Cummings Peter (Orcid ID: 0000-0002-9766-2216)
Jankowski Eric (Orcid ID: 0000-0002-3267-1410)
Palmer Jeremy (Orcid ID: 0000-0003-0856-4743)
Siepmann Ilja (Orcid ID: 0000-0003-2534-4507)
Matsumoto Ray (Orcid ID: 0000-0002-9124-3512)

Open-Source Molecular Modeling Software in Chemical Engineering Focusing on the Molecular Simulation Design Framework

Authors:

Peter T. Cummings[1], Clare M[c]Cabe[1,2], Christopher R. Iacovella[1], Akos Ledeczi[3], Eric Jankowski[4], Arthi Jayaraman[5], Jeremy C. Palmer[6], Edward J. Maginn[7], Sharon C. Glotzer[8], Joshua A. Anderson[8], J. Ilja Siepmann[9,10], Jeffrey Potoff[11], Raymond A. Matsumoto[1], Justin B. Gilmer[12], Ryan S. DeFever[7], Ramanish Singh[9,10], Brad Crawford[11]

Affiliations

1. Department of Chemical and Biomolecular Engineering and Multiscale Modeling and Simulation Center, Vanderbilt University, Nashville, TN
2. Department of Chemistry, Vanderbilt University, Nashville, TN
3. Department of Electrical Engineering and Computer Science and Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN
4. Micron School of Materials Science and Engineering, Boise State University, Boise, ID
5. Departments of Chemical and Biomolecular Engineering and of Materials Science and Engineering, University of Delaware, Newark, DE
6. Department of Chemical and Biomolecular Engineering, University of Houston, Houston, TX
7. Department of Chemical and Biomolecular Engineering, University of Notre Dame, Notre Dame, IN
8. Departments of Chemical Engineering, of Materials Science, and of Physics, University of Michigan, Ann Arbor, MI
9. Department of Chemistry and Chemical Theory Center, University of Minnesota
10. Department of Chemical Engineering and Materials Science, University of Minnesota, MN
11. Department of Chemical Engineering and Materials Science, Wayne State University, Detroit, MI

This is the author manuscript accepted for publication and has undergone full peer review but has not been through the copyediting, typesetting, pagination and proofreading process, which may lead to differences between this version and the Version of Record. Please cite this article as doi: 10.1002/aic.17206

12. Interdisciplinary Graduate Program in Materials Science and Multiscale Modeling and Simulation Center, Vanderbilt University, Nashville, TN

1. Background

Molecular simulation has emerged as an important sub-field of chemical engineering, due in no small part to the leadership of Keith Gubbins. A characteristic of the chemical engineering molecular simulation community is the commitment to freely share simulation codes and other key software components required to perform a molecular simulation under open-source licenses and distribution on public repositories such as GitHub. Here we provide an overview of open-source molecular modeling software in Chemical Engineering, with focus on the Molecular Simulation Design Framework (MoSDeF). MoSDeF is an open-source Python software stack that enables facile use of multiple open-source molecular simulation engines, while at the same time ensuring maximum reproducibility.

Molecular simulation is a methodology for predicting the collective (in particular, thermodynamic and transport) properties of systems from information about how the molecules in the system interact with each other. That “information” can be obtained on-the-fly from quantum mechanics but, in most molecular simulations, it is encoded in a mathematical function, called a force field, that attempts to include all the intermolecular interactions between molecules (electrostatic interactions, van der Waals repulsive and attractive interactions) as well as intramolecular interactions (e.g., bond stretching, bond angle bending, and torsional interactions). More specifically, a force field is a representation of the total potential energy associated with interactions of all the atoms in the system (obtained by summing over all the molecules), which

can be differentiated with respect to the position of an atom to obtain the force exerted on that atom. Force fields can be derived from first principles calculations (e.g., quantum chemistry calculations) and/or experimental data; thus, generally force fields are semi-empirical. For inhomogenous systems (e.g., a fluid adsorbed on a surface or into a pore), the force field includes models for how the molecules interact with atoms in the surfaces or with an external field. Assuming that the molecular simulation runs long enough to attain equilibrium, and that the system is large enough or configured to eliminate unwanted surface effects (through so-called periodic boundary conditions), for a given force field, molecular simulation can provide essentially exact information about the properties of the system, obtained by averaging over the configurations generated in the simulation. Two major types of molecular simulations are routinely performed: molecular dynamics (MD), in which Newton's equations, or a convenient variation thereof, are solved for the dynamics of each atom in the system, and Monte Carlo (MC) simulation, in which configurations of the system are generated via a Markov chain process that asymptotically are distributed according to the appropriate equilibrium ensemble probability (e.g., for systems at constant molecule number N, volume V, and temperature T, the Boltzmann distribution, in which configurations have probability $\propto e^{-E/k_B T}$, where E is the energy of the system and k_B is Boltzmann's constant). In either case, the raw output of the simulation is configurations of the system (known as a trajectory) that can then be analyzed to compute properties. From an MD simulation, the trajectory will consist of positions and velocities for all atoms in the system over the course of the simulation; a typical MD simulation will employ a time step of 10^{-15}s, so that a 10-100 ns trajectory covers $10^7 - 10^8$ steps. For a 100,000-atom simulation (a typical system size with current computational resources), a trajectory file can be of the order of terabytes, so that statistical analysis of such files can be thought of as a particular kind of "big data" problem.

Molecular simulation began in the 1950s with simple systems such as hard spheres (MC[1] and MD[2,3]) and in the 1960s with the Lennard-Jones fluid (MC[4] and MD[5]). For such monatomic systems, the force field is very simple, specifying the interaction energy between spherically symmetric molecules. Beginning in the 1970s, molecular simulation was introduced to the field of chemical engineering primarily by Keith Gubbins, the honoree of this Founders issue of *AIChE Journal*. Keith is known and admired internationally and across many disciplines not only for his contributions in molecular theory (which have been seminal, such as Gray-Gubbins perturbation theory and the statistical associating fluid theory, or SAFT, equation of state) but also for his research in molecular simulation. One of the earliest Gubbins simulation papers[6] from 1979 has been cited almost 1000 times. [As an aside, his postdoctoral trainee co-author on this paper, Dominic Tildesley, was for many years a successful academic in the UK before joining Unilever, where he established one of the world's premier industrial molecular modeling groups, eventually rising to Vice President of Discovery Platforms; Tildesley also co-authored one of the seminal text books on molecular simulation[7].] Keith's influence on the field of chemical engineering in relation to molecular simulation can be measured in programming at AIChE Annual Meetings (which in the early 1980s had no sessions on molecular simulation in contrast to today when a whole programming area – the Computational Molecular Science and Engineering Forum, Area 21 – is largely focused on molecular simulation) and in papers presented at Properties and Phase Equilibria for Process and Product Design conference series established in 1977 (in which the first molecular simulation paper was presented in 1980, and by 2007 more than half the presentations involved molecular simulation and/or molecular theory). Since its early days, molecular simulation has become a workhorse in science and industry. The promise of being able to predict collective

properties from molecular interactions, and the attendant insight gained, have made molecular simulation (both MD and MC) an ideal and indispensable capability in materials science, biology, medicine (specifically, drug discovery) and engineering. There are commercial entities that market molecular simulation software (e.g., BIOVIA and Schrödinger). A 2002 international comparative study on molecular modeling (of which molecular simulation constitutes a major component) documented the widespread use of molecular modeling in industry, including many chemical, drug, and personal care product companies[8].

The authors of this Perspective article are all beneficiaries of the trail-blazing efforts of Keith Gubbins in establishing molecular simulation as an accepted and respected subfield of chemical engineering. Today, molecular simulation is taught in most chemical engineering departments in the U.S. at the graduate level, and is increasingly available as an elective at the undergraduate level or even offered as a first-year seminar to incoming undergraduate students. It has become one of the major focuses of the educational foundation, CAChE (Computer Aids for Chemical Engineering Education, cache.org), which established a molecular modeling task force in 1998. CACHE runs a highly successful technical conference, Foundations of Molecular Modeling and Simulation (fomms.org, held every three years since 2000) that has produced many educational resources to enable chemical engineers to teach and utilize molecular simulation in the classroom. In 2012, Keith Gubbins was awarded the FOMMS Medal for his numerous and long-standing contributions to the molecular simulation community. In addition to prodigious research contributions, he has authored seminal textbooks[9], including the two-volume definitive treatise on the theory of molecular fluids[10,11] that is an essential part of the library of any serious statistical mechanician interested in molecular fluids.

2. Development of molecular simulation tools in the chemical engineering community

Although molecular simulation (MD and MC) transcends disciplinary boundaries as noted above, chemical engineers have been particularly active in developing algorithms that compute properties of strong interest to the chemical engineering community (ChEC). One example is vapor-liquid phase equilibria, which is of enduring interest to the ChEC due to separation processes. Thus, a molecular simulation methodology for computing phase equilibrium directly and efficiently, the Gibbs ensemble MC (GEMC) algorithm, was developed in 1987 within the ChEC by Panagiotopoulos[12]. Phase equilibria can involve differences in densities between phases of several orders of magnitude; likewise, in chemical manufacturing there can be wide ranges of state conditions. Hence, along with the development of algorithms, the ChEC has also been at the forefront of developing force fields that are accurate over wide ranges of state conditions, such as the TraPPE family of force fields optimized for vapor-liquid equilibrium (see the extensive resources at http://trappe.oit.umn.edu) and the Gaussian charge polarizable model (GCPM) for water[13] that correctly predicts water's phase equilibria, thermodynamic, transport and dielectric properties over wide ranges of temperature and pressure. By contrast, much of the molecular simulation community in other disciplines is focused on properties at or near ambient conditions (including ambient conditions for biological systems).

Table 1 here

Table 1. Examples of open-source molecular simulation codes and related supporting utilities developed within the chemical engineering molecular modeling community. Website links

are to home pages of the codes or to code repositories. In addition, several other open-source codes emerging from the chemical engineering community are highlighted.

In this regard, the molecular simulation community in chemical engineering is particularly noted for sharing methods and capabilities by making software developed within the community freely available under open-source licenses, as described in a recent review article[30]. A recent, more general review of open-source molecular modeling software is provided by Pirhadi *et al.*[31] Table 1 provides examples of open-source molecular simulation tools developed within the ChEC, divided into simulation codes and other utilities. Similar to GEMC, many of these algorithms developed are primarily implemented within MC and hence it is not surprising that the bulk of open-source simulation engines developed within the ChEC (see Table 1) are for performing MC simulations. The need for community-developed simulation engines, whether they are MD or MC, stems from the fact that such codes have become increasingly difficult to develop, extend, and maintain for a single individual or single research group. This is due not only to an ever growing set of features and algorithms, but also due to changes in computing hardware utilized in a research environment: we have been through the era of vector architectures (e.g., Cray, Hitachi), parallel vector computers (a small number of coupled vector processors, such as Cray YMP), massively parallel shared memory computers (MPP, such as the Intel Paragon, in which a large number of the same commodity central processor units – CPUs – used in deskside computers are linked together and communicate over a communication network), multicore processors (such as Intel Xeon that has gone from 6 cores to more than 50) both stand alone and as part of an MPP, and more recently the inclusion of massively multicore graphical processing units (GPUs, which have migrated from the gaming industry into scientific computing and data manipulation). A modern

supercomputer typically consists of nodes, connected via an interconnect (from vendors such as Mellanox and Intel)[ab], where each node houses multiple commodity multicore CPUs and GPUs. This is the dominant architecture of the supercomputers on the top 500 list of the fastest computers in the world[32], with the top 5 supercomputers having between 1.5 and in excess of 10.5 million total computing cores at the time of writing; designing and maintaining simulation codes that perform efficiently on these rapidly evolving computer architectures is a significant challenge. Beyond community developed simulation engines, we have also seen the rise of other community developed utilities to support simulation, e.g., in the form of general analysis packages as well as software that makes it easier to accurately and reproducibility initialize configurations, apply force fields to molecules, and create input files for a variety of simulation engines.

Figure 1 here

Figure 1. Typical steps involved in performing a molecular simulation. If all steps are scriptable, the entire process can be encased in a loop over hundreds or thousands of chemistry, composition, and/or state conditions combinations to enable screen for desirable properties. Background colors refer to: initialization steps (blue), simulation run time steps (red), and system analysis steps (yellow).

For several decades, the open-software movement has been making its presence felt in the chemical engineering community. Open-source software offers many advantages over proprietary

[a] These interconnects can vary from standardized ethernet connections to more specialized, proprietary high performance interconnects from various vendors. At the time of writing, the current top 500 list includes numerous systems with propriety interconnects such as Mellanox Infiniband (now owned by Nvidia), Intel Omni-Path, Cray Aries, and Fujitsu Tofu along with standard ethernet connections ranging from 10G to 100G.

codes. First, they are universally available and do not contain any hidden parameters. This makes verification of results published using these codes much more feasible than for proprietary codes. Indeed, some scholarly journals have taken the position of considering only manuscripts for publication in which molecular modeling calculations were performed using open-source codes or source code that is made available to reviewers. Second, open-source codes are available at no cost, which means that the codes can be downloaded and used by researchers throughout the world, removing barriers for scientific progress. Third, open-source codes typically attract a community of users and/or developers, so that bugs are discovered and eliminated quickly, often overnight; in the case of proprietary software, bugs are typically only fixed during update cycles, which may be months apart, or may even go unnoticed, since the code cannot be inspected by users. The downside of open-source software is that, since there is no revenue stream in the usual sense (sale of software), the sustainability of an open-source code over decades can be questionable. However, codes can reach a level of usage such that the effort to maintain and improve the code is taken on by the user community; `LAMMPS` has arguably reached this position. Also, for some open-source codes there is an alternative revenue stream. For example, Red Hat is the biggest contributor and supporter of the open-source Linux operating system. It makes money by writing, selling, and supporting business-oriented middleware that runs within Linux, as well as selling consulting services to companies switching to Linux for their enterprise software. The commercial Scienomics MAPS platform for materials and process simulations embeds some of the open-source MD and MC codes, such as `LAMMPS`, `Cassandra`, and `MCCCS-Towhee`. Enthought, Inc. is a software company based in Austin, Texas, that develops and markets scientific and analytic computing solutions using primarily the Python programming language; its commercial activities underwrite the widely used open-source SciPy (Scientific Python) package.

In the remainder of this Perspective, as an example of ChEC open-source software, we focus our discussion on the Molecular Simulation Design Framework (MoSDeF), to which all the authors are contributors. MoSDeF is a set of Python tools to facilitate the initialization and parameterization of systems, with the goal of enabling transparent and reproducible molecular simulation workflows that, at the same time, are user-friendly and extensible.

3. Molecular Simulation Design Framework (MoSDeF)

As shown in Figure 1, performing a molecular simulation, whether MD or MC, requires multiple steps: building an initial configuration of the system, selecting and applying a force field, generating a syntactically correct input file (or files) for a target simulation engine, equilibration (to relax the system from its initial configuration – e.g., a crystal – to a configuration characteristic of equilibrium – e.g., liquid), production run to generate a trajectory, and analysis of the trajectory (e.g., averaging over the trajectory to compute thermodynamic and/or structural properties, perform visualization, etc.). Often reliability and statistics are improved by running multiple independent trajectories using the same workflow. Accomplishing these steps in a way that is both accurate and reproducible can be a significant challenge. For example, the application of a force field is a frequent source of error in simulations; for a system composed of moderately complex molecules (such as an ionic liquid) the force field can have a hundred or more parameters that must be provided, offering multiple opportunities for errors (e.g., use of incompatible units, use of parameter values from a publication containing a typographical error, incorrect application of parameters due to logic errors or because of ambiguous definition of parameter usage, etc.). While the use of a community developed, open-source simulation engine may help to reduce the

likelihood of fundamental errors in algorithms underlying the simulations, such codes cannot necessarily prevent users from providing parameters that are inconsistent with the intended usage.

Typically, many of these steps are performed within a given research group by a single graduate student, often making use of *ad hoc*, in-house software, even if open-source simulation engines are used. This approach has several shortcomings that can make simulations more prone to error, limit the extensibility, and hamper reproducibility. For example, the various tools used to accomplish these steps may only be loosely coupled and require manipulation, editing, and/or modification of the tools and/or data by the user. This manipulation may introduce errors and make it difficult to reproducibly capture the exact procedures employed. The need for human manipulation may also limit the ability to use such workflows in applications that require automation, such as parameter screening studies or within the context of larger workflows (e.g., to predict phase equilibrium within a process simulator). The use of in-house software itself, which is typically not open-source or freely available, creates numerous roadblocks as well. Someone wishing to reproduce a simulation would be required to write their own software to accomplish the same tasks. The development of such software may be time consuming and publications often do not provide sufficient detail regarding the procedures used to initialize and parameterize simulations. Furthermore, without access to the original source code, it is not possible to ascertain the quality of the software; that is, to know whether it has undergone sufficient validation or if there are errors and bugs that ultimately impact the accuracy of the reported results.

The Molecular Simulation Design Framework (MoSDeF)[33] is designed to address these issues of automation/efficiency, accuracy, and reproducibility in molecular simulation. MoSDeF is an open-

source Python library built upon the scientific Python software stack with three major components: `mBuild` (for constructing initial configurations of systems) and `foyer` (for applying force fields). The third component, `GMSO` (General Molecular Simulation Object), is currently under development and is designed to be a general, flexible way of encapsulating the information required to define a simulation topology in a simulation engine in an agnostic manner. All of the capabilities of `MoSDeF` are scriptable, thus making the tools inherently reproducible, as well as suitable for automated calculations (e.g., screening). MoSDeF is implemented as a set of composable/modular tools, where each "subpackage" (i.e., module) is designed such that it can be used within MoSDeF, or as a standalone package, allowing MoSDeF to more easily integrate with other community efforts. This also allows the framework to be more easily modified, tested, extended, and have fewer bugs than a monolithic approach. MoSDeF leverages libraries including `packmol`[34], `parmed`[35], `openmm`[36,37], and `openbabel`[38] to maximize compatibility with simulation engines. The interoperability and integration of `mBuild`, `foyer`, and `GMSO` distinguish MoSDeF from other initialization and simulation management packages that are tailored for specific engines (e.g., `ambertools`[39], `playmol`[40]) and which may also require coordination of workflows across multiple languages (e.g., `topotools`[40]), complicating data provenance. That is, MoSDeF tools enable the initialization, simulation, and analysis workflows of entire scientific studies to be defined in python scripts. Performing a simulation using MoSDeF, combined with dissemination of simulation scripts on a service such as Github, enables a molecular simulation to be published as a TRUE (transparent, reproducible, usable by others, and extensible) simulation[41].

MoSDeF has its origins in a decade of National Science Foundation (NSF)-supported collaborative research at Vanderbilt University involving researchers from chemical engineering and computer science[42–44], the latter affiliated with the Institute for Software Integrated Systems (ISIS)[45]. ISIS is a leading academic software engineering research center, and is the originator of the concept of model-integrated computing (MIC)[46]. MIC is a systems engineering approach that focuses on the creation of domain specific modeling languages to capture the essential features of the individual components of a given process, at the level of abstraction that is appropriate for the end users. Due to abstraction, processes are described at a meta level that allows tasks to be coupled together to execute scientific or engineering workflows. MIC has been deployed in applications as diverse as managing auto assembly lines and processing health records. MIC design principles, domain-specific modeling languages, and the general philosophy of abstraction have shaped the development of MoSDeF. In particular, MoSDeF attempts to be simulation-engine-agnostic, treating the concept of a molecular simulation at a meta level, above the specifics of the simulation engines. The tools within MoSDeF are designed to fully describe a system: implementation relies on writers to instantiate syntactically correct input files for specific engines from this information. MoSDeF was initially developed to support several commonly used open-source MD codes (`LAMMPS`[47], `GROMACS`[48] and `HOOMD-blue`[49]) and has since grown to support open-source MC simulation engines, namely `Cassandra`[16] and `GOMC`[18]. In the Supplementary Information, we provide details on how to install MoSDeF through various hosting systems (`anaconda`, `docker`, from source using `github`, etc.) on Apple OSX, Linux, and Windows platforms. Below we describe each of the three key components. Source code, tutorials, documentation, and related publications can be accessed from mosdef.org and/or github.com/mosdef-hub/.

3.1. mBuild

As shown in Figure 1, the first step in a simulation workflow typically involves defining the configuration of the atoms (or more generally, particles) in the system. The `mBuild` Python library[50,51] has been developed to be a general, customizable tool for constructing arbitrarily complex system configurations in a programmatic fashion (i.e., scriptable). Key to the `mBuild` library is its underlying `Compound` data structure. A `Compound` is a general "container" that can describe effectively anything: an atom, a collection of atoms, a molecule, a generic point particle, a collection of `Compounds`, operations on the underlying `Compounds` and/or data, etc. `Compounds` can be duplicated, rotated, translated, scaled, etc. to construct a system. `Compounds` can also contain information regarding connections between the atoms, by defining either fixed `Bonds` within a `Compound` or by adding `Ports` that allow connections to be made between separate `Compounds`. `Ports` define both location and orientation of a connection; in atomistic systems, the number of `Ports` and their locations are typically representative of the underlying chemistry. For example, Figure 2 shows Python code that defines a CH_2 moiety with two C-H `Bonds` and two `Ports`. In order to create a connection between two `Compounds`, a user simply states which `Ports` should connect and `mBuild` automatically performs translations and reorientations, creating a new (composite) `Compound` (see Klein *et al*.[50] for more details). As such, this allows complex systems to be built-up from smaller, interchangeable pieces that can be *connected*, through the use of the concept of generative modeling.[50] This design approach allows for declaratively expressing repetitive structures, such as polymer chains and planar tilings (as

used in Figure 2) and also allows significant modifications to system structure/chemistry to be made with only minimal changes to the initialization routines.

Figure 2 here

Figure 2: Python script that uses mBuild to define a class for a -CH_2- group, create a polymer composed of multiple -CH_2- groups, and connects copies of this polymer to a surface. Note for simplicity, the terminal -CH_3 group is not shown. Additional mBuild tutorials and example scripts are available online at https://github.com/mosdef-hub/mbuild_tutorials.

3.2. `Foyer`

After a system configuration is initialized, the interactions between all constituents must be defined before a system can be simulated (as shown in Figure 1), i.e., the force field must be applied to the system. The `Foyer` library[52] has been developed as a general tool for applying force fields to molecular systems (i.e., atom-typing), that provides a standardized approach to defining chemical context and atom-typing rules[22,53]. In `Foyer`, the forcefield parameters and the rules that dictate parameter usage are stored together in a standardized XML file, separate from the code used to evaluate them. Usage rules are encoded by using a combination of a SMARTS-based annotation scheme, which defines the chemical context associated with a given parameter, and `overrides` that define rule precedence. SMARTS is a language designed for describing molecular patterns,[54] thus allowing information about the bonded environment of an atom to be efficiently and clearly encoded in a format that is both human and machine readable. For example, the chemical context of a terminal methyl group (-CH_3) in an alkane can be expressed as `[C;X4](C)(H)(H)H`. In this annotation, `[C;X4]` indicates that the atom of interest is a carbon (`C`), with 4 total bonds (`X4`)

and `(C)(H)(H)H` provides the identity of those 4 bonds (1 carbon, 3 hydrogens). Figure 3 shows a snippet from the Foyer XML forcefield file demonstrating how these usage rules can be encoded, using select parameters from OPLS-AA force field (See Klein et al.[22] for more details). By separating the usage rules and parameters from the software used to evaluate them, the Foyer library does not need to change if changes are made to a force field file. As such, this allows the implementation of novel and "custom" force fields without the need to write new software, which simplifies the process of disseminating and evolving forcefields, and increases reproducibility of work by making it clear not just what force field was used, but how it was applied to the system. A complimentary approach not requiring SMARTS and `overrides` is to make molecule-specific XML files available (e.g., via webpages such as http://trappe.oit.umn.edu).

Figure 3 here

Figure 3: Foyer snippet illustrating how three Carbon atom types can be defined, with rules for precedence and chemical context, in a human- and computer-readable format.

3.3. General Molecular Simulation Object (`GMSO`)

With a system initialized and parameterized, the information in the system topology must be written to a file for a simulation engine. While the information required by different simulation engines is, generally speaking, the same, the structure and format of the data file(s) passed to simulation engines is typically unique to the engine itself. Generating these files accurately, especially for a wide range of unique simulation engines, can be non-trivial. The current version

of MoSDeF relies upon the use open-source utilities `parmed`[35] and `OpenMM`[37,55] to store this information; these tools along with native MoSDeF code, include parsers to generate syntactically correct data files. In this approach, a single simulation topology can be used to generated input files for a variety of simulation engines, allowing different engines and methodologies (e.g., MC and MD) to be applied to the same system. While effective, these backend codes do not have general support for the breadth of simulation engines and force fields we aim to include. To this end, the General Molecular Simulation Object (`GMSO`) has been under development with the goal of becoming the *de facto* backend data structure of the MoSDeF. The goal of `GMSO` is to serve as a general container for all of the relevant system information (e.g., the fully parameterized system), stored in a simulation engine agnostic way. `GMSO` is designed with interoperability and support for various functional forms as a first-class feature. For example, `GMSO` builds upon the idea of Foyer XML data file, shown in Fig. 3, but provides further meta data; this includes encoding the functional forms of the potentials in the force field (those that can be expressed in computer algebraic inputs) using the `sympy` Python library. `GMSO` is also structured to make it easier to add data file writers, allowing `GMSO` support to be extended and customized. Because `GMSO` supports user-defined analytic equations for force field components, it future-proofs `GMSO` for new developments in force fields, such as those being pursued by several of the authors.

3.4. Computational Screening and Automation using MoSDeF

Since all the functions of MoSDeF are scriptable, when combined with a workflow management tool such as `signac/signac flow`[21], it is relatively trivial to perform computational screening of the properties of systems by looping over chemistries and/or conditions and calculating relevant properties from the simulations. The MoSDeF/`signac` combination has been used to screen the

impact on nanolubrication properties of end-group chemistry of self-assembled alkylsilane tethers on amorphous silica surfaces[23], leading to a machine-learning-derived model connecting end-group cheminformatic descriptors with tribological properties of interest. In another example[56], the diffusivities of ions in organic solvents were screened for 22 different solvents, revealing a pattern in this large data set (ion diffusivity proportional to solvent diffusivity) that was in contrast with previous, primarily experimental findings (ion diffusivity proportional to solvent dipole moment). The computational screening finding were confirmed in subsequent experimental studies utilizing quasi-elastic neutron scattering[57] and NMR[58].

3.5 Expanding MoSDeF

As noted earlier, the genesis of MoSDeF was a series of NSF grants to Vanderbilt PIs Cummings, M^{c}Cabe, Iacovella, and Ledezci[42–44]. A recent collaborative NSF grant[59] has funded groups from the universities of Michigan (Glotzer and Anderson), Notre Dame (Maginn), Minnesota (Siepmann), Delaware (Jayaraman), Houston (Palmer), Wayne State (Potoff), and Boise State (Jankowski) universities to work together to expand MoSDeF's capabilities, including the collaborative design and development of the aforementioned `GMSO` backend. This collaboration is resulting in increasing integration with `HOOMD-blue`, integration with MC codes `Cassandra` and `GOMC`, and the first principles MD/MC code `CP2K`; additionally, MoSDeF has been integrated more closely with Michigan's `signac` workflow management tools. In the case of `Cassandra`, for example, using MoSDeF existing utilities and adding additional capabilities resulting from the Vanderbilt/Notre Dame collaboration, the complexity of setting up a simulation has been reduced from 9 steps (including 3 requiring user editing of files) to a single python script using MoSDeF; this, in turn, has enabled computational screening with `Cassandra`. Other groups, including

Houston, Boise State, and Delaware, are focusing on developing modules to implement complex workflows and analyses involved in phase equilibrium calculations and construction of intricate molecular models. Building the modules around the MoSDEF framework will enable these workflows to be performed in a reproducible fashion with a variety of widely used simulation engines.

An example of the capabilities enabled by this collaboration is given in the Supplementary Information (SI). Inspired by the honoree of this special issue, Keith Gubbins, in the SI we report the use of five different simulation codes (the open-source MC codes `Cassandra` and `GOMC`, the open-source MD codes `LAMMPS` and `GROMACS`, and the open -source first principles MD code `CP2K`) to repeat calculations reported by Striolo *et al*.[60] on the adsorption of water into carbon slit pores. The latter were groundbreaking simulations for their time and the paper has been cited ~200 times (Google Scholar). The paper reported adsorption/desorption isotherms, demonstrating the hysteresis seen in experiment, as well as density profiles and orientational structure of the adsorbed water into carbon slit pores. The Striolo *et al*. simulations were performed using in-house codes; thus, they are almost impossible to reproduce in detail. In the SI, we show that we can reproduce the adsorption/desorption isotherms reported by Striolo *et al*. to within an acceptable degree using `Cassandra` and `GOMC`; more importantly, we show that by using the MoSDeF tools to create the simulations, we can easily test multiple engines, and show we get excellent agreement between the two different MC codes. Having used the technique of GEMC in both `Cassandra` and `GOMC`, we establish the number of water molecules in the pore at a given external pressure. We then perform *NVT* (constant number of molecules, volume and temperature) simulations using multiple

codes. We find remarkable agreement for the water structure inside the pore between the MC engines `Cassandra` and `GOMC` and MD engines `LAMMPS` and `GROMACS`. The use of MoSDeF (`mbuild` to build the simulation systems and `foyer` to apply the force fields) is absolutely essential to obtaining consistency between these calculations. The first principles MD code `CP2K` with interactions described on-the-fly via Kohn-Sham density functional theory produces similar, but not identical, results for water structure, thereby allowing us to identify differences in water-substrate interactions. The fact that one can move the simulated system between all of these codes fairly effortlessly, thanks to the use of the MoSDeF tools and its meta-level abstraction of the concept of molecular simulation, is a very significant step forward for the simulation community. Moreover, the SI contains all the instructions needed for the reader to download and run all the utilities and codes needed to reproduce the reported calculations exactly, hence qualifying these as TRUE simulations.[41]

3.6 Future Directions and Challenges

It is clear that the role of modeling and simulation in engineering and scientific research will continue to grow as computing power advances and new methods and simulation engines are developed. Ironically, the more powerful and capable modeling tools become, the more difficult it is to ensure that the results of these simulations can be reproduced by others and that the numerous details that go into running the simulations are validated and justified. This can lead to a "crisis in confidence" in the accuracy of simulation research. We believe that efforts at developing tools and workflows that focus on transparency and accuracy of simulations are therefore essential, and that the MoSDeF tools described here are an important step in helping improve the reliability of simulations.

Given the backgrounds of the authors, the major focus up to this point has been on classical force field-driven Monte Carlo and molecular dynamics simulations, although as demonstrated in this Perspective, MoSDeF can also be applied to the ab initio code CP2K. In the future, we would like to see the scope of MoSDeF expanded to include other ab initio codes, reactive methods and analysis tools. MoSDeF has been developed to be very flexible, so it can be adapted to work with additional packages as well as future computer architectures and programming structures. A key challenge in realizing this vision is resources, both financial and human. We have been fortunate to have support from the National Science Foundation to create MoSDeF. Its expansion and long-term sustainability will require that the research community see the value in MoSDeF and commit to supporting it. There are many examples of the research community supporting open source simulation packages, but the key is that users find value in the tool and in extending its capabilities. We hope we have demonstrated the value of MoSDeF and that other researchers will become involved in its maintenance and growth.

4. Summary and Conclusions

In this Perspective article, we have described our efforts at developing the Molecular Simulation Design Framework (MoSDeF), a collection of open source tools that not only make the design and execution of molecular simulations easier, but they also help enable the simulations to be "TRUE": transparent, reproducible, usable by others, and extensible. The collection of tools enables system setup, atom typing, force field assignment, and job management. MoSDeF is designed to be compatible with a wide range of simulation engines and force fields. As an example of MoSDeF's

capabilities, we undertook the modeling of the sorption and diffusion of water in a carbon slit pore, something Keith Gubbins and co-workers did many years ago. We show that we can seamlessly integrate five different simulation packages in the study and that consistent results are obtained between the different packages. We hope this article stimulates other researchers to not only adopt MoSDeF in their work, but to also contribute to its continued expansion and development.

We dedicate this Perspective to our colleague, mentor, and friend, Keith Gubbins. The authors of this article wish to express their deep gratitude to Keith for all he has done for our community. We wish him many more years of productive science.

Acknowledgements

The preparation of this Perspective article has been supported by a National Science Foundation grants OAC-1835874 to Vanderbilt University, OAC-1835612 to the University of Michigan, OAC-1835630 to the University of Notre Dame, OAC-1835067 to the University of Minnesota, OAC-1835613 to the University of Delaware, OAC-1835593 to Boise State University, OAC-1835713 to Wayne State University, and OAC-1835560 to the University of Houston.

Literature Cited

1. Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E. Equation of State Calculations by Fast Computing Machines. *J Chem Phys*. 1953;21(6):1087-1092. doi:10.1063/1.1699114
2. Alder BJ, Wainwright TE. Phase transition for a hard sphere system. *J Chem Phys*. 1957;27(5):1208-1209. doi:10.1063/1.1743957
3. Alder BJ, Wainwright TE. Studies in Molecular Dynamics. I. General Method. *J Chem Phys*. 1959;31(2):459-466. doi:10.1063/1.1730376
4. Wood WW, Parker FR. Monte Carlo Equation of State of Molecules Interacting with the Lennard-Jones Potential. I. A Supercritical Isotherm at about Twice the Critical Temperature. *J Chem Phys*. 1957;27(3):720-733. doi:10.1063/1.1743822
5. Rahman A. Correlations in the Motion of Atoms in Liquid Argon. *Phys Rev*. 1964;136(2A):A405-A411. doi:10.1103/PhysRev.136.A405
6. Nicolas JJ, Gubbins KE, Streett WB, Tildesley DJ. Equation of state for the lennard-jones fluid. *Mol Phys*. 1979;37(5):1429-1454. doi:10.1080/00268977900101051
7. Allen MP, Tildesley DJ. *Computer Simulation of Liquids*. Vol 1. Second. Oxford University Press; 2017. doi:10.1093/oso/9780198803195.001.0001
8. Westmoreland PR, Kollman PA, Chaka AM, et al. *Applying Molecular and Materials Modeling*. Dordrecht: Springer Netherlands; 2002. doi:10.1007/978-94-017-0765-7
9. Reed TMK, Gubbins KE. *Applied Statistical Mechanics: Thermodynamic and Transport Properties of Fluids*. McGraw-Hill; 1973.

https://books.google.com/books?id=w_tQAAAAMAAJ.

10. Gray CG, Gubbins KE, Joslin CG. *Theory of Molecular Fluids: I: Fundamentals*. OUP Oxford; 1984. https://books.google.com/books?id=3mz2RcnnMGwC.
11. Gray CG, Gubbins KE, Joslin CG. *Theory of Molecular Fluids: Volume 2: Applications*. OUP Oxford; 2011. https://books.google.com/books?id=4xr8jwEACAAJ.
12. Panagiotopoulos AZ. Direct determination of phase coexistence properties of fluids by monte carlo simulation in a new ensemble. *Mol Phys*. 1987;61(4):813-826. doi:10.1080/00268978700101491
13. Paricaud P, Předota M, Chialvo AA, Cummings PT. From dimer to condensed phases at extreme conditions: Accurate predictions of the properties of water by a Gaussian charge polarizable model. *J Chem Phys*. 2005;122(24):244511. doi:10.1063/1.1940033
14. Anderson JA, Lorenz CD, Travesset A. General purpose molecular dynamics simulations fully implemented on graphics processing units. *J Comput Phys*. 2008;227(10):5342-5359. doi:10.1016/j.jcp.2008.01.047
15. Glaser J, Nguyen TD, Anderson JA, et al. Strong scaling of general-purpose molecular dynamics simulations on GPUs. *Comput Phys Commun*. 2015;192:97-107. doi:10.1016/j.cpc.2015.02.028
16. Shah JK, Marin-Rimoldi E, Mullen RG, et al. Cassandra: An open source Monte Carlo package for molecular simulation. *J Comput Chem*. 2017;38(19):1727-1739. doi:10.1002/jcc.24807
17. Dubbeldam D, Calero S, Ellis DE, Snurr RQ. RASPA: molecular simulation software for adsorption and diffusion in flexible nanoporous materials. *Mol Simul*. 2016;42(2):81-101.

doi:10.1080/08927022.2015.1010082

18. Nejahi Y, Soroush Barhaghi M, Mick J, et al. GOMC: GPU Optimized Monte Carlo for the simulation of phase equilibria and physical properties of complex fluids. *SoftwareX*. 2019;9:20-27. doi:10.1016/j.softx.2018.11.005
19. Schultz AJ, Kofke DA. Etomica : An object-oriented framework for molecular simulation. *J Comput Chem*. 2015;36(8):573-583. doi:10.1002/jcc.23823
20. Ramasubramani V, Dice BD, Harper ES, Spellings MP, Anderson JA, Glotzer SC. freud: A software suite for high throughput analysis of particle simulation data. *Comput Phys Commun*. 2020;254:107275. doi:10.1016/j.cpc.2020.107275
21. Adorf CS, Dodd PM, Ramasubramani V, Glotzer SC. Simple data and workflow management with the signac framework. *Comput Mater Sci*. 2018;146:220-229. doi:10.1016/j.commatsci.2018.01.035
22. Klein C, Summers AZ, Thompson MW, et al. Formalizing atom-typing and the dissemination of force fields with foyer. *Comput Mater Sci*. 2019;167(May):215-227. doi:10.1016/j.commatsci.2019.05.026
23. Summers AZ, Gilmer JB, Iacovella CR, Cummings PT, McCabe C. MoSDeF, a Python Framework Enabling Large-Scale Computational Screening of Soft Matter: Application to Chemistry-Property Relationships in Lubricating Monolayer Films. *J Chem Theory Comput*. 2020;16(3):1779-1793. doi:10.1021/acs.jctc.9b01183
24. Thompson MW, Gilmer JB, Matsumoto RA, et al. Towards molecular simulations that are transparent, reproducible, usable by others, and extensible (TRUE)*. *Mol Phys*. 2020;0(0):e1742938. doi:10.1080/00268976.2020.1742938

25. Lin ST, Sandler SI. A priori phase equilibrium prediction from a segment contribution solvation model. *Ind Eng Chem Res*. 2002;41(5):899-913. doi:10.1021/ie001047w

26. Bell IH, Mickoleit E, Hsieh CM, et al. A Benchmark Open-Source Implementation of COSMO-SAC. *J Chem Theory Comput*. 2020;16(4):2635-2646. doi:10.1021/acs.jctc.9b01016

27. Fortunato ME, Colina CM. pysimm : A python package for simulation of molecular systems. *SoftwareX*. 2017;6:7-12. doi:10.1016/j.softx.2016.12.002

28. Martin TB, Gartner TE, Jones RL, Snyder CR, Jayaraman A. pyPRISM: A Computational Tool for Liquid-State Theory Calculations of Macromolecular Materials. *Macromolecules*. 2018;51(8):2906-2922. doi:10.1021/acs.macromol.8b00011

29. Schweizer KS, Curro JG. Integral-equation theory of the structure of polymer melts. *Phys Rev Lett*. 1987;58(3):246-249. doi:10.1103/PhysRevLett.58.246

30. Cummings PT, Gilmer JB. Open-source molecular modeling software in chemical engineering. *Curr Opin Chem Eng*. 2019;23:99-105. doi:10.1016/j.coche.2019.03.008

31. Pirhadi S, Sunseri J, Koes DR. Open source molecular modeling. *J Mol Graph Model*. 2016;69:127-143. doi:10.1016/j.jmgm.2016.07.008

32. The TOP500 project. https://top500.org.

33. Molecular Simulation Design Framework (MoSDeF) Homepage. https://mosdef.org.

34. Martínez L, Andrade R, Birgin EG, Martínez JM. PACKMOL: a package for building initial configurations for molecular dynamics simulations. *J Comput Chem*. 2009;30(13):2157-2164. doi:10.1002/jcc.21224

35. ParmEd — ParmEd documentation. http://parmed.github.io/ParmEd/html/index.html.

Published February 18, 2018. Accessed April 18, 2018.

36. Eastman P, Swails J, Chodera JD, et al. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. Gentleman R, ed. *PLOS Comput Biol*. 2017;13(7):e1005659. doi:10.1371/journal.pcbi.1005659

37. Eastman P, Pande VS. OpenMM: A Hardware Independent Framework for Molecular Simulations. *Comput Sci Eng*. 2015;12(4):34-39. doi:10.1109/MCSE.2010.27

38. O'Boyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR. Open Babel: An open chemical toolbox. *J Cheminform*. 2011;3(1):33. doi:10.1186/1758-2946-3-33

39. Case DA, Belfon K, Ben-Shalom IY, et al. AMBER 2020. AMBER 2020 Reference Manual.

40. Abreu CRA. Playmol. https://github.com/atoms-ufrj/playmol. Published 2018.

41. Thompson MW, Gilmer JB, Matsumoto RA, et al. Towards molecular simulations that are transparent, reproducible, usable by others, and extensible (TRUE). *Mol Phys*. 2020;0(0):e1742938. doi:10.1080/00268976.2020.1742938

42. NSF Award # CBET-1028374 Collaborative Research: CDI-Type II: Cyber-Enabled Design of Functional Nanomaterials. https://www.nsf.gov/awardsearch/showAward?AWD_ID=1028374.

43. NSF Award # ACI-1047828 - SI2-SSI: Development of an Integrated Molecular Design Environment for Lubrication Systems (iMoDELS) (PI: Cummings). https://www.nsf.gov/awardsearch/showAward?AWD_ID=1047828.

44. NSF Award # ACI-1535150 - SI2-SSE: Development of a Software Framework for Formalizing Forcefield Atom-Typing for Molecular Simulation (PI: Iacovella). https://www.nsf.gov/awardsearch/showAward?AWD_ID=1535150.

45. Institute for Software Integrated Systems. https://www.isis.vanderbilt.edu/. Accessed November 24, 2019.

46. Sztipanovits J, Karsai G. Model-integrated computing. *Computer (Long Beach Calif)*. 1997;30(4):110-111.

47. Plimpton S. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *J Comput Phys*. 1995;117(1):1-19. doi:10.1006/jcph.1995.1039

48. Abraham MJ, Murtola T, Schulz R, et al. Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*. 2015;1-2:19-25. doi:10.1016/j.softx.2015.06.001

49. Anderson JA, Glaser J, Glotzer SC. HOOMD-blue: A Python package for high-performance molecular dynamics and hard particle Monte Carlo simulations. *Comput Mater Sci*. 2020;173:109363. doi:10.1016/j.commatsci.2019.109363

50. Klein C, Sallai J, Jones TJ, Iacovella CR, McCabe C, Cummings PT. A Hierarchical, Component Based Approach to Screening Properties of Soft Matter. In: Snurr RQ, Adjiman CS, Kofke DA, eds. *Foundations of Molecular Modeling and Simulation. Molecular Modeling and Simulation (Applications and Perspectives)*. Springer, Singapore; 2016:79-92. doi:10.1007/978-981-10-1128-3_5

51. mBuild Github repository. https://github.com/mosdef-hub/mbuild. Accessed August 17, 2018.

52. Foyer Github repository. https://github.com/mosdef-hub/foyer. Accessed August 10, 2020.

53. Iacovella CR, Sallai J, Klein C, Ma T. Idea Paper : Development of a Software Framework for

Formalizing Forcefield Atom-Typing for Molecular Simulation. In: *4th Workshop on Sustainable Software for Science: Practice and Experiences (WSSSPE4)*. ; 2016.

54. Daylight Theory: SMARTS - A Language for Describing Molecular Patterns. http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html. Published August 16, 2017. Accessed April 18, 2018.

55. SimTK: OpenMM: Project Home. https://simtk.org/projects/openmm. Published February 18, 2018. Accessed April 18, 2018.

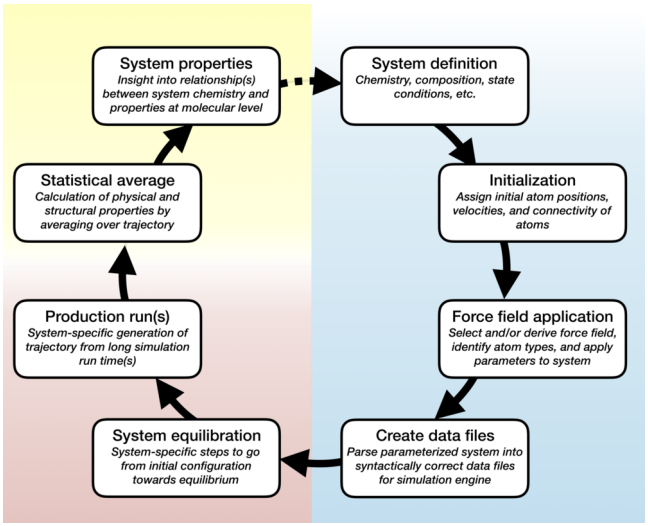
56. Thompson MW, Matsumoto R, Sacci RL, Sanders NC, Cummings PT. Scalable Screening of Soft Matter: A Case Study of Mixtures of Ionic Liquids and Organic Solvents. *J Phys Chem B*. 2019;123(6):1340-1347. doi:10.1021/acs.jpcb.8b11527

57. Osti NC, Matsumoto RA, Thompson MW, Cummings PT, Tyagi M, Mamontov E. Microscopic Dynamics in an Ionic Liquid Augmented with Organic Solvents. *J Phys Chem C*. 2019;123(32):19354-19361. doi:10.1021/acs.jpcc.9b05119

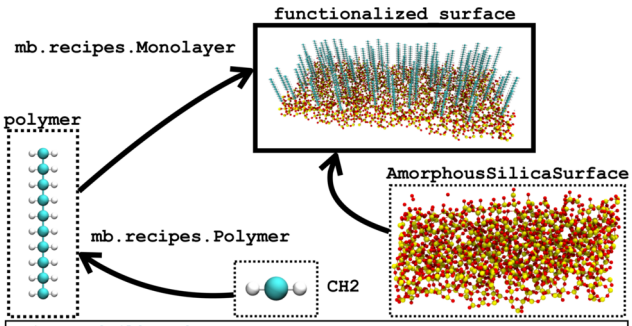
58. Cui J, Kobayashi T, Sacci RL, Matsumoto RA, Cummings PT, Pruski M. Diffusivity and Structure of Room Temperature Ionic Liquid in Various Organic Solvents. *J Phys Chem B*. 2020:submitted for publication.

59. NSF Award # OAC-1835874 Collaborative Research: NSCI Framework: Software for Building a Community-Based Molecular Modeling Capability Around the Molecular Simulation Design Framework (MoSDeF). https://www.nsf.gov/awardsearch/showAward?AWD_ID=1835874.

60. Striolo A, Chialvo AA, Cummings PT, Gubbins KE. Water adsorption in carbon-slit nanopores. *Langmuir*. 2003;19(20):8583-8591. doi:10.1021/la0347354



AIC_17206_AIChE-20-23421-accepted-Figure1.tif




```
import mbuild as mb
from mbuild.lib.atoms import H
from mbuild.recipes import recipes
from mbuild.lib.surfaces import AmorphousSilicaSurface

class CH2(mb.Compound):
    def __init__(self):
        super(CH2, self).__init__()
        carbon = mb.Particle(pos=[0.0, 0.0, 0.0], name='C')
        hydrogen0 = mb.Particle(pos=[0.1, 0.0, 0.0], name='H')
        hydrogen1 = mb.Particle(pos=[-0.1, 0.0, 0.0], name='H')
        self.add([carbon, hydrogen0, hydrogen1])
        self.add_bond((carbon, hydrogen0))
        self.add_bond((carbon, hydrogen1))
        self.add(mb.Port(anchor=self[0], orientation=[0, 1, 0],
                         separation=0.07), label='up')
        self.add(mb.Port(anchor=self[0], orientation=[0, -1, 0],
                         separation=0.07), label='down')

polymer = recipes.Polymer(monomers=CH2(), n=10)
locations = mb.pattern.Grid2DPattern(n=10, m=10)
functionalized_surface = recipes.Monolayer(AmorphousSilicaSurface(),
                                           polymer, backfill=H(),
                                           pattern=locations,
                                           tile_x=2, tile_y=1,)
```

AIC_17206_AIChE-20-23421-accepted-Figure2.tif

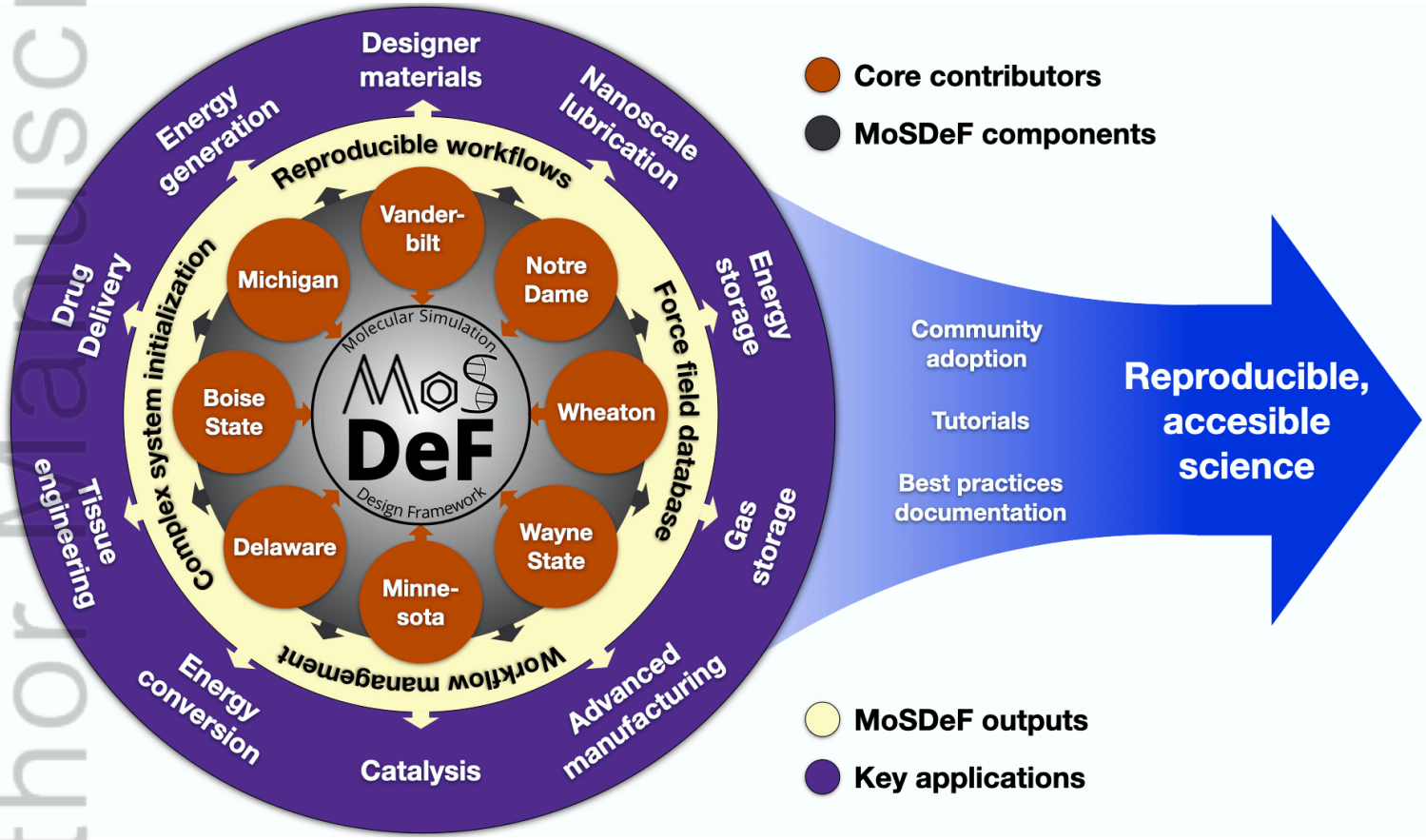
```
<ForceField>
 <AtomTypes>
 ...
 <Type name="opls_135" class="CT" element="C" mass="12.01100" \\
       def="[C;X4](C)(H)(H)H" desc="alkane CH3" \\
       doi="10.1021/ja9621760"/>
 <Type name="opls_148" class="CT" element="C" mass="12.01100" \\
       def="[C;X4]([C;%opls_145])(H)(H)H" \\
       desc="toluene CH3" overrides="opls_135" \\
       doi="10.1021/ja9621760"/>
 <Type name="opls_145" class="CA" element="C" mass="12.01100" \\
       def="[C;X3;r6]1[C;X3;r6][C;X3;r6][C;X3;r6][C;X3;r6][C;X3;r6]1" \\
       desc="aromatic C in 6-membered ring" \\
       overrides="opls_141,opls_142" doi="10.1021/ja9621760"/>
 ...
 </AtomTypes>
</ForceField>
```

AIC_17206_AIChE-20-23421-accepted-Figure3.tif

Simulation Codes			
Code	Description	Originator	Website
HOOMD-blue (Hierarchical Object Oriented MD)	First MD code specifically written for high performance on GPUs. It has evolved into a general-purpose MD and MC particle simulation toolkit optimized for execution on both GPUs and CPUs.	Joshua Anderson and Glotzer group[14,15]	https://glotzerlab.engin.umich.edu/hoomd-blue/index.html
Cassandra	MC code with focus on complex molecular systems (e.g., ionic liquids)	Maginn group[16]	https://cassandra.nd.edu
RASPA	MC/MD code for simulating adsorption and diffusion of molecules in flexible nanoporous materials	Snurr, Dubbeldam, Calero and Vlugt groups[17]	https://www.iraspa.org/RASPA/
GOMC (GPU-optimized MC)	GPU-accelerated general purpose MC	Potoff group[18]	http://gomc.eng.wayne.edu
MCCCS-Towhee MCCCS-MN	MC for complex chemical systems code with a focus on phase and adsorption equilibria	Marcus Martin Siepmann group	http://towhee.sourceforge.net
Etomica	Java-based MC/MD for use in education, with modules to demonstrate molecular basis for many phenomena	Kofke group[19]	http://www.etomica.org/app/modules/
Other Utilities			
iRASPA	GPU-accelerated visualization package	Snurr, Dubbeldam, Calero and Vlugt groups[17]	https://www.iraspa.org/index.html
freud	Library for analyzing MD/MC simulation trajectories for metrics such as the radial distribution function and various order parameters	Glotzer group[20]	https://freud.readthedocs.io/en/stable/
signac/signac-flow	Framework to manage and scale large simulation workflows, facilitating data reuse, sharing, and reproducibility.	Glotzer group[21]	https://signac.io, https://docs.signac.io/en/latest/
MoSDeF (Molecular Simulation Design Framework)	Python-based software package for building complex molecular systems, applying forcefields, and generating input files for various simulation engines	Cummings and McCabe groups[22–24]	https://mosdef.org
COSMO-SAC (COnductor-like Screening Model-Segment Activity Coefficient)	Package to predict activity coefficients from quantum chemistry calculations for use in phase equilibria calculations relevant to chemical engineering separations	Sandler, Lin and Bell groups[25,26]	https://github.com/usnistgov/COSMOSAC
pysimm (python simulation interface for molecular modeling)	Python package for facilitating building of amorphous polymer systems and application of force fields	Colina group[27]	https://pysimm.org, https://github.com/polysimtools/pysimm
pyprism (python-based Polymer Reference Interaction Site Model (PRISM))	Python-based, open-source framework for conducting PRISM theory calculations, with a user-friendly scripting interface for setting up and numerically solving the PRISM equations	Jayaraman group[28,29]	https://pyprism.readthedocs.io/en/latest/
Notable Open-Source Codes Emerging from other Chemical Engineering Communities			
GNU OCTAVE	C++-based open-source high-level language, primarily intended for numerical computations; compatible with MATLAB	John W. Eaton	https://www.gnu.org/software/octave/
DWSIM5	Mixed code open-source chemical process simulator, compatible with the	Daniel Medeiros	https://github.com/DanWBR/dwsim5

	CAPE-OPEN standards for interoperability (https://www.colan.org)		

Table 1. Examples of open-source molecular simulation codes and related supporting utilities developed within the chemical engineering molecular modeling community. Website links are to home pages of the codes or to code repositories. In addition, several other open-source codes emerging from the chemical engineering community are highlighted.



AIC_17206_AIChE-20-23421-accepted-Table of Contents Figure.tif