ARTICLE OPEN

Check for updates

PySAGES: flexible, advanced sampling methods accelerated with GPUs

Pablo F. Zubieta Rico[1], Ludwig Schneider[1], Gustavo R. Pérez-Lemus[1], Riccardo Alessandri[1], Siva Dasetty[1], Trung D. Nguyen[2], Cintia A. Menéndez[1], Yiheng Wu[1], Yezhi Jin[1], Yinan Xu[1], Samuel Varner[1,3], John A. Parker[2], Andrew L. Ferguson[1], Jonathan K. Whitmer[4] and Juan J. de Pablo[1]✉

Molecular simulations are an important tool for research in physics, chemistry, and biology. The capabilities of simulations can be greatly expanded by providing access to advanced sampling methods and techniques that permit calculation of the relevant underlying free energy landscapes. In this sense, software that can be seamlessly adapted to a broad range of complex systems is essential. Building on past efforts to provide open-source community-supported software for advanced sampling, we introduce PySAGES, a Python implementation of the Software Suite for Advanced General Ensemble Simulations (SSAGES) that provides full GPU support for massively parallel applications of enhanced sampling methods such as adaptive biasing forces, harmonic bias, or forward flux sampling in the context of molecular dynamics simulations. By providing an intuitive interface that facilitates the management of a system's configuration, the inclusion of new collective variables, and the implementation of sophisticated free energy-based sampling methods, the PySAGES library serves as a general platform for the development and implementation of emerging simulation techniques. The capabilities, core features, and computational performance of this tool are demonstrated with clear and concise examples pertaining to different classes of molecular systems. We anticipate that PySAGES will provide the scientific community with a robust and easily accessible platform to accelerate simulations, improve sampling, and enable facile estimation of free energies for a wide range of materials and processes.

npj Computational Materials (2024)10:35 ; https://doi.org/10.1038/s41524-023-01189-z

INTRODUCTION

Molecular simulations are extensively used in a wide range of science and engineering disciplines. As their use has grown for the discovery of new phenomena and the interpretation of sophisticated experimental measurements, so has the complexity of the systems that are considered. Classical atomistic molecular dynamics (MD) simulations are generally limited to microsecond time scales and length scales of tens of nanometers. For systems that are characterized by rugged free energy landscapes, such time scales can be inadequate to ensure sufficient sampling of the relevant phase space, and advanced methods must therefore be adopted to overcome free energy barriers. In that regard, it is useful and increasingly common to identify properly chosen collective variables (CVs), which are generally differentiable functions of the atomic coordinates of the system; then, biases can be applied to explore the space defined by such CVs, thereby overcoming barriers and enhancing sampling of the thermally accessible phase space.

The rapid growth of hardware accelerators such as GPUs or TPUs, or specialized hardware designed for fast MD computations[1], has provided researchers with increased opportunities to perform longer simulations of larger systems. GPUs, in particular, provide a widely accessible option for fast simulations, and several software packages, such as HOOMD-blue[2], OpenMM[3], JAX MD[4], LAMMPS[5], and Gromacs[6], are now available for MD simulations on such devices.

As mentioned above, enhanced sampling methods seek to surmount the high energy barriers that separate multiple metastable states in a system, while facilitating the calculation of relevant thermodynamic quantities as functions of different CVs such as free-energy surfaces (FES). Several libraries, such as PLUMED[7], Colvars[8], and our own SSAGES package[9], provide out-of-the-box solutions for performing enhanced sampling MD simulations.

Among the various enhanced sampling methods available in the literature, some of the most recently devised schemes rely on machine learning (ML) strategies to approximate free energy surfaces and their gradients (generalized forces)[10–13]. Similarly, algorithms for identifying meaningful CVs that correlate with high variance or slow degrees of freedom are based on deep neural networks[14–19]. These advances serve to highlight the need for seamless integration of ML frameworks with existing MD software libraries.

To date, there are no solutions that combine enhanced sampling techniques, hardware acceleration, and ML frameworks to facilitate enhanced-sampling MD simulations on GPUs. While some MD libraries that support GPUs provide access to a limited set of enhanced sampling methods[3,6,20–22], there are currently no packages that enable users to take advantage of all of these features within the same platform and in the same backend-agnostic fashion that tools such as PLUMED and SSAGES have provided for CPU-based MD simulations.

Here we present PySAGES, a Python Suite for Advanced General Ensemble Simulations. It is a free, open-source software package written in Python and based on JAX that follows the design ideas of SSAGES and enables users to easily perform enhanced-sampling MD simulations on CPUs, GPUs, and TPUs. PySAGES can currently be coupled with HOOMD-blue, LAMMPS, OpenMM,

[1]Pritzker School of Molecular Engineering, The University of Chicago, Chicago, IL 60637, USA. [2]Research Computing Center, The University of Chicago, Chicago, IL 60637, USA. [3]Division of Chemistry and Chemical Engineering, California Institute of Technology, Pasadena, CA 91125, USA. [4]Department of Chemical and Biomolecular Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. ✉email: depablo@uchicago.edu

JAX MD, and ASE – and by extension from the latter to CP2K, Quantum ESPRESSO, VASP and Gaussian, among others. At this time, PySAGES offers the following enhanced sampling methods: Umbrella Sampling, Metadynamics, Well-tempered Metadynamics, Forward Flux Sampling, String Method, Adaptive Biasing Force, Artificial neural network sampling, Adaptive Biasing Force using neural networks, Combined Force Frequency, and Spectral Adaptive Biasing Force. PySAGES also includes some of the most commonly used CVs and, importantly, defining new ones is relatively simple, as long as they can be expressed in terms of the NumPy interface provided by JAX. All CVs can be automatically differentiated through JAX functional transforms. PySAGES is highly modular, thereby allowing for the easy implementation of new methods as they emerge, even as part of a user-facing script.

In the following sections, we provide a general overview of the design and implementation of PySAGES, and present a series of examples to showcase its flexibility for addressing research problems in different application areas. We also discuss its performance in GPUs and present a few perspectives on how to grow and improve the package to cover more research use cases through future development, as well as community involvement and contributions.

METHODS

We begin by briefly outlining the core components of PySAGES, how they interact, and how communication with each backend allows PySAGES to bias a simulation during runtime. A summary of the execution workflow of PySAGES along with a mapping of the user interface with the main stages of the simulation and the interaction with the backends, is illustrated in Fig. 1.

To provide a uniform user interface while minimizing disruption to preexisting workflows, PySAGES only requires the user to wrap their traditional backend scripting code into *simulation generator* functions. This approach accommodates the heterogeneity of Python interfaces across the different simulation backends supported by PySAGES. An example of a simulation generator function and how a traditional OpenMM script can be modified to perform an enhanced-sampling MD simulation is depicted in Fig. 2.

At the start of a simulation, the simulation generator function is called to instantiate as many replicas of the simulation as needed. Then, for each replica, PySAGES queries the particle information and the device that the backend will be using. In addition, during this initial stage, PySAGES also performs automatic differentiation of the collective variables via JAX's `grad` transform – required to estimate the biasing forces, and generates specialized initialization and updating routines for the user-declared sampling method.

Like SSAGES, PySAGES wraps the simulation information into an object called a `Snapshot`. This object exposes the most important simulation information, such as particle positions, velocities, and forces in a backend- and device-agnostic format. To achieve this, PySAGES uses DLPack – for C++ based MD libraries – to directly access the contents of the backend-allocated buffers for the different particle properties without creating data copies whenever possible.

Once the setup of both the simulation and sampling method is completed, PySAGES hands control back to the backend, which will run for a given number of time steps or until some other stopping criteria is reached. In order to exchange information back and forth, PySAGES adds a force-like object or function to the backend which gets called as part of the time integration routine. Here, the sampling method state gets updated and the computed biasing forces are added to the backend net forces.

Finally, the information collected during an enhanced simulation is returned and can be used for calculation of the free energy as a function of the selected CVs. Any particular simulation that requires a longer convergence time can be stopped and then

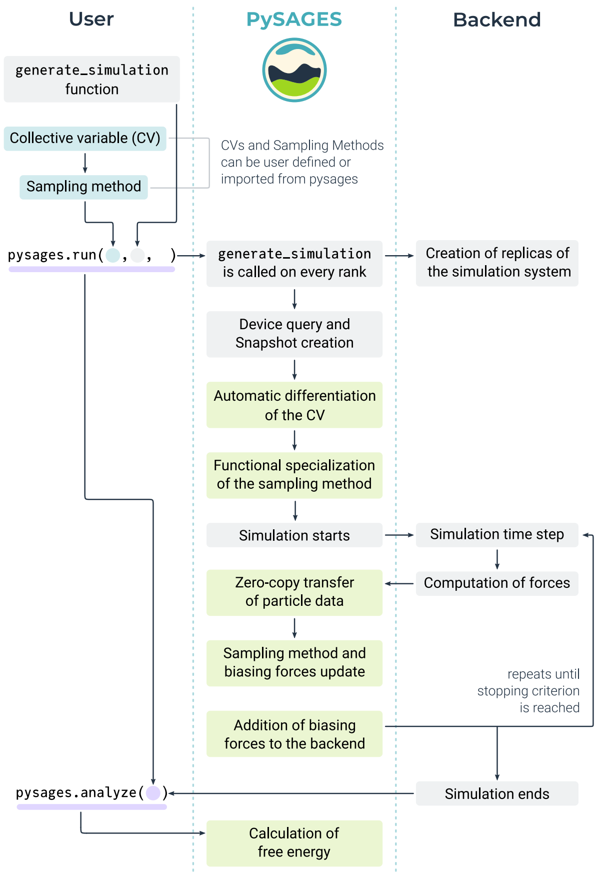
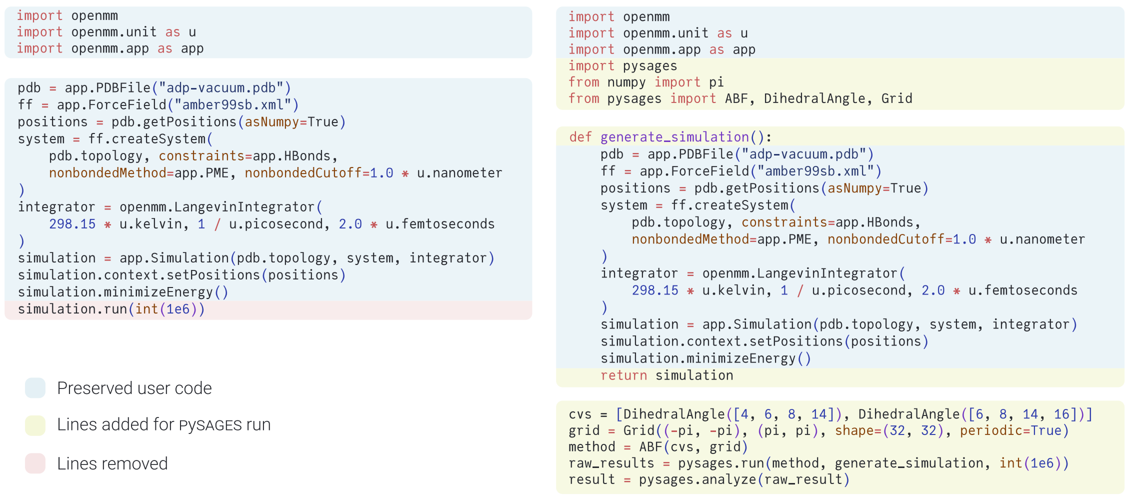

User
PySAGES
Backend
generate_simulation function
Collective variable (CV)
Sampling method
CVs and Sampling Methods can be user defined or imported from pysages
pysages.run(, ,)
generate_simulation is called on every rank
Creation of replicas of the simulation system
Device query and Snapshot creation
Automatic differentiation of the CV
Functional specialization of the sampling method
Simulation starts
Simulation time step
Zero-copy transfer of particle data
Computation of forces
Sampling method and biasing forces update
repeats until stopping criterion is reached
Addition of biasing forces to the backend
pysages.analyze( )
Simulation ends
Calculation of free energy

Fig. 1 The PySAGES simulation flowchart. For a simulation, a user sets up a script that declares the CV and sampling methods to be used. PySAGES initializes the biasing computation and launches the simulation in the backend. At each time step, PySAGES wraps the particle data, computes a biasing force, if required, and modifies the backend information accordingly.

restarted by saving the state of the sampling method and the last snapshot of a simulation. Unlike SSAGES, PySAGES offers a user-friendly `analyze` interface that simplifies the process of performing post-simulation analysis, including the automatic calculation of free energies based on the chosen sampling method. This feature can greatly reduce the time and effort required to gain valuable insights from simulations.

PySAGES offers an easy way to leverage different parallelism frameworks including MPI with the same uniform fronted available to run enhanced sampling simulations. This is achieved via Python's `concurrent.futures` interface. In particular, for MPI parallelism, the user only needs to pass an additional `MPIPoolExecutor` (from `mpi4py`) to PySAGES' `run` method. If the user selects a method such as `UmbrellaSampling`, the workload for each image will be distributed across available MPI nodes. On the other hand, for most of the sampling methods, the parallelization interface allows the user to run multiple replicas of the same system to enable, for instance, analysis of the uncertainties associated to computing the free energy of a given system.

To ensure the reproducibility and correctness of our implementation and to follow software engineering best practices, we have implemented a comprehensive unit tests suite, and leverage GitHub's continuous integration services. In addition, we use trunk.io to adhere to quality standards as well as to ease the collaboration of developers.



```python
import openmm
import openmm.unit as u
import openmm.app as app

pdb = app.PDBFile("adp-vacuum.pdb")
ff = app.ForceField("amber99sb.xml")
positions = pdb.getPositions(asNumpy=True)
system = ff.createSystem(
    pdb.topology, constraints=app.HBonds,
    nonbondedMethod=app.PME, nonbondedCutoff=1.0 * u.nanometer
)
integrator = openmm.LangevinIntegrator(
    298.15 * u.kelvin, 1 / u.picosecond, 2.0 * u.femtoseconds
)
simulation = app.Simulation(pdb.topology, system, integrator)
simulation.context.setPositions(positions)
simulation.minimizeEnergy()
simulation.run(int(1e6))
```

```python
import openmm
import openmm.unit as u
import openmm.app as app
import pysages
from numpy import pi
from pysages import ABF, DihedralAngle, Grid

def generate_simulation():
    pdb = app.PDBFile("adp-vacuum.pdb")
    ff = app.ForceField("amber99sb.xml")
    positions = pdb.getPositions(asNumpy=True)
    system = ff.createSystem(
        pdb.topology, constraints=app.HBonds,
        nonbondedMethod=app.PME, nonbondedCutoff=1.0 * u.nanometer
    )
    integrator = openmm.LangevinIntegrator(
        298.15 * u.kelvin, 1 / u.picosecond, 2.0 * u.femtoseconds
    )
    simulation = app.Simulation(pdb.topology, system, integrator)
    simulation.context.setPositions(positions)
    simulation.minimizeEnergy()
    return simulation

cvs = [DihedralAngle([4, 6, 8, 14]), DihedralAngle([6, 8, 14, 16])]
grid = Grid((-pi, -pi), (pi, pi), shape=(32, 32), periodic=True)
method = ABF(cvs, grid)
raw_results = pysages.run(method, generate_simulation, int(1e6))
result = pysages.analyze(raw_result)
```

Fig. 2 Example of how to use the Python interface for PySAGES. It is easy to extend existing MD scripts with PySAGES to perform enhanced-sampling MD, with minimal changes to the code. In general, the only requirement is for the user to wrap the code that defines the simulation system into a *simulation generator* function.

Enhanced Sampling Methods

While we assume the reader has some basic understanding of enhanced sampling methods, here we provide an overview of these techniques. We direct readers interested in learning more about the fundamentals of enhanced sampling to a number of excellent recent review articles[23–31]. In addition, we discuss the general structure of how enhanced sampling methods are implemented within PySAGES, and also present a summary of the various methods already available in the library.

Enhanced sampling methods are a class of simulation techniques that manipulate regular MD simulations in order to more effectively sample the configuration space. In MD a collective variable, ξ, is typically a function of the positions of all particles, $\hat{\xi}(\{r_i\})$.

For a given statistical ensemble (such as the canonical, NVT), the corresponding free energy can be written as $A = -k_\mathrm{B}T\ln(Z)$, where A is the Helmholtz free energy and Z is the canonical partition function. To make explicit the dependency of the free energy on ξ, let us write down the partition function:

$$Z(\xi) \propto \int d^N r_i\, \delta(\hat{\xi}(\{r_i\}) - \xi)\, e^{-U(\{r_i\})/k_\mathrm{B}T} \tag{1}$$

Normalizing this partition function gives us the probability of occurrence, $p(\xi) = Z(\xi)/(\int d\xi Z(\xi))$, for configurations in the CV subspace. Substituting this probability into the expression for the free energy, we get:

$$A(\xi) = -k_\mathrm{B}T\ln(p(\xi)) + C \tag{2}$$

where C is a constant.

If we take the derivative of the free energy with respect to ξ we get

$$\frac{dA(\xi)}{d\xi} = \frac{\int d^N r_i \frac{dU}{d\xi}\, \delta(\hat{\xi}(\{r_i\}) - \xi)\, e^{-U(\{r_i\})/k_\mathrm{B}T}}{\int d^N r_i\, \delta(\hat{\xi}(\{r_i\}) - \xi)\, e^{-U(\{r_i\})/k_\mathrm{B}T}} = \left\langle \frac{dU}{d\xi} \right\rangle_\xi, \tag{3}$$

where $\langle \dots \rangle_\xi$ denotes the conditional average.

The goal of CV-based enhanced sampling methods is to accurately determine either $p(\xi)$ or $dA(\xi)/d\xi$ – from which $A(\xi)$ can be recovered – in a computationally tractable manner.

In PySAGES, the implementation of sampling methods follows the JAX functional style programming model. New methods are implemented as subclasses of the `SamplingMethod` class, and are required to define a `build` method. This method returns two methods, `initialize` and `update`, used as part of the process of biasing the simulation. For readers familiar with JAX MD, these could be thought of as analogues to the higher-level functions returned by JAX MD's `simulate` integration methods. The `initialize` method allocates all the necessary helper objects and stores them in a `State` data structure, while the `update` method uses the information from the simulation at any given time to update the `State`.

While PySAGES allows new methods to be written seamlessly as part of Python scripts used to set up molecular dynamics simulations, it also provides out-of-the-box implementations of several of the most important known sampling methods. We list and briefly detail them next.

Harmonic biasing. One simple way to sample a specific region of the phase space is to bias the simulation around a point ξ_0 with harmonic bias. This adds a quadratic potential energy term to the Hamiltonian that increases the potential energy as a system moves away from the target point: $\mathcal{H}_\mathrm{b} = \mathcal{H} + k/2\,(\xi - \xi_0)^2$, where $k > 0$ is the spring constant. The unbiased probability distribution $p(\xi)$ can be recovered by dividing the biased distribution by the known weight of the bias $p(\xi) = p_\mathrm{b}(\xi)\,/\,e^{-k/2(\xi-\xi_0)^2/k_\mathrm{B}T}$.

The disadvantage of this approach is that it can only be used to explore the free energy landscape near a well-know point in phase space. This may not be sufficient for many systems, where the free energy landscape is complex.

Umbrella sampling. This is a technique that traditionally builds on harmonic biasing by combining multiple harmonically-biased simulations. It is a well-known method for exploring a known path in phase space to obtain a free energy profile along that path[32]. Typically, a path between to point of interest is described by N points in phase space, ξ_i. At each of these points, a harmonically biased simulation is performed, and the resulting occurrence histograms are combined to obtain a single free energy profile.

In PySAGES, we implement umbrella integration for multi-dimensional CVs. This method approximates the forces acting on the biasing points and integrates these forces to find the free energy profile $A(\xi)$, and allows to explore complex high-dimensional free energy landscapes.

Improved string method. When only the endpoints are known, but not the path itself, the improved (spline-based) string method can be used to find the mean free energy pathway (MFEP) between these two endpoints[33]. The spline-based string method improves upon the original string method by interpolating the MFEP using cubic-splines. In this method, the intermediate points of the path are moved according to the recorded mean forces acting on them, but only in the direction perpendicular to the contour of the path. This ensures that distances between the points along the path remain constant.

This method has been widely used and has been shown to be an effective way to find the MFEP between two points in the phase space[33].

Adaptive Biasing Force Sampling. The adaptive biasing force (ABF) sampling method is a technique used to map complex free-energy landscapes. It can be applied without prior knowledge of the potential energy of the system, as it generates on-the-fly estimates of the derivative of the free energy at each point along the integration pathway. ABF works by introducing an additional force to the system that biases the motion of the atoms, with the strength and direction of the bias continuously updated during the simulation. In the long-time limit, this yields a Hamiltonian with no average force acting along the transition coordinate of interest, resulting in a flat free-energy surface and allowing the system to display accelerated dynamics, thus providing reliable free-energy estimates[34,35]. Similarly to SSAGES, PySAGES implementation of ABF is based on the algorithm described in[35].

Metadynamics. This is another popular approach for enhancing the sampling of complex systems. In metadynamics[36], a bias potential is applied along one or more CVs in the form of Gaussian functions. The height and width (σ) of these Gaussians are controlled by the user. The Gaussian bias potentials are cumulatively deposited at user-defined intervals during the simulation. In standard metadynamics, the height of the Gaussian bias potentials is fixed.

In contrast, for well-tempered metadynamics (WTMD)[37] simulations, the height of the Gaussian bias potentials is adjusted at each timestep using a preset temperature based bias factor. This scaling of Gaussian heights in WTMD leads to faster convergence compared to standard metadynamics, as it restricts the range of free energy explored to a range defined by the bias factor.

In PySAGES, we have implemented both standard metadynamics and WTMD. The well-tempered variant is activated when a user sets a value for the bias factor. To improve the computational performance, we have added optional support for storing the bias potentials in both on a pre-defined grid. This allows users to trade-off accuracy for faster simulations, depending on their needs.

Forward flux sampling. Forward flux sampling (FFS) belongs to a different family of enhanced sampling methods than the ones described above. In the previously described methods, the free energy change from a region in the phase space (A) to the region of interest (B) is calculated by applying a bias to the system. In FFS no bias is added and instead an efficient selection of trajectories that crosses the phase space from A to B is performed. Since no bias is used, the intrinsic dynamics of the system is conserved and therefore kinetic and microscopic information of the transition path can be studied[38]. In PySAGES we have implemented the direct version of FFS[39,40].

Artificial neural networks sampling. Artificial neural networks sampling (ANN)[10] employs regularized neural networks to directly approximate the free energy from the histogram of visits to each region of the CV space, and generates a biasing force that avoids ringing and boundary artifacts[10], which are commonly observed in methods such as metadynamics or basis functions sampling[41]. This approach is effective at quickly adapting to diverse free energy landscapes by interpolating undersampled regions and extrapolating bias into new, unexplored areas.

The implementation on PySAGES offers more flexible approaches to network regularization than SSAGES, which uses Bayesian regularization.

Force-biasing using neural networks. Force-biasing using neural networks (FUNN)[11] is based upon the same idea as ANN, that is, relying on artificial neural networks to provide continuous functions to bias a simulation. As opposed to ANN, which uses a histogram of visits to CV space, FUNN updates its network parameters by training on the ABF estimates for the mean forces as the simulation advances. This method shares all of the features of ABF, but the smooth approximation of the generalized mean force it produces enables much faster convergence to the free energy of a system compared to ABF.

Combined force frequency sampling. The combined force frequency sampling (CFF) method[12] combines the speed of generalized-force based techniques such as ABF or FUNN with the advantages of frequency-based methods like metadynamics or ANN. Notable improvements over earlier force-based methods include eliminating the need for hyperparameters to dampen early-time estimates, automating the integration of forces to generate the free energy, and providing an explicit expression for the free energy at all times, enabling the use of replica exchange or reweighing.

In principle, by using sparse storage of histograms, it should be possible to scale the method to higher dimensions without encountering memory limitations, such optimization is however not yet implemented in PySAGES.

Spectral adaptive biasing force. Spectral ABF[42] is a method that follows the same principle as neural-network-based sampling methods, in that it builds a continuous approximation to the free energy. However, in contrast to methods like FUNN it does so by fitting exponentially convergent basis functions expansions, and could be thought as a generalization of the Basis Functions Sampling Method. In contrast to the latter, and similar to CFF, it allows for the recovery of an explicit expression for the free energy of a system. It is an extremely fast method in terms of both runtime and convergence.

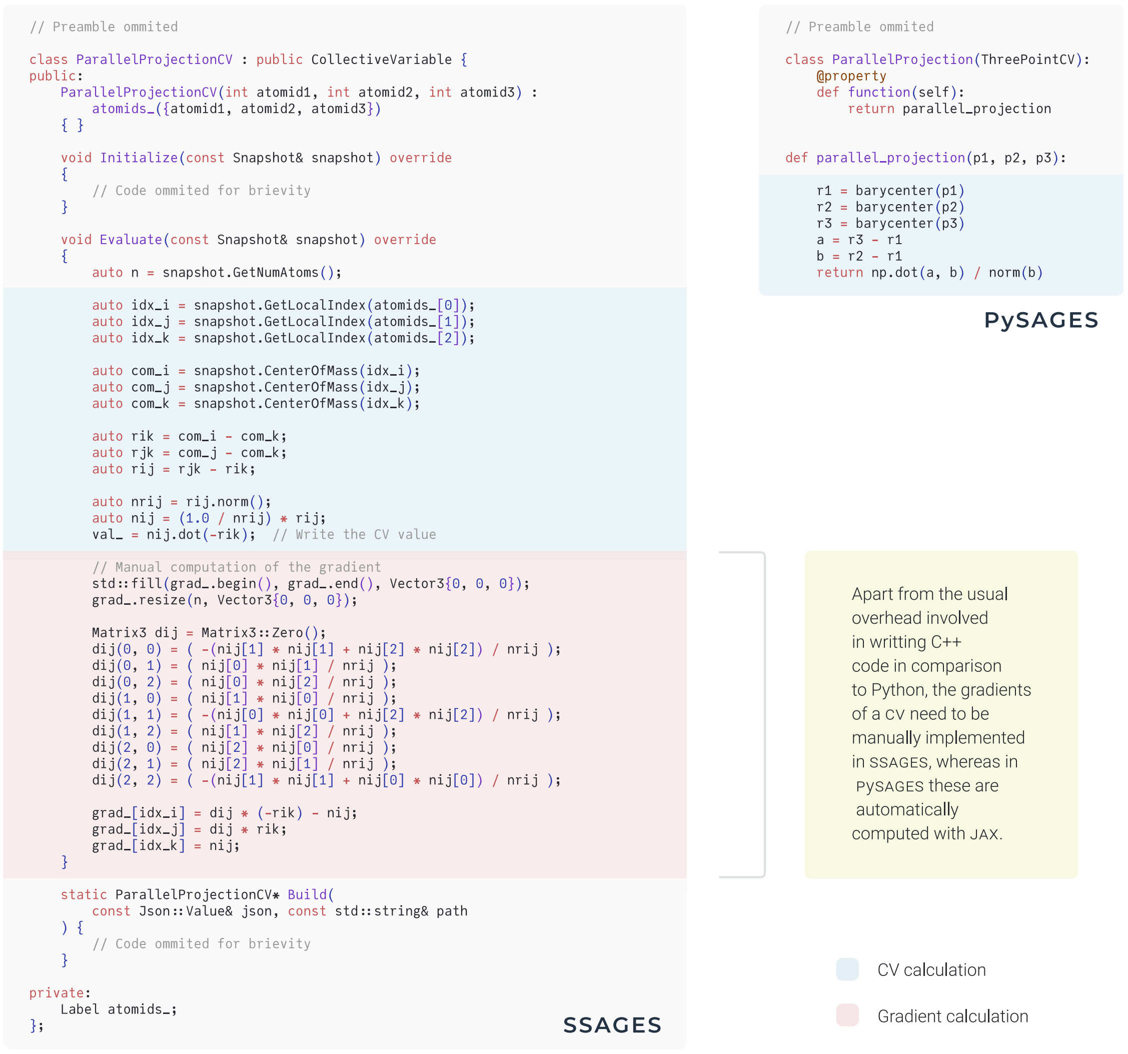
Collective variables

As previously mentioned, enhanced sampling calculations commonly involve the selection of a CV. An appropriate CV for a given system could simply be the distance between the centers of mass of two groups of atoms, but could be a complex specialized quantity.

Below, we list a set of CVs predefined in PySAGES, sorted by the number of groups of atom coordinates necessary for their use:

1. `TwoPointCV`. This subclass is for CVs that need two groups for their definition. This includes `Distance` and `Displacement` (vector).
2. `ThreePointCV`. Subclass of CVs with three groups of atoms, such as `Angle`.
3. `FourPointCV`. Subclass of CVs with four groups of atoms, such as `DihedralAngle`.
4. `AxisCV`. Subclass of CVs that are projected on a determinate axis. This includes `Component` and `PrincipalMoment`.
5. `CollectiveVariable` General base class for all CVs. In PySAGES, CVs that directly derive from this class, and do not belong to the previous groups, include: `RingPhaseAngle`, `RingAmplitude`, `RadiusofGyration`, `Asphericity`, `Acylindricity`, `ShapeAnisotropy`, `RingPuckeringCoordinates`[43] (vector).

In PySAGES we provide users with a simple framework for defining CVs, which are automatically differentiated with JAX. To



```cpp
// Preamble ommited

class ParallelProjectionCV : public CollectiveVariable {
public:
    ParallelProjectionCV(int atomid1, int atomid2, int atomid3) :
        atomids_({atomid1, atomid2, atomid3})
    { }

    void Initialize(const Snapshot& snapshot) override
    {
        // Code ommited for brievity
    }

    void Evaluate(const Snapshot& snapshot) override
    {
        auto n = snapshot.GetNumAtoms();

        auto idx_i = snapshot.GetLocalIndex(atomids_[0]);
        auto idx_j = snapshot.GetLocalIndex(atomids_[1]);
        auto idx_k = snapshot.GetLocalIndex(atomids_[2]);

        auto com_i = snapshot.CenterOfMass(idx_i);
        auto com_j = snapshot.CenterOfMass(idx_j);
        auto com_k = snapshot.CenterOfMass(idx_k);

        auto rik = com_i - com_k;
        auto rjk = com_j - com_k;
        auto rij = rjk - rik;

        auto nrij = rij.norm();
        auto nij = (1.0 / nrij) * rij;
        val_ = nij.dot(-rik);  // Write the CV value

        // Manual computation of the gradient
        std::fill(grad_.begin(), grad_.end(), Vector3{0, 0, 0});
        grad_.resize(n, Vector3{0, 0, 0});

        Matrix3 dij = Matrix3::Zero();
        dij(0, 0) = ( -(nij[1] * nij[1] + nij[2] * nij[2]) / nrij );
        dij(0, 1) = ( nij[0] * nij[1] / nrij );
        dij(0, 2) = ( nij[0] * nij[2] / nrij );
        dij(1, 0) = ( nij[1] * nij[0] / nrij );
        dij(1, 1) = ( -(nij[0] * nij[0] + nij[2] * nij[2]) / nrij );
        dij(1, 2) = ( nij[1] * nij[2] / nrij );
        dij(2, 0) = ( nij[2] * nij[0] / nrij );
        dij(2, 1) = ( nij[2] * nij[1] / nrij );
        dij(2, 2) = ( -(nij[1] * nij[1] + nij[0] * nij[0]) / nrij );

        grad_[idx_i] = dij * (-rik) - nij;
        grad_[idx_j] = dij * rik;
        grad_[idx_k] = nij;
    }

    static ParallelProjectionCV* Build(
        const Json::Value& json, const std::string& path
    ) {
        // Code ommited for brievity
    }

private:
    Label atomids_;
};
```

SSAGES

```python
// Preamble ommited

class ParallelProjection(ThreePointCV):
    @property
    def function(self):
        return parallel_projection


def parallel_projection(p1, p2, p3):

    r1 = barycenter(p1)
    r2 = barycenter(p2)
    r3 = barycenter(p3)
    a = r3 - r1
    b = r2 - r1
    return np.dot(a, b) / norm(b)
```

PySAGES

Apart from the usual overhead involved in writting C++ code in comparison to Python, the gradients of a cv need to be manually implemented in SSAGES, whereas in PySAGES these are automatically computed with JAX.

🟦	CV calculation
🟥	Gradient calculation

Fig. 3 Example of how to write a CV in PySAGES. On the left is the same CVs written in SSAGES and on the right the PySAGES version. In general, the only requirement is for the user to write the CV as a differentiable function in JAX.

illustrate this, we compare how to write the calculation of a CV that measures the projection of the vector between two groups of atoms over the axis that passes by other two groups, in both SSAGES and PySAGES (see Fig. 3). In PySAGES the gradient calculation is done automatically whereas in SSAGES it has to be coded explicitly.

Data-driven and differentiable CVs discovered using artificial neural networks (e.g. autoencoders)[15,18,24,44,45] with arbitrary featurizations of atoms can in principle be implemented in PySAGES based on the above general abstract classes of CVs.

RESULTS AND DISCUSSION

To evaluate a software package like PySAGES, we must consider at least two factors: physical correctness and computational performance.

First, to assess the correctness of the enhanced sampling methods implemented in PySAGES, we present in Supplementary Discussion 1 the free-energy landscape for the dihedral angles ϕ and ψ of alanine dipeptide (ADP). This example is commonly used to benchmark new enhanced sampling algorithms. Similarly, we also show in Supplementary Discussion 1 the free-energy as a function of the dihedral angle of butane. Our results show that PySAGES reproduces the expected free-energy landscapes using different methods and backends. In Section "Example applications of enhanced sampling with PySAGES," we further investigate the applicability and correctness of PySAGES beyond these simple model systems.

In addition, we showcase how extensible PySAGES is by discussing how to implement a complex collective variable to model the interface between two materials. This example also shows the power of differential programming for CV declaration in PySAGES.

Finally, we demonstrate the performance of PySAGES on GPUs with two different backends in the Section "Performance." In particular, we compare the performance of enhanced sampling simulations to the performance of pure MD simulations, as well as other enhanced sampling implementations.

Example applications of enhanced sampling with PySAGES

To demonstrate the versatility and effectiveness of PySAGES in different contexts, we present several examples of how enhanced sampling methods can be used to gain valuable insights in various fields including biology, drug design, materials engineering, polymer physics, and ab-initio simulations. These examples showcase how PySAGES can be used in diverse research areas and the utility of different enhanced sampling methods and backends.

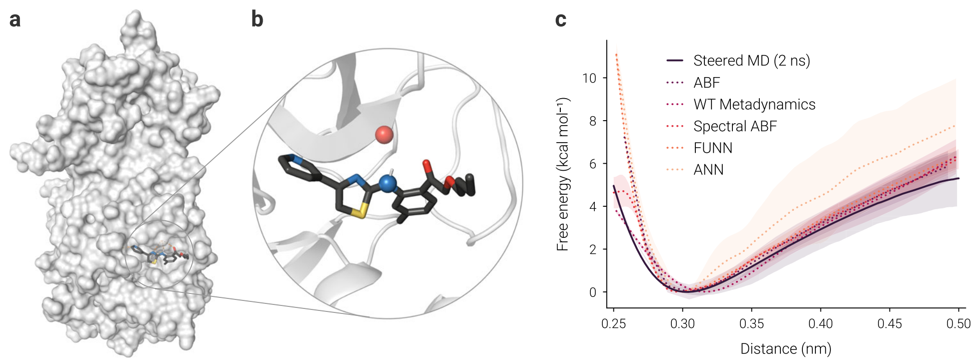


Fig. 4 Dynamical Undocking (DuCK) method in detail. For a proposed binding mode obtained from classical docking, a short run using MD simulations is carried out and the most stable receptor-ligand native contact is selected from that run. **a** Protein-ligand complex. **b** Close-up to the binding pocket, the collective variable is the distance between the atoms marked as spheres. **c** Comparison between different methods in PySAGES for DuCK calculations averaged over 5 different replicas for each method (shaded regions represent the standard deviation). The reference, Steered MD simulations were run for 2 ns. In comparison, different methods in PySAGES are used considering simulation period 10 times shorter: only ANN[10] provides inferior performance against the reference; Spectral ABF[42] or FUNN[11] give the best performance, being able to converge faster to the reference free energy with one order of magnitude simulation time reduction.

Overall, these examples confirm that the enhanced sampling methods implemented in PySAGES work as intended and provide results consistent with existing literature.

Structural stability of protein-ligand complexes for drug discovery. High-throughput docking techniques are a widely-used computational technique in drug lead discovery. However, these techniques are limited by the lack of information about protein conformations and the stability of ligands in the docked region[46]. To address this issue, the Dynamical Undocking (DuCK) method was developed to evaluate the structural stability of the ligand binding by calculating the work required to break the most important native contact (hydrogen bond interactions) in the protein-ligand complex[47]. It is important to highlight several key aspects regarding this technique: in DuCK, the quantity of interest is the work necessary to depart from the ideal hydrogen bond configuration and reach the quasi-bound state (defined in practice as the point along the simulation where the work profile exhibits the highest value). Therefore, we only consider the response of the ligand-protein complex to a small perturbation with respect to the ideal binding configuration. More importantly, as the unbound state is not considered and never reached, this quantity cannot provide information about the binding free energy. Instead, the main objective here is to simply indicate if the hydrogen bond under investigation gives rise to a (local) minimum in the free-energy landscape and to estimate the depth of that minimum. Thus, DuCK is complementary and orthogonal to classical docking, making both techniques useful for drug discovery. We note here that DuCK can be slow to converge when combined with traditional enhanced sampling techniques[47], making it unsuitable for high-throughput drug discovery protocols.

Here, we demonstrate how PySAGES with OpenMM can be used efficiently in drug discovery applications, where the user-friendly interface, native parallel capabilities, and enhanced sampling methods with fast convergence are synergistically combined to accelerate the virtual screening of ligand databases. In this example, we study the main protease (Mpro) of SARS-CoV-2 virus (PDB: 7JU7[48]), where the ligands were removed and the monomer A was selected as the docking receptor. A ligand with SMILES string `CCCCOCC(=O)c1ccc(C)cc1N[C@H]1N[C@@H](c2cccnc2)CS1` was docked using rDock[49]. The best scoring pose was used to initialize the system, which was simulated using the ff14SB[50], TIP3P[51], and GAFF[52] force fields. A 10 ns equilibration procedure was carried out to find the most stable hydrogen bond between the ligand and the protein. The last frame of this equilibration was then used to initialize the enhanced sampling calculations in PySAGES with ABF, metadynamics, FUNN, ANN, and Spectral ABF. These methods were compared against the same system simulated using Amber20[53] with Steered Molecular Dynamics (see Fig. 4c). Our results suggest that we can reduce the simulation time by an order of magnitude using recently developed enhanced sampling methods like Spectral ABF or FUNN. This can greatly accelerate the drug discovery process and help identify potential drug leads more quickly.

Fission of a diblock copolymer spherical domain. We now investigate the fission of a single spherical domain of a diblock copolymer using a coarse-grained model. We use a soft, coarse-grained dissipative particle dynamics (DPD) model published in previous studies[54,55]. The model consists of $n = 200$ chains with $N = 256$ beads each, representing a liquid polymer melt. The first $N_A = 16$ beads in each chain are type A, while the remaining $N_B = 240$ are type B.

A standard DPD potential is used to enforce incompressibility with a repulsion parameter of $A_{ii} = 5\, k_B T/\sigma^2$. However, a higher interaction of $A_{AB} = A_{ii} + \Delta A\, k_B T/\sigma^2$, with $\Delta A \in [0.1, 0.4]$ is applied between unlike particles to create a repulsion that leads to a microphase separation. A Flory-Huggins parameter $\Delta A \propto \chi N > 0$ can characterize this phase separation. The interaction range of this non-bonded potential is 1σ, as well as the range of the DPD thermostat that keeps the temperature at $T = 1\, k_B T = 1\, \epsilon$.

In addition, a harmonic spring force with zero resting length is used to connect the beads to polymer chains with a spring constant of $k = 16/3\, k_B T/\sigma^2$, resulting in an average bond length of $b_0 = 0.75\, \sigma$. The equilibrium phase for this polymer melt is a BCC phase of spherical A droplets inside a B melt[56]. However, we confine the polymer to a tight cubic simulation box of length $L_0 = 10\, \sigma$, which results in a single A spherical domain in the B matrix. We integrate the simulation with a time step of $\Delta t = 10^{-3}\, \tau$ and each simulation is equilibrated for $t = 1000\, \tau$, followed by a production run of $t = 1000\, \tau$ as well. A discussion of the GPU performance of this system with and without PySAGES can be found in the Section "Performance."

After defining the diblock copolymer system, the next step is to define a CV within the system. In this case, we are interested in the fission of the single spherical A domain into two equally sized smaller A domains. To achieve this, we divide the polymer chains into two groups: the first $n = 100$ chains are going to form the first small domain (blue in Fig. 5) and the second $n = 100$ chains form the second spherical domain (red in Fig. 5). To define and enforce the separation of the two groups, we define our CV as the distance, R, between the center of mass of the blue A-tails and the center of mass of the red A-tails. Initially, without biasing, the two

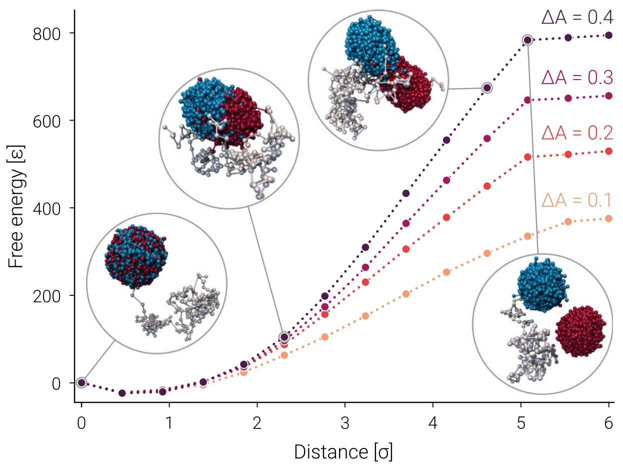


Fig. 5 Free energy landscape of the fission of a spherical diblock-copolymer domain. The chain ends forming the spherical domain are split into two groups (blue) and (red), the other chain ends not visible for clarity except for a single chain (grey). Initially, a single spherical domain is formed, but as we constraint the center of mass between the blue and red groups further, the domain first elongates and then separates completely. During this separation, the free energy continuously increases and the increase is steeper for high repulsion between unlike type ΔA. As soon as the domain is separated, the free energy plateaus.

groups form a single spherical domain and blue and red polymer tails are well mixed, as shown at small $R < 1\sigma$ in Fig. 5.

To study the separation of the spherical domain, we use harmonic biasing (see Section "Enhanced Sampling Methods") to enforce a separation distance R_0 between the two groups. The high density in the system, $\sqrt{\mathcal{N}} = \rho_0 R_0^3 / N \approx 344$, leads to low fluctuations and suppression of unfavorable conformations. Therefore, we use a high spring force constant of $k_{CV} = 1500\,\epsilon/\sigma^2$ to facilitate the separation.

We investigate a separation of $R \in [0, 6]\,\sigma$ with 14 replicas and use umbrella integration (see Section "Enhanced Sampling Methods") to determine the free energy profile, as shown in Fig. 5. As we increase the external separation distance R_0, we observe how the single domain splits into two. At a low separation distance $R < 2\,\sigma$, the single domain is mostly undeformed, but the two groups separate inside the single spherical domain. Increasing the separation distance further goes beyond the dimensions of the spherical domain, leading to the deformation of the domain into an elongated rod-like shape. The two groups still maintain a connection to minimize the AB interface.

At a separation between $4\,\sigma$ and $5\,\sigma$ the deformation becomes so strong, that the penalty of forming two separate A and B spherical domains is lower than the entropic penalty of keeping a single domain elongated with the AB interface joining both subdomains of the droplet. After the separation, the free energy landscape remains indifferent to the separation, since there is no longer interaction between the two domains.

The free energy profile of separation is controlled by the repulsion of unlike types $\chi N \propto \Delta A$. The stronger the repulsion, the more energy is necessary to enlarge the AB surface area for the fission. For the strongest interaction $\Delta A = 0.4\,\epsilon$, the total free energy barrier reaches about $800\,\epsilon$, while for the lowest $\Delta A = 0.1\,\epsilon$ it remains below $400\,\epsilon$. Both barriers are orders of magnitude larger than thermal fluctuations $1\,k_BT = 1\,\epsilon$, so a spontaneous separation is not expected and the fission can only be studied via enhanced sampling.

It is interesting to note that at the lowest separation distance ($R_0 = 0$) it is not the lowest free energy state. Enforcing perfect mixing is not favorable, as the two groups naturally want to separate slightly optimizing the entropy of the chain end-tails.

Liquid crystal anchoring in aqueous interfaces. Liquid crystals (LCs), materials that flow like liquids but have anisotropic properties as crystals, have been used lately as prototypes for molecular sensors at interfaces given the high sensitivity in their anchoring behavior relative to small concentration of molecules at aqueous interfaces. The presence of molecules at the interface changes drastically the free energy surface of LC molecules relative to their orientation and distance to such interface. In this example, we consider the LC 4-cyano-4′-pentylbiphenyl (5CB) at the interface of pure water and sodium lauryl sulfate (SDS). For 5CB and water, previous work has focused on obtaining the free energy surface of a 5CB at the water interface[57]. In our case, hybrid anchoring conditions have been imposed on a 16 nm slab of 1000 5CB molecules in the nematic phase (300 K) interacting with a 3 nm slab of water with 62 molecules of (SDS) at one of the interfaces. The force fields used are: the united atom model for 5CB developed by Tiberio et al.[58], TIP3P[51] for water, GAFF[52] and Lipid 17 for SDS. The CVs chosen to study this system are the distance of the center of mass of one molecule of 5CB at each one of the interfaces (see Supplementary Discussion 2), and the tilt orientation of the same molecule with respect to the z axis of the box. For the enhanced sampling method, we choose FUNN (see Section "Enhanced Sampling Methods"). The free energy surfaces for the pure water and with SDS at the interface are both displayed in Fig. 6. We can observe that the free energy surface of pure water shows a minimum corresponding to a parallel orientation to the surface with a similar shape that one calculated in[57]. On the contrary, the presence of SDS transforms the minimum to a maximum in the same relative position and orientation to the interface (Fig. 6a), moving now the minima to a perpendicular orientation of 5CB to the interface, in agreement to the experimental observation of change from planar to homeotropic anchoring in the presence of SDS in water.

Ab initio enhanced sampling simulations. In the domain of ab initio simulations, particularly when applied to heterogeneous catalysis and electrochemistry, accounting accurately for the dynamic and entropic contributions to the free energy is crucial for a precise characterization of the observed phenomena[23]. The inherent limitations of classical force fields preclude the accurate simulation of essential bond-breaking processes, which are central to catalysis. This necessitates the implementation of MD simulations based on first-principles calculations.

In scenarios where reactive events are hindered by high free energy barriers, the incorporation of enhanced sampling techniques into a simulation becomes an essential component of such calculations. By integrating PySAGES with ASE[59], we offer access to a broad spectrum of first-principles calculators. To illustrate this, we have used CP2K[60] as a simulator to model the dissociation of NaCl in water, a representative reaction that requires enhanced sampling[61,62]. In this example, the CV is defined as the separation distance between a sodium and a chlorine atom. The NVT ab initio molecular dynamics simulation presented here was executed using a double-ζ basis set[63] with an energy cutoff of 400 Ry, in conjunction with GTH pseudopotentials[64]. The PBE functional was used with Grimme D3 vdW correction[65,66]. The simulation involved a cubic box of edge length 15 Å containing 113 water molecules, one sodium, and one chlorine atom, corresponding to a density of 1000 kg m^{-3}. Once the system was equilibrated under 300 K for a total of 150 ps, we implemented Spectral ABF as our enhanced sampling method of choice (refer to Section "Enhanced Sampling Methods").

After a total of 150 ps of sampling, the results of enhanced sampling, displayed in Fig. 7, reveal two free energy minima along the CV axis. The first minimum (CV = 2.8 Å) represents the undissolved NaCl molecule (contact-ion pair), while the second (CV = 4.75 Å) corresponds to the solvated Na$^+$ and Cl$^-$ ions (solvent-separated-ion pair). This free energy profile aligns well

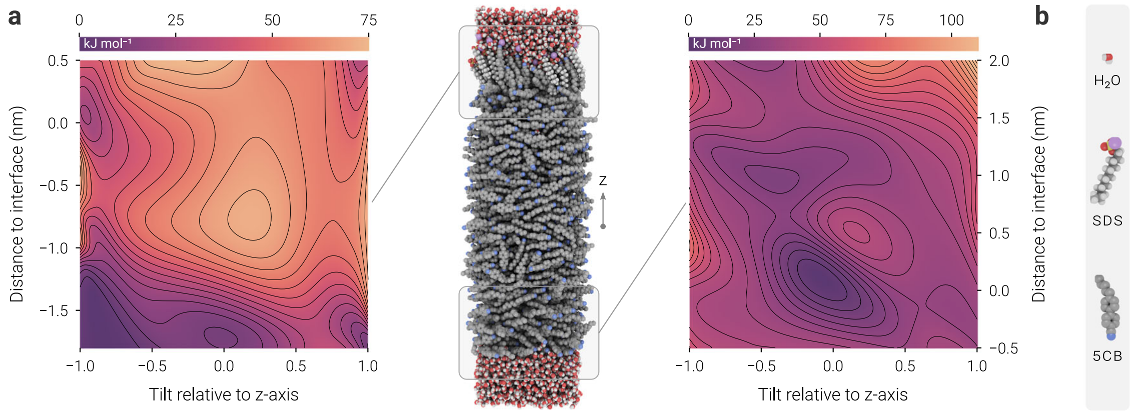


Fig. 6 Free energy surface (FES) of 5CB in a hybrid anchoring slab with SDS and water. Center: Snapshot of the system **a** FES of 5CB molecule near the water--SDS interface. **b** FES of 5CB near a pure water interface. Both FES were obtained with PySAGES and OpenMM using the FUNN method.

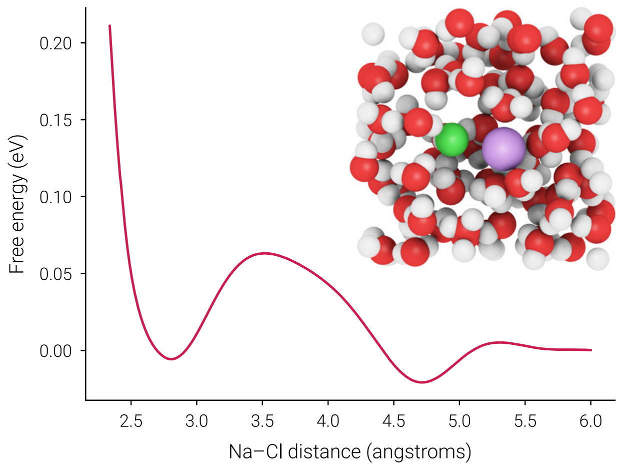


Fig. 7 Free energy (T = 300 K) of the Na–Cl distance when in solution. Calculations where performed with ASE + CP2K using Spectral ABF in PySAGES.

with previously reported values[61,62], further affirming the accuracy and reliability of the combination of PySAGES and ASE.

Furthermore, the Spectral ABF method offered a marked increase in efficiency when applied to this problem. A previous study, employing a constrained-MD method[61], required a total of 126 ps of simulation time for a limited sampling with 21 windows. By contrast, the current calculation accelerated the process, requiring a total sampling time of 150 ps for 140 bins. The efficiency of the Spectral ABF method is further illustrated by comparison to another study[62] that applied ABF, where the total sampling time of 3.0 ps per sampling bin is decreased to 1.1 ps per sampling bin in the present example.

Enhanced sampling with machine learned force fields. Deep neural networks (NN) force fields can retain the accuracy of ab initio MD while allowing for computational costs similar to those of classical MD. Through ASE it is possible to access NN potentials such as DeepMD[67], and the Gaussian Approximation Potential (GAP)[68]. Additionally, JAX MD allows to leverage more general NN potentials that can be used in enhanced sampling calculations. Coupling of PySAGES with ASE or JAX MD can be used in active learning of NN force fields by efficiently sampling rare events using any of the enhanced sampling methods provided by PySAGES. For example, it could allow to implement the strategy described in Ref. [69] where parallel tempering metadynamics was used to generate accurate NN force field in urea decomposition in water.

To test the capabilities of PySAGES to handle different NN force fields, we have selected three different systems trained with the methods mentioned above. First, we studied a graph neural network (GNN) model of a Si crystal[70] with PySAGES and JAX MD. In this case, a crystalline Si system of 64 atoms was used, and the CV was the Si–Si distance for the the crystal. The results of Fig. 8a show that for this model, the minimum in the free energy corresponds almost exactly to the experimental value for the Si–Si nearest distance of 2.35 Å.

Next, in Fig. 8b, a GAP potential was used for Si–H amorphous mixtures[71]. In this case, a system of 244 atoms was used, and the collective variable is the bond angle between a triad of Si–Si–H atoms in the mixture. The global minimum in free energy agrees with the histogram taken from unbiased simulations reported in[71].

Lastly, for DeepMD, we use a pre-trained model for water, where the enhanced sampling system is one single water molecule in vacuum, the collective variable is the internal angle of the molecule and the sampling method is ABF (see Section "Enhanced Sampling Methods"). The results in Fig. 8c show that the minimum for this free energy profile is around 105 degrees, which is within the range of the experimental value.

Case study: A collective variable for interfaces. When two immiscible liquids are in contact with each other, the density of one liquid experiences a gradual change in the vicinity of the interface with the other. This transition region is the liquid-liquid interface and its position has high importance in many studies (as in our LCs example above). However, defining the location of such an interface is not a trivial task since it generally fluctuates as the simulation progresses. As a representative CV for the interface, we can utilize the position of the point where the gradient of the density is maximized. More formally, let $\rho(x)$ denote the density of a liquid of interest at a coordinate x on the perpendicular axis. We would like to find the location of the interface:

$$I = \arg\max_{x} |\rho'(x)| \tag{4}$$

However, the density function $\rho(x)$ is not directly measurable in a molecular simulation, as the coordinates of atoms are discrete. To obtain an approximation of $\rho(x)$, we divide the coordinates into multiple bins, each with a width of δ, and create a histogram $p(x)$ that records the number of atoms falling into the bin around position x. In other words,

$$p(x) = \sum_{i=1\ldots n} [|x_i - x| < \delta/2] \tag{5}$$

in which x_i denotes the coordinate of atom i. As written above, $p(x)$ is non-differentiable. Therefore, as in other works[72], we utilize the

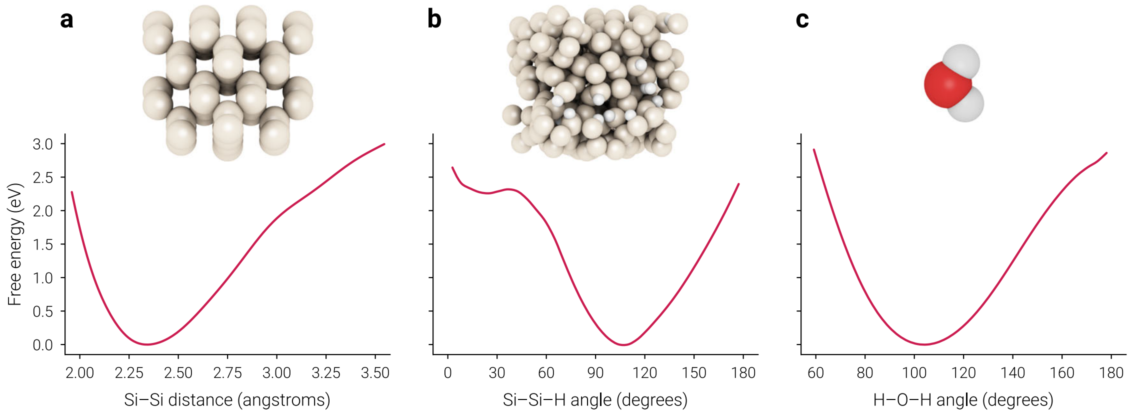


Fig. 8 Free energy calculation for different systems modeled with machine-learned force fields. a Si--Si distance of a GNN model with JAX MD. **b** Si--Si--H angle of GAP model with ASE. **c** Water internal angle from a DeepMD model with ASE.

kernel density trick with a Gaussian kernel to modify $p(x)$. The modified $\tilde{p}(x)$, is defined as:

$$\tilde{p}(x) = \sum_{i=1\ldots n} \exp\left(-\frac{(x_i - x)^2}{2\sigma^2}\right) \tag{6}$$

in which σ is a hyperparameter that decides the width of the Gaussian kernel. Then, the gradient of the density can be approximated as:

$$\tilde{p}'(x) = \frac{\tilde{p}(x+\delta/2) - \tilde{p}(x-\delta/2)}{\delta} \tag{7}$$

and we calculate the location of the interface as $I = \arg\max_x |\tilde{p}'(x)|$. The arg max operator is also non-differentiable. As a result, we replace it with a softmax function that transforms the raw input into a probability. Denote the m bins as $j = 1,\ldots,m$, and finally we calculate the location of the interface as:

$$I = \frac{\sum_j x_j \exp|\tilde{p}'(x_j)|}{\sum_j \exp|\tilde{p}'(x_j)|} \tag{8}$$

As demonstrated in the code snippet for this CV, provided in Supplementary Discussion 2, PySAGES allows for the concise and straightforward implementation of complex CVs such as this one.

Performance

Our analysis revealed that PySAGES is at least ~14–15 times faster than SSAGES on an Nvidia V100 GPU machine. To obtain this estimate, we ran enhanced sampling using umbrella sampling along the center of mass distance between two spherical polymer domains to measure the free energy landscape of the fission of a spherical diblock-copolymer blend (Fig. 5) described in Section "Example applications of enhanced sampling with PySAGES." For support and compatibility across libraries and MD engine versions, we estimated the performance with SSAGES v0.9.2-alpha and PySAGES v0.3.0 using HOOMD-blue v2.6.0 and HOOMD-blue v2.9.7, respectively.

We highlight that the performance of PySAGES, particularly in terms of scaling with the number of GPUs, is primarily a reflection of the simulation backend's performance. As a result, our subsequent analysis concentrates on the relationship between PySAGES and the backend and the impact this interaction has on performance. For a comprehensive performance evaluation of the individual backends, we refer readers to the specific literature associated with each, such as that provided for HOOMD-blue.

GPU utilization analysis. PySAGES is designed to execute every compute-intensive step of a simulation on the GPU and have zero copy instruction between GPU device and host CPU memory for its explicit backends for HOOMD-blue[2] and OpenMM[3], while still providing Python code for the user through JAX[73]. In this section, we investigate the calculation efficiency of PySAGES by examining two example systems, one for each backend.

For HOOMD-blue, we consider a system of highly coarse-grained DPD diblock-copolymers as discussed in Section "Example applications of enhanced sampling with PySAGES." The simulation box contains a total of $nN = 51\,200$ particles at a density of $\rho = 51.2/\sigma^3$, which we use for benchmarking purposes with an Nvidia V100 GPU hosted on an Intel Xeon Gold 6248R CPU @ 3.00GHz. Running only with HOOMD-blue v2.9.7 we achieve an average time steps per second (TPS) of 754, which is the expected high performance of HOOMD-blue on GPUs.

Figure 9a shows a detailed profiled timeline during the execution of a single time step. During 1.8 ms, HOOMD-blue spends the most computational effort on the calculation of pairwise DPD forces. It can be noted that HOOMD-blue is designed to have almost no idle time of the GPU during a time step. As soon as PySAGES is added we observe that an additional computation takes part to calculate the CV and add the forces to every particle. This causes a small period of idle of the GPU, since the execution also requires action of the Python runtime interface with JAX. In the future, we plan to launch the calculation of CV asynchronously with the regular force calculation, which would hide this small CPU-intensive GPU idle time. However, we measure that the total delay due to the extra computation is only about 247 μs only. We regard this to be an acceptable overhead given PySAGES advantages such as user-friendly definition of CVs.

In order to connect multiple points in CV space we can use enhanced sampling methods such as umbrella sampling or the improved string method (see Section "Enhanced Sampling Methods") to calculate the MFEP. Common for these advanced sampling methods is that multiple replicas of the system are simulated. With PySAGES we easily parallelize their execution using the Python module `mpi4py` and its `MPIPoolExecutor`. This enables us to execute replicas of the simulations on multiple GPUs even as they span different host machines. In our example, we used 14 replicas for umbrella integration with 7 Nvidia V100 GPUs. The use of a single V100 GPU to execute the simulations with $5 \cdot 10^5$ time steps for all replicas takes 2 hours and 59 minutes. Ideal scaling with 7 GPUs would reduce the time to solution to about 26 minutes. With our MPI-parallel implementation, we achieve a time-to-solution of 28 minutes. Synchronization overhead and nonparallel aspects like final analysis sum up to 2 minutes or about 9% overhead. This multi-GPU implementation via MPI enables automatically efficient enhanced sampling in high-performance computing (HPC) environments. We emphasize that this multi-GPU parallelism exists in conjunction with the

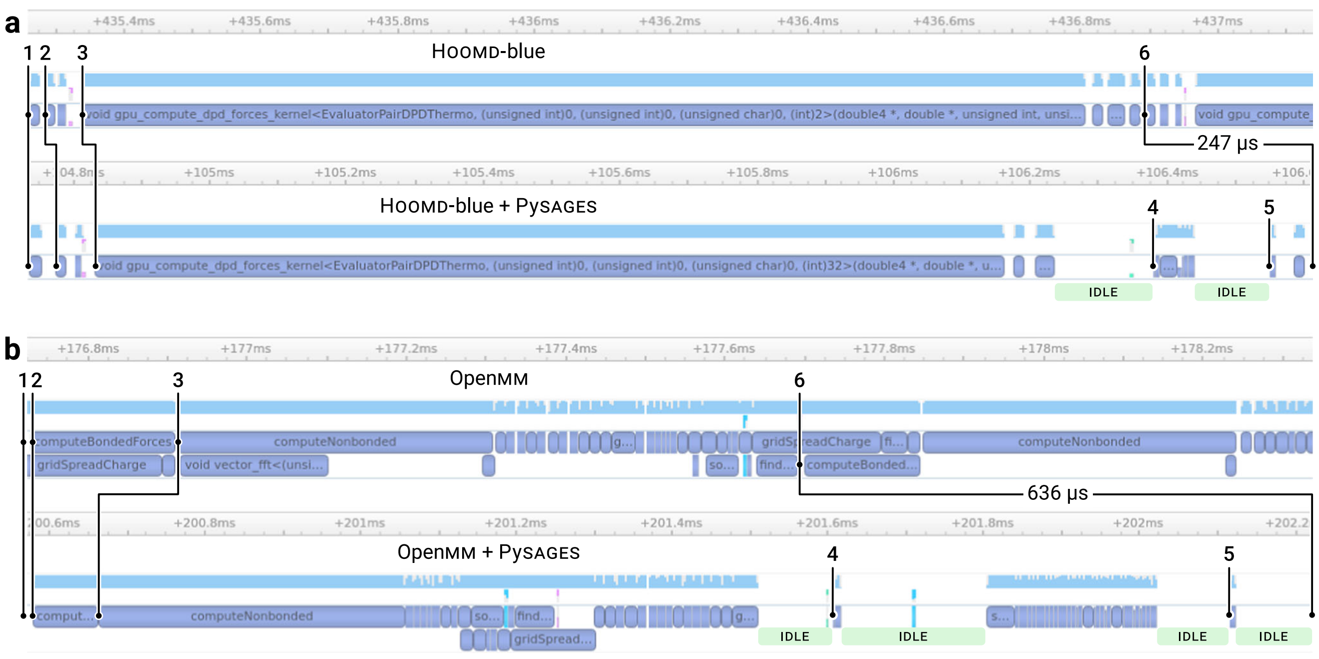


Fig. 9 Profiled timelines for a single-time step of unbiased and biased execution with HOOMD-blue and OpenMM. The profiles were recorded with Nvidia Nsight systems on an Nvidia V100 GPU. Light-blue represents the GPU activity while dark-blue represents individual CUDA compute kernels. The numbered lines indicate the same compute steps in all simulations: (1) start of integration step, (2) compute of bond forces, (3) pair-wise forces, (4) calculation of the CV, (5) addition of the harmonic biasing force to the backend, and (6) end of the integration step. (4) and (5) are PySAGES only and are executed on the GPU. We observe GPU idle time during the PySAGES Python coordination with GPU--JAX/CuPy (green bar), but note that there is no memory copies even within the GPU memory. **a** 1.8 ms of recorded HOOMD-blue execution. The top row shows a vanilla HOOMD-blue simulation step, while the bottom row shows a PySAGES/HOOMD-blue simulation with harmonic biasing of a center of mass CV. The additional time for CV biasing per time step is 247 *μs*. **b** 1.6 ms recorded execution time line of an OpenMM OPLS simulation of 40,981 particles as polymers with PME summation for long-range Coulomb forces. OpenMM works with asynchronous GPU kernel execution, which leads to less linearly-sorted timelines. Overall, the performance degradation is more pronounced with OpenMM compared to HOOMD-blue.

parallelism provided by the simulation backend. Furthermore, this level of parallelism can only be effectively utilized in sampling methods that depend on multiple samples.

For enhanced sampling methods that are designed for single replica simulations, we offer an implementation that allows multiple replicas to run in parallel, known as embarrassingly parallel computing. In this situation, the build-in analysis averages the results from multiple replicas and estimates uncertainties.

In the previous section, we have demonstrated the fast GPU interoperability between PySAGES and HOOMD-blue via JAX. However, the concept of PySAGES is to develop enhanced sampling methods independently of the simulation backend, so here we demonstrate that similar performance can be achieved with OpenMM. Since OpenMM focuses on all-atom simulations, we simulate an all-atom model of a polymer with the BigSMILES[74] notation `{[$]CC([$])(C)C(OCC(O)CSC1=CC=C(F)C(F)=C1)=O}` with an OPLS-AA force field[75,76] including long-range Coulomb forces via particle mesh Ewald (PME). We simulate a bulk system of 40mers with 31 macromolecules present, adding up to 40,981 atoms. As a proof of concept, we calculated the center of mass for every polymer chain and biased it harmonically via PySAGES. As a performance metric, we evaluate how many simulation nano-seconds are executed per day on the same hardware configuration as a the HOOMD-blue example above. For the unbiased, pure OpenMM simulation we achieve a performance of ≈136 ns per day. For the PySAGES biased simulation, we achieve a performance of ≈75 ns per day, equating to a biasing overhead of approximately 50%. Figure 9b shows a similar time series analysis as for HOOMD-blue.

It is notable that OpenMM's execution model makes more use of parallel execution of independent kernels, which also changes the order of execution compared to HOOMD-blue. As a result, the same GPU synchronization changes the execution more drastically than in HOOMD-blue. Additionally, a single time step for this system is faster executed compared to HOOMD-blue, making the synchronization overhead more noticeable. In this case, parallelization of PySAGES and OpenMM is projected to have a bigger performance advantage. Furthermore, we notice that the calculation of the center of mass and the biasing of all 31 polymer chains is more costly than the single CV in the previous example. The combination of these factors explain the higher PySAGES overhead for this OpenMM simulation, but overall performance is good and significantly better for alternative implementations that require CV calculations on the GPU.

Outlook

We have introduced PySAGES, a library for enhanced sampling in molecular dynamics simulations, which allows users to utilize a variety of enhanced sampling methods and collective variables, as well as to implement new ones via a simple Python and JAX-based interface.

We showed how PySAGES can be used through a number of example applications in different fields such as drug design, materials engineering, polymer physics, and ab-initio MD simulations. We hope that these convey to the reader the flexibility and potential of the library for addressing a diverse set of problems in a high-performance manner.

As our analysis showcased, for large problems, PySAGES can perform biased simulation well over one order of magnitude faster than a library such as SSAGES even when the backend already performs computations on a GPU.

Nevertheless, as with any newly developed software, PySAGES will continue to undergo improvements. In the near term, we plan to optimize PySAGES-side computations to run fully asynchronously with the computation of the forces of the backend, which will further enhance its current performance. We also invite the community to contribute to the development of PySAGES, whether by suggesting new features, reporting bugs, or contributing code.

Overall, we believe that PySAGES provides a useful tool for researchers interested in performing molecular and ab-initio simulations in multiple fields, due to its user-friendly framework for defining and using sampling methods and collective variables, as well as its high performance on GPU devices.

Looking further ahead, we are excited about the potential for PySAGES to enable fully end-to-end differentiable free energy calculations. This will provide new possibilities for force-field and materials design, which would drive significant advances in these areas.

DATA AVAILABILITY

The data supporting the findings of this study is available within this article, its Supplementary Information, and the GitHub repository https://github.com/SSAGESLabs/PySAGES-examples.

CODE AVAILABILITY

The code for PySAGES is available in the GitHub repository: https://github.com/SSAGESLabs/PySAGES.

Received: 7 April 2023; Accepted: 12 December 2023;
Published online: 14 February 2024

REFERENCES

1. Shaw, D. E. 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*, 41–53 (IEEE, 2014).
2. Anderson, J. A., Glaser, J. & Glotzer, S. C. HOOMD-blue: A Python package for high-performance molecular dynamics and hard particle Monte Carlo simulations. *Comput. Mater. Sci.* **173**, 109363 (2020).
3. Eastman, P. et al. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. *PLOS Comput. Biol.* **13**, 1–17 (2017).
4. Schoenholz, S. & Cubuk, E. D. JAX, M.D. a framework for differentiable physics. In *NeurIPS*, vol. 33, 11428–11441 (2020).
5. Thompson, A. P. et al. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Comp. Phys. Comm.* **271**, 108171 (2022).
6. Abraham, M. J. et al. Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* **1**, 19–25 (2015).
7. Tribello, G. A., Bonomi, M., Branduardi, D., Camilloni, C. & Bussi, G. PLUMED 2: New feathers for an old bird. *Comput. Phys. Commun.* **185**, 604–613 (2014).
8. Fiorin, G., Klein, M. L. & Hénin, J. Using collective variables to drive molecular dynamics simulations. *Mol. Phys.* **111**, 3345–3362 (2013).
9. Sidky, H. et al. SSAGES: Software suite for advanced general ensemble simulations. *J. Chem. Phys.* **148**, 044104 (2018).
10. Sidky, H. & Whitmer, J. K. Learning free energy landscapes using artificial neural networks. *J. Chem. Phys.* **148**, 104111 (2018).
11. Guo, A. Z. et al. Adaptive enhanced sampling by force-biasing using neural networks. *J. Chem. Phys.* **148**, 134108 (2018).
12. Sevgen, E., Guo, A. Z., Sidky, H., Whitmer, J. K. & de Pablo, J. J. Combined force-frequency sampling for simulation of systems having rugged free energy landscapes. *J. Chem. Theory Comput.* **16**, 1448–1455 (2020).
13. Wang, D. et al. Efficient sampling of high-dimensional free energy landscapes using adaptive reinforced dynamics. *Nat. Comput. Sci.* **2**, 20–29 (2022).
14. Schwantes, C. R. & Pande, V. S. Improvements in Markov state model construction reveal many non-native interactions in the folding of NTL9. *J. Chem. Theory Comput.* **9**, 2000–2009 (2013).
15. Chen, W. & Ferguson, A. L. Molecular enhanced sampling with autoencoders: On-the-fly collective variable discovery and accelerated free energy landscape exploration. *J. Comput. Chem.* **39**, 2079–2102 (2018).
16. Mardt, A., Pasquali, L., Wu, H. & Noé, F. VAMPnets for deep learning of molecular kinetics. *Nat. Commun.* **9**, 1–11 (2018).
17. Chen, W., Sidky, H. & Ferguson, A. L. Capabilities and limitations of time-lagged autoencoders for slow mode discovery in dynamical systems. *J. Chem. Phys.* **151**, 064123 (2019).
18. Chen, W., Sidky, H. & Ferguson, A. L. Nonlinear discovery of slow molecular modes using state-free reversible VAMPnets. *J. Chem. Phys.* **150**, 214114 (2019).
19. Sidky, H., Chen, W. & Ferguson, A. L. Molecular latent space simulators. *Chem. Sci.* **11**, 9459–9467 (2020).
20. Lee, T.-S. et al. GPU-accelerated molecular dynamics and free energy methods in Amber18: performance enhancements and new features. *J. Chem. Inf. Model.* **58**, 2043–2050 (2018).
21. Phillips, J. C. et al. Scalable molecular dynamics on CPU and GPU architectures with namd. *J. Chem. Phys.* **153**, 044130 (2020).
22. Kobayashi, C. et al. GENESIS 1.1: A hybrid-parallel molecular dynamics simulator with enhanced sampling algorithms on multiple computational platforms. *J. Comput. Chem.* **38**, 2193–2206 (2017).
23. Piccini, G. et al. Ab initio molecular dynamics with enhanced sampling in heterogeneous catalysis. *Catal. Sci. Technol.* **12**, 12–37 (2022).
24. Sidky, H., Chen, W. & Ferguson, A. L. Machine learning for collective variable discovery and enhanced sampling in biomolecular simulation. *Mol. Phys.* **118**, e1737742 (2020).
25. Jackson, N. E., Webb, M. A. & de Pablo, J. J. Recent advances in machine learning towards multiscale soft materials design. *Curr. Opin. Chem. Eng.* **23**, 106–114 (2019).
26. Tiwary, P. & van de Walle, A.*A Review of Enhanced Sampling Approaches for Accelerated Molecular Dynamics*, 195–221 (Springer, Cham, 2016).
27. Wang, A.-h, Zhang, Z.-c & Li, G.-h Advances in enhanced sampling molecular dynamics simulations for biomolecules. *Chin. J. Chem. Phys.* **32**, 277 (2019).
28. Miao, Y. & McCammon, J. A. Unconstrained enhanced sampling for free energy calculations of biomolecules: a review. *Mol. Simul.* **42**, 1046–1055 (2016).
29. Yang, Y. I., Shao, Q., Zhang, J., Yang, L. & Gao, Y. Q. Enhanced sampling in molecular dynamics. *J. Chem. Phys.* **151**, 070902 (2019).
30. Abrams, C. & Bussi, G. Enhanced sampling in molecular dynamics using metadynamics, replica-exchange, and temperature-acceleration. *Entropy* **16**, 163–199 (2014).
31. Limongelli, V. Ligand binding free energy and kinetics calculation in 2020. *Wiley Interdiscip. Rev. Comput. Mol. Sci.* **10**, e1455 (2020).
32. Kästner, J. Umbrella sampling. *Wiley Interdiscip. Rev. Comput. Mol. Sci.* **1**, 932–942 (2011).
33. Weinan, E., Ren, W. & Vanden-Eijnden, E. Simplified and improved string method for computing the minimum energy paths in barrier-crossing events. *J. Chem. Phys.* **126**, 164103 (2007).
34. Comer, J. et al. The adaptive biasing force method: Everything you always wanted to know but were afraid to ask. *J. Phys. Chem. B* **119**, 1129–1151 (2015).
35. Darve, E., Rodríguez-Gómez, D. & Pohorille, A. Adaptive biasing force method for scalar and vector free energy calculations. *J. Chem. Phys.* **128**, 144120 (2008).
36. Laio, A. & Parrinello, M. Escaping free-energy minima. *PNAS* **99**, 12562–12566 (2002).
37. Barducci, A., Bussi, G. & Parrinello, M. Well-tempered metadynamics: a smoothly converging and tunable free-energy method. *Phys. Rev. Lett.* **100**, 020603 (2008).
38. Hussain, S. & Haji-Akbari, A. Studying rare events using forward-flux sampling: Recent breakthroughs and future outlook. *J. Chem. Phys.* **152**, 060901 (2020).
39. Allen, R. J., Warren, P. B. & ten Wolde, P. R. Sampling rare switching events in biochemical networks. *Phys. Rev. Lett.* **94**, 018104 (2005).
40. Allen, R. J., Frenkel, D. & ten Wolde, P. R. Simulating rare events in equilibrium or nonequilibrium stochastic systems. *J. Chem. Phys.* **124**, 024102 (2006).
41. Whitmer, J. K., Chiu, C.-c, Joshi, A. A. & De Pablo, J. J. Basis function sampling: A new paradigm for material property computation. *Phys. Rev. Lett.* **113**, 190602 (2014).
42. Zubieta Rico, P. F. & de Pablo, J. J. Sobolev sampling of free energy landscapes. Preprint at: https://arxiv.org/abs/2202.01876 (2022).
43. Cremer, D. & Pople, J. A. General definition of ring puckering coordinates. *J. Am. Chem. Soc.* **97**, 1354–1358 (1975).
44. Ribeiro, J. M. L., Bravo, P., Wang, Y. & Tiwary, P. Reweighted autoencoded variational Bayes for enhanced sampling (RAVE). *J. Chem. Phys.* **149**, 072301 (2018).
45. Wehmeyer, C. & Noé, F. Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics. *J. Chem. Phys.* **148**, 241703 (2018).
46. Sethi, A., Joshi, K., Sasikala, K. & Alvala, M. Molecular docking in modern drug discovery: Principles and recent applications. In Gaitonde, V., Karmakar, P. & Trivedi, A. (eds.) *Drug Discovery and Development*, vol. 2, chap. 3, 1–21 (IntechOpen, 2019).
47. Ruiz-Carmona, S. et al. Dynamic undocking and the quasi-bound state as tools for drug discovery. *Nat. Chem.* **9**, 1755–4349 (2017).
48. Drayman, N. et al. Masitinib is a broad coronavirus 3CL inhibitor that blocks replication of SARS-CoV-2. *Science* **373**, 931–936 (2021).

49. Ruiz-Carmona, S. et al. rDock: A fast, versatile and open source program for docking ligands to proteins and nucleic acids. *PLOS Comput. Biol.* **10**, 1–7 (2014).
50. Maier, J. A. et al. ff14SB: Improving the accuracy of protein side chain and backbone parameters from ff99SB. *J. Chem. Theory Comput.* **11**, 3696–3713 (2015).
51. Jorgensen, W. L., Chandrasekhar, J., Madura, J. D., Impey, R. W. & Klein, M. L. Comparison of simple potential functions for simulating liquid water. *J. Chem. Phys.* **79**, 926–935 (1983).
52. Wang, J., Wolf, R. M., Caldwell, J. W., Kollman, P. A. & Case, D. A. Development and testing of a general amber force field. *J. Comput. Chem.* **25**, 1157–1174 (2004).
53. Case, D. A. et al. *Amber 2020* (2020).
54. Schneider, L., Heck, M., Wilhelm, M. & Müller, M. Transitions between lamellar orientations in shear flow. *Macromolecules* **51**, 4642–4659 (2018).
55. Schneider, L. & Müller, M. Rheology of symmetric diblock copolymers. *Comput. Mater. Sci.* **169**, 109107 (2019).
56. Matsen, M. W. The standard gaussian model for block copolymer melts. *J. Phys.: Condens. Matter* **14**, R21 (2001).
57. Ramezani-Dakhel, H. et al. Understanding atomic-scale behavior of liquid crystals at aqueous interfaces. *J. Chem. Theory Comput.* **13**, 237–244 (2017).
58. Tiberio, G., Muccioli, L., Berardi, R. & Zannoni, C. Towards in silico liquid crystals. realistic transition temperatures and physical properties for n-cyanobiphenyls via molecular dynamics simulations. *ChemPhysChem* **10**, 125–136 (2009).
59. Larsen, A. H. et al. The atomic simulation environment—a python library for working with atoms. *J. Phys.: Condens. Matter* **29**, 273002 (2017).
60. Kühne, T. D. et al. CP2K: An electronic structure and molecular dynamics software package - Quickstep: Efficient and accurate electronic structure calculations. *J. Chem. Phys.* **152**, 194103 (2020).
61. Timko, J., Bucher, D. & Kuyucak, S. Dissociation of NaCl in water from ab initio molecular dynamics simulations. *J. Chem. Phys.* **132**, 114510 (2010).
62. Zhang, C. et al. Dissociation of salts in water under pressure. *Nat. Commun.* **11**, 3037 (2020).
63. VandeVondele, J. & Hutter, J. Gaussian basis sets for accurate calculations on molecular systems in gas and condensed phases. *J. Chem. Phys.* **127**, 114105 (2007).
64. Goedecker, S., Teter, M. & Hutter, J. Separable dual-space gaussian pseudopotentials. *Phys. Rev. B* **54**, 1703–1710 (1996).
65. Perdew, J. P., Burke, K. & Ernzerhof, M. Generalized Gradient Approximation made simple [Phys. Rev. Lett. 77, 3865 (1996)]. *Phys. Rev. Lett.* **78**, 1396–1396 (1997).
66. Grimme, S., Ehrlich, S. & Goerigk, L. Effect of the damping function in dispersion corrected density functional theory. *J. Comput. Chem.* **32**, 1456–1465 (2011).
67. 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.* **120**, 143001 (2018).
68. Bart¢k, A. P., Payne, M. C., Kondor, R. & Csányi, G. Gaussian Approximation Potentials: The Accuracy of Quantum Mechanics, without the Electrons. *Phys. Rev. Lett.* **104**, 136403 (2010).
69. Yang, M., Bonati, L., Polino, D. & Parrinello, M. Using metadynamics to build neural network potentials for reactive events: the case of urea decomposition in water. *Catal. Today* **387**, 143–149 (2022).
70. Cubuk, E. D., Malone, B. D., Onat, B., Waterland, A. & Kaxiras, E. Representations in neural network based empirical potentials. *J. Chem. Phys.* **147**, 024104 (2017).
71. Unruh, D., Meidanshahi, R. V., Goodnick, S. M., Csányi, G. & Zimányi, G. T. Gaussian approximation potential for amorphous si : H. *Phys. Rev. Mater.* **6**, 065603 (2022).
72. Wang, W., Wu, Z., Dietschreit, J. C. B. & Gómez-Bombarelli, R. Learning pair potentials using differentiable simulations. *J. Chem. Phys.* **158**, 044113 (2023).
73. Frostig, R., Johnson, M. J. & Leary, C. Compiling machine learning programs via high-level tracing. In *SysML Conference* (SysML, 2018).
74. Lin, T.-S. et al. BigSMILES: a structurally-based line notation for describing macromolecules. *ACS Cent. Sci.* **5**, 1523–1531 (2019).
75. Jorgensen, W. L., Maxwell, D. S. & Tirado-Rives, J. Development and testing of the OPLS all-atom force field on conformational energetics and properties of organic liquids. *J. Am. Chem. Soc.* **118**, 11225–11236 (1996).
76. Schneider, L. et al. In silico active learning for small molecule properties. *Mol. Syst. Des. Eng.* **7**, 1611–1621 (2022).

ACKNOWLEDGEMENTS

This work is supported by the Department of Energy, Basic Energy Sciences, Materials Science and Engineering Division, through the Midwest Integrated Center for Computational Materials (MICCoM). L. S. is grateful for the support of the Eric and Wendy Schmidt AI in Science Postdoctoral Fellowship at the University of Chicago. R. A. is supported by the Dutch Research Council (NWO Rubicon 019.202EN.028). S. V. is supported by the U. S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Department of Energy Computational Science Graduate Fellowship under Award Number DE-SC0022158. The authors also acknowledge the Research Computing Center of the University of Chicago for computational resources.

AUTHOR CONTRIBUTIONS

P. F. Z.R. conceptualized and lead the development of the software. P. F. Z.R., L. S. and J. A. P. contributed to the design the software architecture. P. F. Z.R., L. S., G. R. P.-L., R. A., S. D., Y. W., Y. X., T. D. N. and J. A. P. took part on the development of the software. All the group of developers, C. A. M., Y. J., and S. V. validated the code and contributed to the generation of the data for the examples presented. A. L. F., J. K. W. and J. J. de P. supervised the work. All authors discussed the results and contributed to the manuscript.

COMPETING INTERESTS

A. L. F. is a co-founder and consultant of Evozyne, Inc. and a co-author of US Patent Applications 16/887,710 and 17/642,582, US Provisional Patent Applications 62/853,919, 62/900,420, 63/314,898, and 63/479,378 and International Patent Applications PCT/US2020/035206 and PCT/US2020/050466.

ADDITIONAL INFORMATION

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41524-023-01189-z.

Correspondence and requests for materials should be addressed to Juan J. de Pablo.