Journal of Cheminformatics

SOFTWARE

Open Access

MDSuite: comprehensive post-processing tool for particle simulations

Samuel Tovey[1], Fabian Zills[1], Francisco Torres-Herrador[2,3,4], Christoph Lohrmann[1], Marco Brückner[1] and Christian Holm[1*]

Abstract

Particle-Based (PB) simulations, including Molecular Dynamics (MD), provide access to system observables that are not easily available experimentally. However, in most cases, PB data needs to be processed after a simulation to extract these observables. One of the main challenges in post-processing PB simulations is managing the large amounts of data typically generated without incurring memory or computational capacity limitations. In this work, we introduce the post-processing tool: MDSuite. This software, developed in Python, combines state-of-the-art computing technologies such as TensorFlow, with modern data management tools such as HDF5 and SQL for a fast, scalable, and accurate PB data processing engine. This package, built around the principles of FAIR data, provides a memory safe, parallelized, and GPU accelerated environment for the analysis of particle simulations. The software currently offers 17 calculators for the computation of properties including diffusion coefficients, thermal conductivity, viscosity, radial distribution functions, coordination numbers, and more. Further, the object-oriented framework allows for the rapid implementation of new calculators or file-readers for different simulation software. The Python front-end provides a familiar interface for many users in the scientific community and a mild learning curve for the inexperienced. Future developments will include the introduction of more analysis associated with ab-initio methods, colloidal/macroscopic particle methods, and extension to experimental data.

Keywords Molecular dynamics, Computational physics, Material properties, High performance computing, TensorFlow, FAIR data

Introduction

Particle-based (PB) simulations, of which we declare Molecular dynamics (MD) to be a subset, are a cornerstone of modern computational physics, chemistry, biology, and engineering. The benefit of PB simulations comes in their access to system observables and properties that are otherwise unavailable experimentally. Whilst the method itself involves the sampling of configuration space under constraints imposed by defined interactions, it is the analysis of these configurations that leads to insights in medicine [1–5], battery technology [6–15], astrophysics [16], materials engineering [17–21], and much more. When running simulations, one is faced with the choice of either performing On-The-Fly (OTF) analysis, or post-processing their simulation data to extract information. In the case of OTF analysis, it is often difficult to know beforehand parameters such as measurement range, number of samples, or bin resolution, that will be required in the calculations. For this reason, post-processing of

*Correspondence:
Christian Holm
holm@icp.uni-stuttgart.de
[1] Institute for Computational Physics, Universität Stuttgart, Stuttgart, Germany
[2] Aeronautics and Aerospace Department, von Karman Institute for Fluid Dynamics, Rhode-St-Genese, Belgium
[3] Thermo and Fluid Dynamics (FLOW), Vrije Universiteit Brussel, Brussels, Belgium
[4] Laboratory for Chemical Technology (LCT), Ghent University, Ghent, Belgium

simulation results has become a preferred solution, particularly in cases where simulations are computationally expensive. Another important consideration is the number of simulations that will be studied. In the case of temperature or al-chemical comparison, often several large simulations will be run under varying conditions. In these cases, it is convenient to have access to post-processing tools that can track and analyse each simulation as well as compare the results of the analysis. Several post-processing tools have been developed in the past and are used by scientists around the world including MDAnalysis [22], PyLAT [23], pyLOOS, mdtraj [24], pytraj [25], freud [26], and VMD [27] to name some of the most popular. These programs, each having their own strengths, follow a similar workflow:

1 Load a trajectory object
2 Perform analysis
3 Close trajectory object

Whilst this method meets the demands of many users in the PB community, it is positioned more as a tool for the analysis of single simulations and less as an integral part of the scientific process. In the direction of managing many simulations, there also exists several data management tools including datreant [28] and signac [29]. These packages allow scientists to build large data workflows and track their states as they use other libraries to perform analysis. Put together, the post-processing libraries as well as data management packages can assist scientists in handling their simulations so long as interoperability is well handled by the users. A downside of course is that the post processing and the experiment handling is separated which can result in a more complex interface and reduced usability.

In this paper we introduce MDSuite, a post-processing program for PB simulations that allows for the collection and analysis of simulation data from many simulations under a common framework. MDSuite differs from the previously mentioned programs in that it takes raw simulation data and generates a new database more suited for rapid post-processing. This database structure, hereafter referred to as MDS-DB, allows data to be loaded only when it is pertinent to the analysis being performed. Furthermore, the structure is such that properties of the particles (i.e. positions, stress, and forces) are separated by species at the time of database construction, thereby eliminating expensive computational overhead during analysis. Furthermore, MDSuite uses an enveloping object that we have named the `Project` class, in which the results of any number of simulations may be contained, analysed, and compared to one another.

The proceeding work is structured as follows. First we discuss the guiding principles of the MDSuite Python library, introducing the `Project` and `Experiment` class objects as well as how they work together to create an optimised and user-friendly analysis environment. This is followed by a discussion into the MDS-DB structure used by MDSuite to store simulation data. We then introduce the different properties that can currently be computed using the MDSuite code including information regarding the implementation of the analysis. After the introduction of the calculators, a brief performance review is presented in order to outline how the MDSuite code has been written to take full advantage of the latest high performance computing systems. Finally, we introduce the outlook of the MDSuite package including the extension beyond simulations as well as general performance optimization.

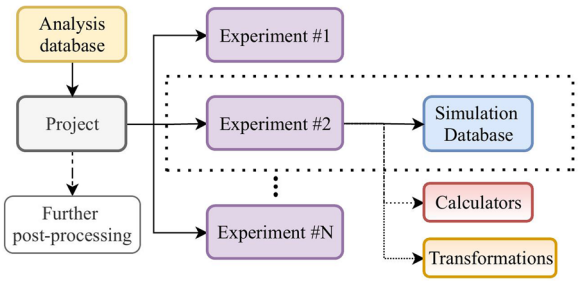

Fig. 1 Structure of a MDSuite project. Each experiment stores data in its own database and has access to the Calculators and Transformations

Implementation

Software architecture

The governing principle behind MDSuite is the simplification of PB investigations. MDSuite approaches this by providing an infrastructure in which simulation data and analysis results from any number of simulations may be stored, (re-)analyzed, and compared.

The two integral components of this infrastructure are the `Project` class and the `Experiment` class (Fig. 1).

The `Experiment` class contains all the information pertaining to a single simulation including particle trajectories, net system properties such as fluxes, and thermodynamic information such as temperature and pressure. This data is stored in a compressed manner within the MDS-DB. Simulation data can be added to the `Experiment` at one time or iteratively over the course of a simulation as might be necessary in the case where several runs are required. Further to the storage of trajectory data, the `Experiment` class connects to the MDSuite calculators ("Calculators" section) which are used to perform analysis. The results of the analyses, including any series data generated in the process, are also stored by the `Experiment` class in an SQL database (SQL-DB). This means that once an analysis has been performed,

the raw data such as an autocorrelation function, or a radial distribution function, as well as the outcome of the analysis such as a diffusion coefficient or coordination number, may be accessed at any time without needing to repeat the calculations. Furthermore, the use of an SQL database structure allows for the storage of the parameters used in the calculation such as data range, correlation time, or bin spacing. This means that analyzed data can be queried from a database and compared along with the parameters of the calculation. One may also use this feature for further processing such the common practice of fitting the autocorrelation function with double exponentials for the determination of the thermal conductivity with all necessary information about the computation stored. An additional important feature of the trajectory data stored by the `Experiment` class is the assigned version. If a computation is performed with a fixed set of parameters before additional trajectory data is added, it is important that when this computation is called again after the addition of data, that it is recomputed and not simply loaded from the database. This is handled by a version system that assigns a new number to each state of the MDS-DB as trajectory data is added. If new data is parsed into the database, the version will be updated and the computations performed on the old set of data will become stale such that computations with identical parameters may be re-performed.

The `Project` class acts as a container for multiple `Experiment` objects and can be used to collect property values for each of the classes and compare them with respect to a desired parameter. Experiments can be added and removed from the `Project` class at any time during an investigation and there is no need to know beforehand how many will be performed. Fundamental simulation properties such as thermostat temperature, time-step, and species information are stored in the SQL-DB for each experiment. Due to the internal use of SI units in MDSuite, simulations performed using different systems of units or simulation engines can still be directly compared with one another. As an example, consider the comparisons of two inter-atomic potentials, each constructed with a specific unit system in mind. Utilizing the MDSuite framework eliminates the necessity of unit conversion whilst also providing a simple means for comparing analysis results.

SQL-DB

As was mentioned earlier, MDSuite utilizes an SQL database for the management of simulation parameter data and results of computations. This database is built using the SQLAlchemy package [30] which allows for multiple back-end SQL engines. Data stored within the database can either be series type data, e.g. an autocorrelation function, or the results of some analysis such as diffusion coefficients along with the uncertainty associated with this measurement. This data will also have associated with it meta-data including computation range, correlation time, and number of configurations. This type of record keeping is essential in the communication of results in case of publication as is in line with the FAIR data principles [31]. FAIR data guidelines are a framework for improving the way scientists approach the Findability, Accessibility, Interoperability, and Reuse of digital assets [31]. In MDSuite, the storage of all computation results along with their parameters provides an environment for such a framework as these results are freely accessible from outside MDSuite, they can be hosted online, they will provide data in standard data types, and provide a complete parameter set describing how the data was produced. Utilization of the SQL-DB not only allows for simple navigation of data through the `Project` class, but also for the use of generic SQL database navigation tools for studying data graphically.

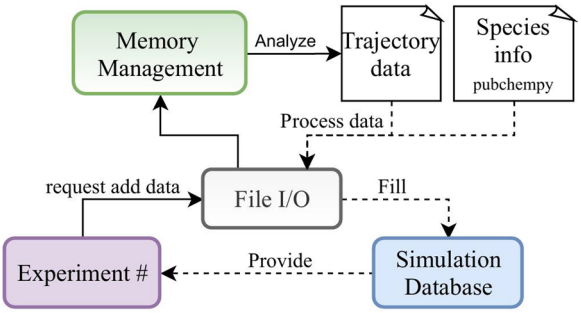

Fig. 2 Database generation logic. Solid lines indicate actions and dashed lines data flow

MDS-DB

The database generation process is one of the unique features of MDSuite. Typically, MD codes dump the results of simulations into data files of varying format which may contain trajectory information of the particles or net system information such as fluxes and temperatures. In order to avoid complications with calculators, MDSuite reads simulation data and constructs a database based upon the HDF5 hierarchical data format [32]. Currently, MDSuite uses a slight variation of the established H5MD database structure [33], however, this will change in the near future and be brought completely inline with this standard. The database construction process is depicted in Fig. 2. When data is added to an `Experiment`, the `File I/O` module will choose the corresponding reader with which to process the data. Simultaneously, the `Memory Management` module will analyze the original trajectory file and decide the appropriate batching size

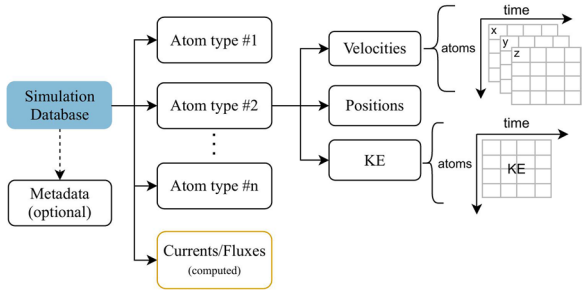

Fig. 3 Typical structure of the simulation database. For each atom type, sub-groups for each variable are created. Data is then stored in tables. Currents or fluxes are also stored as global quantities

to process the data. This allows one to efficiently parse large datasets on a machine with limited memory. As the batches of data are read, they are stored in the simulation database. Additionally, the PubChemPy [34] library is called to expand species information to real element specific data such as mass and charge which is then stored in the SQL-DB.

The MDS-DB, rather than trivially storing configurations of atoms, splits the information into groups and sub-groups by property (Fig. 3). As an example, in the generation of an MDS-DB for a system of NaCl, positions, forces, and velocities for all sodium and chloride atoms in the simulation will be split first into the groups Na and Cl and then into sub-groups of positions, velocities, and forces. This extends naturally to the use of coarse-grained descriptions where molecules are used as the groups and center of mass positions and velocities are stored. In the case of system wide properties such as global flux (eg. thermal flux), data is simply stored in a group with the name of the flux. Currents, fluxes, or other global properties can also be computed inside MDSuite using the transformations and stored in the MDS-DB. With this structure, any simulation data may be stored in a simple, compressed format. The motivation behind such a partitioning scheme is rooted in programming simplicity, memory safety, and improved computational performance. Programming simplicity becomes evident when looking at calculator definitions. In MDSuite, each calculator has a set of properties that it requires to perform the calculation. With a partitioned database scheme this becomes universal across file formats and makes writing new calculators more efficient. Furthermore, with species-wise separation it becomes simple to perform computations on only a selection of species in a simulation. Beyond simplicity in writing calculators, memory safety also becomes easier to handle with data being split by property. This is because when MDSuite is called to compute a diffusion coefficient for a single species, it can load only the specific positions of the single species rather than having to search through all trajectory data and filter it for the desired components. In this way, memory usage calculations can be performed on a well-defined amount of data. Finally, while using a partitioning scheme may increase the initial processing time when creating an experiment, it means that data is read from a file precisely once and during analysis there is no need for searching through trajectory data.

As different MD simulation engines have different output formats, MDSuite implements readers to pre-process simulation data and prepare it for storage. MDSuite utilizes the Chemfiles library [35] as a foundation for file-reading thereby giving access to over 21 different formats including those for large simulations engines such as GROMACS [36], AMBER [37], CHARMM [38] and LAMMPS [39, 40]. Furthermore, the MDSuite file readers are written such that it is simple to add new readers for custom simulation outputs. This has been used, for example, to store the output of an ESPResSo [41] simulation of a bacteria study, initially stored as a pandas [42, 43] `DataFrame`

User interface

The aim of the MDSuite user interface is to allow for easy and understandable communication with the `Calculators` and `Transformations`. To this end, MDSuite is distributed as a Python package with a streamlined API. That is to say, users typically require only a single package import and only access methods from a single class. To aid in usability, many user-focused methods have been added to perform simple tasks such as accessing stored data, changing names of MDS database groups, and updating physical properties of species. These features keep the focus on scientific work rather than computational baggage. This usability is particularly noticeable when using the package through Jupyter notebooks [44, 45], as this allows direct interaction with results as they are calculated.

Code sample 1 demonstrates the MDSuite API for the analysis of two molten salt systems of NaCl and LiCl.

```
import mdsuite as mds

project = mds.Project(name="Molten_Salts")

# Add simulation data from two simulations
NaCl = project.add_experiment(
    name="NaCl",
    units="metal",
    temperature=1400.0,
    simulation_data="NaCl.lammpstraj"
)
LiCl = project.add_experiment(
    name="LiCl",
    units="real",
    temperature=1600.0,
    simulation_data="LiCl.xyz"
)

# Compute diffusion coefficients for both
# diffusion_data = project.run.EinsteinDifusionCoefficients(
    data_range=200, correlation_time = 50
)

# Compute the RDF for NaCl
rdf_data = NaCl.run.RadialDistributionFunction(zeta=6)
cn_data = NaCl.run.CoordinationNumbers(rdf_data)

# Compute ionic conductivity for LiCl
ic_data = LiCl.run.GreenKuboIonicConductivity()
```

Code sample 1 Example of MDSuite API for the analysis of two molten salts. In this case,self-diffusion coefficients are computed for both salts whereas the radial distribution functions and ionic conductivitiy are computed only for the NaCl and LiCl respectively.

Calculators

A driving factor in the design logic of MDSuite was the desire to be able to perform a multitude of analyses on a simulation trajectory under a single framework. The construction of the MDS-DB allows for data to be rapidly loaded based on the particle type and property under consideration. With this approach, a number of methods for analysis become possible in a fast and memory safe manner. This section discusses the different analysis options available to the MDSuite user with some details about their implementation.

All analyses in MDSuite are built from the `Calculator` parent class. Every other calculator inherits from this class and implements the actual computation. The benefit of this is that the `Experiment` class is agnostic towards the inner workings of calculator objects and interfaces with them in the same manner, regardless of the computation being performed. Furthermore, by structuring the calculator implementations as children of a parent class, additional calculators may be easily added by users.

The execution logic of a calculator is depicted in Fig. 4. When a `Project` requests that a particular calculation be performed, the corresponding `Calculator` will

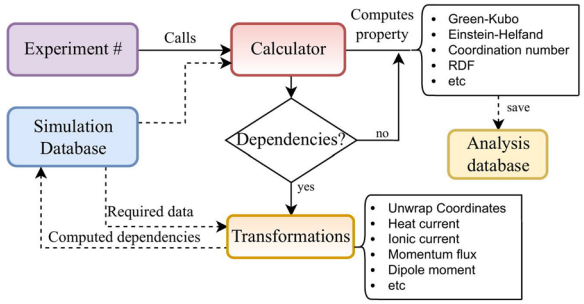

Fig. 4 Execution logic of a `Calculator`. Continuous lines depict actions, dashed lines depict data flow

be called. This `Calculator` will request the required data from the MDS-DB ("MDS-DB" section). If this data is not available, the `Calculator` will make use of the appropriate `Transformation` to compute it. The computed data will be stored in the MDS-DB to avoid re-computation in the event that the analysis is repeated. Once all the dependencies are available, the calculation of the property will proceed. Finally, the results will be stored in the SQL-DB.

All calculators in MDSuite may be called simply from the `Project` class with:

```
project = mdsuite.Project(...)
project.run.ComputationName(...)
```

In order to optimize the performance of the calculators, as well as to take advantage of modern hardware, MDSuite utilizes the TensorFlow Python library [46] to allow not only for optimized tensor calculations, but also for immediate deployment on GPU and parallel processing with a small increase in import time due to device registration. TensorFlow is also used in the construction of optimized data pipelines for the computations where TensorFlow Datasets, used for elements of batching, are employed. Memory safety is handled both with configuration-wise batching and atom-wise mini-batching if necessary (see Additional file 1). The performance improvements of the TensorFlow library are discussed in more detail in "Performance" section where strong-scaling tests have been performed.

The remainder of this section will discuss the available calculators.

Structural properties

The computation of structural properties in MDSuite largely revolves around the radial distribution function. The following section outlines the various structural analyses available to users as well as what each of them describes.

Radial distribution function

Radial distribution functions (RDFs) describe the particle density of species α at a distance r from a particle of species β [47]. This may be calculated directly from the positions of particles within a simulation by:

$$g_{\alpha\beta}(r) = \frac{N_\alpha N_\beta}{V} \sum_{i=1}^{N_\alpha} \sum_{j=1}^{N_\beta} \langle \delta(r - |\mathbf{r}_i^\alpha - \mathbf{r}_j^\beta|) \rangle, \tag{1}$$

where $g_{\alpha\beta}$ is the RDF of species α with respect to species β, N_α is the number of particles of species α, V is the box volume, and $\mathbf{r}_i$ is the position of particle i.

Angular distribution function

Angular distribution functions (ADFs), similar to the RDF, describe the distribution of the angle between three atoms. ADFs can be used to better understand bonding between specific species as well as to identify breakdown in structure under changing environments. An ADF is computed with reference to three atoms in the system and is calculated by:

$$g_{\alpha\beta\gamma}(\theta) = \frac{1}{N_\alpha N_\beta N_\gamma} \sum_{ijk \in \alpha\beta\gamma} \frac{1}{|r_{ij}|^\zeta |r_{ik}|^\zeta} \langle \delta(\theta - \theta_{ijk}) \rangle, \tag{2}$$

where the angle being measured is between the triangle formed with the atom of species α at the centre and ζ is a sensitivity factor used to highlight peaks and relevant distances. This sensitivity index can be useful in filtering out angles occurring at large distances from the central particle. This is because, during the ADF calculation, no information about distance between the particles is used in the calculation of the angles.

Coordination numbers

Coordination numbers describe the number of particles surrounding a reference particle at a chosen distance and are related to the ordering in the system [48]. In the case of a crystalline structure, this reduces simply to the number of particles bonded to the reference particle [49]. In

a liquid system this becomes a less rigorous definition and the coordination numbers are often calculated at different points along the RDF referred to as coordination shells [48]. MDSuite uses the definition of coordination numbers taken from Waseda [48]:

$$n^i_{\alpha\beta} = 4\pi\rho \int_{r_{i-1}}^{r_i} \mathrm{drr}^2 \mathrm{g}_{\alpha\beta}(\mathrm{r}), \tag{3}$$

where $i \geq 1$ is the coordination number to be calculated associated with a chosen coordination shell. The integration range is taken at minimums between successive peaks in the RDF. MDSuite uses the Golden-Section search algorithm [50] along with a Savitzky-Golay [51] filter to identify the minimums in the RDFs and perform the analysis correctly (see Additional file 1). The parameters of the filter are tunable at runtime.

Potential of mean-force

Potential of mean-force (PMF) is a measurement of how the free energy of a system changes as a function of interatomic distance and may be calculated directly from the RDF as [52, 53]:

$$w_{\alpha\beta}(r) = -k_\mathrm{B} T \ln g_{\alpha\beta}(r). \tag{4}$$

The PMF ($w_{\alpha\beta}$), computed between species α and β, can be used to understand effective binding strength between atomic species [14]. MDSuite utilizes the minima finding algorithm discussed for the coordination numbers to identify the minimum value of the PMF.

Kirkwood-Buff integrals

Kirkwood-Buff (KB) integrals are a fundamental component of the Kirkwood-Buff solution theory developed by John G. Kirkwood and Frank P. Buff aimed at relating microscopic properties of a solution to its thermodynamic quantities [54]. The integrals are calculated directly from an RDF by:

$$G_{\alpha\beta} = 4\pi \int_0^\infty \mathrm{drr}^2(\mathrm{g}_{\alpha\beta}(\mathrm{r}) - 1). \tag{5}$$

The KB integral ($G_{\alpha\beta}$), computed between species α and β, can be used to better understand preferential binding between species in a system [55].

Dynamic properties

Dynamic calculations are those that take place with respect to time. As such, there exists correlation between properties of particles and therefore, species attention must be paid when performing computations. In all dynamic calculations in MDSuite, ensembling is performed to ensure correct statistics [56]. This ensembling is represented by the angled brackets in each of the calculations to follow. In each computation, the user may set a desired correlation time to decide over what time range this ensembling occurs and thereby also computing correct errors.

Typically in particle simulations there are two approaches to the computation of dynamic properties, Green-Kubo, and Einstein-Helfand. MDSuite implements both of these calculation approaches and therefore, a brief discussion of each is included.

Green-Kubo calculations

The first method discussed is the Green-Kubo approach [57–59] which uses autocorrelation functions to compute dynamic properties from a flux in the system by:

$$\lambda = \frac{\omega^{\mathrm{GK}}}{d \cdot V} \int_0^\infty dt \langle \boldsymbol{\eta}(t) \cdot \boldsymbol{\eta}(0) \rangle, \tag{6}$$

where ω^{GK} is some calculation specific pre-factor, d is the dimension, and $\boldsymbol{\eta}$ is the flux on which autocorrelation is computed. In reality, the integral cannot be taken to infinity and rather a cutoff value must be chosen at which to integrate the system. As this can take optimization, MDSuite allows the user to set an integration range and then computes the running integral, along with the uncertainty, on the function up to this value. This approach allows users to identify a converged value for the integral. This running integral can be seen in Fig. 5.

Einstein-Helfand computations

In the case of an Einstein-Helfand computation, rather than use an autocorrelation function we look at the gradient of a mean square displacement. The property for which we look at the mean square displacement is often the integral of the flux used in the corresponding Green-Kubo approach. In this case one would like to solve:

$$\lambda = \lim_{t \to \infty} \frac{\omega^{\mathrm{EH}}}{2 \cdot d \cdot t} \langle |\boldsymbol{\zeta}(t) - \boldsymbol{\zeta}(0)|^2 \rangle, \tag{7}$$

where $\boldsymbol{\zeta}$ is the integrated flux, ω^{EH} is a property species pre-factor, and d is the dimension.

In the remainder of this section each dynamic calculator available in MDSuite is discussed.

Viscosity

Viscosity is a fluids resistance to deformation. In MDSuite, the viscosity can be calculated using both the Green-Kubo relation and the Einstein-Helfand relation [60]. Assuming

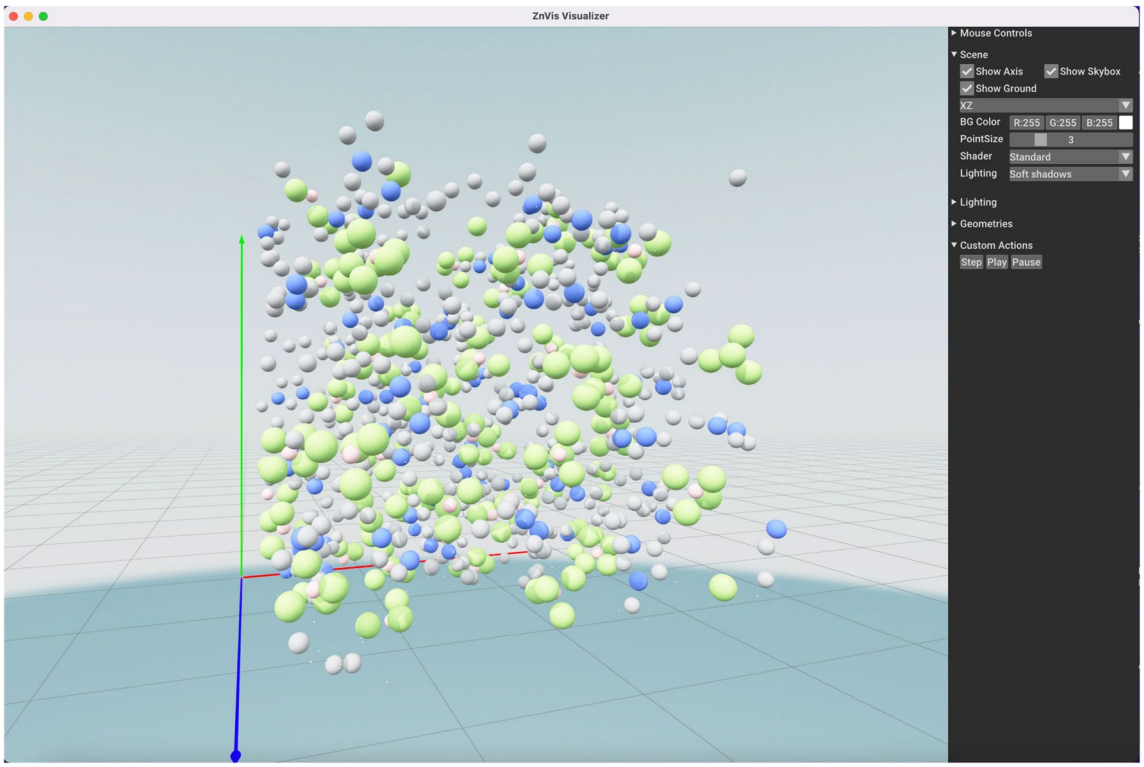

Fig. 5 A snapshot from an MD simulation of the BMIM-BF4 ionic liquid

isotropy, the equation to compute viscosity by means of Green-Kubo reads [60]:

$$\kappa = \frac{V}{3k_{\mathrm{B}}T}\int_0^\infty \mathrm{dt}\langle \mathrm{P_{a,b}}(0)\cdot \mathrm{P_{a,b}}(\mathrm{t})\rangle \qquad (\mathrm{a}\neq \mathrm{b}\in\{\mathrm{x},\mathrm{y},\mathrm{z}\}) \tag{8}$$

where V is the volume of the system, k_{B} is the Boltzmann constant, T is the temperature, and $P_{a,b}(t)$ are the off-diagonal components of the stress tensor at time t. The Einstein-Helfand approach uses a limit observation to determine the viscosity [60] with:

$$\kappa = \frac{1}{Vk_{\mathrm{B}}T}\lim_{t\to\infty}\frac{1}{2t}\langle[L^i_{a,b}(t)-L^i_{a,b}(0)]^2\rangle, \tag{9}$$

where $L_{a,b}(t)$ is defined as:

$$L^i_{a,b}(t) = p^i_a(t)\cdot r^i_b(t), \tag{10}$$

and where p^i_a and $r_b(t)$ are the momentum and position of particle i respectively.

Self-diffusion coefficients

Self-diffusion coefficients describe the average rate at which atoms of species α diffuse through the bulk. They may be calculated from a Green-Kubo relation by:

$$D_\alpha = \frac{1}{3N_\alpha}\int_0^\infty \mathrm{dt}\sum_{\mathrm{i}}^{\mathrm{N}_\alpha}\langle \mathbf{v}^\alpha_{\mathrm{i}}(\mathrm{t})\cdot\mathbf{v}^\alpha_{\mathrm{i}}(0)\rangle, \tag{11}$$

where $\mathbf{v}^\alpha_i$ is the velocity of particle i of species α. MDSuite also provides the corresponding Einstein relation, written [47]:

$$D_\alpha = \frac{1}{2\cdot d\cdot t}\frac{1}{N_\alpha}\lim_{t\to\infty}\sum_i^{N_\alpha}\langle|\mathbf{r}^\alpha_i(t)-\mathbf{r}^\alpha_i(0)|^2\rangle, \tag{12}$$

where d is the dimension of the system.

Distinct-diffusion coefficients

A related property to the self-diffusion coefficients are the so-called distinct diffusion coefficients which provide information about atom correlation in a system [61]. The Green-Kubo relation implemented in MDSuite is written:

$$D_{\alpha\beta} = \frac{1}{3}\frac{1}{N_\alpha\cdot N_\beta}\int_0^\infty \mathrm{dt}\sum_{\mathrm{i}}^{\mathrm{N}_\alpha}\sum_{\mathrm{j}}^{\mathrm{N}_\beta}\langle \mathbf{v}^\alpha_{\mathrm{i}}(\mathrm{t})\cdot\mathbf{v}^\beta_{\mathrm{j}}(0)\rangle, \tag{13}$$

where summation is performed over distinct atoms i and j. In the case of $\alpha=\beta$, the condition $i\neq j$ is enforced.

The corresponding Einstein formulation is also implemented as:

$$D_{\alpha\beta} = \frac{1}{N_\alpha \cdot N_\beta} \frac{1}{2 \cdot d \cdot t} \lim_{t\to\infty} \sum_{i}^{N_\alpha} \sum_{j}^{N_\beta} \langle (\mathbf{r}_i^\alpha(t) - \mathbf{r}_i^\alpha(0)) \cdot (\mathbf{r}_j^\beta(t) - \mathbf{r}_j^\beta(0)) \rangle. \tag{14}$$

Ionic conductivity

The ionic conductivity of a system describes the ability of a material to dissipate electrical energy based on the mobility and charge of the constituent ions. Ionic conductivity may be calculated by the Green-Kubo relation as [62]:

$$\sigma = \frac{\beta}{3V} \int_0^\infty \mathrm{dt} \langle \mathbf{J}^\sigma(\mathrm{t}) \cdot \mathbf{J}^\sigma(0) \rangle, \tag{15}$$

where $\beta = \frac{1}{k_\mathrm{B} T}$, V is the volume of the system, and $\mathbf{J}^\sigma = q \sum_i^N z_i \mathbf{v}_i$ is the ionic current. An alternative approach, denoted the Einstein-Helfand method, involves studying the mean square displacement of the translational dipole moment as:

$$\sigma = \lim_{t\to\infty} \frac{\beta}{2 \cdot d \cdot t} \langle |\mathbf{M}(t) - \mathbf{M}(0)|^2 \rangle, \tag{16}$$

where $\mathbf{M} = q \sum_i^N z_i \mathbf{r}_i$ is the translational dipole moment of the system. To further probe the effects of ion correlation in a system, MDSuite also calculates the Nernst-Einstein ionic conductivity which uses the self-diffusion coefficients to approximate the property as

$$\sigma^{NE} = \frac{q^2 \beta}{3V} \left(\sum_\alpha x_\alpha z_\alpha^2 D_\alpha \right), \tag{17}$$

where x_α is the mass fraction of species α, z_α is the ion charge, and D_α is the self-diffusion coefficient. The Nernst-Einstein approach suffers from the absence of correlation effects, often resulting in an over-estimate for the ionic conductivity [61]. Whilst the Green-Kubo and Einstein-Helfand approaches can be used, it is also possible to correct the Nernst-Einstein equation using the distinct diffusion coefficients by [61]

$$\sigma^{CNE} = \frac{q^2 \beta}{3V} \left(\sum_\alpha x_\alpha z_\alpha^2 D_\alpha + \sum_{\beta\gamma} x_\beta x_\gamma z_\beta z_\gamma D_{\beta\gamma} \right), \tag{18}$$

where the symbols are as above and the $D_{\beta\gamma}$ is the distinct diffusion coefficient for the β, γ pair. MDSuite implements both the Einstein-Helfand and Green-Kubo approaches for the full ionic conductivity as well as the self and distinct diffusion coefficients thereby enabling the calculation of the Nernst-Einstein and corrected Nernst-Einstein conductivities.

Thermal conductivity

The thermal conductivity of a system describes the ability of a material to transport heat. At a macroscopic scale, thermal transport by means of conduction is described by Fourier's law:

$$\mathbf{q} = -\lambda \nabla T. \tag{19}$$

In Eq. 19, λ is defined as the thermal conductivity of the material and can be expressed using a Green-Kubo relation as:

$$\lambda = \frac{V}{3 k_\mathrm{B} T^2} \int_0^\infty \mathrm{dt} \langle \mathbf{J}^\kappa(0) \cdot \mathbf{J}^\kappa(\mathrm{t}) \rangle, \tag{20}$$

where $\mathbf{J}^\kappa$ is the heat-flux defined as:

$$\mathbf{J}^\kappa(t) = \frac{1}{V} \left[\sum_i e_i \mathbf{v}_i(t) - \frac{1}{2} \sum_{i<j} \mathbf{F}_{ij}(t) (\mathbf{v}_i(t) + \mathbf{v}_j(t)) \mathbf{r}_{ij}(t) \right]. \tag{21}$$

The equivalent Einstein-Helfand relation is also implemented for this quantity as

$$\kappa = \frac{1}{V k_\mathrm{B} T^2} \lim_{t\to\infty} \frac{1}{2t} \langle |\mathcal{J}(t) - \mathcal{J}(0)|^2 \rangle \tag{22}$$

where $\mathcal{J}(t)$ is defined as the integrated heat current. This current can be stored during a simulation or computed in MDSuite as:

$$\mathcal{J}(t) = \sum_{i=1}^{N} e_i(t) \cdot \mathbf{r}_i(t), \tag{23}$$

where $e_i(t)$ and $\mathbf{r}_i(t)$ are the energy and the position vector of the i-th particle in the simulation respectively. In all cases, the per-particle energy must be printed during the simulation run in order for MDSuite to perform the computation. This can be done for example, using the LAMMPS [39, 40] `compute pe/atom` command which will dump the energy of an atom up to the specified cutoff including non-local terms such as electrostatics. In its current state, MDSuite makes no assumptions about inter-particle interactions.

Transformations

In MDSuite, transformations are defined as operations on simulation data that yield time-dependent results. Because these properties exist at each time step, they are also stored under new groups in the MDS-DB.

Transformation may be called from a `Project` by:

```
project = mdsuite.Project(...)
project.run.TransformationName(...)
```

In the case where a transformation is required for a calculator to run, it will be called automatically by the MDSuite dependency handler. Whilst many of the transformations have already been discussed in the calculator section including ionic current, translational dipole moment, thermal flux, and the integrated heat current, MDSuite also offers some unique transformations which warrant additional discussion.

Coordinate (Un)wrapping

A core component of particle-based simulations is the use of periodic boundary conditions (PBC) to mimic a bulk system of infinite particles [47]. When post-processing is then performed on these systems, the application of PBC sometimes must be reversed in order to retrieve for example, correct dynamics, or to complete a molecular structure. In MDSuite, two approaches may be taken to unwrapping coordinates, box-hopping detection and scaling by image numbers stored during a simulation.

Molecule mapping

A notable transformation in MDSuite is the molecular mapping module. In this module, a distance search is used to perform graph decomposition on a configuration in order to map free particles into molecule groups. These groups can then be used with any of the aforementioned calculators thereby allowing for the construction and analysis of coarse-grained representations. To further improve the accuracy of this method, approximate graph isomorphism checks may be applied to ensure that the molecular graph built is approximately isomorphic with a reference graph constructed from the SMILES string (see Additional file 1). While molecule mapping module is very flexible and can be used to construct any number of molecule groups, it is currently hampered by computational limitations. In its current state, the initial construction of groups, usually performed on one or two configurations, is an $\mathcal{O}(N^2)$ operation whereas the mapping of these groups over the full trajectory is $\mathcal{O}(N)$. Current work is underway in constructing more accurate and faster approaches to this molecular mapping.

Customization

An important aspect of any post-processing tool is an ability to adapt its features for custom analysis. Through the use of object oriented programming, MDSuite provides users with the ability to subclass the parent calculator classes and in doing so, take full advantage of the database interface, memory safety, and performance. Due to the Python interpreter, these additions can be made without any reinstallation of the MDSuite software. Detailed information about this process is provided on the MDSuite developer documentation page at https://mdsuite.readthedocs.io/en/main/_developer_docs/implementing_calculators.html.

Software and development

Aside from the features and performance of MDSuite, the development process is of utmost importance for stable and usable software. MDSuite utilizes a test-driven development approach where both unit tests and integration tests are used to cover the code base. Furthermore, continuous integration is used to ensure that before any code is added to the main branch the package installs for all supported Python versions, the documentation builds, and all of the tests and example scripts pass. MDSuite adopts the semantic versioning approach [63] and aims for a small number of major changes. MDSuite is released under the OSI approved Eclipse Public License 2.0 (EPLv2) and can be installed on Linux, Windows, and MacOS operating systems via pip

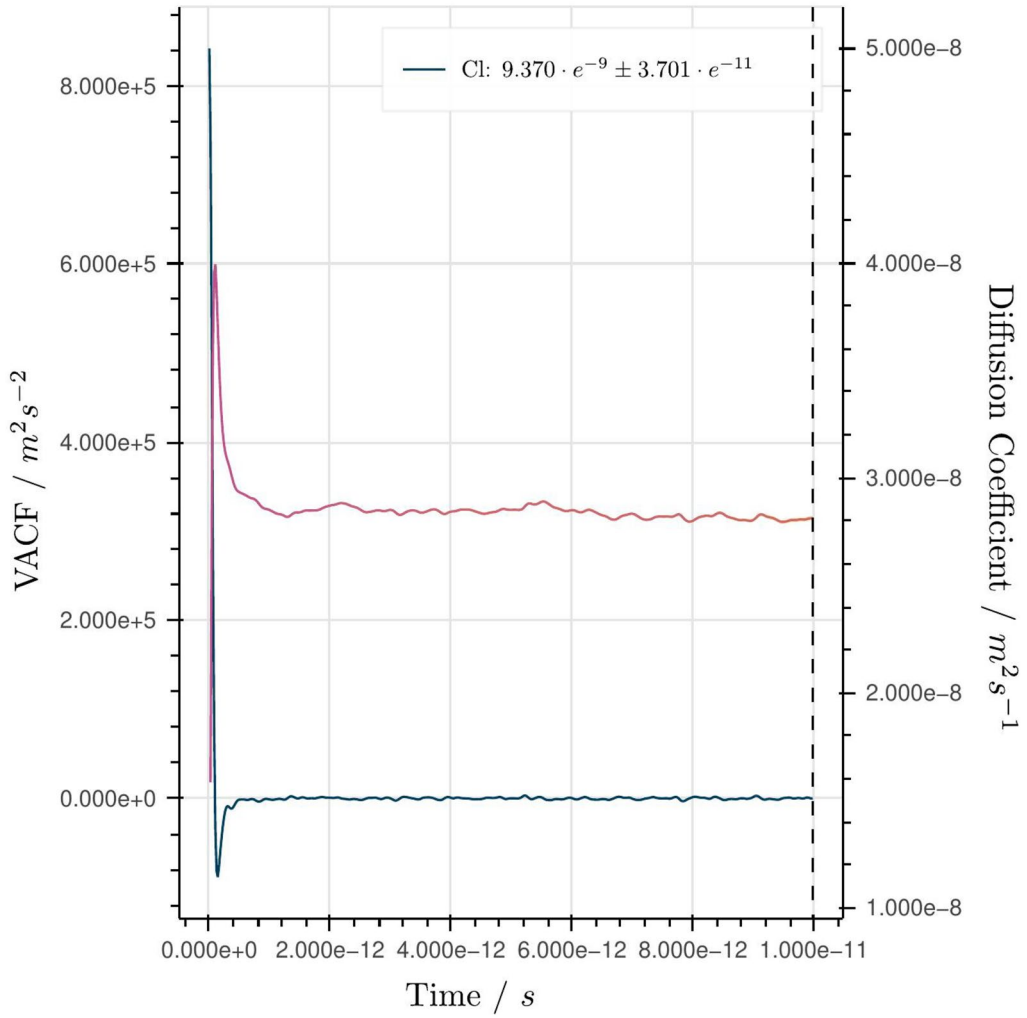

Fig. 6 Interactive plot environment offered by Bokeh [65]. In this case, we see a velocity autocorrelation function along with the cumulative integral to identify a point of convergence

```
pip install mdsuite
```

or directly from the github repository at https://github.com/zincware/MDSuite.

Data visualization

Data visualization and exploration is of fundamental importance particularly in the age of big-data. In MDSuite, focus is placed on exploratory data analysis, that is, plots and visualizations that allow users to move through the data and identify or operate on regions of interest.

Three-dimensional visualization

The ability to visualize a simulation or any particle trajectory can lead to better insights and intuition during analysis. MDSuite offers a rudimentary visualization module based on the ZnVis particle simulation visualizer []. ZnVis is built on top of the Open3D data processing engine [64] which utilizes a C++ backend for the visualization of point-cloud and mesh data. In MDSuite, ZnVis is used to display particles in an interactive window (Figs. 5 and 6), visualize the trajectory of small simulations, and capture

snapshots of a simulation at a specific time step as a png file.

Two-dimensional visualization

In cases where a full three-dimensional visualization is not required, MDSuite utilizes the Bokeh [65] library to construct interactive plots for data analysis and exploration. These plots, displayed within a web-browser at the end of a calculation, are fully interactive in that they enable exploration of plotted data through zooming, sliding, and hovering over points in the plot. In addition to the automatic plotting features, user have complete access to the raw calculation data for custom plotting.

Results and discussion

Performance

Beyond providing functionality, MDSuite has been developed to maximise the performance of computations. Much of the performance of the MDSuite calculators and transformations arises from the heavy use of TensorFlow [46] and the formulation of computations as tensor operations. By utilizing TensorFlow for these operations the computations can immediately be performed in parallel and on GPU with little to no additional software. This, coupled with the memory manager, results in memory safe and computationally efficient calculations. To assess the performance of MDSuite, a strong-scaling test has been performed using the RDF calculator as it presents both a memory and time intensive operation that is frequently used in computational studies. Strong-scaling measures the improvement in the time of a computation with respect to additional computational resources. In this study, the CPU access of TensorFlow was limited from 1 to 96 cores and the RDF computation performed 10 times to ensure correct statistics. In addition to the strong-scaling test, the calculations were also performed on several devices to assess performance improvements with respect to accelerators including a GTX 1070 and RTX 2080 GPU. In the strong-scaling test, 3 configurations of 33'000 oxygen atoms were used in the RDF computation. To additionally test the memory management capabilities of MDSuite, a 108'000 atom simulation of liquid argon was performed using the LAMMPS [39] simulation engine generating 5000 time-steps in the process , the trajectory of which was used in the device scaling tests.. In the device test, an RDF calculation has been performed on an increasing number of configurations, or frames, of this simulation. In the largest case (80 configurations), $80 \cdot 108000 = 8640000$ atoms are used in the RDF computation on devices with as little as 8 GB of memory. Fig. 7 outlines the results of these experiments. It is clear when studying the figure that the parallelization of the computation results in improved speeds.

In addition to plotting simply the computation time, the speedup factor is also included on the second y-axis of Fig. 7. This plot shows the factor speedup upon the introduction of additional resources and demonstrates how MDSuite natively scales to more capable computational devices. Turning attention now to the device scaling test, it can be seen that the deployment of the calculation onto a GPU results in a substantial improvement in computation speed over even the 96 core CPU computation. Furthermore, due to the use of the TensorFlow library [46] no specific GPU code was required for the acceleration. While the scaling of the MDSuite library is effective at improving computation times, it is not perfect scaling, i.e, with increasing resources vs computations

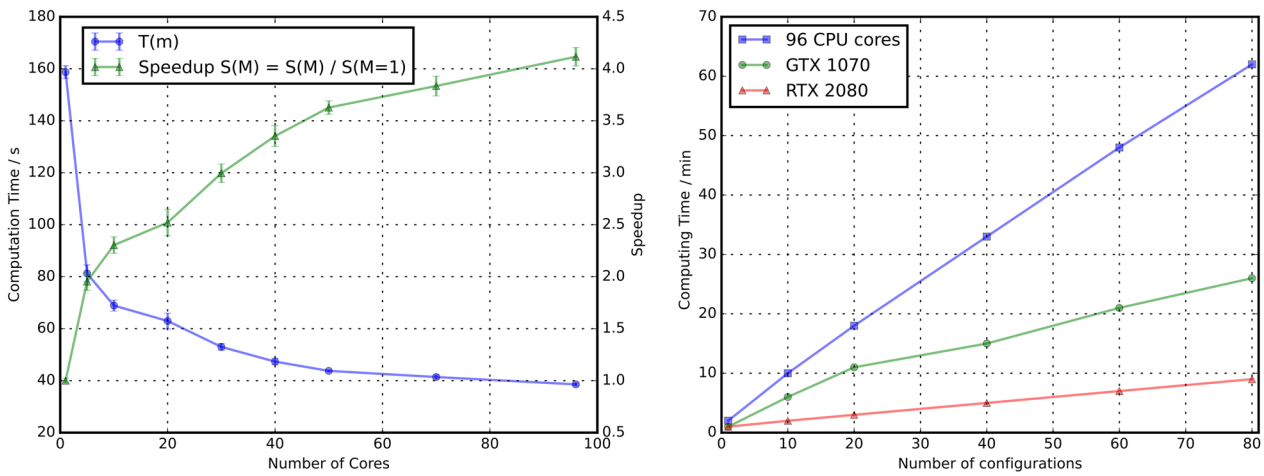

Fig. 7 (left) Strong-scaling of the RDF calculation on 99000 Oxygen atoms. (right) GPU time scaling with respect to number of 108'000 argon atom configurations in an RDF computation

the time plot will have a non-zero gradient. This may be partially addressed with continued optimization of the calculators, however, on some fundamental level this will always be limited by the performance of the TensorFlow library.

Conclusion

We have introduced a new post-processing suite for particle-based simulations capable of combining the simulation data from several investigations, performing analysis on each, and comparing the outcomes of this analysis under a single framework. The Python library consists of an easy-to-use API thereby promoting accessibility to those in the community unfamiliar with programming.

Implemented methods of analysis combine libraries for tensor operations such as TensorFlow in order to optimize performance and provide better support for GPU and cluster deployment. Beyond a direct focus on performance, MDSuite also provides a memory-safe framework and in doing so, allows for the analysis of million atoms systems on desktop machines.

Whilst the current state of MDSuite is capable of a wide range of analysis, there is always more to be done. In future releases, development will focus on the extension of MDSuite calculators to other fields including colloidal studies, biological systems, and ideally, areas in high-energy physics.

The outlook of computational methods is promising. With the ever-increasing capabilities of interaction models and simulation engines, the possibility for new discoveries continues to grow. MDSuite offers a new, innovate means to combine scientific research with computational methods under a common framework.

Abbreviations

PB	Particle based
MD	Molecular dynamics
MDS-DB	MDSuite database used for the storage of simulation data
SQL-DB	SQL database used by MDSuite to store the analysis of computations
RDF	Radial distribution function
ADF	Angular distribution function
KB	Kirkwood-Buff
PMF	Potential of mean-force
GPU	Graphics processing unit
CPU	Central processing unit
API	Application programming interface
PBC	Periodic boundary conditions
GK	Green-Kubo
EH	Einstein-Helfand

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s13321-023-00687-y.

Additional file 1. Additional information regarding molecule mapping, data processing, and memory management algorithms.

Acknowledgements

S.T would like to acknowledge the invaluable technical discussion with Johannes Zeman, Kai Szuttor, and Rudolf Weeber at the Institue for Computational Physics. S.T would also like to thank Ganesh Sivaraman and Anand Narayanan Krishnamoorthy for early testing of the software.

Author contributions

ST developed the initial MDSuite code base, worked on the software, and wrote the paper. FTH contributed to the software and wrote the paper. CL contributed to the development of the software. FZ and MB both contributed to the software and to the paper. CH supervised the project and edited the paper. All authors read the paper, contributed edits and approved of its final form. All authors read and approved the final manuscript.

Funding

Open Access funding enabled and organized by Projekt DEAL. The research of F. T-H is supported by SB PhD fellowship 1S58718N of the Research Foundation Flanders (FWO). C.H and S.T acknowledge financial support from the German Funding Agency (Deutsche Forschungsgemeinschaft DFG) under Germany's Excellence Strategy EXC 2075-390740016, and S. T was supported by a LGF stipend of the state of Baden-Württemberg. C.H, F.Z, and S.T acknowledge support from SPP 2363-"Utilization and Development of Machine Learning for Molecular Applications—Molecular Machine Learning" Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), Project-No 497249646. C.L and C.H acknowledge support from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), project number 327154368, SFB 1313.

Availability of data and materials

Project name: MDSuite Project home page: https://github.com/zincware/MDSuite Operating system(s): Platform independent Programming language: Python3 License: EPL v2.0

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Received: 10 March 2022 Accepted: 22 January 2023
Published online: 11 February 2023

References

1. Kreissl P, Holm C, Weeber R (2021) Frequency-dependent magnetic susceptibility of magnetic nanoparticles in a polymer solution: a simulation study. Soft Matter 17:174–183. https://doi.org/10.1039/D0SM01554G

2. Salo-Ahen OMH, Alanko I, Bhadane R, Bonvin AMJJ, Honorato RV, Hossain S, Juffer AH, Kabedev A, Lahtela-Kakkonen M, Larsen AS, Lescrinier E, Marimuthu P, Mirza MU, Mustafa G, Nunes-Alves A, Pantsar T, Saadabadi A, Singaravelu K, Vanmeert M (2021) Molecular dynamics simulations in drug discovery and pharmaceutical development. Processes. https://doi.org/10.3390/pr9010071
3. Durrant JD, McCammon JA (2011) Molecular dynamics simulations and drug discovery. BMC Biol 9(1):71. https://doi.org/10.1186/1741-7007-9-71
4. De Vivo M, Masetti M, Bottegoni G, Cavalli A (2016) Role of molecular dynamics and related methods in drug discovery. J Med Chem 59(9):4035–4061. https://doi.org/10.1021/acs.jmedchem.5b01684
5. Zhao H, Caflisch A (2015) Molecular dynamics in drug design. Eur J Med Chem 91:4–14. https://doi.org/10.1016/j.ejmech.2014.08.004
6. Zeman J, Kondrat S, Holm C (2021) Ionic screening in bulk and under confinement. J Chem Phys 155(20):204501. https://doi.org/10.1063/5.0069340
7. Sivaraman G, Guo J, Ward L, Hoyt N, Williamson M, Foster I, Benmore C, Jackson N (2021) Automated development of molten salt machine learning potentials: application to LiCL. J Phys Chem Lett 12(17):4278–4285. https://doi.org/10.1021/acs.jpclett.1c00901. (**PMID: 33908789**)
8. Uhlig F, Zeman J, Smiatek J, Holm C (2018) First-principles parametrization of polarizable coarse-grained force fields for ionic liquids. J Chem Theory Comput 14(3):1471–1486. https://doi.org/10.1021/acs.jctc.7b00903. (**PMID: 29357238**)
9. Deringer VL (2020) Modelling and understanding battery materials with machine-learning-driven atomistic simulations. J Phys Energy 2(4):041003. https://doi.org/10.1088/2515-7655/abb011. (**Publisher: IOP Publishing**)
10. Franco AA, Rucci A, Brandell D, Frayret C, Gaberscek M, Jankowski P, Johansson P (2019) Boosting rechargeable batteries R &D by multiscale modeling: myth or reality? Chem Rev 119(7):4569–4627. https://doi.org/10.1021/acs.chemrev.8b00239
11. Sun Y, Yang T, Ji H, Zhou J, Wang Z, Qian T, Yan C (2020) Boosting the optimization of lithium metal batteries by molecular dynamics simulations: a perspective. Adv Energy Mater 10(41):2002373. https://doi.org/10.1002/aenm.202002373
12. Muralidharan A, Chaudhari MI, Pratt LR, Rempe SB (2018) Molecular dynamics of lithium ion transport in a model solid electrolyte interphase. Sci Rep 8(1):10736
13. Breitsprecher K, Holm C, Kondrat S (2018) Charge me slowly, I am in a hurry: optimizing charge-discharge cycles in nanoporous supercapacitors. ACS Nano 12(10):9733–9741. https://doi.org/10.1021/acsnano.8b04785
14. Tovey S, Narayanan Krishnamoorthy A, Sivaraman G, Guo J, Benmore C, Heuer A, Holm C (2020) DFT accurate interatomic potential for molten NaCl from machine learning. J Phys Chem C 124(47):25760–25768. https://doi.org/10.1021/acs.jpcc.0c08870
15. Breitsprecher K, Janssen M, Srimuk P, Mehdi BL, Presser V, Holm C, Kondrat S (2020) How to speed up ion transport in nanopores. Nat Commun 11(1):6085. https://doi.org/10.1038/s41467-020-19903-6
16. Zaverkin V, Molpeceres G, Kästner J (2021) Neural-network assisted study of nitrogen atom dynamics on amorphous solid water—II. Diffusion. Mon Notices Royal Astron Soc 510(2):3063–3070. https://doi.org/10.1093/mnras/stab3631
17. Sivaraman G, Krishnamoorthy AN, Baur M, Holm C, Stan M, Csányi G, Benmore C, Vázquez-Mayagoitia Á (2020) Machine-learned interatomic potentials by active learning: amorphous and liquid hafnium dioxide. NPJ Comput Mater 6(1):104. https://doi.org/10.1038/s41524-020-00367-7
18. Zaverkin V, Netz J, Zills F, Köhn A, Kästner J (2022) Thermally averaged magnetic anisotropy tensors via machine learning based on Gaussian moments. J Chem Theory Comput. https://doi.org/10.1021/acs.jctc.1c00853. (**(PMID: 34882425)**)
19. de Tomas C, Suarez-Martinez I, Marks NA (2016) Graphitization of amorphous carbons: a comparative study of interatomic potentials. Carbon 109:681–693
20. Desai S, Li C, Shen T, Strachan A (2017) Molecular modeling of the microstructure evolution during carbon fiber processing. J Chem Phys 147(22):224705. https://doi.org/10.1063/1.5000911. (**Publisher: American Institute of Physics**)
21. Salaway RN, Zhigilei LV (2014) Molecular dynamics simulations of thermal conductivity of carbon nanotubes: Resolving the effects of computational parameters. Int J Heat Mass Transfer 70:954–964
22. Michaud-Agrawal N, Denning EJ, Woolf TB, Beckstein O (2011) MDAnalysis: a toolkit for the analysis of molecular dynamics simulations. J Comput Chem 32(10):2319–2327. https://doi.org/10.1002/jcc.21787
23. Humbert MT, Zhang Y, Maginn EJ (2019) PyLAT: python LAMMPS analysis tools. J Chem Inf Model 59(4):1301–1305. https://doi.org/10.1021/acs.jcim.9b00066
24. McGibbon RT, Beauchamp K, Harrigan M, Klein C, Swails J, Hernández C, Schwantes C, Wang L-P, Lane T, Pande V (2015) Mdtraj: a modern open library for the analysis of molecular dynamics trajectories. Biophys J 109(8):1528–1532. https://doi.org/10.1016/j.bpj.2015.08.015
25. Roe DR, Cheatham TE (2013) PTRAJ and CPPTRAJ: software for processing and analysis of molecular dynamics trajectory data. J Chem Theory Comput 9(7):3084–3095. https://doi.org/10.1021/ct400341p. (**PMID: 26583988**)
26. Ramasubramani V, Dice BD, Harper ES, Spellings MP, Anderson JA, Glotzer SC (2020) freud: A software suite for high throughput analysis of particle simulation data. Comput Phys Commun 254:107275. https://doi.org/10.1016/j.cpc.2020.107275
27. Humphrey W, Dalke A, Schulten K (1996) VMD—visual molecular dynamics. J Mol Graph 14:33–38
28. David L Dotson, Sean L Seyler, Max Linke, Richard J Gowers (2016) Oliver Beckstein: datreant: persistent, Pythonic trees for heterogeneous data. In: Sebastian Benthall, Scott Rostrup (eds.) Proceedings of the 15th Python in Science Conference, pp. 51–56. https://doi.org/10.25080/Majora-629e541a-007
29. Adorf CS, Dodd PM, Ramasubramani V, Glotzer SC (2018) Simple data and workflow management with the signac framework. Comput Mater Sci 146:220–229. https://doi.org/10.1016/j.commatsci.2018.01.035
30. Bayer M (2012) Sqlalchemy. In: Brown A, Wilson G. (eds.) Thearchitecture of open source applications volume II: structure, scale, and a few more fearless hacks. aosabook.org. http://aosabook.org/en/sqlalchemy.html. Accessed 03 Feb 2022.
31. ...Wilkinson MD, Dumontier M, Aalbersberg IJ, Appleton G, Axton M, Baak A, Blomberg N, Boiten J-W, da Silva Santos LB, Bourne PE, Bouwman J, Brookes AJ, Clark T, Crosas M, Dillo I, Dumon O, Edmunds S, Evelo CT, Finkers R, Gonzalez-Beltran A, Gray AJG, Groth P, Goble C, Grethe JS, Heringa J, 't Hoen PAC, Hooft R, Kuhn T, Kok R, Kok J, Lusher SJ, Martone ME, Mons A, Packer AL, Persson B, Rocca-Serra P, Roos M, van Schaik R, Sansone S-A, Schultes E, Sengstag T, Slater T, Strawn G, Swertz MA, Thompson M, van der Lei J, van Mulligen E, Velterop J, Waagmeester A, Wittenburg P, Wolstencroft K, Zhao J, Mons B (2016) The fair guiding principles for scientific data management and stewardship. Sci Data 3(1):160018. https://doi.org/10.1038/sdata.2016.18
32. Collette A (2013) Python and HDF5. O'Reilly Media, Sebastopol.
33. de Buyl P, Colberg PH, Höfling F (2014) H5md: a structured, efficient, and portable file format for molecular data. Comp Phys Commun 185(6):1546–1553. https://doi.org/10.1016/j.cpc.2014.01.018
34. Kim S, Chen J, Cheng T, Gindulyte A, He J, He S, Li Q, Shoemaker BA, Thiessen PA, Yu B, Zaslavsky L, Zhang J, Bolton EE (2018) PubChem 2019 update: improved access to chemical data. Nucleic Acids Res 47(D1):1102–1109. https://doi.org/10.1093/nar/gky1033
35. Fraux G, Fine J, Ezavod, Barletta GP, Scalfi L, Dimura M: Chemfiles/chemfiles: Version 0.9.3. https://doi.org/10.5281/zenodo.3653157.
36. Lindahl, Abraham, Hess, van der Spoel (2021) ROMACS 2021.4 Manual. Zenodo https://doi.org/10.5281/zenodo.5636522
37. Case DA, Cheatham TE III, Darden T, Gohlke H, Luo R, Merz KM Jr, Onufriev A, Simmerling C, Wang B, Woods RJ (2005) The amber biomolecular simulation programs. J Comput Chem 26(16):1668–1688. https://doi.org/10.1002/jcc.20290
38. Brooks BR, Bruccoleri RE, Olafson BD, States DJ, Swaminathan S, Karplus M (1983) Charmm: a program for macromolecular energy, minimization, and dynamics calculations. J Comput Chem 4(2):187–217. https://doi.org/10.1002/jcc.540040211
39. Plimpton S (1995) Fast parallel algorithms for short-range molecular dynamics. J Comput Phys 117(1):1–19. https://doi.org/10.1006/jcph.1995.1039

40. Thompson AP, Aktulga HM, Berger R, Bolintineanu DS, Brown WM, Crozier PS, in 't Veld, P.J., Kohlmeyer, A., Moore, S.G., Nguyen, T.D., Shan, R., Stevens, M.J., Tranchida, J., Trott, C., Plimpton SJ, (2022) Lammps—a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. Comp Phys Commun 271:108171. https://doi.org/10.1016/j.cpc.2021.108171
41. Weik F, Weeber R, Szuttor K, Breitsprecher K, de Graaf J, Kuron M, Landsgesell J, Menke H, Sean D, Holm C (2019) Espresso 4.0—an extensible software package for simulating soft matter systems. Eur Phys J Spec Top 227(14):1789–1816. https://doi.org/10.1140/epjst/e2019-800186-9
42. pandas development team, T.: Pandas-dev/pandas: Pandas. https://doi.org/10.5281/zenodo.3509134
43. Wes McKinney: Data Structures for Statistical Computing in Python. In: Stéfan van der Walt, Jarrod Millman (eds.) Proceedings of the 9th Python in Science Conference, pp. 56–61 (2010). https://doi.org/10.25080/Majora-92bf1922-00a
44. Pérez F, Granger BE (2007) IPython: a system for interactive scientific computing. Comput Sci Eng 9(3):21–29. https://doi.org/10.1109/MCSE.2007.53. (**Publisher: IEEE Computer Society**)
45. Kluyver T, Ragan-Kelley B, Pérez F, Granger B, Bussonnier M, Frederic J, Kelley K, Hamrick J, Grout J, Corlay S, Ivanov P, Avila D, Abdalla S, Willing C, Team JD (2016) Jupyter Notebooks—a publishing format for reproducible computational workflows. In: Loizides F, Scmidt B. (eds.) Positioning and power in academic publishing: players, agents and agendas, IOS Press, pp 87–90.
46. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S et al (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. Available via Tensorflow. https://www.tensorflow.org/about/bib. Accessed 04 Feb 2022.
47. Frenkel D, Smit B (2002) Understanding molecular simulation, 2nd edn. Academic Press, San Diego. https://doi.org/10.1016/B978-012267351-1/50006-7. Publication Title: Understanding Molecular Simulation (Second Edition)
48. Waseda Y (1980) The Structure of non-crystalline materials: liquids and amorphous solids. Advanced Book Program. McGraw-Hill International Book Company, New York.
49. Muller P (1994) Glossary of terms used in physical organic chemistry (IUPAC Recommendations 1994). Pure Appl Chem 66(5):1077–1184. https://doi.org/10.1351/pac199466051077. (**Place: Berlin, Boston Publisher: De Gruyter**)
50. Kiefer J, Wolfowitz J (1952) Stochastic estimation of the maximum of a regression function. Ann Math Stat 23(3):462–466. https://doi.org/10.1214/aoms/1177729392
51. Savitzky A, Golay MJE (1964) Smoothing and differentiation of data by simplified least squares procedures. Anal Chem 36(8):1627–1639. https://doi.org/10.1021/ac60214a047
52. Smiatek J, Heuer A, Wagner H, Studer A, Hentschel C, Chi L (2013) Coat thickness dependent adsorption of hydrophobic molecules at polymer brushes. J Chem Phys 138(4):044904. https://doi.org/10.1063/1.4789305
53. Smiatek J, Wohlfarth A, Holm C (2014) The solvation and ion condensation properties for sulfonated polyelectrolytes in different solvents-a computational study. New J Phys 16(2):025001. https://doi.org/10.1088/1367-2630/16/2/025001. (**Publisher: IOP Publishing**)
54. Kirkwood JG, Buff FP (1951) The statistical mechanical theory of solutions I. J Chem Phys 19(6):774–777. https://doi.org/10.1063/1.1748352
55. Kobayashi T, Reid JESJ, Shimizu S, Fyta M, Smiatek J (2017) The properties of residual water molecules in ionic liquids: a comparison between direct and inverse kirkwood-buff approaches. Phys Chem Chem Phys 19:18924–18937. https://doi.org/10.1039/C7CP03717A
56. Janke W (2002) Statistical analysis of simulations: data correlations and error estimation. In: Grotendorst J, Marx D, Muramatsu A (eds) Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms. NIC Series, vol 10. John von Neumann Institute for Computing, Jülich, pp 423-445.
57. Green MS (1952) Markoff random processes and the statistical mechanics of time-dependent phenomena. J Chem Phys 20(8):1281–1295. https://doi.org/10.1063/1.1700722
58. Kubo R (1957) Statistical-mechanical theory of irreversible processes. I. General theory and simple applications to magnetic and conduction problems. J Phys Soc Japan 12(6):570–586. https://doi.org/10.1143/JPSJ.12.570
59. Kubo R, Toda M, Hashitsume N (1991) Statistical physics II: nonequilibrium statistical mechanics, 2nd edn. Springer Series in Solid-State Sciences, Springer Ser. Solid-State Statistical Physics. Springer, Berlin Heidelberg. https://doi.org/10.1007/978-3-642-58244-8
60. Kinaci A, Haskins JB, Çağın T (2012) On calculation of thermal conductivity from Einstein relation in equilibrium molecular dynamics. J Chem Phy 137(1):014106. https://doi.org/10.1063/1.4731450. (**Publisher: American Institute of Physics**)
61. Kashyap HK, Annapureddy HVR, Raineri FO, Margulis CJ (2011) How is charge transport different in ionic liquids and electrolyte solutions? J Phys Chem B 115(45):13212–13221. https://doi.org/10.1021/jp204182c. (**PMID: 22022889**)
62. Gillan MJ (1991) The molecular dynamics calculation of transport coefficients. Phys Scripta T39:362–366. https://doi.org/10.1088/0031-8949/1991/t39/057. (**Publisher: IOP Publishing**)
63. Lam P, Dietrich J, Pearce DJ (2020) Putting the semantics into semantic versioning. arXiv:2008.07069
64. Zhou Q-Y, Park J, Koltun V (2018) Open3D: a modern library for 3D data processing. arXiv:1801.09847
65. Bokeh Development Team (2018) Bokeh: python library for interactive visualization. https://bokeh.pydata.org/en/latest/

Publisher's Note