faster and generalize better, and the growing trend is to properly combine data-driven and physics-informed paradigms.[94,98,99]

Along this line, AlphaFold2[15] is a prime illustration of the potency of combining these two realms. The training objectives of AlphaFold2 are composed of multiple terms, some of which are supervised and necessitate labels (or data), while some are totally unsupervised and inspired by basic physics rules. For example, the incorporation of steric clashes along with improper bond length penalties is evidently derived from the force field in MD simulations. We anticipate that with AI-enhanced MS, numerous challenging problems in chemistry and biophysics, in addition to protein folding, can also be resolved by exploiting both amassed data and physics rules.

AUTHOR INFORMATION

Corresponding Authors

Yi Isaac Yang – *Institute of Systems Biology, Shenzhen Bay Laboratory, 518055 Shenzhen, China;* orcid.org/0000-0002-5599-0975; Email: yangyi@szbl.ac.cn

Yi Qin Gao – *Changping Laboratory, Beijing 102200, China; Institute of Systems Biology, Shenzhen Bay Laboratory, 518055 Shenzhen, China; Beijing National Laboratory for Molecular Sciences, College of Chemistry and Molecular Engineering, Peking University, 100871 Beijing, China;* orcid.org/0000-0002-4309-9376; Email: gaoyq@pku.edu.cn

Authors

Jun Zhang – *Changping Laboratory, Beijing 102200, China;* orcid.org/0000-0002-8760-6747

Dechin Chen – *Institute of Systems Biology, Shenzhen Bay Laboratory, 518055 Shenzhen, China;* orcid.org/0000-0003-0084-488X

Yijie Xia – *Beijing National Laboratory for Molecular Sciences, College of Chemistry and Molecular Engineering, Peking University, 100871 Beijing, China*

Yu-Peng Huang – *Beijing National Laboratory for Molecular Sciences, College of Chemistry and Molecular Engineering, Peking University, 100871 Beijing, China;* orcid.org/0000-0002-9978-2393

Xiaohan Lin – *Beijing National Laboratory for Molecular Sciences, College of Chemistry and Molecular Engineering, Peking University, 100871 Beijing, China*

Xu Han – *Beijing National Laboratory for Molecular Sciences, College of Chemistry and Molecular Engineering, Peking University, 100871 Beijing, China;* orcid.org/0000-0003-4417-9340

Ningxi Ni – *Huawei Hangzhou R&D Centre, Huawei Technologies Co., Ltd., Hangzhou 310052, China*

Zidong Wang – *Huawei Hangzhou R&D Centre, Huawei Technologies Co., Ltd., Hangzhou 310052, China*

Fan Yu – *Huawei Hangzhou R&D Centre, Huawei Technologies Co., Ltd., Hangzhou 310052, China*

Lijiang Yang – *Beijing National Laboratory for Molecular Sciences, College of Chemistry and Molecular Engineering, Peking University, 100871 Beijing, China*

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jctc.3c00214

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China (No. 2022ZD0115003), National Natural Science Foundation of China (22050003, 92053202, 21821004, and 21927901 to Y.Q.G. and 22273061 and 22003042 to Y.I.Y.), and CAAI-Huawei MindSpore Open Fund to J.Z. The authors thank Dr. Yao-Kun Lei, Dr. Sirui Liu, Dr. Xing Che, and Dr. Maodong Li for useful discussion and gratefully acknowledge the support of MindSpore and CANN used for this research.

ABBREVIATIONS

AD	automatic differentiation
AI	artificial intelligence
AP	automatic parallelization
CANN	compute architecture for neural networks
CPU	central processing unit
CUDA	compute unified device architecture
GPU	graphic processing unit
MD	molecular dynamics
MS	molecular simulation(s)
NPU	neural processing unit
TPU	tensor processing unit

REFERENCES

(1) Brooks, B. R.; Bruccoleri, R. E.; Olafson, B. D.; States, D. J.; Swaminathan, S.; Karplus, M. Charmm: A Program for Macromolecular Energy, Minimization, and Dynamics Calculations. *J. Comput. Chem.* **1983**, *4*, 187–217.

(2) Pearlman, D. A.; Case, D. A.; Caldwell, J. W.; Ross, W. S.; Cheatham, T. E.; DeBolt, S.; Ferguson, D.; Seibel, G.; Kollman, P. Amber, a Package of Computer Programs for Applying Molecular Mechanics, Normal Mode Analysis, Molecular Dynamics and Free Energy Calculations to Simulate the Structural and Energetic Properties of Molecules. *Comput. Phys. Commun.* **1995**, *91*, 1–41.

(3) Berendsen, H. J. C.; van der Spoel, D.; van Drunen, R. Gromacs: A Message-Passing Parallel Molecular Dynamics Implementation. *Comput. Phys. Commun.* **1995**, *91*, 43–56.

(4) Nelson, M. T.; Humphrey, W.; Gursoy, A.; Dalke, A.; Kalé, L. V.; Skeel, R. D.; Schulten, K. NAMD: A Parallel, Object-Oriented Molecular Dynamics Program. *Int. J. Supercomput. Appl. High Perform. Comput.* **1996**, *10*, 251–268.

(5) Plimpton, S. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *J. Comput. Phys.* **1995**, *117*, 1−19.
(6) Eastman, P. K.; Pande, V. S. OpenMM: A Hardware-Independent Framework for Molecular Simulations. *Comput. Sci. Eng.* **2010**, *12*, 34−39.
(7) Huang, Y.-P.; Xia, Y.; Yang, L.; Wei, J.; Yang, Y. I.; Gao, Y. Q. SPONGE: A GPU-Accelerated Molecular Dynamics Package with Enhanced Sampling and AI-Driven Algorithms. *Chin. J. Chem.* **2022**, *40*, 160−168.
(8) Gang, C.; Guobo, L.; Songwen, P.; Baifeng, W. Gpgpu Supported Cooperative Acceleration in Molecular Dynamics. In *2009 13th International Conference on Computer Supported Cooperative Work in Design*, Santiago, April 22−24, 2009; IEEE: New York, 2009; pp 113−118.
(9) Baker, J. A.; Hirst, J. D. Molecular Dynamics Simulations Using Graphics Processing Units. *Mol. Inform.* **2011**, *30*, 498−504.
(10) Case, D. A.; Cheatham, T. E., III; Darden, T.; Gohlke, H.; Luo, R.; Merz, K. M., Jr.; Onufriev, A.; Simmerling, C.; Wang, B.; Woods, R. J. The Amber Biomolecular Simulation Programs. *J. Comput. Chem.* **2005**, *26*, 1668−1688.
(11) Salomon-Ferrer, R.; Case, D. A.; Walker, R. C. An Overview of the Amber Biomolecular Simulation Package. *WIREs Comput. Mol. Sci.* **2013**, *3*, 198−210.
(12) Shaw, D. E.; Deneroff, M. M.; Dror, R. O.; Kuskin, J. S.; Larson, R. H.; Salmon, J. K.; Young, C.; Batson, B.; Bowers, K. J.; Chao, J. C.; et al. Anton, a Special-Purpose Machine for Molecular Dynamics Simulation. *Commun. ACM* **2008**, *51*, 91−97.
(13) Shaw, D. E.; Grossman, J. P.; Bank, J. A.; Batson, B.; Butts, J. A.; Chao, J. C.; Deneroff, M. M.; Dror, R. O.; Even, A.; Fenton, C. H.; et al. Anton 2: Raising the Bar for Performance and Programmability in a Special-Purpose Molecular Dynamics Supercomputer. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, November 16−21, 2014; pp 41−53.
(14) Shaw, D. E.; Adams, P. J.; Azaria, A.; Bank, J. A.; Batson, B.; Bell, A.; Bergdorf, M.; Bhatt, J.; Butts, J. A.; Correia, T.; et al. Anton 3: Twenty Microseconds of Molecular Dynamics Simulation before Lunch. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*; Association for Computing Machinery: St. Louis, MO, 2021; Article 1.
(15) Jumper, J.; Evans, R.; Pritzel, A.; Green, T.; Figurnov, M.; Ronneberger, O.; Tunyasuvunakool, K.; Bates, R.; Žídek, A.; Potapenko, A.; et al. Highly Accurate Protein Structure Prediction with Alphafold. *Nature* **2021**, *596*, 583−589.
(16) Noé, F.; Olsson, S.; Köhler, J.; Wu, H. Boltzmann Generators: Sampling Equilibrium States of Many-Body Systems with Deep Learning. *Science* **2019**, *365*..
(17) Behler, J. Perspective: Machine Learning Potentials for Atomistic Simulations. *J. Chem. Phys.* **2016**, *145*, 170901.
(18) Bartók, A. P.; De, S.; Poelking, C.; Bernstein, N.; Kermode, J. R.; Csányi, G.; Ceriotti, M. Machine Learning Unifies the Modeling of Materials and Molecules. *Sci. Adv.* **2017**, *3*, e1701816.
(19) Zhang, L.; Han, J.; Wang, H.; Car, R.; E, W. Deepcg: Constructing Coarse-Grained Models Via Deep Neural Networks. *J. Chem. Phys.* **2018**, *149*, 034101.
(20) Zhang, L.; Han, J.; Wang, H.; Car, R.; E, W. Deep Potential Molecular Dynamics: A Scalable Model with the Accuracy of Quantum Mechanics. *Phys. Rev. Lett.* **2018**, *120*, 143001.
(21) Barrett, R.; Chakraborty, M.; Amirkulova, D.; Gandhi, H.; Wellawatte, G.; White, A. HOOMD-TF: GPU-Accelerated, Online Machine Learning in the HOOMD-Blue Molecular Dynamics Engine. *J. Open Source Softw.* **2020**, *5*, 2367.
(22) Schoenholz, S.; Cubuk, E. D. In *JAX MD: A Framework for Differentiable Physics, Neural Information Processing Systems 33*, Vancouver, 2020; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., Lin, H., Eds.; Curran Associates, Inc.: New York, 2020.
(23) Doerr, S.; Majewski, M.; Pérez, A.; Krämer, A.; Clementi, C.; Noe, F.; Giorgino, T.; De Fabritiis, G. TorchMD: A Deep Learning Framework for Molecular Simulations. *J. Chem. Theory Comput.* **2021**, *17*, 2355−2363.
(24) Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX conference on Operating Systems Design and Implementation (OSDI '16)*, Savannah, 2016; Keeton, K., Roscoe, T., Eds.; USENIX Association: Berkeley, CA, 2016; pp 265−283.
(25) Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Neural Information Processing Systems 32*, Vancouver, 2019; Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R., Eds.; Curran Associates, Inc.: New York, 2019.
(26) *MindSpore*; https://www.mindspore.cn/ (accessed 2020).
(27) *MindSPONGE: Simulation Package Towards Next Generation Molecular Modelling*; https://gitee.com/mindspore/mindscience/tree/master/MindSPONGE (accessed 2021).
(28) Zhang, J.; Lei, Y.-K.; Zhang, Z.; Chang, J.; Li, M.; Han, X.; Yang, L.; Yang, Y. I.; Gao, Y. Q. A Perspective on Deep Learning for Molecular Modeling and Simulations. *J. Phys. Chem. A* **2020**, *124*, 6745−6763.
(29) Behler, J.; Parrinello, M. Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. *Phys. Rev. Lett.* **2007**, *98*, 146401.
(30) Smith, J. S.; Isayev, O.; Roitberg, A. E. Ani-1: An Extensible Neural Network Potential with Dft Accuracy at Force Field Computational Cost. *Chem. Sci.* **2017**, *8*, 3192−3203.
(31) Zhang, Y.; Hu, C.; Jiang, B. Embedded Atom Neural Network Potentials: Efficient and Accurate Machine Learning with a Physically Inspired Representation. *J. Phys. Chem. Lett.* **2019**, *10*, 4962−4967.
(32) Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; Dahl, G. E. Neural Message Passing for Quantum Chemistry. In *The 34th International Conference on Machine Learning*; Doina, P., Yee Whye, T., Eds.; ML Research Press: Sydney, 2017; pp 1263−1272.
(33) Schütt, K. T.; Sauceda, H. E.; Kindermans, P.-J.; Tkatchenko, A.; Müller, K.-R. Schnet − a Deep Learning Architecture for Molecules and Materials. *J. Chem. Phys.* **2018**, *148*, 241722.
(34) Unke, O. T.; Meuwly, M. Physnet: A Neural Network for Predicting Energies, Forces, Dipole Moments, and Partial Charges. *J. Chem. Theory Comput.* **2019**, *15*, 3678−3693.
(35) Zhang, J.; Zhou, Y.; Lei, Y.-K.; Yang, Y. I.; Gao, Y. Q. *Molecular CT: Unifying Geometry and Representation Learning for Molecules at Different Scales*. arXiv preprint arXiv:2012.11816, **2020**.
(36) Liu, Z.; Lin, L.; Jia, Q.; Cheng, Z.; Jiang, Y.; Guo, Y.; Ma, J. Transferable Multilevel Attention Neural Network for Accurate Prediction of Quantum Chemistry Properties Via Multitask Learning. *J. Chem. Inf. Model.* **2021**, *61*, 1066−1082.
(37) Wang, H.; Zhang, L.; Han, J.; E, W. Deepmd-Kit: A Deep Learning Package for Many-Body Potential Energy Representation and Molecular Dynamics. *Comput. Phys. Commun.* **2018**, *228*, 178−184.
(38) Zhang, Y.; Wang, H.; Chen, W.; Zeng, J.; Zhang, L.; Wang, H.; E, W. Dp-Gen: A Concurrent Learning Platform for the Generation of Reliable Deep Learning Based Potential Energy Models. *Comput. Phys. Commun.* **2020**, *253*, 107206.
(39) Thaler, S.; Zavadlav, J. Learning Neural Network Potentials from Experimental Data Via Differentiable Trajectory Reweighting. *Nat. Commun.* **2021**, *12*, 6884.
(40) Wieder, M.; Fass, J.; Chodera, J. D. Teaching Free Energy Calculations to Learn from Experimental Data. *bioRxiv* **2021**, 2021.
(41) Wang, X.; Li, J.; Yang, L.; Chen, F.; Wang, Y.; Chang, J.; Chen, J.; Zhang, L.; Yu, K. Dmff: An Open-Source Automatic Differentiable Platform for Molecular Force Field Development and Molecular Dynamics Simulation. *ChemRxiv* **2022**..
(42) Baydin, A. G.; Pearlmutter, B. A.; Radul, A. A.; Siskind, J. M. Automatic Differentiation in Machine Learning: A Survey. *J. Mach. Learn. Res.* **2018**, *18*, 5596−5637.

https://doi.org/10.1021/acs.jctc.3c00214
J. Chem. Theory Comput. 2023, 19, 4338−4350

(43) Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. Learning Representations by Back-Propagating Errors. *nature* **1986**, *323*, 533−536.
(44) Maclaurin, D.; Duvenaud, D.; Adams, R. P. Autograd: Effortless Gradients in Numpy. In *ICML 2015 AutoML workshop*, 2015.
(45) Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M. J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S. *JAX: Composable Transformations of Python+ NumPy Programs*; 2018.
(46) Innes, M.; Edelman, A.; Fischer, K.; Rackauckas, C.; Saba, E.; Shah, V. B.; Tebbutt, W. *A Differentiable Programming System to Bridge Machine Learning and Scientific Computing*. arXiv preprint arXiv:1907.07587, **2019**.
(47) Barducci, A.; Bonomi, M.; Parrinello, M. Metadynamics. *Wiley Interdiscip. Rev. Comput. Mol. Sci.* **2011**, *1*, 826−843.
(48) Torrie, G. M.; Valleau, J. P. Nonphysical Sampling Distributions in Monte Carlo Free-Energy Estimation: Umbrella Sampling. *J. Comput. Phys.* **1977**, *23*, 187−199.
(49) Laio, A.; Parrinello, M. Escaping Free-Energy Minima. *Proc. Nat. Acad. Sci. U.S.A.* **2002**, *99*, 12562−12566.
(50) Niu, H.; Piaggi, P. M.; Invernizzi, M.; Parrinello, M. Molecular Dynamics Simulations of Liquid Silica Crystallization. *proceedings of the national academy of sciences of the united states of america* **2018**, *115*, 5348−5352.
(51) Li, M.; Zhang, J.; Niu, H.; Lei, Y.-K.; Han, X.; Yang, L.; Ye, Z.; Yang, Y. I.; Gao, Y. Q. Phase Transition between Crystalline Variants of Ordinary Ice. *J. Phys. Chem. Lett.* **2022**, *13*, 8601−8606.
(52) Bonomi, M.; Branduardi, D.; Bussi, G.; Camilloni, C.; Provasi, D.; Raiteri, P.; Donadio, D.; Marinelli, F.; Pietrucci, F.; Broglia, R. A.; et al. PLUMED: A Portable Plugin for Free-Energy Calculations with Molecular Dynamics. *Comput. Phys. Commun.* **2009**, *180*, 1961−1972.
(53) Giorgino, T. How to Differentiate Collective Variables in Free Energy Codes: Computer-Algebra Code Generation and Automatic Differentiation. *Comput. Phys. Commun.* **2018**, *228*, 258−263.
(54) Fiorin, G.; Klein, M. L.; Hénin, J. Using Collective Variables to Drive Molecular Dynamics Simulations. *Mol. Phys.* **2013**, *111*, 3345−3362.
(55) Sugita, Y.; Okamoto, Y. Replica-Exchange Molecular Dynamics Method for Protein Folding. *Chemical physics letters* **1999**, *314*, 141−151.
(56) Raiteri, P.; Laio, A.; Gervasio, F. L.; Micheletti, C.; Parrinello, M. Efficient Reconstruction of Complex Free Energy Landscapes by Multiple Walkers Metadynamics. *J. Phys. Chem. B* **2006**, *110*, 3533−3539.
(57) Henkelman, G.; Uberuaga, B. P.; Jónsson, H. A Climbing Image Nudged Elastic Band Method for Finding Saddle Points and Minimum Energy Paths. *J. Chem. Phys.* **2000**, *113*, 9901−9904.
(58) Valsson, O.; Tiwary, P.; Parrinello, M. Enhancing Important Fluctuations: Rare Events and Metadynamics from a Conceptual Viewpoint. *Annu. Rev. Phys. Chem.* **2016**, *67*, 159−184.
(59) Abraham, M. J.; Murtola, T.; Schulz, R.; Páll, S.; Smith, J. C.; Hess, B.; Lindahl, E. Gromacs: High Performance Molecular Simulations through Multi-Level Parallelism from Laptops to Supercomputers. *SoftwareX* **2015**, *1−2*, 19−25.
(60) Thompson, A. P.; Aktulga, H. M.; Berger, R.; Bolintineanu, D. S.; Brown, W. M.; Crozier, P. S.; in 't Veld, P. J.; Kohlmeyer, A.; Moore, S. G.; Nguyen, T. D.; et al. Lammps - a Flexible Simulation Tool for Particle-Based Materials Modeling at the Atomic, Meso, and Continuum Scales. *Comput. Phys. Commun.* **2022**, *271*, 108171.
(61) Amadei, A.; Linssen, A. B.; Berendsen, H. J. Essential Dynamics of Proteins. *Proteins: Struct., Funct., Bioinf.* **1993**, *17*, 412−425.
(62) Buchete, N.-V.; Hummer, G. Coarse Master Equations for Peptide Folding Dynamics. *J. Phys. Chem. B* **2008**, *112*, 6057−6069.
(63) E, W.; Vanden-Eijnden, E. Transition-Path Theory and Path-Finding Algorithms for the Study of Rare Events. *Annu. Rev. Phys. Chem.* **2010**, *61*, 391−420.
(64) Chodera, J. D.; Singhal, N.; Pande, V. S.; Dill, K. A.; Swope, W. C. Automatic Discovery of Metastable States for the Construction of Markov Models of Macromolecular Conformational Dynamics. *J. Chem. Phys.* **2007**, *126*, 155101.
(65) Ferguson, A. L.; Panagiotopoulos, A. Z.; Kevrekidis, I. G.; Debenedetti, P. G. Nonlinear Dimensionality Reduction in Molecular Simulation: The Diffusion Map Approach. *Chem. Phys. Lett.* **2011**, *509*, 1−11.
(66) Ceriotti, M.; Tribello, G. A.; Parrinello, M. Simplifying the Representation of Complex Free-Energy Landscapes Using Sketch-Map. *Proc. Natl. Acad. Sci. U. S. A.* **2011**, *108*, 13023−13028.
(67) Pérez-Hernández, G.; Paul, F.; Giorgino, T.; De Fabritiis, G.; Noé, F. Identification of Slow Molecular Order Parameters for Markov Model Construction. *J. Chem. Phys.* **2013**, *139*, 015102.
(68) Zhang, J.; Lei, Y.-K.; Che, X.; Zhang, Z.; Yang, Y. I.; Gao, Y. Q. Deep Representation Learning for Complex Free-Energylandscapes. *journal of physical chemistry letters* **2019**, *10*, 5571−5576.
(69) Zhang, J. *Information Distilling of Metastability*. https://gitee.com/mindspore/mindscience/tree/master/MindSPONGE/tutorials/IDM.
(70) Becker, S.; Hinton, G. E. Self-Organizing Neural Network That Discovers Surfaces in Random-Dot Stereograms. *Nature* **1992**, *355*, 161.
(71) Best, R. B.; Hummer, G.; Eaton, W. A. Native Contacts Determine Protein Folding Mechanisms in Atomistic Simulations. *Proc. Natl. Acad. Sci. U. S. A.* **2013**, *110*, 17874−17879.
(72) Frauenfelder, H.; Sligar, S. G.; Wolynes, P. G. The Energy Landscapes and Motions of Proteins. *Science* **1991**, *254*, 1598−1603.
(73) Eyring, H. The Activated Complex in Chemical Reactions. *J. Chem. Phys.* **1935**, *3*, 107−115.
(74) Onsager, L. Initial Recombination of Ions. *Phys. Rev.* **1938**, *54*, 554.
(75) Best, R. B.; Hummer, G. Reaction Coordinates and Rates from Transition Paths. *Proc. Natl. Acad. Sci. U. S. A.* **2005**, *102*, 6732−6737.
(76) Zhang, J.; Lei, Y.-K.; Zhang, Z.; Han, X.; Li, M.; Yang, L.; Yang, Y. I.; Gao, Y. Q. Deep Reinforcement Learning of Transition States. *Phys. Chem. Chem. Phys.* **2021**, *23*, 6888−6895.
(77) Hummer, G. From Transition Paths to Transition States and Rate Coefficients. *J. Chem. Phys.* **2004**, *120*, 516−523.
(78) Bolhuis, P. G.; Chandler, D.; Dellago, C.; Geissler, P. L. Transition Path Sampling: Throwing Ropes over Rough Mountain Passes, in the Dark. *Annu. Rev. Phys. Chem.* **2002**, *53*, 291−318.
(79) Faradjian, A. K.; Elber, R. Computing Time Scales from Reaction Coordinates by Milestoning. *J. Chem. Phys.* **2004**, *120*, 10880−10889.
(80) Gao, Y. Q. An Integrate-over-Temperature Approach for Enhanced Sampling. *J. Chem. Phys.* **2008**, *128*, 064105.
(81) Yang, Y. I.; Shao, Q.; Zhang, J.; Yang, L.; Gao, Y. Q. Enhanced Sampling in Molecular Dynamics. *J. Chem. Phys.* **2019**, *151*, 070902.
(82) Zhang, J.; Yang, Y. I.; Noé, F. Targeted Adversarial Learning Optimized Sampling. *journal of physical chemistry letters* **2019**, *10*, 5791−5797.
(83) Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. In *Generative Adversarial Nets, Advances in neural information processing systems*. arXiv:1406.2661, **2014**, 2672−2680.
(84) Arjovsky, M.; Chintala, S.; Bottou, L. *Wasserstein Generative Adversarial Networks*. In *International conference on machine learning*, 2017; pp 214−223.
(85) Williams, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine learning* **1992**, *8*, 229−256.
(86) Zhang, J.; Lei, Y.-K.; Yang, Y. I.; Gao, Y. Q. Deep Learning for Variational Multiscale Molecular Modeling. *J. Chem. Phys.* **2020**, *153*, 174115.
(87) Ingraham, J. B.; Riesselman, A. J.; Sander, C.; Marks, D. Learning Protein Structure with a Differentiable Simulator. In *International Conference on Learning Representations*, 2019.
(88) Wang, W.; Axelrod, S.; Gómez-Bombarelli, R. *Differentiable Molecular Simulations for Control and Learning*. arXiv preprint arXiv:2003.00868, **2020**.

(89) Chen, T. Q.; Rubanova, Y.; Bettencourt, J.; Duvenaud, D. Neural Ordinary Differential Equations. In *Neural Information Processing Systems*, 2018; pp 6572−6583.
(90) Andrychowicz, M.; Denil, M.; Gomez, S.; Hoffman, M. W.; Pfau, D.; Schaul, T.; Shillingford, B.; De Freitas, N. Learning to Learn by Gradient Descent by Gradient Descent. *Advances in Neural Information Processing Systems*; 2016; Vol. *29*.
(91) Sanchez-Gonzalez, A.; Bapst, V.; Cranmer, K.; Battaglia, P. *Hamiltonian Graph Networks with Ode Integrators*. arXiv preprint arXiv:1909.12790, **2019**.
(92) Li, Y.; He, H.; Wu, J.; Katabi, D.; Torralba, A. *Learning Compositional Koopman Operators for Model-Based Control*. arXiv preprint arXiv:1910.08264, **2019**.
(93) Morton, J.; Jameson, A.; Kochenderfer, M. J.; Witherden, F. Deep Dynamical Modeling and Control of Unsteady Fluid Flows. *Advances in Neural Information Processing Systems*; 2018; Vol. *31*.
(94) Kochkov, D.; Smith, J. A.; Alieva, A.; Wang, Q.; Brenner, M. P.; Hoyer, S. Machine Learning−Accelerated Computational Fluid Dynamics. *Proc. Natl. Acad. Sci. U. S. A.* **2021**, *118*, No. e2101784118.
(95) Tamayo-Mendoza, T.; Kreisbeck, C.; Lindh, R.; Aspuru-Guzik, A. Automatic Differentiation in Quantum Chemistry with Applications to Fully Variational Hartree−Fock. *ACS central science* **2018**, *4*, 559−566.
(96) Li, L.; Hoyer, S.; Pederson, R.; Sun, R.; Cubuk, E. D.; Riley, P.; Burke, K. Kohn-Sham Equations as Regularizer: Building Prior Knowledge into Machine-Learned Physics. *Physical review letters* **2021**, *126*, 036401.
(97) Raissi, M.; Perdikaris, P.; Karniadakis, G. E. Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations. *J. Comput. Phys.* **2019**, *378*, 686−707.
(98) Pfau, D.; Spencer, J. S.; Matthews, A. G.; Foulkes, W. M. C. Ab Initio Solution of the Many-Electron Schrödinger Equation with Deep Neural Networks. *Physical Review Research* **2020**, *2*, 033429.
(99) Kirkpatrick, J.; McMorrow, B.; Turban, D. H.; Gaunt, A. L.; Spencer, J. S.; Matthews, A. G.; Obika, A.; Thiry, L.; Fortunato, M.; Pfau, D.; et al. Pushing the Frontiers of Density Functionals by Solving the Fractional Electron Problem. *Science* **2021**, *374*, 1385−1389.

https://doi.org/10.1021/acs.jctc.3c00214
J. Chem. Theory Comput. 2023, 19, 4338−4350