Scymol
A python-based software package for initializing and running molecular dynamics simulations using LAMMPS

Assaf, Eli I.; Maalouf, Elsa; Liu, Xueyan; Lin, Peng; Erkens, Sandra

DOI
10.1016/j.softx.2025.102044

Publication date
2025

Document Version
Final published version

Published in
SoftwareX

SoftwareX 29 (2025) 102044

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original Software Publication

Scymol: A python-based software package for initializing and running molecular dynamics simulations using LAMMPS

Eli I. Assaf [a,*], Elsa Maalouf [b,**], Xueyan Liu [a], Peng Lin [a], Sandra Erkens [a]

[a] Delft University of Technology, Delft, The Netherlands
[b] Baha and Walid Bassatne Department of Chemical Engineering and Advanced Energy, American University of Beirut, Beirut, Lebanon

ARTICLE INFO

Keywords:
Molecular Dynamics Simulations
Computational Chemistry
LAMMPS
Graphical User Interface
High throughput
Software package

ABSTRACT

Scymol is a Python-based software package specifically designed to facilitate the setup and execution of molecular simulations in LAMMPS. It comes equipped with a user-friendly interface, which simplifies the process of initializing molecular systems and defining simulation parameters. Moreover, the software generates and executes LAMMPS simulation sequences, enabling researchers to establish comprehensive simulation schemes, such as heating or deformation cycles, in a single run. Through its successful application in diverse research projects and its modular design, Scymol demonstrates considerable promise as an indispensable tool for researchers aiming to carry out molecular dynamics simulations without sacrificing complexity or high-throughput capabilities in their methodologies.

Metadata

Nr	Code metadata description	*Please fill in this column*
C1	Current code version	*V1.0.0 (major.minor.patch)*
C2	Permanent link to code/ repository used for this code version	*https://github.com/eli-ams/scymol*
C3	Permanent link to reproducible capsule	*https://codeocean.com/capsule/1160181/tree*
C4	Legal code license	*GNU General Public License (GPL)*
C5	Code versioning system used	*None*
C6	Software code languages, tools and services used	Python 3.9+, Qt5, Rdkit, Pysimm, MPI, LAMMPS
C7	Compilation requirements, operating environments and dependencies	See "Installation" on README.md in *https://github.com/eli-ams/scymol*
C8	If available, link to developer documentation/manual	See README.md in *https://github.com/eli-ams/scymol*
C9	Support email for questions	e.i.assaf@tudelft.nl

1. Motivation and significance

The application of Statistical Mechanics (SM) and Molecular Dynamics (MD) was confined to the domains of physics, chemistry, and scientific computing [1,2]. However, these methodologies have now gained traction in disciplines outside core areas because of their accurate representation of novel materials without the need for extensive experimental validation [3]. Over the years, molecular simulation engines such as LAMMPS [4] (Large-scale Atomic/Molecular Massively Parallel Simulator) and GROMACS [5] (Groningen Machine for Chemical Simulations) have emerged as field standards. Despite their comprehensive capabilities in facilitating diverse simulation configurations, using these software packages appropriately presents its own set of challenges [6].

Users should have a solid understanding of the chemistry of the systems they are simulating, including the nature of the phenomena to be captured and the necessary conditions. Physically, it is essential to understand how classical mechanics and statistical methods interplay to model atomic interactions [7]. On the computational side, these methods require careful use of state-of-the-art algorithms and computational resources, making expertise in numerical and High-Performance Computing (HPC) crucial [8,9]. Moreover, users should be well-versed in LAMMPS-specific scripting including knowledge in its source code [10]. Additionally, users should be prepared to navigate Unix-based operating systems on which many LAMMPS simulations run.

Commercially available software tools like Materials Studio [11],

* Corresponding author.
** Co-corresponding author.
E-mail addresses: e.i.assaf@tudelft.nl (E.I. Assaf), em40@aub.edu.lb (E. Maalouf).

https://doi.org/10.1016/j.softx.2025.102044
Received 30 September 2024; Received in revised form 13 December 2024; Accepted 9 January 2025
Available online 15 January 2025

MedeA [12], Scienomics, and QuantumEspresso [13] have been developed to simplify the process of setting up and running MD. However, these tools have limitations in terms of cost, flexibility, and compatibility with open-source alternatives. They often rely on graphical user interfaces that, while user-friendly and comprehensive, hinder the automation of high-throughput studies. Open-source solutions such as Avogadro [14], OpenBabel [15], Visual Molecular Dynamics (VMD) [16], and Ovito [17] offer some functionalities for MD preparation and analysis but are not built to be a comprehensive solution around LAMMPS. These tools still necessitate manual efforts to bridge gaps in simulation workflows, particularly in areas that involve the initialization of LAMMPS simulations.

Additionally, renowned Python-based packages such as Chemistry at HARvard Macromolecular Mechanics [18] (CHARMM) and its web-based tool CHARMM-GUI, as well as the Molecular Simulation Design Framework [19] (MosDef), offer direct solutions for preparing, executing, and analyzing Molecular Dynamics simulations but present certain limitations. For instance, CHARMM is not open-source, and access to its underlying code is governed by restrictive licensing conditions. CHARMM-GUI serves as an online tool for generating simulation inputs, but the actual simulations are performed locally on compatible MD engines, resulting in an interrupted workflow. In contrast, while MosDef is open source, it does not include a UI and instead relies extensively on Python scripting for setting up and running simulations. Similarly, other open-source packages, such as PySoftK [20] and Polymatic [21], provide robust solutions for setting up and running MD simulations. However, they also rely heavily on Python scripting skills for their execution, which makes them less accessible to researchers with limited scripting proficiency or those not specifically focused on the fields for which the software is supported and designed.

In the current landscape, many researchers find themselves combining functionalities from different software packages to establish and execute LAMMPS-based simulations [22–28]. This situation is less than ideal for scientists whose expertise should be focused on answering important questions in their lines of study rather than figuring out how to conduct complex simulations effectively and efficiently. For this reason, the development of a software package like Scymol proves valuable, as it aims to provide a simple and straightforward platform for initiating and running MD simulations, from start to end, with the intent of lowering the barriers related to technical proficiency to run LAMMPS simulations. Moreover, Scymol aims to establish a foundation for a growing library of LAMMPS simulation routines, designed to evolve over time and address the need for predicting material properties through a standardized set of sequences of LAMMPS operations. This is particularly beneficial for researchers engaged in technical scientific projects, such as Civil and Pavement Engineers, who seek a straightforward introduction to the use of Molecular Dynamics simulations and anticipate a program that is continuously developed to meet their specific objectives.

The development of a software package for automating the initialization of molecular systems and preparing LAMMPS simulations has been accelerated by the capabilities offered by Python libraries such as Openbabel, Openmm [29], RDKit [30], and Pysimm [31]. These libraries, although not explicitly designed to interface with LAMMPS, facilitate various tasks within the MD simulation workflow, thereby simplifying the development of more comprehensive software packages such as Scymol.

2. Software description

Scymol is a Python-based software package featuring a graphical User Interface (UI) developed in PyQt5. The software is designed to simplify the process of setting up and executing molecular simulations using LAMMPS. It eliminates much of the complexity associated with traditional simulation setups by offering intuitive features such as drag-and-drop interfaces for configuring various LAMMPS stages. Users can set up complex simulations of hydrocarbon mixtures involving deformation and heating cycles by simply providing the SMILES notation or other universally recognized molecular formats. Scymol is implemented in Python 3.12 and is organized into three main components: the *frontend*, the *fron2back*, and the *backend*.

The frontend is designed primarily as a UI developed using PyQt5 and QtDesigner [32]. It offers a simple yet comprehensive interface where users can add or delete molecules, modify a number of configuration parameters, and set up sequences of LAMMPS simulation stages. The frontend employs the RDKit module to parse the inputs provided via the UI, preparing the chemical system for subsequent processing by the backend.

The fron2back functions as the link between the frontend and the backend. It collects all the data in the elements of the frontend and processes them to produce the inputs readable by the backend. It also spawns, monitors, and controls the state of the jobs executed from the UI.

The backend functions as an automated engine for executing LAMMPS simulations based on user-defined inputs. It performs tasks such as initializing molecular positions, generating necessary LAMMPS inputs, and managing the execution of individual LAMMPS processes through a Message Passing Interface (MPI) [33]. It also handles the input and output files produced between LAMMPS stages and aggregates the results. The backend uses the RDKit, Pysimm, and Numba [7] Python packages to execute these functionalities. Importantly, the backend is designed to operate independently from the frontend. It reads all the required information from an inputs.py file, which the fron2back generates, to perform its operations. This file is both readable and modifiable, allowing users to adapt it to custom Python algorithms and run it in environments that may not support UIs.

2.1. Software architecture

An illustration of the principal elements and their intercommunication is provided in Fig. 1, while Fig. 2 offers a snapshot of the main UI. The entire package resides in a single directory, denoted as */*. Within this directory, there are several subdirectories—*/backend, /frontend, /front2back*, and */output*—as well as individual files such as */main.py*, */static_functions.py*, and */prechecks.py*. These components are organized functionally and are detailed as follows.

2.1.1. main.py

The *main.py* file is pivotal for initializing the UI elements and instancing the MainWindow class. This class serves as the main application window and holds methods for initializing various attributes, including data tables and molecule objects. It also configures window properties and maps interface signals to corresponding actions. To initialize an instance of Scymol, *main.py* must be executed from / (e.g., using the command *python /main.py*).

2.1.2. frontend/

The frontend directory functions as a custom module responsible for building the UI. This directory contains various Python modules and subdirectories that serve different functionalities:

2.1.2.1. main_window.py. Contains the Python class essential for constructing the UI of Scymol's MainWindow instance. It encompasses elements from all the tabs (Tab1 through Tab5) as well as the menu bar located at the top of the MainWindow.

2.1.2.2. lammps_flowchart_window.py. Contains Python modules dedicated to initializing instances of the LammpsFlowchartWindow class. This window is crucial for enabling users to set up sequential LAMMPS simulation substages through an intuitive drag-and-drop interface. An associated subdirectory, frontend/dialog_windows/, contains subclasses

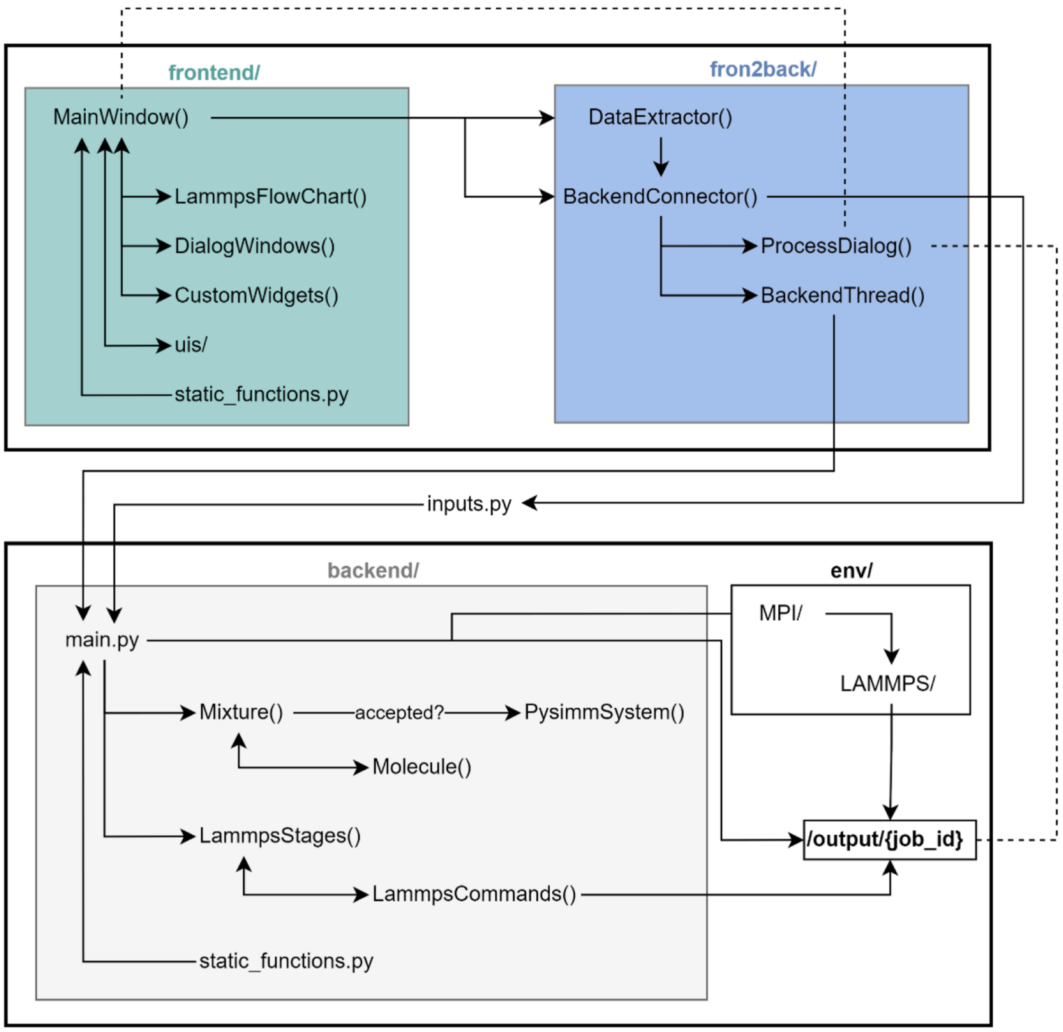
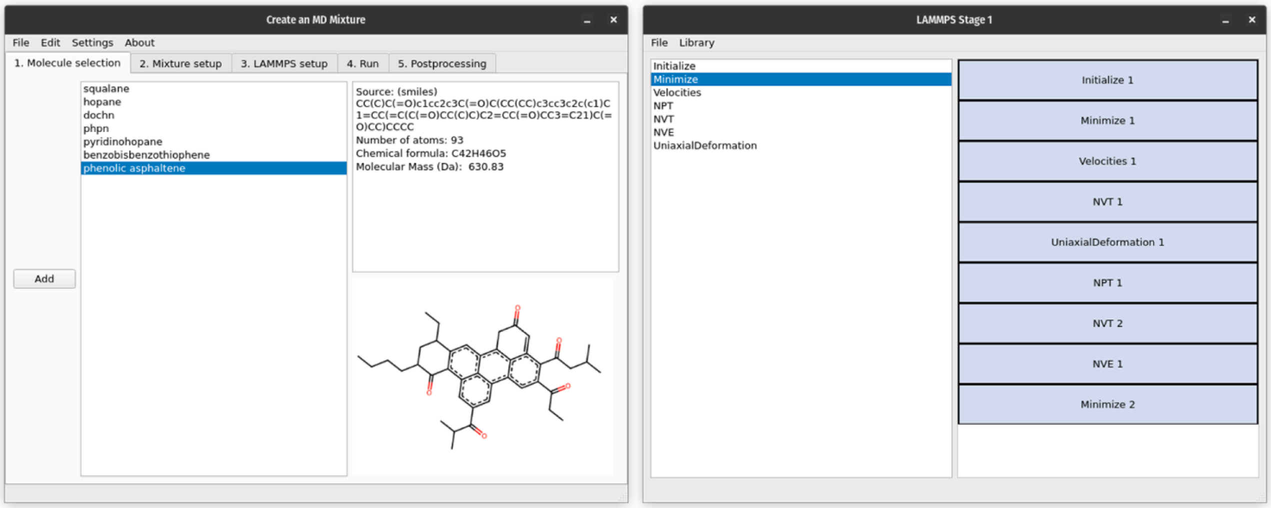


Fig. 1. Schematic of a standard job execution in the program, detailing directories, files, and class relationships. Bolded rectangles denote independent processes; blue rectangles specify the core directories in Scymol's root. Solid arrows indicate direct code interactions, and dashed arrows indicate signal-based communication.



squalane
hopane
dochn
phpn
pyridinohopane
benzobisbenzothiophene
phenolic asphaltene

```
Source: (smiles)
CC(C)C(=O)c1cc2c3C(=O)C(CC(CC)c3cc3c2c(c1)C
1=CC(=C(C(=O)CC(C)C)C2=CC(=O)CC3=C21)C(=
O)CC)CCCC
Number of atoms: 93
Chemical formula: C42H46O5
Molecular Mass (Da):  630.83
```

Initialize
Minimize
Velocities
NPT
NVT
NVE
UniaxialDeformation

Fig. 2. Depiction of Tab 1 and the LAMMPS Flowchart Interface in Scymol. The user interface in Scymol is partitioned into tabs, facilitating the sequential preparation of simulation inputs. These tabs also serve as a structured guide through the various functionalities offered by the program.

for displaying windows where users can adjust parameters specific to the LAMMPS substages initialized by the user.

2.1.2.3. molecule.py. Allows for the creation of molecule instances when a user imports molecules into the system. Utilizing RDKit's utilities, this class holds pertinent information about the molecule while checking if it possesses a stable structure, well-defined bonds, angles, dihedrals, and so forth, ensuring its compatibility with LAMMPS.

2.1.2.4. static_functions.py. A standalone Python module comprising utility functions that are independent of class states or methods. These functions execute common tasks like computing chemical information and file input/output operations.

2.1.2.5. /dialog_windows. This directory contains modules essential for initializing dialog windows that are directly triggered from the MainWindow instance. For example, a dialog window for entering the SMILES notation of a molecule is initialized from this directory.

2.1.2.6. /custom_widgets. While Scymol predominantly employs standard Qt widgets, this directory focuses on the creation and customization of widgets that introduce custom functionalities or that override the behavior of standard widgets.

2.1.2.7. /context_menus. Given the potential complexity of context menus, which may offer a multitude of options and necessitate the initialization of various submodules, this directory organizes the Python files required for initializing instances of context menus.

2.1.3. /front2Back

The front2back/ directory facilitates a link between the frontend's state and the backend's input requirements for constructing and executing molecular systems. This directory comprises several Python modules, each serving distinct functionalities:

2.1.3.1. BackendConnector.py. Manages the translation of UI interactions into an *inputs.py* file and handles the lifecycle of backend simulation processes. Jobs are stored in *output/<job_id>*.

2.1.3.2. RunningProcessDialog.py. Creates a dialog window instance to display simulation output from a backend-generated log file and allows for simulation termination.

2.1.3.3. BackendThread.py. Spawns a subprocess instance to run newly submitted jobs, capturing *stdout* and *stderr* messages in a log file for real-time feedback.

2.1.3.4. DataExtractor.py. Extracts data from UI widgets into a dictionary, facilitating the generation of the *inputs.py* file by BackendConnector.py for submitting simulations.

2.1.4. /backend

The /backend directory serves as a specialized module in Scymol, responsible for the core functionalities required for executing simulations. This module is designed to function independently, relying solely on the inputs.py file prepared by the /frontend for running a job. The directory is organized into various Python files and modules, each tailored for specific tasks:

2.1.4.1. lammps_commands.py. This file houses the *LammpsCommands* class, which offers methods for defining and appending a range of LAMMPS commands to a simulation script. It encompasses a large number of basic LAMMPS scripting protocols, such as LAMMPS *fixes* and *computes [34]*, commonly used to build LAMMPS scripts. Structured comments, the addition of custom code, and variable management are also integral to this class.

2.1.4.2. lammps_stages.py. This file introduces the *LammpsStage* class, central to generating full LAMMPS scripts by calling commands from the *LammpsCommands* instance.

2.1.4.3. molecule.py. This file contains the *Molecule* class, which creates instances for each molecule based on the SMILES string from the *inputs.py* file. The class uses Rdkit's functionalities to initialize, minimize, and equilibrate molecules into valid conformers, just like in the *Molecule* class of the *frontend/*.

2.1.4.4. mixture.py. This file contains the *Mixture* class which stores a collection of molecule instances initialized using the *Molecule* class. It provides methods involving mixture-wide computations, including molecule sorting, force field assignment, and potential energy calculations, among others.

2.1.4.5. pysimm_system.py. This class interfaces with the PySIMM library to translate RdKit objects into LAMMPS-compatible inputs, thereby facilitating the molecular system's initialization for simulations using LAMMPS.

2.1.4.6. inputs.py. This is a fundamental file containing comprehensive input parameters and configuration settings, serving as the cornerstone for the backend's operations. All the information required by the /backend to run a job through the backend is contained here.

2.1.4.7. log_functions.py. This module is responsible for logging job executions in the backend. It enables the creation of log files with varying levels of details (*minimal, normal,* or *verbose*), aiding in debugging and tracking the progress of submitted jobs. The log file is stored in *output/<job_id>/log.*

2.1.4.8. static_functions.py. Similar to its counterparts in the *frontend* and *front2back* modules, it offers functions for common tasks, distinct from other modules to maintain the backend's independence.

2.1.4.9. main.py. This Python script is pivotal for running a complete job. It operates independently of the *frontend* and *front2back* modules, relying solely on the *inputs.py* file. It can be called by using the *python main.py –job_id ⟨job_id⟩* command.

2.1.4.10. /lammps_presets_library.py. This module contains predefined LAMMPS substages, offering a flexible and expandable set of options for simulations. The included presets involve *Initialize, Minimize, Velocities, NPT, NVT, NVE*, and *Uniaxial Deformation*, all widely used LAMMPS routines [34] in the scientific community [1,8].

2.2. Software functionalities

Scymol has a user interface that guides users through the necessary steps for simulation setup, ensuring complete utilization of the software's features. While the interface is important for gathering and organizing user inputs, the computational capabilities are primarily contained within the backend architecture. These core functionalities, along with their respective user interface elements, are presented as follows.

2.2.1. Frontend

2.2.1.1. Molecule Selection (Tab 1). This tab enables users to create a molecular set for use in their simulation. Molecules can be added either through their SMILES notation or from a list of predefined molecule structure files such as .mol and .pdb. As molecules are loaded, they are added to a "List of Molecules" section (1.1), their key chemical information is displayed in a "Description" section (1.2), and an interactive 2D representation appears in a "Drawing" section (1.3). The order of the molecules in the list, which users can adjust via drag-and-drop, is reflected in the generated LAMMPS input files.

2.2.1.2. Mixture Setup (Tab 2). This tab provides tools for configuring a molecular mixture. A "Setup" table (2.1) lists the molecules loaded in Tab 1 and allows users to specify the number of each molecule and their initial orientation. The "Settings" section (2.2) offers parameters crucial for the initial placement of molecules in the simulation box without overlap. An "Information" section (2.3) provides an overview of the mixture, including the total number of molecules, average molecular mass, and molecular formula.

2.2.1.3. LAMMPS Setup (Tab 3). This tab consists of two sections: "Force Field" (3.1) and "Stages" (3.2). The Force Field section offers a selection of commonly used force fields, namely GAFF and GAFF2 [35], PCFF [36], CHARMM [37], and TIP3P [38]. The program loads the atomic types and charges and checks if the selected force field is applicable to cover all the interactions in the system. The Stages list lets users design a sequenced list of LAMMPS stages. Each stage added is expandable, revealing a LAMMPS Flowchart menu, where users can set up a sequence of LAMMPS substages. A library is included to further aid users in setting up commonly used LAMMPS sequences (e.g., a heat cycle). Furthermore, each substage can be explored into, presenting configuration options for each substage (e.g., to set the temperature in an NVT substage).

2.2.1.4. Run (Tab 4). This tab allows users to specify the location of LAMMPS (setting 4.1) and the parallelization library (e.g., MPI) (setting 4.2). By default, the program includes precompiled versions of LAMMPS and OpenMPI, but users can choose their own versions. Setting 4.3 permits users to select the number of cores to be allocated for the job.

2.2.1.5. Postprocessing (Tab 5). Although not a primary focus of the program, this tab enables users to view log files from previous jobs and to display computed properties, thereby offering a preliminary view of both computational and physical aspects of the simulations.

2.2.2. Backend

The backend follows a specific workflow as outlined in Fig. 3. The backend of Scymol supports two operational modes. The first mode focuses on initializing new atomistic systems from scratch and performing LAMMPS simulations, while the second continues simulations using pre-existing atomistic systems, such as those generated by prior LAMMPS runs. These modes are selected based on the configuration of the inputs. py file, which provides all the necessary parameters to define and execute a simulation workflow. This design allows Scymol's backend to function independently of the frontend or front2back components, ensuring flexibility and usability in diverse simulation scenarios. The following describes the detailed functionalities involved in initializing, setting up, and running simulations under the first mode of operation.

2.2.2.1. Molecule Initialization. Individual molecules are parsed from standard atomistic modeling formats into fully stable 3D conformers using RDKit. This involves three key steps: (a) initializing atomic positions in three-dimensional space, (b) determining bonding connectivity and bond order, and (c) generating multiple conformers and selecting the one with the lowest intramolecular energy. This ensures the stability of each molecule before its inclusion in the simulation box.

2.2.2.2. Mixture Initialization. The initialized molecules are placed within a low-density simulation box using a low-discrepancy distribution method. The intermolecular potential energy of the system is evaluated to ensure there is no atomic overlap. If the energy exceeds a

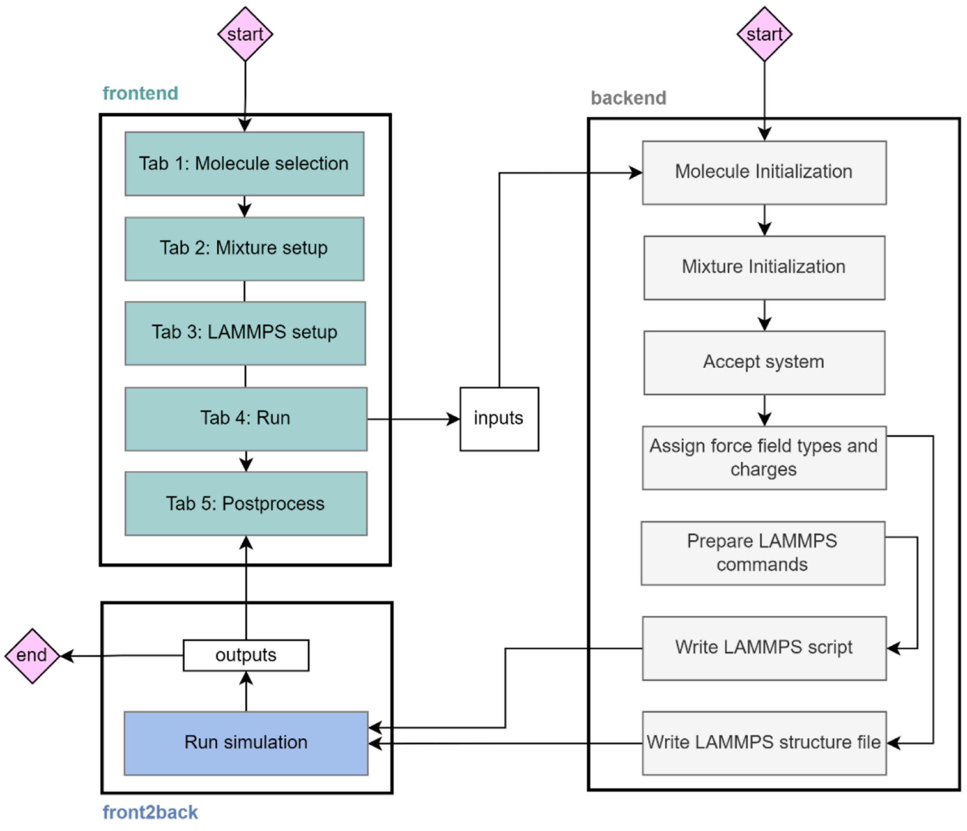


Fig. 3. Diagram showing how the software functionalities combine to aid users in generating LAMMPS input files and running the simulations as needed.

predefined threshold, the placement process is repeated. This iterative approach guarantees a physically plausible starting configuration for subsequent simulations.

2.2.2.3. Assignment of Force Field Types and Charges. Atom types and charges are assigned to the molecules using the PySimm package, which provides a library of predefined force fields. Each unique molecule type is processed using PySimm's apply_forcefield() function, and the resulting parameters are then reused for identical molecules in the system to optimize computational efficiency.

2.2.2.4. LAMMPS Scripting. Scymol generates a series of LAMMPS commands organized into structured simulation routines, which are compiled into scripts based on user specifications. These routines include fundamental LAMMPS scripting commands such as *units, boundary, atom_style, pair_style, bond_style*, and various fixes (e.g., *NVT, NPT, NVE*). Each routine is designed to be self-contained, enabling seamless sequencing and uninterrupted execution of LAMMPS simulations.

2.2.2.5. Generation of Data and Input files. Two essential files are created for each simulation: the structure.dat file, which contains atomic coordinates, types, bonds, angles, dihedrals, and box dimensions, and the input.dat file, which specifies the LAMMPS commands to be executed. These files are generated using a combination of RDKit and PySimm to ensure consistency and compatibility with LAMMPS.

2.2.2.6. Execution of LAMMPS/MPI Jobs. All required simulation files are structured within the execution environment to enable standalone operation by MPI/LAMMPS. Although the backend initiates the process and remains connected to retrieve results, it is designed to allow MPI/LAMMPS to function independently. Additional routines can be executed sequentially as dictated by the simulation workflow. Additional routines, including those not exclusive to LAMMPS, can be executed sequentially as dictated by the simulation workflow.

3. Illustrative examples

3.1. Example 1

The objective of this example is to create an equilibrated model of a bitumen sample representative of AA1 70/100 bitumen from Supplier "A." The goal is to validate its equilibrium density against both experimental measurements and models obtained using conventional Molecular Dynamics (MD) methodologies. The molecular structures and mass compositions are based on the work of Greenfield et al. [39,40], consisting of 12 different molecule types, totaling 608 molecules. Detailed information about the molecular structures, along with their SMILES notations, is provided in Table 1.

To initialize and equilibrate the system, the molecules are first placed in a simulation box with a low initial density—approximately one-tenth of the expected final bitumen density—to minimize atomic overlaps. The system is then imported into LAMMPS, where the GAFF2 force field is applied. The initialization process involves a sequence of compression and equilibration stages, beginning with uniaxial compression at progressively decreasing rates to simulate true strain. This is followed by equilibration under NPT and NVT ensembles to achieve a stable system at 298 K and 1 atm. Finally, an NVE stage is performed to evaluate system stability and ensure energy conservation, allowing for results collection. This multistep approach is widely accepted for modeling condensed amorphous molecular systems.

Using the Scymol framework, the process begins by entering the names and SMILES notations of the 12 molecule types into the Molecule Selection tab (Tab 1). Scymol automatically generates the molecular objects and 2D representations. The desired quantities of each molecule type are specified in the Mixture Setup tab (Tab 2), as shown in Table 1. Molecules are placed in the simulation box using the Sobol Distribution Method, ensuring even spatial distribution while avoiding overlaps. All other parameters for this step are left at their default values. In the LAMMPS Setup tab (Tab 3), the GAFF2 force field is selected, and a LAMMPS stage is added to the simulation workflow.

The specific stages of the LAMMPS workflow are defined in the LAMMPS Flowchart Window. Users can either select the predefined "Uniaxial Compression" stage from the library or manually assemble the required sequence of substages: Initialize, Minimize, Assign Velocities, NVT, Uniaxial Deformation, NPT, NVT, and NVE. Each stage is configured with its respective parameters to match the experimental conditions.

The simulation is initiated by clicking the "Run" button (4.4), which triggers the associated subprocesses. A Running Process Dialog is displayed, allowing users to monitor progress in real time. Upon completion, the output directory contains all relevant data, including the initial system configuration, LAMMPS input and output files, trajectory files, and energy logs. These outputs can be used for further analysis, such as comparing the simulated density with experimental values or examining energy and pressure profiles during equilibration.

3.2. Example 2

In the second example, the density of bitumen is measured at elevated temperatures of 60, 135, and 160°C. Rather than being confined to the initialization techniques native to the software, the program can seamlessly continue from existing LAMMPS simulations. To accomplish this, the equilibrated molecular model generated in Example 1 serves as the starting point for this simulation. The system undergoes a sequence of heating and equilibration substages designed to elevate the temperature while allowing sufficient time for pressure

Table 1
The molecules selected to represent bitumen in the examples of this study and their corresponding SMILES notation.

Name	Number	SMILES
Squalane	16	CC(CCCC(C)CCCCC(C)CCCC(C)CCCC(C)C)CCCC(C)C
Hopane	8	CC12CCC3C(C)(CCCC3(C)C)C1CCC1C2(C)CCC2C(CCC12C)C(C)CCCCCC
Dioctylcyclohexane naphthalene	168	CCCCCCCCC1Cc2cc3ccccc3cc2CC1CCCCCCCC
Perhydrophenanthrene naphthalene	160	Cc1cc2cc3C4CCC5CCC(CC5C4CCc3cc2cc1CC)Cc1cccc(CCC)c1
Quinolinohopane	24	CC12CCC3C(C)(CCCC3(C)C)C1CCC1C2(C)CCC2c3c(nc4ccc(cc4c3C)CCCCC)CC12C
Thioisorenieratane	24	Cc1c(CCC(C)CCCC(C)CCc2sc(CCC(C)CCc3c(C)c(C)ccc3C)c(C)c2)c(C)ccc1C
Benzobisbenzothiophene	104	O=S1c2cc3c(cc2c2ccccc12)sc1ccccc13
Pyridinohopane	24	CC12CCC3C(C)(CCCC3(C)C)C1CCC1C2(C)CCC2c3c(nc(cc3C)CCCCC)CC12C
Trimethylbenzeneoxane	32	CC1(Oc2c(CC1)c(C)cc(C)c2C)CCCC(C)CCCC(C)CCCC(C)C
Phenolic asphaltene	16	CC(C)Cc1cc2c3c(cc4c(CC(CC4CC)CCCC)c3c1)c1cc(O)cc3c1c2cc(CCC)c3CCC(C)C
Pyrrolic asphaltene	16	CC(CCc1cc2c(cc1CCC)c1c3c4c5c6c(c7cc[NH]c7cc61)c(cc5c(c1CC(CC)c5cc(CCC(C)C)c6CCc2c3c6c5c41)C(C)CCC)CCCCCC(C)CC)CC
Thiophenic asphaltene	16	Cc1cc(CCC(C)CCC)c2CC(CCC)c3cc4sc5c6c4c4c3c2c1cc4c1cc2c(CC(CC2CC)CCCC)c(cc5CC(C)C)c61

equilibration to reach reliable density values.

To integrate the data from Example 1 into Scymol, the 'File, Load, from previous lammps' option in the top menu bar is utilized. This action imports both the structural data file and the final trajectory data from the first example. All default settings are retained, and the simulation is initiated in the usual manner.

3.3. Example 3

In the third example, the objective is to demonstrate the capacity of Scymol's backend module to function autonomously from its frontend, particularly in a high-performance computing (HPC) environment that supports standard installations of Python, MPI, and LAMMPS. This is especially relevant for computational workloads requiring enhanced performance, as in the case of the multiple heating-equilibration cycles in Example 2. The simulation is executed on an HPC system (DelftBlue, operated under a SLURM [41] job scheduler and RedHat operating system) using 32 CPU cores.

To start, a new directory (e.g., /root) is created, and the backend/ folder from Example 2 is copied into it. Key parameters in the backend/inputs.py file are adjusted to align with the HPC's configurations. Specifically, the number of processes is set to 32, the MPI location is set to either 'srun' or 'mpiexec' (with 'srun' being recommended for SLURM-managed systems [42]), and the LAMMPS location is set to 'lmp'. This level of customization offers experienced users the ability to manually adjust simulation parameters directly in the inputs.py file, without the need for the frontend interface.

A job script tailored for HPC execution is manually created. The script must import Python, MPI, and LAMMPS, and initiate the simulation with a command akin to 'srun python main.py –job_id <int: job_id>'. The job proceeds to execute from start to finish in a manner analogous to using Scymol's frontend. This example not only establishes the backend's capability to run independently on an HPC, but also illustrates the ease with which experienced users can modify input parameters to generate different molecular systems efficiently.

4. Impact

Scymol is an open-source Python package designed to facilitate the setup and execution of molecular dynamics simulations using LAMMPS. By addressing the complexities of simulation workflows, including molecule initialization, force field assignment, and script preparation, it has proven instrumental in both research and industrial applications. For instance, it supported the KPE-CEAB project in conducting sensitivity analyses on bituminous materials and has been utilized by companies investigating the impact of additives in hydrocarbon mixtures [43,44].

The development of Scymol is rooted in the need to provide a structured and accessible approach to molecular dynamics simulations. Its architecture integrates the collective expertise of our research group, consolidating computational methods, data, and findings into a unified and shareable framework. This approach not only facilitates reproducibility and transferability of simulations but also ensures that workflows are cumulative, enabling the systematic development of new methodologies and applications.

Its dual-interface design—offering a graphical user interface for ease of use and a backend for more advanced/automated job configurations—ensures usability across a wide range of expertise levels. This makes Scymol a practical solution for researchers aiming to incorporate molecular dynamics simulations into their work with minimal barriers, especially for those involved in Civil and Pavement Engineering fields.

5. Future Direction

As of now, Scymol is self-sufficient and performs the tasks needed; however, it serves as the foundation for the development of a more advanced software package. The future development of Scymol is expected to follow two primary paths: (1) our group will extend this version of Scymol to address specific needs in Civil and Pavement Engineering, and (2) a broader community of developers may further enhance Scymol to tackle general molecular simulation challenges. The following outlines potential areas for future work:

- **Advanced LAMMPS Routines**

  The integration of more advanced LAMMPS routines that extend beyond basic subroutines, such as NPT or NVT dynamics, is envisioned. These routines would incorporate simpler LAMMPS commands into encapsulated workflows, enabling the setup of simulation schemes for determining complex, but critical material properties, including cohesive energy density, shear viscosity, thermal conductivity, among others. These features would be accessible through the LAMMPS Flowgraph Window.

- **Non-LAMMPS Routines**

  Currently, the functionality in Tab 3 – Stage 3.2 is limited to invoking only LAMMPS-related runs. Future versions could incorporate additional tasks not related to LAMMPS to enhance task automation at runtime. For example, this could encompass stages for preparing molecular systems, as currently implemented in Tabs 1 and 2 of Scymol, enabling a more integrated approach to system initialization and manipulation with LAMMPS executions.

- **Enhanced Molecular File Input/Output System**

  To overcome limitations in file compatibility, Scymol could adopt an intermediary package like OpenBabel. This would enable support for a wide range of molecular file formats and facilitate a more robust access to a number of molecular input files.

- **Improved Force Field Handling**

  Developing a native system for force field atomic typing and charge assignment would enhance Scymol's applicability. Currently reliant on Pysimm, this improvement would allow the integration of new force fields, including all-atom, united-atom, and coarse-grained models, expanding the software's versatility.

- **3D Visualization Tools**

  The implementation of a 3D visualization tab, potentially utilizing OpenGL, would allow users to visualize and manipulate atomistic systems interactively. Such a feature would greatly enhance user experience and provide intuitive insights into the progress of molecular simulations.

- **Robust Job Submission System**

  Scymol could benefit from a networked job submission framework, enabling the execution of tasks on remote servers. Such functionality would streamline job management in high-performance computing environments and foster collaborative workflows.

- **Implementation of other Python-based atomistic design packages**

  As discussed earlier in this manuscript, numerous software solutions effectively address specific niche areas within computational chemistry. Similar to how Scymol utilizes OpenBabel for file format standardization and normalization, components of other packages could be integrated to enhance and expand Scymol's functionality.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work the author(s) used OpenAI's ChatGPT4.0 to simplify verbose paragraph descriptions. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

CRediT authorship contribution statement

Eli I. Assaf: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Elsa Maalouf:** Writing – review & editing, Supervision, Project administration, Conceptualization. **Xueyan Liu:** Writing – review & editing, Supervision, Resources, Project administration, Funding acquisition, Formal analysis. **Peng Lin:** Writing – review & editing, Validation, Supervision, Data curation, Conceptualization. **Sandra Erkens:** Writing – review & editing, Supervision, Resources, Project administration, Methodology, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper

Acknowledgements

This paper/article is created under the research program Knowledge-based Pavement Engineering (KPE). KPE is a cooperation between Rijkswaterstaat, TNO, and TU Delft in which scientific and applied knowledge is gained about asphalt pavements and which contributes to the aim of Rijkswaterstaat to be completely climate neutral and to work according to the circular principle by 2030. The opinions expressed in these papers are solely from the authors.

References

[1] Shell MS. Thermodynamics and statistical mechanics: an integrated approach. Cambridge University Press; 2015.
[2] Van Gunsteren WF, Berendsen HJ. Computer simulation of molecular dynamics: methodology, applications, and perspectives in chemistry. Angew Chem Int Ed Engl 1990;29(9):992–1023.
[3] Vlachakis D, Bencurova E, Papangelopoulos N, Kossida S. Current state-of-the-art molecular dynamics methods and applications. Adv Protein Chem Struct Biol 2014; 94:269–313.
[4] Thompson AP, Aktulga HM, Berger R, Bolintineanu DS, Brown WM, Crozier PS, in 't Veld PJ, Kohlmeyer A, Moore SG, Nguyen TD, Shan R, Stevens MJ, Tranchida J, Trott C, Plimpton SJ. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. Comput Phys Commun 2022;271:108171.
[5] Abraham MJ, Murtola T, Schulz R, Páll S, Smith JC, Hess B, Lindahl E. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX 2015;1-2:19–25.
[6] Sharma S, Kumar P, Chandra R. Chapter 7 - Applications of BIOVIA materials studio, LAMMPS, and GROMACS in various fields of science and engineering. In: Sharma S, editor. Molecular dynamics simulation of nanocomposites using biovia materials studio, lammps and gromacs. Elsevier; 2019. p. 329–41.
[7] Lam SK, Pitrou A, Seibert S. Numba: A llvm-based python jit compiler. In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC; 2015. p. 1–6.
[8] Ercolessi F. A molecular dynamics primer, spring college in computational physics. Trieste: ICTP; 1997. p. 19.
[9] Stegailov VV, Orekhov ND, Smirnov GS. HPC hardware efficiency for quantum and classical molecular dynamics. In: Parallel Computing Technologies: 13th International Conference, PaCT 2015, Petrozavodsk, Russia, August 31-September 4, 2015, Proceedings 13. Springer; 2015. p. 469–73.
[10] FrantzDale B, Plimpton SJ, Shephard MS. Software components for parallel multiscale simulation: an example with LAMMPS. Eng Comput 2010;26:205–11.
[11] Akkermans RL, Spenley NA, Robertson SH. Monte Carlo methods in materials studio. Mol Simul 2013;39(14-15):1153–64.
[12] France-Lanord A, Rigby D, Mavromaras A, Eyert V, Saxe P, Freeman C, Wimmer E. MedeA®: Atomistic simulations for designing and testing materials for micro/nano electronics systems. In: 2014 15th International Conference on Thermal, Mechanical and Mulit-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE). IEEE; 2014. p. 1–8.
[13] Giannozzi P, Baroni S, Bonini N, Calandra M, Car R, Cavazzoni C, Ceresoli D, Chiarotti GL, Cococcioni M, Dabo I, Dal Corso A, de Gironcoli S, Fabris S, Fratesi G, Gebauer R, Gerstmann U, Gougoussis C, Kokalj A, Lazzeri M, Martin-Samos L, Marzari N, Mauri F, Mazzarello R, Paolini S, Pasquarello A, Paulatto L, Sbraccia C, Scandolo S, Sclauzero G, Seitsonen AP, Smogunov A, Umari P, Wentzcovitch RM. QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials. J Phys Condens Matter 2009;21(39):395502.
[14] Hanwell MD, Curtis DE, Lonie DC, Vandermeersch T, Zurek E, Hutchison GR. Avogadro: an advanced semantic chemical editor, visualization, and analysis platform. J Cheminform 2012;4(1):17.
[15] 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.
[16] Humphrey W, Dalke A, Schulten K. VMD: visual molecular dynamics. J Mol Graph 1996;14(1):33–8.
[17] Stukowski A. Visualization and analysis of atomistic simulation data with OVITO–the Open Visualization Tool. Model Simul Mat Sci Eng 2009;18(1):015012.
[18] Brooks BR, Brooks III CL, Mackerell Jr AD, Nilsson L, Petrella RJ, Roux B, Won Y, Archontis G, Bartels C, Boresch S. CHARMM: the biomolecular simulation program. J Comput Chem 2009;30(10):1545–614.
[19] Cummings PT, McCabe C, Iacovella CR, Ledeczi A, Jankowski E, Jayaraman A, Palmer JC, Maginn EJ, Glotzer SC, Anderson JA, Siepmann JI, Potoff J, Matsumoto RA, Gilmer JB, DeFever RS, Singh R, Crawford B. Open-source molecular modeling software in chemical engineering focusing on the Molecular Simulation Design Framework. AIChE J 2021;67(3):e17206.
[20] Ziolek RM, Santana-Bonilla A, López-Ríos de Castro R, Kühn R, Green M, Lorenz CD. Conformational heterogeneity and interchain percolation revealed in an amorphous conjugated polymer. ACS Nano 2022;16(9):14432–42.
[21] Abbott LJ, Hart KE, Colina CM. Polymatic: a generalized simulated polymerization algorithm for amorphous polymers. Theor Chem Acc 2013;132:1–19.
[22] Cao H, Cao X, Zhao X, Guo D, Liu Y, Bian J. Molecular dynamics simulation of wax molecules aggregational crystallization behavior during cooling of crude oil mixture. Case Stud Therm Eng 2022;37:102298.
[23] Gao Y, Zhang Y, Yang Y, Zhang J, Gu F. Molecular dynamics investigation of interfacial adhesion between oxidised bitumen and mineral surfaces. Appl Surf Sci 2019;479:449–62.
[24] Lu G, Zhang X, Shao C, Yang H. Molecular dynamics simulation of adsorption of an oil-water-surfactant mixture on calcite surface. Pet Sci 2009;6:76–81.
[25] Ma X, Wu J, Liu Q, Ren W, Oeser M. Molecular dynamics simulation of the bitumen-aggregate system and the effect of simulation details. Constr Build Mater 2021;285:122886.
[26] Ren S, Liu X, Lin P, Gao Y, Erkens S. Molecular dynamics simulation on bulk bitumen systems and its potential connections to macroscale performance: Review and discussion. Fuel 2022;328:125382.
[27] Xu M, Yi J, Feng D, Huang Y. Diffusion characteristics of asphalt rejuvenators based on molecular dynamics simulation. Int J Pavement Eng 2019;20(5):615–27.
[28] Long Z, Tang X, Ding Y, Miljković M, Khanal A, Ma W, You L, Xu F. Influence of sea salt on the interfacial adhesion of bitumen–aggregate systems by molecular dynamics simulation. Constr Build Mater 2022;336:127471.
[29] Eastman P, Swails J, Chodera JD, McGibbon RT, Zhao Y, Beauchamp KA, Wang L-P, Simmonett AC, Harrigan MP, Stern CD. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. PLoS Comput Biol 2017;13(7): e1005659.
[30] Landrum G. RDKit: A software suite for cheminformatics, computational chemistry, and predictive modeling. Greg Landrum 2013;8:31.
[31] Fortunato ME, Colina CM. pysimm: A python package for simulation of molecular systems. SoftwareX 2017;6:7–12.
[32] Summerfield M. Rapid gui programming with python and qt: the definitive guide to pyqt programming (paperback). Pearson Education; 2007.
[33] Clarke L, Glendinning I, Hempel R. The MPI message passing interface standard. In: Programming Environments for Massively Parallel Distributed Systems: Working Conference of the IFIP WG 10.3, April 25–29, 1994. Springer; 1994. p. 213–8.
[34] LAMMPS Official Documentation. https://docs.lammps.org/fix.html. (Accessed 10th November 2023.
[35] Sprenger K, Jaeger VW, Pfaendtner J. The general AMBER force field (GAFF) can accurately predict thermodynamic and transport properties of many ionic liquids. J Phys Chem B 2015;119(18):5882–95.
[36] Sun H, Mumby SJ, Maple JR, Hagler AT. An ab initio CFF93 all-atom force field for polycarbonates. J Am Chem Soc 1994;116(7):2978–87.
[37] Vanommeslaeghe K, Hatcher E, Acharya C, Kundu S, Zhong S, Shim J, Darian E, Guvench O, Lopes P, Vorobyov I. CHARMM general force field: A force field for drug-like molecules compatible with the CHARMM all-atom additive biological force fields. J Comput Chem 2010;31(4):671–90.
[38] Mark P, Nilsson L. Structure and dynamics of the TIP3P, SPC, and SPC/E water models at 298 K. J Phys Chem A 2001;105(43):9954–60.
[39] Li DD, Greenfield ML. Chemical compositions of improved model asphalt systems for molecular simulations. Fuel 2014;115:347–56.
[40] Ren S, Liu X, Lin P, Erkens S, Xiao Y. Chemo-physical characterization and molecular dynamics simulation of long-term aging behaviors of bitumen. Constr Build Mater 2021;302:124437.
[41] Yoo AB, Jette MA, Grondona M. Slurm: Simple linux utility for resource management. In: Workshop on job scheduling strategies for parallel processing. Springer; 2003. p. 44–60.

[42] Comprés I, Mo-Hellenbrand A, Gerndt M, Bungartz H-J. Infrastructure and api extensions for elastic execution of mpi applications. In: Proceedings of the 23rd European MPI Users' Group Meeting; 2016. p. 82–97.

[43] Assaf EI, Liu X, Lin P, Ren S, Erkens S. Predicting the properties of bitumen using machine learning models trained with force field atom types and molecular dynamics simulations. Mater Des 2024;246:113327.

[44] Assaf EI, Liu X, Lin P, Ren S, Erkens S. Predicting the diffusion coefficients of rejuvenators into bitumens using molecular dynamics, machine learning, and force field atom types. Mater Des 2024;248:113502.