Monte Carlo molecular simulations with FEASST version 0.25.1

Harold W. Hatch,[1, a)] Daniel W. Siderius,[1] and Vincent K. Shen[1]
Chemical Informatics Research Group, Chemical Sciences Division, National Institute of Standards and Technology, Gaithersburg, Maryland 20899-8380, USA

(Dated: 5 September 2024)

FEASST is an open-source Monte Carlo software package for particle-based simulations. This software, which was released in 2017, has been used to study phase equilibrium, self-assembly, aggregation or gelation in biological materials, colloids, polymers, ionic liquids and adsorption in porous networks. We highlight some of the unique features available in FEASST, such as flat-histogram grand canonical ensemble, Gibbs ensemble, and Mayer-sampling simulations with support for anisotropic models and parallelization with flat-histogram and prefetching. We also discuss how the challenges of supporting a variety of Monte Carlo algorithms were overcome by an object-oriented design. This also allows others to extend classes, which improves software interoperability, as inspired by LAMMPS classes and user packages. This article describes version 0.25.1 with benchmarks, compilation instructions, and introductory tutorials for running, restarting and testing simulations, user guidelines, software design strategies, alternative interfaces and the test-driven development strategy. (This is an author reprint of https://doi.org/10.1063/5.0224283).

I. INTRODUCTION

Since the inception of Monte Carlo (MC) simulations over seventy years ago,[1] MC software currently available to the molecular simulation community includes but is not limited to MUSIC,[2] BOSS,[3] MCPRO,[3] Towhee,[4] Etomica,[5] DL_MONTE,[6] Cassandra,[7] Faunus,[8] RASPA,[9] GOMC,[10] BRICK-CFCMC,[11] SIMONA,[12] ms2,[13] ESPResSo,[14] MCCCS-MN,[15] and the Free Energy and Advanced Sampling Simulation Toolkit (FEASST).[16,17] Despite the number of available MC software, none have usage metrics[18] within an order of magnitude of popular molecular dynamics (MD) software such as GROMACS[19] and LAMMPS.[20,21] The aim of FEASST development is to make an MC software package that is as easy to use and customize as LAMMPS.

FEASST is an open-source MC molecular simulation software package available at `https://pages.nist.gov/feasst` with 76 and 22 thousand lines of C++ source and automated tests, respectively, to implement the following MC methods.[16] Metropolis,[1] Mayer-sampling (MSMC),[22] Wang-Landau[23] or transition-matrix[24] MC may be performed in the microcanonical, canonical (NVT), isothermal-isobaric, grand-canonical (μVT), semi-grand, Gibbs[25] or expanded ensembles.[26] Interaction potentials include square well, Kern-Frenkel,[27] Lennard-Jones (LJ), Mie, (screened) Coulomb, precomputed tables and the Ewald sum. Cell and neighbor lists are also implemented. Bonded potentials include rigid, square well, harmonic, and FENE[28] bonds and angles with TraPPE[29] and Ryckaert-Bellemans dihedrals.[30] Particles may be confined by hard shapes such as slabs, cylinders, spheres, and supertoroids, and the union or intersection of multiple shapes. Confined fluids may also be simulated with table and rigid site potentials with 2D Ewald corrections.[31] A large assortment of trial moves, such as translation, rotation, crankshaft, pivot, rigid cluster, alchemical transformation, (dual-cut) configurational bias (CB and DCCB),[32,33] aggregation volume bias,[34,35] reptation and Jacobian-Gaussian branching[36] have been implemented.

FEASST[17] has been used previously to study colloids, porous networks, ionic liquids, and water and biological molecules. Studies of assembly and gelation of colloids include trimers,[37,38] cubes,[39] binary superlattices,[40,41] cylinders[42,43] and supertoroids.[44] FEASST has also been used to study the assembly of porous networks,[45] entropy correlations,[46] Henry's law constants,[47] and water adsorption[48] in metal-organic frameworks (MOFs). Additional applications include coarse-grained simulations of monoclonal antibodies,[49] ionic liquids,[50] and the calculation of thermodynamic derivatives[51] to predict structural quantities,[52] virial coefficients[53] and explore phase the behavior using Gaussian process regression.[54] Example results obtained with FEASST are highlighted and benchmarked in Section II with a summary of 54 Python-based tutorials available online.

Benchmark studies of two different kinds of MC-specific parallelization methods were performed with FEASST using OpenMP. The first method simultaneously simulates different parts of the macrostate distribution with transition-matrix MC simulations.[55,56] The second method simultaneously prefetches multiple attempted trials from the same configuration, and reconstructs the serial Markov chain.[57]

In Section III, we describe all required steps to run a bulk LJ simulation in the NVT ensemble with FEASST. This example illustrates the ease with which FEASST may be compiled, run, restarted, and tested and provides enough information for readers to get started with the more complex and numerous tutorials present in the larger online documentation that is summarized in Section II K.

One of the major FEASST development challenges was to support not only Metropolis acceptance criteria,

[a)] Electronic mail: harold.hatch@nist.gov

but also flat-histogram Monte Carlo (FHMC), MSMC, Widom insertion and deletion, Henry's law constant and isosteric heat of adsorption calculations.[47] FHMC simulations require flexibility in both the implementation of macrostates and bias functions to enable simulations in μVT and expanded ensembles using Wang-Landau,[23] and transition-matrix and their combination[58,59] with parallelization.[60] In addition, MSMC differs from many-particle simulations because only two particles are simulated for the second virial coefficient (B_2) without periodic boundary conditions (PBCs). MSMC allows hard particle overlap, old configurations are not considered in trial acceptance, CB flexibility is performed differently,[49,61] and the lack of PBCs makes Ewald and long range van der Waals corrections unnecessary. The challenges associated with supporting Metropolis MC, FHMC and MSMC acceptance criteria in the same software are overcome by an object-oriented design described in Section IV that allows modular trial acceptance criteria that are further extendable to address future challenges.

The nomenclature of FEASST classes is also summarized in Section IV for two reasons. The first reason is that a basic understanding of FEASST classes allows users to anticipate the optional arguments available in simulation input files. The second reason is that the object-oriented design enables other researchers to customize FEASST for specific use cases via a plugin architecture, with the option of contributing code to the user community. Due to the wide variety of specialized sampling techniques in MC, many research groups currently use in-house special purpose MC software. FEASST provides an alternative for research groups to create FEASST plugins for their specific modifications as inspired by LAMMPS user packages. This extensibility is enabled by modern software design such as the factory and visitor patterns,[62] as discussed in Section IV E.

Another major FEASST development challenge was to enable periodic saving or writing of the execution state to file (i.e., checkpointing via serialization) so that the MC simulation may be restarted after interruption (i.e., deserialization). Restarting allows users to request higher priority or shorter queues on high performance computer (HPC) clusters by breaking a longer simulation into a series of shorter simulations. Most online FEASST tutorials include restarting options. Furthermore, user-contributed FEASST classes automatically include convenient (de)serialization functions described in tutorials.

This article ends with a discussion of alternative interfaces, customization, testing and licensing in Section VI. FEASST may be used as a library in C++ and Python as well as a client-server mode, which increases the interoperability of this toolkit with other software. Software interoperability may also be facilitated by establishing interfaces to other software with user plugins. Comparisons with published results and the National Institute of Standards and Technology (NIST) Standard Reference Simulation Website (SRSW)[63] fuel the test-driven development strategy described in this article. Finally, the license of FEASST is discussed before concluding remarks in Section VII.

II. EXAMPLE AND BENCHMARK SIMULATIONS USING FEASST

Here, example simulations are described to highlight some of the unique features available in FEASST. This includes examples of μVT FHMC of LJ, SPC/E water, Kern-Frenkel patchy particles, patchy trimers and TraPPE n-octane. FHMC methods bias the simulation to explore macrostates that are not visited frequently. These types of simulations are useful for sampling systems under conditions that include a transition state, such as those in vapor-liquid equilibrium. In addition, MSMC simulations compute B_2 for LJ particles with temperature extrapolation and atomistic protein-protein interactions with implicit-solvent. Finally, μVT FHMC is used to compute LJ adsorption in a repulsive porous network and CO_2 adsorption in the ZIF-8 adsorbent. Example simulations were run with FEASST version 0.25.0, and the changes in version 0.25.1 were made partially in response to reviewers and did not affect the results shown here. Benchmark central processing unit (CPU) clock time is reported for HPC nodes with dual Intel® Xeon® Silver 4216 CPUs with a total of 32 processors per node at 2.10 GHz base frequency (see the disclaimer at the end of Section VI D). If a simulation reports a clock time of 1 h on 32 processors, then each process spent an hour, resulting in a total of 32 processor-hours.

A. Flat-histogram simulation of a bulk LJ fluid

In this example, the vapor-liquid equilibrium of a bulk, single-component LJ fluid of diameter σ and attractive well ϵ is investigated using μVT FHMC in cubic PBCs of length $L = 8\sigma$ with a constant chemical potential, μ. In this ensemble, the macrostate of FHMC is the number of fluid particles, N. A single simulation was parallelized over 32 processors[60] in an HPC node to compute the high temperature, $k_BT/\epsilon = 1/(\beta\epsilon) = 1.5$, macrostate distribution shown by the red line in Fig. 1, where k_B is the Boltzmann constant. The macrostates were distributed across the parallel processes with an exponential weighting factor.[63] At this supercritical temperature, there is a single maximum in the macrostate probability. This simulation was run for less than 3.5 h on 32 processors. For more simulation details, see the fourth tutorial in the `flat_histogram` plugin. Throughout this article, the names of FEASST classes, arguments and plugins use `this typeset`.

A second simulation at a lower temperature, $k_BT/\epsilon = 0.7$, was parallelized with two HPC nodes with 32 cores each for 16 hours. The first node computed the macrostate distribution up to $N = 375$ particles using ag-

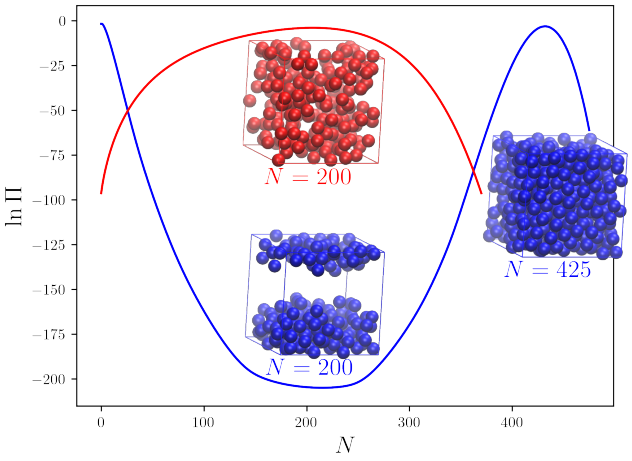

FIG. 1. Natural logarithm of the probability, $\ln \Pi$, of the number of LJ particles, N, obtained by μVT FHMC with $L = 8\sigma$. The red line shows a higher temperature of $k_BT/\epsilon = 1.5$ reweighted to $\beta\mu = -2.227$, and the blue line shows a lower temperature of $k_BT/\epsilon = 0.7$ reweighted to $\beta\mu = -6.257$ for equilibrium between the vapor and liquid phases.[24]

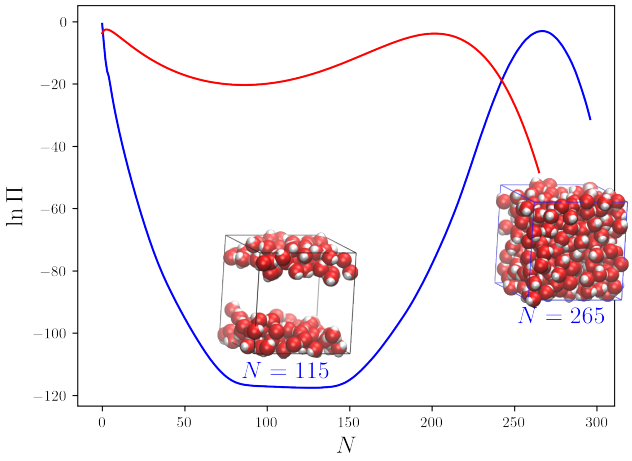

FIG. 2. Natural logarithm of the probability, $\ln \Pi$, of the number of SPC/E water molecules, N, obtained by μVT FHMC with $L = 20$ Å. The red line shows a higher temperature of $T = 525$ K reweighted to $\beta\mu = -8.145$, and the blue line shows a lower temperature of $T = 300$ K reweighted to $\beta\mu = -15.253$ for equilibrium between the vapor and liquid phases.[24]

gregation volume bias (AVB) displacements biased about a distance of 0.9σ to 1.375σ with AVB2,[34,35] AVB4,[48] and insertions and deletions with AVB[64], in addition to standard insertion, deletion and translation moves with a tunable maximum displacement. The second node simulated N from 375 to 475 using ten DCCB steps with a reference potential cutoff at σ. The reference potential utilized a cell list. At this subcritical temperature, the chemical potential, $\beta\mu = -6.257$, may be reweighted to the equilibrium condition of equal probability of vapor and liquid phases. One of the transition states shown in the inset of Fig. 1 for $N = 200$ is a slab of dense fluid.[65] These simulations ran for 16 h on two HPC nodes with 32 processors. For more simulation details, see the tenth tutorial in the `flat_histogram` plugin.

B. Flat-histogram simulation of bulk SPC/E water

The Ewald summation allows FEASST to model long-range charged interactions, as exemplified by the use of FHMC of the fixed point charge SPC/E water[66] model shown in Fig. 2. Fourier-space contributions for each particle are stored separately to allow for the efficient calculation of the change in energy with single-particle perturbations. Ewald vectors, $\mathbf{k}$, were truncated at a squared maximum of $\mathbf{k} \cdot \mathbf{k} = 38$ using a screening length of $\alpha/L = 5.6$.[63] At a temperature near the critical point, $T = 525$ K, as shown by the red line in Fig. 2, standard rigid-body translation, rotation, insertion and deletion trials reasonably converge the macrostate probability distribution. This simulation was run for 22 h on 32 processors. For more simulation details, see the fifth tutorial in the `flat_histogram` plugin.

To facilitate the convergence of the simulations at a lower temperature of $T = 300$ K, shown by the blue line in Fig. 2, additional trials were introduced. The lower densities from 0 to 180 particles were run on an HPC node with 32 processors for 48 h, while 180 to 296 particles were run on a second HPC node with the same number of processors for 117 h. For the first node, standard translations, rotations, insertions and deletions were performed. On the second node, ten DCCB steps were used for insertion and deletion. The reference potential was not simply a shorter cutoff. Instead, only the oxygen sites were considered in order to account for the excluded volume. For the reference, the oxygen sites were modeled as hard spheres with a diameter of 2.857 Å to enable the efficient use of a cell list. DCCB[33,63] greatly improves the sampling and convergence of this simulation. For more simulation details, see the tenth tutorial in the `flat_histogram` plugin.

C. Flat-histogram simulation of Kern-Frenkel patchy particles

Anisotropic interaction models are also supported in FEASST. The Kern-Frenkel[27] model is an orientationally dependent square well of diameter σ, well depth ϵ, and a cutoff of 1.5σ, as illustrated in Fig. 3. At the temperature of $k_BT/\epsilon = 0.7$, the free-energy barrier between the vapor and liquid phases is relatively small compared to the previous results. That is because the simulation is close to the critical temperature. Lower temperatures and smaller attractive coverage lead to complex microstructures.[38,67] This simulation was run for 1.14 h

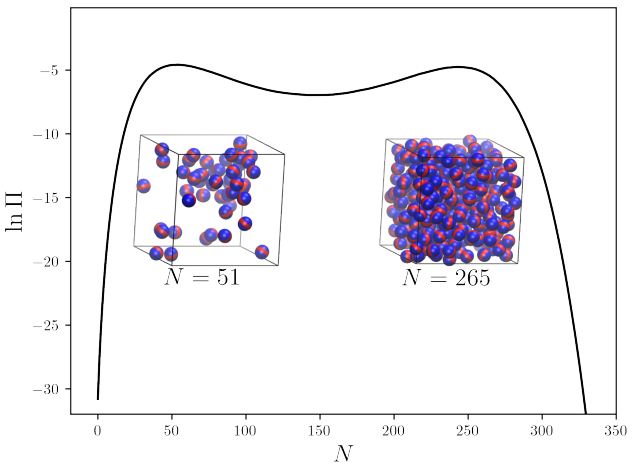

FIG. 3. Natural logarithm of the probability, $\ln \Pi$, of the number of Kern-Frenkel patchy particles, N, obtained by μVT FHMC with $L = 8\sigma$, $k_BT/\epsilon = 0.7$ and $\beta\mu = -3.229$ reweighted for equilibrium between the vapor and liquid density phases.[24] Attractive patches placed on opposite poles of the particle, shown in blue, cover 70% of the surface, with a hemispherical region of no attraction, shown in red.

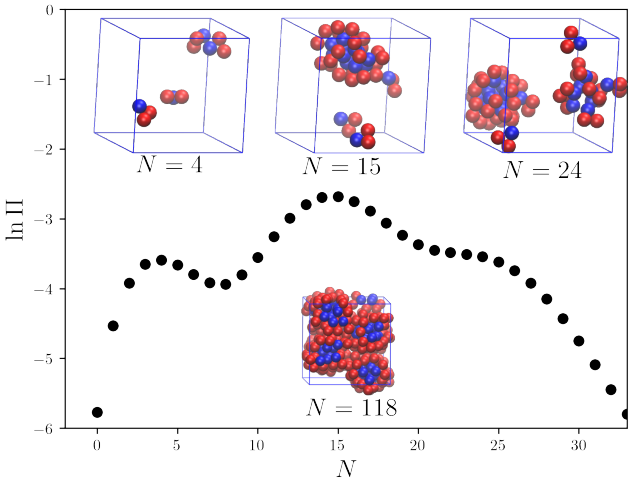

FIG. 4. Natural logarithm of the probability, $\ln \Pi$, of the number of patchy trimer particles, N, obtained by μVT FHMC with $L = 8\sigma$, $k_BT/\epsilon = 1/0.275$ and $\beta\mu = -5$.

on 32 processors. For more simulation details, see the ninth tutorial in the `flat_histogram` plugin.

D. Flat-histogram simulation of patchy trimer particles

In this example, the self-assembly of a patchy trimer model[37] is illustrated in Fig. 4. Each of the three beads of diameter σ in the trimer form an equilateral triangle of length σ. Only one of the beads has an attractive LJ potential with the similar bead on each of the other trimers. Otherwise, all other interactions between

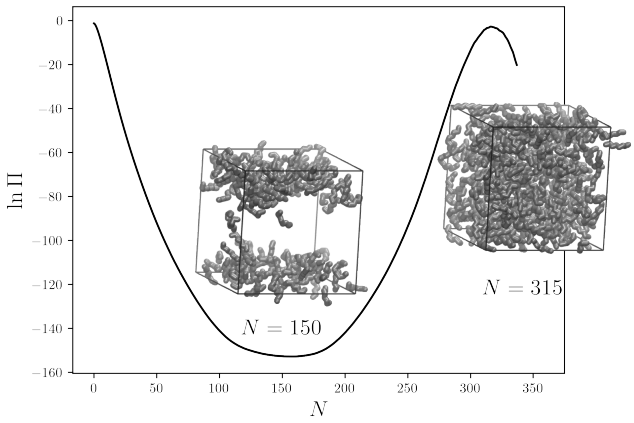

FIG. 5. Natural logarithm of the probability, $\ln \Pi$, of the number of TraPPE united atom n-octane[29] molecules, N, obtained by μVT FHMC with $L = 45$ Å, $T = 350$ K and $\beta\mu = -12.417$ reweighted to equilibrium between the vapor and liquid phases.[24]

beads are purely repulsive with the Weeks-Chandler-Andersen[68] (WCA) potential shift and cutoff at $r_c/\sigma = 2^{1/6}$. This trimer model is similar to colloids synthesized and studied in experiment with a short range attractive interaction due to surface roughness in a solution with depletants.[69,70]

Because only one of the three beads is attractive, trimers self-assemble into micellar structures at relatively low temperatures. The attractive beads occupy the center and the repulsive beads decorate the surface of the micelle. The stability of these microstructures may be reflected in $\ln \Pi$ from μVT FHMC, shown in Fig. 4. For $N = 4$ trimers, there is a maximum in $\ln \Pi$ representative of a homogeneous vapor phase. There is a well-defined minimum in the probability between $N = 4$ and 15 trimers, where $N = 15$ corresponds to the formation of the first stable micelle. Another maximum is observed during the growth of a second stable micelle. The simulation was run for 96 h on 32 processors with $N \leq 100$ (not shown in Fig. 4), For more simulation details, see the eighth tutorial in the `flat_histogram` plugin.

E. Flat-histogram simulation of TraPPE n-octane

FEASST also supports the CB regrowth of molecules with intramolecular flexibility. In the example shown in Fig. 5, the vapor-liquid equilibrium properties of TraPPE n-octane[29] are obtained at $T = 350$ K. The n-octane molecules were regrown using four steps for each site with a DCCB cutoff of 4.091 Å. In addition to rigid body translations and rotations tuned by a maximum parameter, NVT ensemble trials also included regrowths of 1 to 4 atoms at either end of the molecule. Insertions and deletions also utilized the same CB methods for the entire molecule. Similar to Figs. 1 and 2, the stable vapor

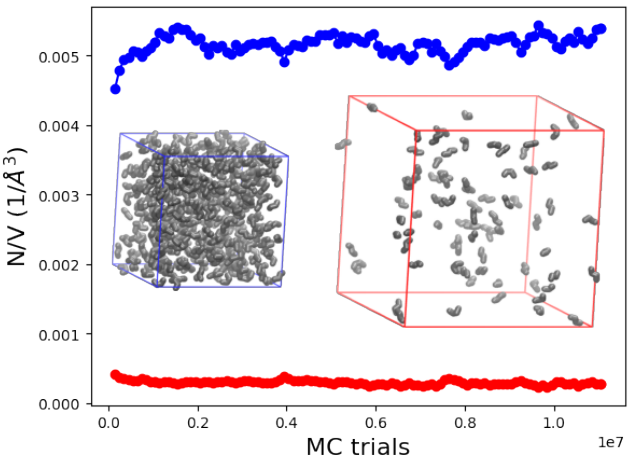

FIG. 6. Equilibration of the density, N/V, of pure component TraPPE united atom n-butane[29] in the Gibbs ensemble with $T = 350$ K, 512 total molecules and 2240 Å^3 total volume. The symbols are blue for the liquid and red for the vapor with the corresponding boundaries in the snapshots.

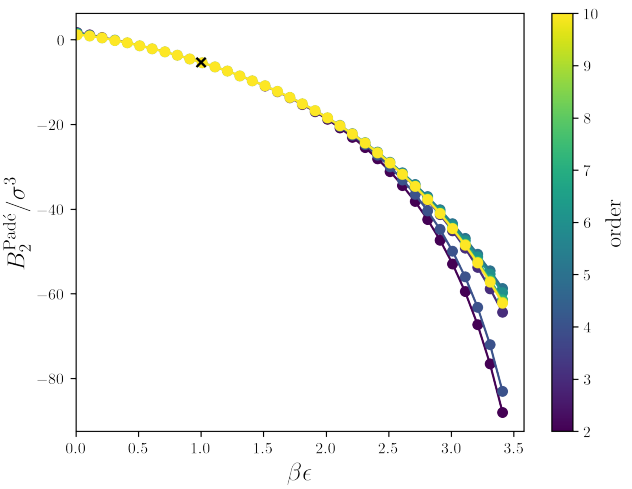

FIG. 7. Second virial coefficient, B_2, of LJ particles obtained from MSMC at $k_BT/\epsilon = 1/(\beta\epsilon) = 1$, and the color symbols show extrapolation[53] with Padé approximants of various orders. The x symbol shows $\frac{3B_2(\beta\epsilon=1)}{2\pi\sigma^3} = -2.535 \pm 0.001$, where $\pm$ indicates the standard deviation of the mean from 32 independent simulations.

and liquid densities are separated by a number of transition states, including the slab-like configuration shown in the inset. This simulation was run for 120 h on 32 processors. For more simulation details, see the eleventh tutorial in the `flat_histogram` plugin.

F. Gibbs ensemble simulation of TraPPE n-butane

FEASST also supports Gibbs ensemble simulations as shown in Fig. 6. Multiple `Configuration` objects can be simulated with volume and particle transfer between them using CB. The interaction potentials in each configuration can be controlled and optimized independently, and more than two configurations are supported. The pressure is computed using the test volume method.[71] This simulation was run for 0.44 h on 1 processor. For more simulation details, see the second tutorial in the `gibbs` plugin.

G. Temperature extrapolation of second virial coefficients (B_2) using MSMC

While the previous examples used μVT FHMC with a variety of isotropic and anisotropic models, FEASST also supports MSMC to compute B_2 with a two particle simulation. Enabling the use of MSMC in the same software as Metropolis and FHMC was a challenge because MSMC does not need to recompute the energy of the old configuration, and the overlap between particles which may typically be immediately rejected in Metropolis MC is allowed in MSMC. The object-oriented design of FEASST makes it easier to interchange Metropolis, FHMC and MSMC acceptance criteria in a modular fashion while re-using the same trial move implementation. Fig. 7 shows the extrapolation of B_2 of LJ particles using MSMC at $\beta\epsilon = 1$ with a hard sphere reference potential of diameter equal to the LJ σ parameter, as detailed in Ref. 53 and also implemented in Ref. 72. In short, the thermodynamic derivatives of the Mayer functions with respect to β, to a given order, are averaged to enable Taylor-series extrapolation, which may be corrected with Padé approximants. This simulation was run for less than 5 min on 32 processors. For more simulation details, see the first tutorial in the `mayer` plugin.

H. MSMC with atomistic proteins in implicit solvent

FEASST supports the use of MSMC to calculate B_2 of rigid atomistic proteins in implicit solvent[73] as shown in Fig. 8 for the lysozyme 4LYT protein databank (PDB) structure.[74] The pdb2pqr30[75] software converted PDB to PQR files with PARSE[76] charges and PROPKA[77] protonation states and Autodock4 parameters[78] assigned with the `coarse_grain_pdb` module in pyfeasst. These simulations were run for 24 h on 32 processors. For more simulation details, see Ref. 79 and the eighth tutorial in the `mayer` plugin.

I. Confinement of LJ in a WCA porous network

Simulations in confinement are also possible in FEASST with hard shapes as well as rigid interaction sites. For example, an LJ fluid may be confined inside of a repulsive porous network that is carved from a cu-

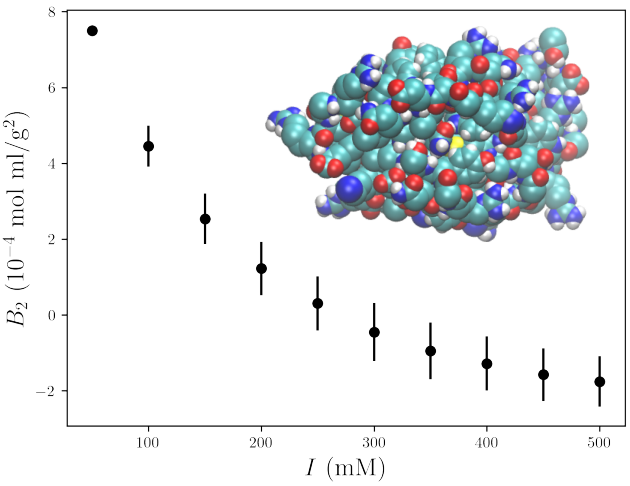

FIG. 8. Second osmotic virial coefficient, B_2, of a rigid atomistic protein model of lysozyme (PDB 4LYT, pH 6, 14.315 kDa, 298.15 K) with implicit solvent computed over a variety of ionic strengths, I, using MSMC with a $\sigma = 30$ Å hard sphere reference potential. Error bars are standard error of the mean of 32 independent simulations.

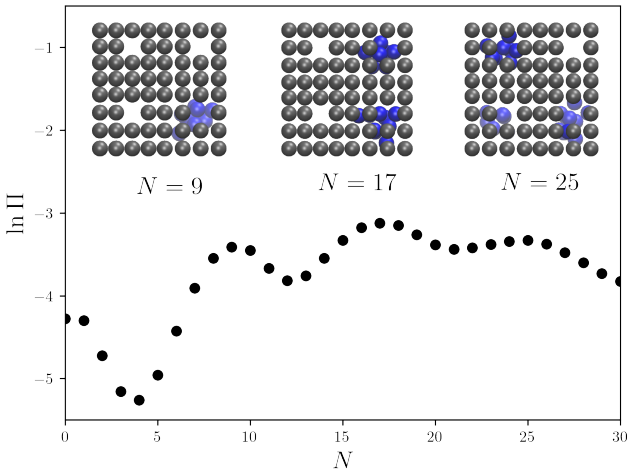

FIG. 9. Natural logarithm of the probability, $\ln \Pi$, of the number of LJ particles, N, shown in blue, obtained by μVT FHMC with $\beta\epsilon = 1/0.3$ and $\beta\mu = -3.6333$. The rigid porous network, shown in gray, was carved from a cubic lattice of purely repulsive WCA particles to form eight pores with connecting channels.[55] The inset shows maximums in the probability when a successive number of pores are filled, similar to maximums from the self-assembled structures shown in Fig. 4.

bic lattice of rigid sites interacting with the LJ fluid using a purely repulsive WCA potential. The porous network has two pores in each dimension (eight total pores) connected by narrow channels, as described previously.[55] The volume, $V = (9\sigma)^3$ in PBCs was held fixed and the LJ particles were inserted and deleted with μVT FHMC. Trial moves included translation, AVB2,[34,35] AVB4[48] and AVB insertion and deletion[64] in addition to standard insertion and deletion. This simulation was run for 1 h on 32 processors with $N \leq 64$ (not shown in Fig. 9). For more simulation details, see the first tutorial in the `confinement` plugin.[55]

J. Adsorption of CO_2 in ZIF-8 using μVT FHMC

FEASST can also compute an adsorption isotherm. In this example, the amount of TraPPE[80] CO_2 in the metal-organic framework (MOF) ZIF-8[81] is determined as a function of pressure using μVT FHMC.[82] The rigid MOF may be treated as a single rigid particle, while only CO_2 is subject to trial translations, rotations, insertions and deletions. The FHMC macrostate is the number of CO_2 molecules, N, and the converged $\ln \Pi$ determines the average loading in the MOF, i.e., $\langle N \rangle$. The relationship between μ and pressure, p, is obtained by an independent simulation of the bulk adsorbate fluid at the same temperature of the confined simulation, after which the $N = 0$ state of the bulk fluid is related to an ideal gas reference state. The $N = 0$ state for a fluid adsorbed in the MOF (or in any confinement) is not equivalent to an ideal gas; the $N = 0$ state can, however, determine the grand potential free energy of the system at some μ relative to that of the MOF.[59,82] The bulk and confined simulations were run for 2 h on 32 processors. For more simulation details, see the second and third tutorials in the `confinement` plugin and earlier work that describes the system setup.[82]

K. Summary of additional tutorials in the online documentation

Including the tutorials highlighted above, the online documentation contains a total of 54 examples in the section titled Tutorials that span a range of different MC methods and interaction potentials. Throughout this article, online documentation sections are underlined. Most online tutorials use pyfeasst to write the text interface files, run the simulations and post-process them. The use of pyfeasst is optional and allows the tutorials to be run either locally or on an HPC with the "run_type" flag, as part of an automatic testing process. Many tutorials support the use of the "help" flag to describe the optional arguments and their default values. To use these scripts on HPC resources, pyfeasst assumes a SLURM queue with optional flags given by "queue_flags." Often the tutorials have unusually short `hours_terminate` so that the simulation will restart, which allows for quick identification of restart errors. We now summarize the tutorials currently available online.

- Single-site LJ potentials are used to demonstrate NVT, μVT, isothermal-isobaric, Gibbs, and temperature expanded ensembles and rigid cluster

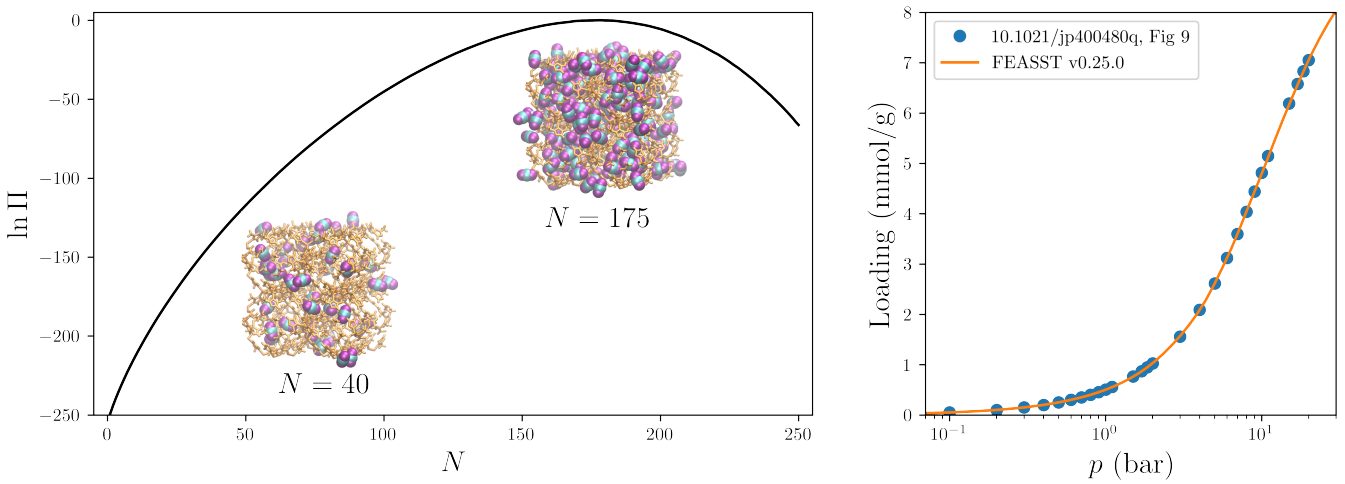

FIG. 10. (Left) Natural logarithm of the probability, $\ln \Pi$, of the number of CO_2 molecules, N, adsorbed into the ZIF-8 MOF, shown in the snapshots as gold for $N = 40$ (1.8 mmol/g) and $N = 175$ (8 mmol/g), at a temperature of 303 K. (Right) Loading of CO_2 as a function of pressure, p, with results from FEASST and previously published results.[82]

moves.

- FHMC simulation tutorials also consider the ideal gas, RPM,[83] hard sphere and EPM2 CO_2.[84] Other FHMC tutorials include CB,[32] aggregation volume bias,[35] efficiency comparisons with parallelization[57] and expanded grand canonical ensemble simulations.[50] Semi-grand ensemble simulations with FHMC include both a binary LJ simulation and a binary CO_2 and N_2 mixture.
- Customization tutorials include user-defined potentials with tables, custom pair-wise models, user-derived periodic analysis and one-time actions.
- MSMC simulations are demonstrated for patchy trimers, TraPPE ethane, Kern-Frenkel patchy particles, rigid-body proteins and coarse-grained antibody models.
- Tutorials with CB simulate freely-jointed chains and 5-mer and 20-mer linear chains. CB examples of angle and branching are also available. Furthermore, temperature extrapolation of the radius of gyration of a linear chain is demonstrated.
- Tutorials in confinement include slab, cylindrical and spherical confinements, as well as building custom confinements with unions and intersections of the above shapes.
- Comparisons with small-angle scattering using the FFTW library is provided, as well as an example for post-processing such calculations from existing trajectories with both simple hard sphere models and coarse-grained antibody models.
- Other tutorials include parallelization with prefetch,[57] anisotropic tabular potentials,[42] netCDF trajectories and block average analysis of correlated data.[85]

III. GETTING STARTED WITH FEASST SIMULATIONS

Using FEASST to perform MC simulations begins with compiling the "fst" and "rst" executables. The syntax of input files to run MC simulations is then discussed, including a description of file formats to define particles and input initial configurations. Energy calculations of reference configurations and restarting simulations from checkpoint files are also introduced. Finally, an example of an input file to perform DCCB[33] insertion and deletion of SPC/E water molecules is explained.

A. Compiling executables and software dependencies

FEASST is designed for Linux command line environments found in HPC clusters. Text 1 shows the typical Bourne-Again SHell (Bash) shell compilation commands for FEASST. Throughout this article, command line examples assume the use of Bash, but FEASST is also compatible with many other shells. Dependencies include a C++ compiler such as g++, CMake, Git and Python 3.[86–89] This is demonstrated on the first line of Text 1 for a variety of package management tools found in Ubuntu, RedHat, Fedora, Rocky, AlmaLinux, Mac or other operating systems. FEASST is designed for minimal required dependencies, and optional dependencies are placed in optional plugins to simplify compilation for most users.

Throughout this tutorial, FEASST is placed in the home directory given by the variable $HOME, as shown on line 2, but FEASST may be compiled anywhere

TEXT 1. Command line compilation of FEASST executables "fst" and "rst."

```
1 #[apt,yum,dnf,brew] install g++ cmake git python3
2 cd $HOME
3 git clone https://github.com/usnistgov/feasst.git
4 mkdir feasst/build && cd $_
5 git checkout v0.25.1
6 cmake ..
7 make install -j$(nproc)
8 #pip install jupyter matplotlib pandas scipy
      ../pyfeasst
```

by replacing the $HOME variable with a different one. Git obtains and manages the version of source code in lines 3 and 5, respectively, and are optional if the user instead downloads the v0.25.1 tag manually from GitHub.[90] In line 4, an out-of-source build directory decouples the source code from the compiled objects. Line 6 uses CMake to search for required dependencies using "$HOME/feasst/CMakeLists.txt," which includes a number of options for additional dependencies such as optional FEASST plugins, GoogleTest, OpenMP, Ccache, Sphinx, SWIG, GCOV, FFTW, pybind11, MPI and NetCDF. On line 7, the "make" command compiles the "fst" and "rst" executables, which read input and checkpoint files, respectively. These two executables should be found in the directory "$HOME/feasst/build/bin." The "make" command is parallelized with the optional "-j" flag.

The last line installs optional Python packages that are required by the online tutorials, but not by the examples in this article. It is important to provide pip a path to the specific pyfeasst directory in FEASST to ensure that the versions match. If "../" is left out of line 8 in Text 1, which assumes that the command is given from the directory "$HOME/feasst/build/," pip may find a PyPI version of pyfeasst that may not work with the specific FEASST version that was installed locally.

B. FEASST simulation text file example

MC simulations are performed by a text input to the "fst" executable with the command

```
$HOME/feasst/build/bin/fst < script.txt
```

where the contents of script.txt provide simulation instructions to the "fst" executable. An example script.txt is given in Text 2, in which FEASST computes the average potential energy of 500 LJ particles in cubic PBCs at a fixed density, $\rho\sigma^3 = 0.003$, and temperature $k_BT\epsilon = 0.9$,[91] with about 5 s of run time.

Lines with a first symbol of "#" are ignored and serve as comments. Otherwise, every line begins with the name of a FEASST class. In line 2, `MonteCarlo` prepares

TEXT 2. FEASST input file for an MC simulation of LJ particles to reproduce the SRSW-reported result for the canonical ensemble average energy.

```
1  # comments begin with the # symbol
2  MonteCarlo
3  RandomMT19937 seed 1572362164
4  Configuration cubic_side_length 55.0321208149104
       particle_type0 /feasst/particle/lj.fstprt
5  Potential Model LennardJones VisitModel
       VisitModelCell min_length max_cutoff
6  Potential VisitModel LongRangeCorrections
7  ThermoParams beta 1.111111 chemical_potential0 -1
8  Metropolis
9  TrialTranslate weight 1 tunable_param 2
10 Checkpoint checkpoint_file lj_checkpoint.fst
       num_hours 0.0001
11 Tune
12 CheckEnergy trials_per_update 1e4 tolerance 1e-8
13 TrialAdd weight 2 particle_type 0
14 Run until_num_particles 500
15 RemoveTrial name TrialAdd
16 Run num_trials 1e5
17 RemoveModify name Tune
18 Log trials_per_write 1e4 output_file lj.csv
19 Energy trials_per_write 1e4 output_file lj_en.csv
20 Movie trials_per_write 1e4 output_file lj.xyz
21 Metropolis num_trials_per_iteration 1e4
       num_iterations_to_complete 1e2
22 Run until_criteria_complete true
```

an MC simulation. Throughout this article, the names of FEASST classes, arguments, and plugins use `this typeset`. Line 3 of Text 2 begins with `RandomMT19937`, which is the name of a class that uses the Mersenne Twister pseudo-random number generator.[92] Unlike in the previous line, the name of the class is followed by an optional argument pair of `seed` and a number to input the value of the seed. Each argument pair is separated by a space, and first has the name of the argument and then the value of that argument (i.e., a dictionary structure). Every non-comment or non-empty line begins with the name of a FEAST class, followed by one or more space-separated pairs of arguments, resulting in an odd number of space-delimited character strings.

Class arguments may be found in the Text Interface section of the FEASST website.[16] As shown in Fig. 11, the arguments of `RandomMT19937` are the same as `Random`, and clicking on the link for `Random` leads to a description of the seed argument. `RandomMT19937` derives from the `Random` base class, as indicated by the text "class `RandomMT19937 : public feasst::Random`." Because `RandomMT19937` is one of the simplest classes in FEASST, it is the easiest to screen capture and show in Fig. 11 in comparison with the more informative documentation for other FEASST classes. Awareness of FEASST object-oriented class structure helps users intuit arguments available to a class, and will be discussed in Sections IV.

FIG. 11. Online documentation shows that `RandomMT19937` is a derived class of `Random`, and accepts the arguments of `Random`, such as seed. This shows why the object-oriented design is important to FEASST users.

In line 4, `Configuration` has two pairs of arguments. Multiple argument pairs may appear in any order without affecting the simulation. The first pair initializes an empty `Domain` of cubic shape for a given PBC length, L, which anticipates the addition of 500 particles for a target density of $\rho\sigma^3 = 0.003$. Manually typing many decimal places is not necessary in FEASST tutorials that use Python formatted strings to generate the input file.

The user must input values with consistent units. For example, the distance units of PBC length, L, should correspond to the same units for the σ parameter of LJ and the positions of the sites. Units of energy should be consistent with the ϵ parameter and the inverse of $\beta = \frac{1}{k_B T}$. Units of pressure are energy per volume. In the special case of Coulomb interactions, the distance units are assumed to be Å, and energy units are kJ/mol and elementary charges. For charge interactions, the `PhysicalConstants` can be customized as an argument in the Configuration class. In addition, the `physical_constants` pyfeasst module provides physical constants for convenience during unit conversion. An example of this can be found in the SPC/E tutorials in the `monte_carlo` plugin, where the inverse temperature is converted from units of inverse Kelvin to mol/kJ. The tenth tutorial in the `flat_histogram` plugin for SPC/E also includes a detailed post processing of macrostate distributions, including unit conversions for density and pressure. In addition, the sixth tutorial in the `flat_histogram` plugin contains an example charge conversion for the RPM model to compare with the results of Ref. 50.

The second argument pair in line 4 defines a type of particle that may exist. No actual particles are physically added to the `Configuration` with this argument. The particle type index of 0 is given at the end of the argument `particle_type0` for the first type of particle defined. FEASST counts from zero following the conventions of C++ and Python. A second particle type would be defined with the argument `particle_type1` for a particle type index of 1. The value of the second argument pair, "/feasst/particle/lj.fstprt" is the path to a file that describes an LJ particle, as discussed in Section III C. The "lj.fstprt" file informs FEASST that particles of type 0 are represented by a single interaction site and inputs the parameters $\epsilon = \sigma = 1$ and $r_c = 3$, where r_c is the interaction cut off distance. The characters "/feasst" in a FEASST input file are always replaced by the base directory of the FEASST compilation. For example, if FEASST was compiled in "$HOME/feasst" then "/feasst" would be replaced with "$HOME/feasst" automatically.

Interactions between sites are defined by lines 5 and 6 using the `Potential` class. Line 5 initializes the `LennardJones` interaction between sites. In this case, the `Model` argument of `Potential` takes a derived class name of `Model` as the value. Similarly, the `VisitModel` argument of `Potential` initializes a cell list with `VisitModelCell` for low density simulations with the cut off as the minimum cell size. Line 6 adds `LongRangeCorrections` by assuming a unit radial distribution function beyond the cutoff distance.[93,94]

Lines $7-12$ enable the NVT ensemble MC simulation of the LJ particles. Thermodynamic parameters are input by the `ThermoParams` class in line 7. The chemical potential, μ, is input with an index corresponding to the particle type. Line 8 initializes `Metropolis` acceptance. We recommend generating FEASST input files with Python formatted strings to input $\beta = 1/0.9$ to maximum precision, as done in the online tutorials, rather than manually inputting 1.111111 without enough decimal places as done in Text 2 to simply keep line 7 on one line in this article's double column format. Line 9 initializes a translation trial move, `TrialTranslate` with a given weighted probability to attempt the `Trial`, and a `Tunable` maximum displacement of 2 in each dimension. Line 10 writes a `Checkpoint` file every `num_hours` that may restart a simulation, as described in Section III E. The `num_hours` is small in this example for illustrative purposes, and should be larger for production simulations. `Tune` changes the maximum displacement parameter to reach a target acceptance. `Tune` is specified without arguments and uses the default class argument to adjust tunable parameters every 10^3 trial moves of a given trial type. On line 12, `CheckEnergy` compares the total potential energy obtained from a series of energy changes over 10^4 trials to the total potential energy recalculated from the entire `Configuration`, and stops the simulation with an error if those total energies are not within 10^{-8}.

Lines $13-15$ randomly add the desired number of particles. First, a `Trial` to add particles is introduced. Then, the simulation is run with both `TrialTranslate` and `TrialAdd` until 500 particles are present. Finally, `TrialAdd` is removed to re-establish the NVT ensemble. Line 16 equilibrates with 10^5 trials while adjusting the tunable maximum displacement parameter, and line 17 removes `Tune` to obey detailed balance in production simulations with a fixed maximum displacement. Lines $18-20$ initialize `Log`, `Energy` and `Movie`, all of which are FEASST `Analyze` classes that periodically output to

files. Line 21 defines the production simulation to be 10^2 iterations of 10^4 trial moves per iteration. Finally, line 22 instructs FEASST to run the production simulation of 10^6 total trials.

The outputs from the three `Analyze` objects are in comma-separated value (CSV) and XYZ-formatted files described in Section III D. While `Log` outputs the instantaneous energy every 10^4 trials, `Energy` outputs the ensemble-average that was accumulated every `Trial`. Thus, `Energy` is more suitable for computing accurate thermodynamic averages while `Log` is more suited toward checking the current status of the simulation. `Log` also prints `Trial` acceptance probabilities and `Tunable` parameters. `Movie` outputs the instantaneous `Site` positions in the XYZ format with a VMD[95] script viewable with the following command:

```
vmd -e lj.xyz.vmd
```

According to the SRSW,[63] the average energy per particle at these conditions is $\langle U/N \rangle_{SRSW} = -2.9787 \times 10^{-2} \pm 3.21 \times 10^{-5}$, where $\pm$ corresponds to the 67% coverage interval of the mean obtained from block averages of size 5×10^7 trials. In Text 2, the average extensive energy output to the file named "lj_en.csv" results in $\langle U/N \rangle_{\text{FEASST}} = -2.9821 \times 10^{-2} \pm 2.43 \times 10^{-4}$, which is statistically equivalent to the SRSW within the 67% confidence level.

The standard error of the mean from the blocking method[85,96] is implemented as described in the `Accumulator` class documentation and the Supporting Information of Ref 55. In short, block standard deviations are obtained by on-the-fly adjustment of block sizes. The reported block standard deviation is the largest block standard deviation among all block sizes that have a minimum of ten blocks and a maximum of $2^6 - 1$ blocks, where 6 is the default value of the `max_block_operations` argument in `Accumulator`. Further examples of block analysis are also provided in the `math` plugin tutorial, although it requires installation of the Python interface using SWIG. By default, not all block operations are stored to avoid slowing down the simulations.

C. Particle files

In FEASST, particles represent atoms, molecules, colloids or coarse-grained models with a number of bonded interaction sites. As in line 4 of Text 2, particle files inform FEASST about the kind of particles allowed in the `Configuration`, but do not add particles to the `Configuration`. Instead, particles may be added with `TrialAdd`, as illustrated in Text 2, or using the `Configuration` argument `add_particles_of_type[i]`, where `[i]` refers to the particle type to add. `Site` coordinates may also be input with an XYZ file as described in Section III D. If no particles are present and there is only one particle type, then the particles are added automatically. Otherwise, if more than one type of particle is present, the order of particles in the XYZ file should be all particles of the first type, all particles of the second type, etc, as described in Section III D.

TEXT 3. FEASST particle file that describes a single-site LJ particle with $\epsilon = \sigma = 1$ and $r_c = 3$ as found in the lj.fstprt file in $HOME/feasst/particle.

```
1  # LAMMPS-inspired data file
2
3  1 sites
4
5  1 site types
6
7  Site Properties
8
9  0 sigma 1.0 epsilon 1.0 cutoff 3.0
10
11 Sites
12
13 0 0 0.0 0.0 0.0
```

An example particle file for a single-site LJ model is shown in Text 3. Comments begin with the "#" character and may only be present at the start of the file. The format is inspired by LAMMPS with major differences, as described in the online documentation section Particle files and units. "Site Properties" are listed with the type index starting from zero and then argument pairs with names that depend upon the `Model`, such as epsilon and sigma for `LennardJones`. The "Sites" have five columns, beginning with the site index, the type of the site and then three coordinates. Typically, the first site is placed on the origin and used as a pivot point for rotation in `TrialParticlePivot` when there are multiple sites. It is recommended that units are listed in the particle files when charge interactions are involved, such as in "$HOME/feasst/particle/spce.fstprt."

D. XYZ and reference configurations

The positions of all particles may be input directly into `Configuration` using an XYZ-formatted file as shown in Text 4 and 5 for the SRSW LJ reference configuration 4.[63] The first line is the number of sites. The second line begins with a placeholder for an optional value (such as an order parameter or macrostate), followed by the dimensions of the periodic cell. The last three numbers in the second line refer to triclinic tilt factors xy, xz and yz as described in the online documentation for `Domain`. The remaining number of lines should match the number of sites.

Text 5 shows a FEASST input file to compute the instantaneous potential energy of the single configuration. In this case, line 2 uses the `Configuration` argument `xyz_file` with a file name of "lj_ref_config.xyz" given in

TEXT 4. FEASST XYZ-formatted file for a single configuration of 30 LJ particles to reproduce the fourth SRSW reference configuration energy. For brevity, the ellipsis on line 6 represents the coordinates for sites $4-30$.

```
30
-1 8.0 8.0 8.0 0.0 0.0 0.0
0 1.077169909511 -1.020988125886 -1.348259447733
0 0.1830884592213 -1.557698231574 -1.782405485883
0 -2.060346185437 3.92737827629 3.889555115290
...
```

TEXT 5. Compute the potential energy of a single reference configuration to high precision using this FEASST input file.

```
MonteCarlo
Configuration xyz_file lj_ref_config.xyz
    particle_type0 /feasst/particle/lj.fstprt
Potential Model LennardJones
Potential VisitModel LongRangeCorrections
ThermoParams beta 0
AlwaysReject
Log output_file lj.csv max_precision true
Run num_trials 1
```

Text 4 and supplied in the Supplemental Online Material in its entirety. Because there is only one particle type given, the number of particles does not need to be initialized and is simply assumed based on the `xyz_file`. If more than one type of particle was initialized, use the `Configuration` arguments `add_particles_of_type0`, `add_particles_of_type1`, etc., to explicitly add the number of particles of each type. When adding particles with this argument, the position in the particle file is used and all particles are overlapping. To input specific coordinates, the XYZ file must provide coordinates in the specific order of all particles of type 0 followed by all particles of type 1, etc.

According to the SRSW,[63] the LJ pair-wise potential energy is $U^{LJ}/\epsilon = -16.790$ and the long-range corrections are $U^{LRC}/\epsilon = -0.54517$. Using Text 4 and 5, these energies are -16.79032130462587 and -0.5451660014945706 respectively, as output into the "lj.csv" file, which is equivalent to the SRSW values within the reported precision.

FEASST also supports simulations in two dimensions. See the file "$HOME/feasst/particle/atom2d.fstprt" as an example. In this case, the XYZ files have third dimension coordinates and PBC lengths of 0.

E. Restarting a FEASST simulation from Checkpoint

`Checkpoint` periodically writes a file that may restart a simulation. Restarting a simulation uses the "rst" executable with the command

```
$HOME/feasst/build/bin/rst lj_checkpoint.fst
```

where "lj_checkpoint.fst" is obtained from the simulation in Text 2. `Checkpoint` also includes a `num_hours_terminate` argument that may safely terminate a simulation before the queued wall clock limit and may be detected in Bash using the command

```
?  != 0
```

`Checkpoint` saves the state of most class data members. This allows restarting a simulation without any discernible difference in results from a simulation that was not restarted. Because all data, including cell and neighbor lists, are written to a `Checkpoint` file, the files may become very large in some cases.

TEXT 6. `TrialGrowFile` example using DCCB insertion and deletion of SPC/E water.

```
TrialGrowFile

particle_type 0 weight 2 transfer true site 0
    num_steps 10 reference_index 0
bond true mobile_site 1 anchor_site 0
    reference_index 0
angle true mobile_site 2 anchor_site 0
    anchor_site2 1 reference_index 0
```

F. Example of configurational bias (CB) insertion of SPC/E water

Although the previous examples considered a bulk fluid with single-site LJ interactions without CB, FEASST also supports CB and DCCB[33] insertions, deletions and regrowth. Using the `TrialGrowFile` class, the CB of particles is described in text files with a similar argument pair structure that may be programmatically generated in Python for more complex molecules.

An example `TrialGrowFile` input file is shown in Text 6 for the insertion and deletion of an SPC/E water molecule with an oxygen atom followed by 2 hydrogens. The file format is described more extensively in the online documentation, but here we will touch on the basic aspects. The first line ensures that the file was intended for `TrialGrowFile`. A `TrialGrowFile` input file may contain multiple trial moves, with each move separated by a blank line. In this case, the only blank line is line 2 because there is only one trial. The trial begins on line 3.

The first line for each trial contains arguments such as the type of particle as well as the weight to select the particular trial from among all trials in `MonteCarlo`. Both particle insertions and deletions are attempted with equal probability due to the `transfer true` argument on line 3. The oxygen atom, with site index 0, is placed with

ten DCCB steps using the first reference potential. This assumes that the first reference potential was already initialized using the `RefPotential Action` as shown in the tenth tutorial in the `flat_histogram` plugin for low temperature water. In this case, the first site is the oxygen atom which contains all excluded volume interactions. Thus, an efficient reference potential for DCCB would be a short-range potential that accounts for exclude volume using a cell list between only oxygen atoms while ignoring hydrogen.

Line 4 places the first hydrogen based on the bonded potential to the oxygen anchor site. Line 5 places the second hydrogen based on the bond to the oxygen as well as the angle with the first hydrogen. Documentation for these arguments is in `TrialGrowFile`. Because both lines 4 and 5 do not include the `num_steps` argument, the default number of steps is 1 such that CB is not used for the hydrogen atoms. Finally, the full potential energy is computed in the DCCB method and used as part of the acceptance criteria.

G. FEASST versions indicate text input file and checkpoint file backwards compatibility

This article describes FEASST version 0.25.1. The first version index refers to the major version of the C++ application programming interface (API) and is currently 0 because the C++ API is under rapid development. We assign a secondary API as the input file interface, with the second version number corresponding to a major version update to the input file interface, with backward incompatible changes, and the third version number corresponding to a minor version update, with backwards compatible changes. The public interface is defined as a dictionary of all class arguments using the tool "$HOME/feasst/dev/tools/analyze_public_interface.py" so that each version is systematically compared with the "dictdiffer" Python tool.

When the second version index, currently 25, increases, input files may expect different arguments. For example, when the second version index increased from 23 to 24, the `file_name` argument of `Checkpoint` was changed to `checkpoint_file`. Backwards incompatible changes such as these are listed in the Change Log of the Text Interface section in the online documentation, and the old arguments may still be supported with a deprecation warning. For example, if the `file_name` argument is given to `Checkpoint` in version 0.24, the simulation will proceed as in version 0.23 with a warning to update the name of the argument. The third version index, currently 0, increases when backwards compatible changes are made, such as introducing new features that do not affect existing features. Although `Checkpoint` changes are made backwards compatible where possible, by giving each class a unique serialization version, `Checkpoint` files may only work with the exact same version in other cases.

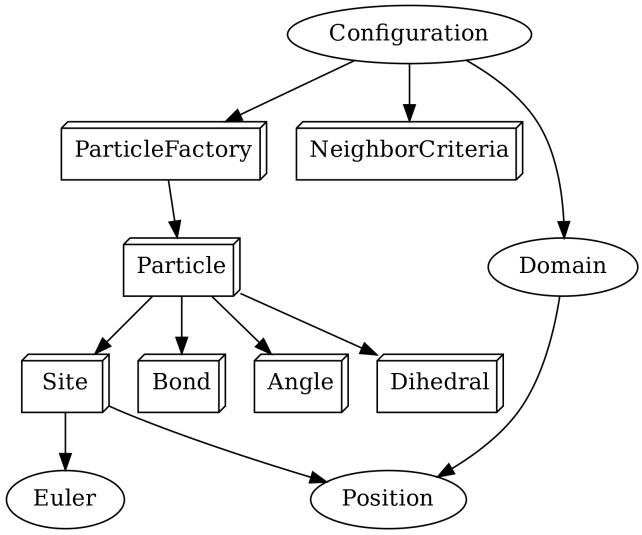

FIG. 12. Flowchart for the `Configuration` class where an oval is a class and arrows point toward classes contained within the originator. The boxes represent multiple classes. To improve readability, not all FEASST classes are shown.

IV. FEASST NOMENCLATURE AND DESIGN

Here, we summarize the design and naming of FEASST classes for several reasons. New users of FEASST benefit from the knowledge of which FEASST classes contain and subclass other classes to anticipate which arguments and options are available to each class. Such an object-oriented design also aids in the customization of FEASST. Instead of building MC simulations with only integers, floating point numbers, and character strings, we use object-oriented programming to create objects specifically designed for MC simulations. This approach facilitates modular, reusable, extensible and maintainable software.

A. The Configuration class

Fig. 12 shows some of the classes contained by `Configuration`. Throughout this article, the names of FEASST classes, arguments and plugins use `this typeset`. Because we cannot assume a constant number of particles, a `Configuration` object is defined both by the number and types of particles that may exist and by the particles that currently exist. Particles that may exist are defined by two separate `ParticleFactory`, one for unique sites and the other for unique particles. A third `ParticleFactory` defines the current particles that exist, but their type and bond information is not duplicated. `Configuration` contains the spatial `Domain` in which particles reside, as well as the `NeighborCriteria` to define neighbors. Each `Particle` contains a collection of sites, and each `Site` contains a `Position` and, optionally, an orientation. A `Particle` may also have many

Bond, Angle or Dihedral objects.

B. The System class

The System class shown in Fig. 13 includes three different PotentialFactory objects. The first is the unoptimized potential, the second is the optimized potential, and the third are reference potentials. Infrequently, the slow unoptimized potential may be compared to the optimized one, while reference configurations may be used for specialized techniques such as MayerSampling and DCCB.[33] Optimized potentials may refer to those that use cell lists, neighbor lists, or a specialized single-purpose code for a particular system (e.g., as demonstrated for a simulation of Lennard-Jones in cuboid PBCs in the opt_lj plugin). A System may also contain a number of Configuration objects, and a BondVisitor, which computes bonded interactions, and ThermoParams, which contains thermodynamic parameters such as temperature, pressure and chemical potential.

PotentialFactory contains a number of Potential objects. A Potential includes both a Model and VisitModel using the visitor design pattern.[62] A Model defines one- or two-body interactions between sites, while a VisitModel sums the interaction energies between a Select part of the Configuration, or the entire Configuration. Model may include one-body interactions, such as confinement, or two-body interactions such as LennardJones. The LennardJones class is derived from Model and ModelTwoBody. This object-oriented design allows derived models such as LennardJones to be easily replaced with other models, or customized without affecting the rest of the code (e.g., modular and extensible). All derived classes can be extended by users as detailed in Section VI B, where base class functions such as energy, which depends on distance and types, can be overridden by the derived class to return a different expression.

VisitModel also has a variety of derived classes, including Ewald, LongRangeCorrections, and VisitModelCell, for cell-lists or intramolecular interactions with VisitModelIntra. VisitModelInner allows extensions to anisotropic interactions, either through Kern-Frenkel patches with VisitModelInnerPatch or through tabular anisotropic potentials with VisitModelInnerTable. VisitModellInner also contains EnergyMap, which stores interaction energies, separation distances, and PBC information in order to support the use of neighbor-lists, rigid cluster translations and rotations, and aggregation volume bias.[35] A few different kinds of EnergyMap include storing the interactions of all particles with EnergyMapAll, or only interacting neighbors with EnergyMapNeighbor, or only store interactions subject to additional criteria such as separation distance or site types.

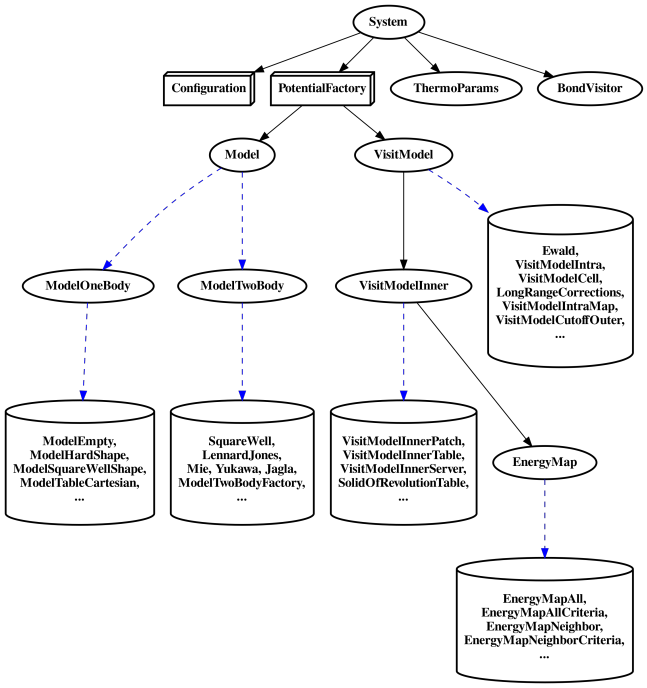

FIG. 13. Flowchart for the System class with symbols and shapes as described in Fig. 12. Classes connected by the blue dashed arrows indicate the base and derived classes, respectively, and cylinders represent many options for derived class.

C. The Trial class

The Trial class is illustrated in Fig. 14. A Trial is performed with a number of TrialStage objects, where each stage has both TrialSelect and Perturb. The Perturb class changes the System to calculate the trial Acceptance while retaining enough information to finalize the Trial if it is accepted, or revert the Trial if it is rejected.

Perturb may utilize a Tunable parameter for a target acceptance. PerturbMove is for Perturb that only changes the positions of the sites. These may include PerturbTranslation, PerturbRotation or more complex perturbations. Other Perturb classes include PerturbAdd and PerturbRemove for μVT ensemble trials, as well as PerturbVolume for the isothermal-isobaric ensemble.

Because MC trials often have 25% or less acceptance probability, for optimization,[97] attempts should perform as little change to the old state as necessary to compute the Acceptance, and only implement the remaining changes after the trial has been accepted. Any changes are reverted upon trial rejection. Furthermore, to enable parallel Prefetch,[57] all these changes are synchronized between processors.

The TrialSelect class involves the selection of the particles or sites that are subject to Perturb. These may involve the selection of particles, specific bonds, angles or dihedrals within the particles. Each TrialStage

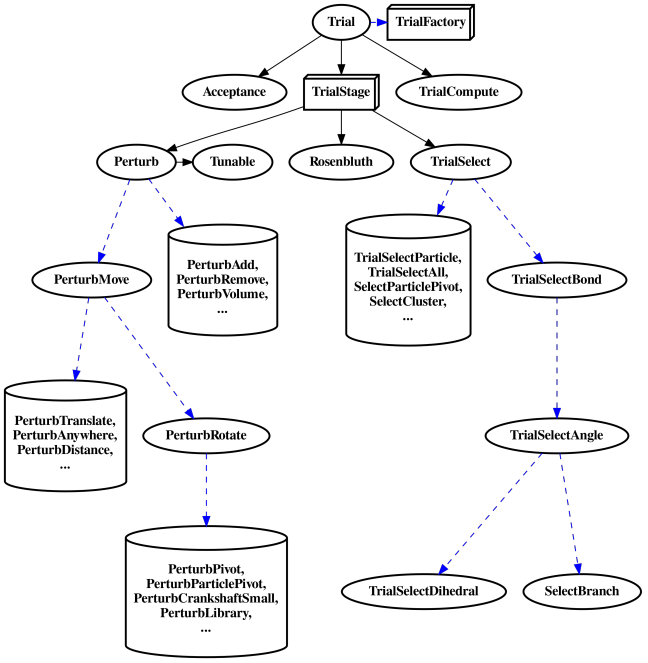

FIG. 14. Flowchart for the `Trial` class with symbols and shapes as described in Fig. 13.

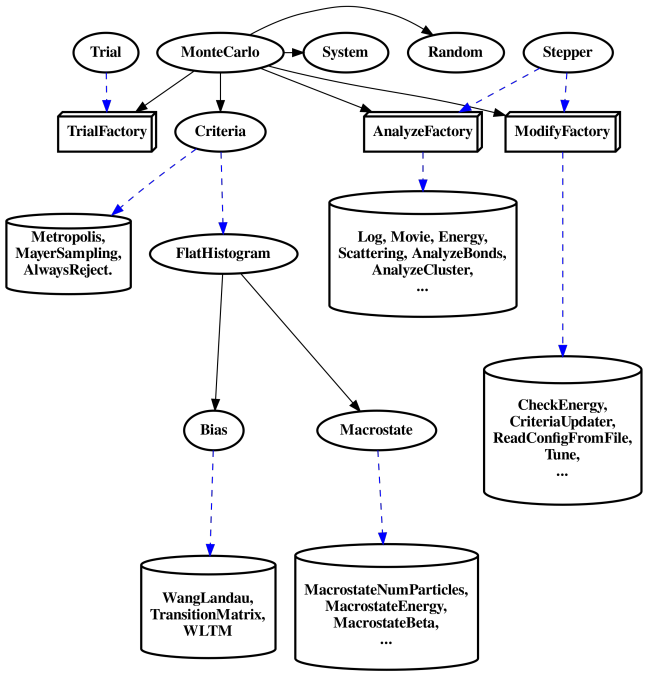

FIG. 15. Flowchart for the `MonteCarlo` class with symbols and shapes as described in Fig. 13.

may be selected using `Rosenbluth`-weighted probabilities. While each `TrialStage` is performed in sequence, the `Acceptance` class records changes to the `System` such as the energy differences of the old and new states. Finally, `TrialCompute` obtains the overall `Trial` acceptance probability.

D. The MonteCarlo class

The `MonteCarlo` class shown in Fig. 15 contains a number of `Trial`, `Analyze` and `Modify` objects as well as a `Random` number generator, the `System` and the acceptance `Criteria`. `Criteria` is another base class from which `Metropolis`, `FlatHistogram` or `MayerSampling` are derived, which allows the `MonteCarlo` class to interchangeably utilize these different methodologies. The `Criteria` function `is_accepted` is overridden by the derived classes to modify how `Trial` attempts are accepted or rejected, in a similar fashion to how `LennardJones` overrides `ModelTwoBody` as discussed in Section IV B.

Further object-oriented design is utilized in `FlatHistogram`, which contains both `Macrostate` definitions, and a `Bias` procedure. Possible `Bias` includes `WangLandau`, `TransitionMatrix` or both, `WLTM`, while `Macrostate` may be the number of particles, energy, the inverse temperature or other possibilities.

`Analyze` or `Modify` classes may be performed after a given number of trials. While `Analyze` does not change the `System`, `TrialFactory` or `Criteria`, `Modify` classes may. These may be important to distinguish especially in parallel `Prefetch`. Example `Analyze` classes include printing energies and trial acceptance probabilities with `Log`, trajectories with `Movie` or analysis of the ensemble average `Energy`. Example `Modify` classes include `Tune` for target acceptance probability or `CheckEnergy` to ensure that the summation of the change in energy does not drift due to numerical precision, and to catch errors in optimized energy evaluations.

E. Design components of a FEASST class

FEASST classes have the following components if they are present in the text user interface, or are derived by those classes. A class constructor takes input argument pairs in the keyword and value syntax described in Section III B. These input arguments are documented with default values, if applicable. Derived classes may also utilize these input arguments. Public functions hide private data.

Checkpointing is enabled by serialization as described by the Standard C++ Organization.[98] Serialization is the conversion of software objects in memory to characters in files, while deserialization is the reverse. The member data of each FEASST class are serialized and deserialized in the same order, starting with a class name if a derived or base class, followed by a unique serialization version for each object, and then serialization of all member data in a specific order that is mirrored in deserialization. Base and derived classes use static maps with a factory design pattern,[62] enabling text interface creation of new objects via class names.

V. GUIDELINES FOR PERFORMING AND TROUBLESHOOTING FEASST SIMULATIONS

First, we recommend verifying that the potential energy is computed as expected based on calculations by another trusted software or researcher by comparison with a reference configuration, as demonstrated in Section III D. Second, find a tutorial that is the closest to what needs to be accomplished, and modify it as necessary based on the Text Interface section of the online documentation. A summary of tutorials is provided in Section II K. Third, reproduce an expected, published or trusted result before simulating models or thermodynamic conditions that have never been performed before. Fourth, run a series of short simulations to optimize simulation parameters, trial weights, and file output frequencies. Checkpoint and restart with a short simulation on a test queue before starting a long simulation that consumes HPC resources.

Users and developers may be found on the GitHub issue tracker,[99] and email list,[100] where users may benefit from previously answered questions. Although any and all reports are appreciated, questions may be easier to address and more quickly answered with the following guidelines as described below.

The first suggestion for preparing an issue is to reproduce the issue with a minimal example that reduces the complexity, run time, and file sizes. This allows others to quickly reproduce and debug the issue. For example, if an issue occurs when using multiple potentials, trials or analysis, remove one at a time until the minimal number is found that still gives the same issue. If the error only occurs after a long simulation, see if the issue may be reproduced more quickly after increasing the frequency of `CheckEnergy` or reducing the number of particles. Another option is to start the simulation from an initial structure. Ideally, the issue may be reproduced in a few seconds on a single processor.

The second suggestion is to minimize the number of necessary files, and lines in files, required to reproduce the issue. The first file to include is the text interface file. In all the online tutorials, Python scripts generate text files that are input to the "fst" executable. This generated text file is often less complex than the Python script. The text file may also reference other files required to reproduce the issue, such as those with the "fstprt" file extension that describe particles. In some cases, CB files read by `TrialGrowFile` may be needed, as well as `TablePotential`, `ShapeFile` and initial XYZ configurations.

After simplifying these files, verify that the same issue may be reproduced using the command

```
$HOME/feasst/build/bin/fst < script.txt
```

where script.txt is the FEASST input file.

The third suggestion is to provide the FEASST version in the description of the issue. The version may be obtained with the command

```
git describe --abbrev=10 --dirty --always
--tags
```

or by providing the git commit from the command "git log." For issues during compilation, include the output of the "cmake .." command from Text 1 starting from an empty build directory.

The fourth suggestion is to provide a backtrace. This can be done with the GNU Debugger[101] (GDB) using the command

```
gdb $HOME/feasst/build/bin/fst
```

followed by the GDB commands "catch throw" for assertion errors, then "r script.txt" and finally "bt" for the backtrace. If the error involves memory issues or a segmentation fault, it may also help to run the simulation in Valgrind[102] using the command

```
valgrind $HOME/feasst/build/bin/fst <
text.txt > text.log 2>&1
```

although Valgrind slows down the simulation.

VI. ALTERNATIVE INTERFACES, CUSTOMIZATION AND TESTING

Here, we discuss the specialized usage of FEASST with alternative APIs, customized classes and plugins, and automation of tests.

A. Interfaces for C++, Python and client-server

The recommended interface for most FEASST use cases is the text interface described in the Secs. II-V of this article. Other alternatives are C++ or Python libraries, text input via a Python module, and client-server mode. The main benefits of the text interface are as follows. Input files may be generated using a scripting language for convenience, and the resulting input file is often easier to test and share with others. Another benefit of the text interface is restarting. The input file is stored in its entirety before each line is executed in sequence. Thus, when a simulation is checkpointed and restarted, the lines that have not finished or have not been performed are executed. In comparison, using FEASST as a library in Python or C++ requires the user to know how far the code proceeded before it was terminated.

There are two types of Python interfaces currently available. The first is in active development and uses pybind11 to expose the text interface commands to Python, as described in the Python Interface section of the online documentation. This allows tasks in Python between

trials, such as additional analysis, checking if the simulation is complete or detecting that an HPC queue will soon preempt the job so that a clean termination and restart are possible. An example is found in "$HOME-/feasst/python/tutorial/test.py."

The second Python interface is documented by a link at the end of the online Python interface section and is not recommended for most applications. This Python interface uses SWIG to expose the entire C++ API to Python. The FEASST SWIG interface has also been difficult for many users to compile using the command

```
cmake -DUSE_SWIG=ON ..
```

during compilation. An example is found in "$HOME-/feasst/tutorial/library/tutorial.py." The SWIG Python interface is not recommended because it is prone to user errors, is difficult to maintain, and may be deprecated in future versions of FEASST.

Another alternative interface is to use FEASST as a library in C++. In this case, nearly all aspects of FEASST are available to the user. The C++ interface is the most flexible because the core of FEASST is written in C++. An example is found in "$HOME/feasst/tutorial/library/tutorial.cpp" and "$HOME/feasst/tutorial/library/CMakeLists.cpp."

There are also specialized client-server mode and MPI options to run FEASST as a server with C++ or Python clients. For example, the classes `ModelServer` and `ModelMPI` query a client for the interaction energy as a function of distance and type. Server-client communication in this way is slow compared to compiled C++ code for Lennard-Jones, and is therefore not recommended except for models that are much more computationally expensive than Lennard-Jones, such as quantum-mechanical or machine-learned potentials. This server-client method is demonstrated in both "$HOME/feasst/plugin/server/tutorial/launch_2_model_server.py" and "$HOME/feasst/plugin/mpi/tutorial/launch_1_model_mpi.py." In comparison with the server example, the MPI example may communicate more quickly but also uses more CPU cycles simply waiting for a response. These methods are available in the server and mpi plugins.

Another use of the client-server interface is for Python clients to give FEASST servers text interface commands line-by-line. This is demonstrated in "$HOME/feasst/-plugin/server/tutorial/tutorial_0_server.ipynb." Another example is given in "$HOME/feasst/plugin/server/tutorial/launch_1_custom_terminate.py" where the client sends a `Run` command to the FEASST server to attempt trial moves until a desired condition is met (e.g., elapsed time). One of the convenient aspects of this kind of client-server approach to running FEASST simulations is that control is given to the client in between intervals of fixed numbers of trials, while the FEASST simulation is still held in memory, which can be used to analyze simulations, as well as interact with the simulations using customized `Action` classes. Although interoperability with other software, such as i-PI,[103] CSLib[104] and MolSSI MDI,[105] is not currently implemented, the FEASST developers hope to better establish such interfaces to simulation engines and machine-learning potentials in future work.

TEXT 7. Command line copy and renaming of an existing FEASST class allows implementation of new potential models without affecting the original FEASST source code.

```
cd $HOME/feasst/plugin/example
sed "s/MODEL_EXAMPLE/NEW_NAME/g"
    include/model_example.h > include/new_name.h
sed "s/model_example\.h/new_name.h/g"
    src/model_example.cpp > src/new_name.cpp
sed -i "s/ModelExample/NewName/g"
    include/new_name.h src/new_name.cpp
```

B. Customizing and extending FEASST classes and plugins

Custom classes and plugins in FEASST are easily created by copying an existing one, renaming and then modifying, as inspired by the modular design of LAMMPS classes and packages. To illustrate the ease of extending FEASST, the following four commands shown in Text 7 are all that is required to create a new `ModelTwoBody` interaction, `NewName`, that is usable in the text input file, after re-compilation shown in Text 1, and may then be further modified without affecting any of the original FEASST source code. This allows for the development of new features without affecting the existing features. The same copy and rename strategy may be applied to any FEASST classes derived from `Random`, `Formula`, `Minimize`, `Solver`, `Shape`, `Model`, `ModelParam`, `PhysicalConstants`, `VisitModel`, `VisitModelInner`, `EnergyMap`, `Bond(Two/Three/Four)Body`, `Analyze`, `Modify`, `Action`, `Constraint`, `TrialSelect`, `Perturb`, `TrialCompute`, `Trial`, `Criteria`, `Macrostate` and `Bias`.

When creating a custom class, first a developer would find an existing class that is related to the new feature. Second, the developer may copy a class header and implementation file, with file extensions ".h" and ".cpp," while renaming these files. Third, the class name in both the header and implementation file is changed, along with the header guards in all capitals and the header include. Finally, FEASST should be recompiled, and optionally the tests and serialization version should change. In a similar fashion, new plugins may also be created by copying an existing plugin and renaming each class. Before recompiling, the plugin should also be added to the "FEASST_PLUGINS" variable in "$HOME/feasst/CMakeLists.txt" as well as one of the new classes added to the "$HOME/feasst/py/depend.py" file. This process is further described in the online documentation section titled

Modify FEASST.

Debugging output can be achieved by setting the VERBOSE_LEVEL variable in the "$HOME/feasst/utils/include/debug.h" file to the desired level and output and test with the following provided macros: FATAL, ERROR, ASSERT, WARN, INFO, DEBUG and TRACE. Examples of the usage of these debugging macros are also provided in the example plugin, along with a description of obtaining user-input arguments and serialization.

C. Test-driven development strategy

FEASST development follows the test-driven strategy. In this strategy, new features begin with a test built around an interface before actual implementation of the feature. This allows developers to first consider how the required information is obtained from the user or other classes. A series of tests are then designed around this interface. Finally, the developer implements the feature and debugs with the automated tests. The automated tests are maintained to test future changes. The benefit of this strategy is an increased focus on testing and design of the user interface, and the cost of this strategy is that the tests increase the number of lines of code that need to be written and maintained. Developers must balance the number of tests against the complexity of the new feature, where there is a trade-off between tests that quickly identify an error and tests that are a burden to maintain.

The GoogleTest C++ unit tests[106] (GTEST) may be compiled by defining the GTEST variable of CMake in line 7 of Text 1 with the command

```
cmake -DUSE_GTEST=ON ..
```

to create the executable "$HOME/feasst/build/bin/unittest." Run only the fast unit tests with the command

```
$HOME/feasst/build/bin/unittest
--gtest_filter=-*LONG
```

to quickly search for possible issues. The unit tests with a suffix of "LONG" if they take more than a fraction of a second and "VERY_LONG" if they take more than a minute. The GTEST filter may also be used to only run tests associated with a specific feature to speed development.

Most Python tutorials in FEASST include a post-process step at the end of the simulation to analyze the final result and compare against published values, if available. Every tutorial in the develop branch is automatically run weekly on an HPC and any errors during the simulation or at the post-processing step are reported to the developers. This helps ensure changes to FEASST do not disrupt previously developed tutorials.

Development of FEASST is closely connected to validated, reference-quality results of molecular simulation provided in the NIST SRSW.[63] The intention behind this close connection between the NIST SRSW and open-source FEASST is to establish traceability between validated results or ground-truth calculations and the computational measurement conducted by FEASST, akin to the traceability chain that links reference materials to the International System of Units. For example, FEASST accurately reproduces the energy of reference configurations provided in the SRSW, which is demonstrated in a tutorial (e.g., see Sec III D) that is regularly re-tested (see above); this is an example of FEASST reproducing a ground-truth calculation since the energy of a particular configuration can be reproduced outside of actual molecular simulation software. A second example is the tutorial in the FEASST documentation that reproduces the average energy of a canonical ensemble of LJ particles at specified density and pressure; this measurement, with associated uncertainty that arises from randomness in its Markov Chain, is compared against MC results from other validated software to confirm the reliability of FEASST for the particular application. Such comparisons rely on satisfactory description of the simulation results (ensemble parameters, system size, forcefield parameters, etc.) so that the simulation can be faithfully reproduced in, for this case, FEASST. The NIST SRSW offers examples of such simulation descriptions and associated results that provide the basis for tests that can be used to validate the new features of FEASST (or other simulation software). We encourage FEASST developers to follow this pattern so that new features are validated against and traceably linked to trusted results.

D. Open license and "as is" disclaimer

FEASST users should not assume that results from FEASST are correct without any testing. Parameters may not have been input correctly, or there could be an error or issue with the particular kind of simulation. There are not enough FEASST developers and available computer resources to test every possible combination of methods and parameters each time the software is modified. See Section V for more guidelines on troubleshooting FEASST simulations.

The full license of FEASST is provided in the online documentation and GitHub repository. To approximately summarize, FEASST is currently provided "as is" by NIST as a public service. NIST cannot be held liable for any damages arising from the use of FEASST. Use of FEASST, improvements, modifications and derivative works should appropriately acknowledge NIST.

purpose.

VII. CONCLUSIONS

We highlight some of the unique features of the open source MC simulation software called FEASST, which has been used previously to study phase behavior, self-assembly, aggregation and gelation of biological materials, colloids, polymers, ionic liquids and adsorption in porous networks. FEASST is parallelized with prefetching and flat-histogram methods. The complete source code and documentation for version 0.25.1 of FEASST are available in the Supplemental Online Material, while more up-to-date information may be found at the `https://pages.nist.gov/feasst` website.[16]

The Free Energy and Advanced Sampling Simulation Toolkit (FEASST) supports many flat-histogram methods, which is why we chose the "Free Energy" part of the name. "Advanced Sampling" refers to the specialized trials available, such as CB. "Toolkit" references the customizable and extensible modular design that supports user-created plugins. Altogether, the acronym hints at a veritable feast of MC methods, where the second "S" helps with search engine optimization. The logo forms an "F" from a two-state free energy shown in blue and a flat-histogram shown in red.

VIII. SUPPLEMENTAL ONLINE MATERIAL

The Supplemental Online Material contains the following:

- feasst-0.25.1.zip and feasst-html-0.25.1.zip contains the complete source code and HTML documentation, respectively, of the specific version of FEASST described in this article. See `https://doi.org/10.18434/M3S095` for the most up-to-date information on FEASST.
- fig_data.zip contains the data for Figs. 1 to 10 in the comma-separated value format.
- script.txt contains Text 2 for convenience without the line numbers.
- lj_ref_config.xyz contains the entire configuration illustrated in Text 4.

[1] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, J. Chem. Phys. **21**, 1087 (1953).
[2] A. Gupta, S. Chempath, M. J. Sanborn, L. A. Clark, and R. Q. Snurr, Mol. Simul. **29**, 29 (2003).
[3] W. L. Jorgensen and J. Tirado–Rives, J. Comput. Chem. **26**, 1689 (2005).
[4] M. G. Martin, Mol. Simul. **39**, 1212 (2013).
[5] A. J. Schultz and D. A. Kofke, J. Comput. Chem. **36**, 573 (2015).
[6] A. V. Brukhno, J. Grant, T. L. Underwood, K. Stratford, S. C. Parker, J. A. Purton, and N. B. Wilding, Mol. Simul. **47**, 131 (2021).
[7] J. K. Shah, E. Marin-Rimoldi, R. G. Mullen, B. P. Keene, S. Khan, A. S. Paluch, N. Rai, L. L. Romanielo, T. W. Rosch, B. Yoo, and E. J. Maginn, J. Comput. Chem. **38**, 1727 (2017).
[8] M. Lund, M. Trulsson, and B. Persson, Source Code Biol. Med. **3**, 1 (2008).
[9] D. Dubbeldam, S. Calero, D. E. Ellis, and R. Q. Snurr, Mol. Simul. **42**, 81 (2016).
[10] Y. Nejahi, M. Soroush Barhaghi, J. Mick, B. Jackman, K. Rushaidat, Y. Li, L. Schwiebert, and J. Potoff, SoftwareX **9**, 20 (2019).
[11] R. Hens, A. Rahbari, S. Caro-Ortiz, N. Dawass, M. Erdős, A. Poursaeidesfahani, H. S. Salehi, A. T. Celebi, M. Ramdin, O. A. Moultos, D. Dubbeldam, and T. J. H. Vlugt, J. Chem. Inf. Model. **60**, 2678 (2020).
[12] M. Penaloza-Amion, E. Sedghamiz, M. Kozlowska, C. Degitz, C. Possel, and W. Wenzel, Front. Phys. **9**, 635959 (2021).
[13] S. Deublein, B. Eckl, J. Stoll, S. V. Lishchuk, G. Guevara-Carrion, C. W. Glass, T. Merker, M. Bernreuther, H. Hasse, and J. Vrabec, Comput. Phys. Commun. **182**, 2350 (2011).
[14] H. J. Limbach, A. Arnold, B. A. Mann, and C. Holm, Comput. Phys. Commun. **174**, 704 (2006).
[15] Y. Sun, R. F. DeJaco, and J. I. Siepmann, Chemical Science **10**, 4377 (2019).
[16] "FEASST Website," `https://doi.org/10.18434/M3S095` (2024).
[17] H. W. Hatch, N. A. Mahynski, and V. K. Shen, J. Res. Natl. Inst. Stan **123**, 123004 (2018).
[18] L. Talirz, L. M. Ghiringhelli, and B. Smit, Living J. Comput. Mol. Sci. **3**, 1483 (2021).
[19] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, SoftwareX **1-2**, 19 (2015).
[20] S. Plimpton, J. Comput. Phys. **117**, 1 (1995).
[21] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, Comp. Phys. Comm. **271**, 108171 (2022).
[22] J. K. Singh and D. A. Kofke, Phys. Rev. Lett. **92**, 220601 (2004).
[23] F. Wang and D. P. Landau, Phys. Rev. Lett. **86**, 2050 (2001).
[24] J. R. Errington and V. K. Shen, J. Chem. Phys. **123**, 164103 (2005).
[25] A. Z. Panagiotopoulos, Mol. Phys. **61**, 813 (1987).
[26] A. P. Lyubartsev, A. A. Martsinovski, S. V. Shevkunov, and P. N. Vorontsov-Velyaminov, J. Chem. Phys. **96**, 1776 (1992).
[27] N. Kern and D. Frenkel, J. Chem. Phys. **118**, 9882 (2003).
[28] R. B. A. Ole Bird, *Dynamics of Polymeric Liquids. Volume 1: Fluid Mechanics*, first edition ed. (John Wiley, New York, 1977).
[29] M. G. Martin and J. I. Siepmann, J. Phys. Chem. B **102**, 2569 (1998).
[30] J.-P. Ryckaert and A. Bellemans, Faraday Discuss. **66**, 95 (1978).
[31] I.-C. Yeh and M. L. Berkowitz, J. Chem. Phys. **111**, 3155 (1999).
[32] J. I. Siepmann and D. Frenkel, Mol. Phys. **75**, 59 (1992).
[33] T. J. H. Vlugt, M. G. Martin, B. Smit, J. I. Siepmann, and R. Krishna, Mol. Phys. **94**, 727 (1998).
[34] B. Chen and J. I. Siepmann, J. Phys. Chem. B **104**, 8725 (2000).
[35] B. Chen and J. I. Siepmann, J. Phys. Chem. B **105**, 11275 (2001).
[36] A. Sepehri, T. D. Loeffler, and B. Chen, J. Chem. Theory Comput. **13**, 1577 (2017).
[37] H. W. Hatch, J. Mittal, and V. K. Shen, J. Chem. Phys. **142**, 164901 (2015).
[38] H. W. Hatch, S.-Y. Yang, J. Mittal, and V. K. Shen, Soft Matter **12**, 4170 (2016).
[39] H. W. Hatch, W. P. Krekelberg, S. D. Hudson, and V. K. Shen, J. Chem. Phys. **144**, 194902 (2016).
[40] N. A. Mahynski, H. Zerze, H. W. Hatch, V. K. Shen, and J. Mittal, Soft Matter **13**, 5397 (2017).
[41] E. Pretti, H. Zerze, M. Song, Y. Ding, N. A. Mahynski, H. W. Hatch, V. K. Shen, and J. Mittal, Soft Matter **14**, 6303 (2018).

[42] H. W. Hatch, N. A. Mahynski, R. P. Murphy, M. A. Blanco, and V. K. Shen, AIP Advances **8**, 095210 (2018).
[43] R. P. Murphy, H. W. Hatch, N. A. Mahynski, V. K. Shen, and N. J. Wagner, Soft Matter **16**, 1279 (2020).
[44] H. W. Hatch and G. W. McCann, J. Res. Natl. Inst. Stand. Technol. **124**, 1 (2019).
[45] T. A. Maula, H. W. Hatch, V. K. Shen, S. Rangarajan, and J. Mittal, Mol. Syst. Des. Eng. **4**, 644 (2019).
[46] C. Rzepa, D. W. Siderius, H. W. Hatch, V. K. Shen, S. Rangarajan, and J. Mittal, J. Phys. Chem. C **124**, 16350 (2020).
[47] D. W. Siderius, H. W. Hatch, and V. K. Shen, J. Phys. Chem. B **126**, 7999 (2022).
[48] D. W. Siderius, H. W. Hatch, and V. K. Shen, J. Phys. Chem. B (2024), 10.1021/acs.jpcb.4c00753.
[49] M. A. Blanco, H. W. Hatch, J. E. Curtis, and V. K. Shen, J. Pharm. Sci. **108**, 1663 (2019).
[50] H. W. Hatch, S. W. Hall, J. R. Errington, and V. K. Shen, J. Chem. Phys. **151**, 144109 (2019).
[51] N. A. Mahynski, H. W. Hatch, M. Witman, D. A. Sheen, J. R. Errington, and V. K. Shen, Mol. Simul. **47**, 395 (2021).
[52] N. A. Mahynski, S. Jiao, H. W. Hatch, M. A. Blanco, and V. K. Shen, J. Chem. Phys. **148**, 194105 (2018).
[53] H. W. Hatch, S. Jiao, N. A. Mahynski, M. A. Blanco, and V. K. Shen, J. Chem. Phys. **147**, 231102 (2017).
[54] J. I. Monroe, W. P. Krekelberg, A. McDannald, and V. K. Shen, J. Chem. Phys. **158**, 164110 (2023).
[55] H. W. Hatch, D. W. Siderius, J. R. Errington, and V. K. Shen, J. Phys. Chem. B **127**, 3041 (2023).
[56] D. W. Siderius, H. W. Hatch, J. R. Errington, and V. K. Shen, AIChE J. **68**, e17686 (2022).
[57] H. W. Hatch, J. Phys. Chem. A **124**, 7191 (2020).
[58] M. S. Shell, P. G. Debenedetti, and A. Z. Panagiotopoulos, J. Chem. Phys. **119**, 9406 (2003).
[59] V. K. Shen and D. W. Siderius, J. Chem. Phys. **140**, 244106 (2014).
[60] K. S. Rane, S. Murali, and J. R. Errington, J. Chem. Theory Comput. **9**, 2552 (2013).
[61] K. R. S. Shaul, A. J. Schultz, and D. A. Kofke, J. Chem. Phys. **135**, 124101 (2011).
[62] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and G. Booch, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. (Addison-Wesley Professional, Reading, Mass, 1994).
[63] V. K. Shen, D. W. Siderius, W. P. Krekelberg, and H. W. Hatch, *NIST Standard Reference Simulation Website* (NIST Standard Reference Database Number 173, National Institute of Standards and Technology, Gaithersburg, MD, 2024).
[64] B. Chen, J. I. Siepmann, K. J. Oh, and M. L. Klein, J. Chem. Phys. **116**, 4317 (2002).
[65] L. G. MacDowell, V. K. Shen, and J. R. Errington, J. Chem. Phys. **125**, 034705 (2006).
[66] H. J. C. Berendsen, J. R. Grigera, and T. P. Straatsma, J. Phys. Chem. **91**, 6269 (1987).
[67] A. Giacometti, F. Lado, J. Largo, G. Pastore, and F. Sciortino, J. Chem. Phys. **132**, 174110 (2010).
[68] J. D. Weeks, D. Chandler, and H. C. Andersen, J. Chem. Phys. **54**, 5237 (1971).
[69] D. J. Kraft, R. Ni, F. Smallenburg, M. Hermes, K. Yoon, D. A. Weitz, A. v. Blaaderen, J. Groenewold, M. Dijkstra, and W. K. Kegel, Proc. Natl. Acad. Sci. USA **109**, 10787 (2012).
[70] J. R. Wolters, G. Avvisati, F. Hagemans, T. Vissers, D. J. Kraft, M. Dijkstra, and W. K. Kegel, Soft Matter **11**, 1067 (2015).
[71] V. I. Harismiadis, J. Vorholz, and A. Z. Panagiotopoulos, J. Chem. Phys. **105**, 8469 (1996).
[72] "GitHub Repo: mayer-extrapolation," `https://github.com/usnistgov/mayer-extrapolation` (2024).
[73] S. Qin and H.-X. Zhou, J. Phys. Chem. B **123**, 8203 (2019).
[74] A. C. M. Young, J. C. Dewan, C. Nave, and R. F. Tilton, J. Appl. Crystallogr. **26**, 309 (1993).
[75] T. J. Dolinsky, J. E. Nielsen, J. A. McCammon, and N. A. Baker, Nucleic Acids Res. **32**, W665 (2004).
[76] D. Sitkoff, K. A. Sharp, and B. Honig, J. Phys. Chem. **98**, 1978 (1994).
[77] M. H. M. Olsson, C. R. Søndergaard, M. Rostkowski, and J. H. Jensen, J. Chem. Theory Comput. **7**, 525 (2011).
[78] R. Huey, G. M. Morris, A. J. Olson, and D. S. Goodsell, J. Comput. Chem. **28**, 1145 (2007).
[79] H. W. Hatch, C. Bergonzo, M. A. M. Blanco, G. Yuan, S. Grudinin, M. Lund, J. E. Curtis, A. V. Grishaev, Y. Liu, and V. K. Shen, J. Chem. Phys. (2024), https://doi.org/10.1063/5.0224809.
[80] J. J. Potoff and J. I. Siepmann, AIChE J. **47**, 1676 (2001).
[81] J. Pérez-Pellitero, H. Amrouche, F. R. Siperstein, G. Pirngruber, C. Nieto-Draghi, G. Chaplais, A. Simon-Masseron, D. Bazer-Bachi, D. Peralta, and N. Bats, Chem. Eur. J. **16**, 1560 (2010).
[82] D. W. Siderius and V. K. Shen, J. Phys. Chem. C **117**, 5861 (2013).
[83] B. Larsen, J. Chem. Phys. **65**, 3431 (1976).
[84] J. G. Harris and K. H. Yung, J. Phys. Chem. **99**, 12021 (1995).
[85] H. Flyvbjerg and H. G. Petersen, J. Chem. Phys. **91**, 461 (1989).
[86] "GCC, the GNU Compiler Collection," `https://gcc.gnu.org/` (2024).
[87] "CMake: A Powerful Software Build System," `https://cmake.org/` (2024).
[88] "git," `https://git-scm.com/` (2024).
[89] "Python," `https://www.python.org/` (2024).
[90] "FEASST GitHub tags," `https://github.com/usnistgov/feasst/tags` (2024).
[91] "NIST Standard Reference Simulation Website: Lennard-Jones Fluid: NVT Monte Carlo," `https://mmlapps.nist.gov/srs/LJ_PURE/mc.htm` (2024).
[92] M. Matsumoto and T. Nishimura, ACM Trans. Model. Comput. Simul. **8**, 3 (1998).
[93] M. P. Allen and D. J. Tildesley, *Computer simulation of liquids* (Clarendon Press, 1989).
[94] D. Frenkel and B. Smit, *Understanding Molecular Simulation: From Algorithms to Applications* (Academic Press, 2002).
[95] W. Humphrey, A. Dalke, and K. Schulten, J. Mol. Graph. **14**, 33 (1996).
[96] A. Grossfield, P. N. Patrone, D. R. Roe, A. J. Schultz, D. W. Siderius, and D. M. Zuckerman, Living J. Comput. Molec. Sci. **1**, 5067 (2019).
[97] R. D. Mountain and D. Thirumalai, Physica A Stat. Mech. Appl. **210**, 453 (1994).
[98] "The Standard C++ Foundation: Serialization and Unserialization," `https://isocpp.org/wiki/faq/serialization` (2024).
[99] "FEASST GitHub issue tracker," `https://github.com/usnistgov/feasst/issues` (2024).
[100] "FEASST Google Group mail list," `https://list.nist.gov/feasst` (2024).
[101] "GDB: The GNU Project Debugger," `https://sourceware.org/gdb/` (2024).
[102] "Valgrind," `https://valgrind.org/` (2024).
[103] V. Kapil, M. Rossi, O. Marsalek, R. Petraglia, Y. Litman, T. Spura, B. Cheng, A. Cuzzocrea, R. H. Meißner, D. M. Wilkins, B. A. Helfrecht, P. Juda, S. P. Bienvenue, W. Fang, J. Kessler, I. Poltavsky, S. Vandenbrande, J. Wieme, C. Corminboeuf, T. D. Kühne, D. E. Manolopoulos, T. E. Markland, J. O. Richardson, A. Tkatchenko, G. A. Tribello, V. Van Speybroeck, and M. Ceriotti, Computer Physics Communications **236**, 214 (2019).
[104] "CSlib, a client/server messaging library for coupling scientific applications," `http://cslib.sandia.gov` (2024).
[105] T. A. Barnes, S. Ellis, J. Chen, S. J. Plimpton, and J. A. Nash, The Journal of Chemical Physics **160**, 214114 (2024).
[106] "GoogleTest," `https://github.com/google/googletest` (2024).