JCTC Journal of Chemical Theory and Computation

pubs.acs.org/JCTC Article

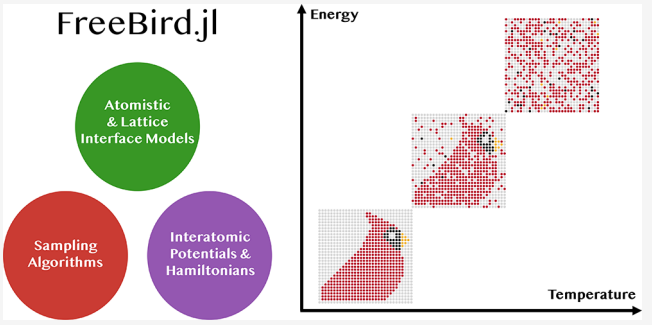
FreeBird.jl: An Extensible Toolbox for Simulating Interfacial Phase Equilibria

Ray Yang, Junchi Chen, Douglas Thibodeaux, and Robert B. Wexler*

Cite This: *J. Chem. Theory Comput.* 2025, 21, 10765−10779 Read Online

ACCESS | Metrics & More | Article Recommendations

ABSTRACT: We present `FreeBird.jl`, an extensible Julia-based platform for computational studies of phase equilibria at generic interfaces. The package supports a range of system configurations, from atomistic solid surfaces to coarse-grained lattice−gas models, with energies evaluated using classical interatomic potentials or lattice Hamiltonians. Both atomistic and lattice systems accommodate single- or multicomponent mixtures with flexibly definable surface and lattice geometries. Implemented sampling algorithms include nested sampling, Wang−Landau sampling, Metropolis Monte Carlo, and, for tractable lattice systems, exact enumeration. Leveraging Julia's type hierarchies and multiple dispatch, `FreeBird.jl` provides a modular interface that allows seamless integration of system definitions, energy evaluators, and sampling schemes. Designed for flexibility, extensibility, and performance, `FreeBird.jl` offers a versatile framework for exploring the thermodynamics of interfacial phenomena.



1. INTRODUCTION

Interfaces are central to technologies addressing today's most pressing challenges, from sustainable heterogeneous catalysis[1] to high-performance semiconductor devices[2] and biomedical innovation.[3] Meeting these challenges requires a multiscale understanding of interfacial processes that can bridge fundamental insights to practical design rules and, ultimately, to commercial technologies.[4,5] At atomic length scales, methods such as low-energy electron diffraction[6] and advanced electron microscopy[7] can resolve solid−vacuum interfaces with sub-Å precision. However, these highly sensitive ex situ probes remain difficult to apply under realistic conditions in which the interface is buried beneath a dense phase or driven far from equilibrium, leaving critical gaps in spatial resolution and experimental accessibility.[8,9]

Because high-resolution in situ probes remain scarce,[10,11] theoretical and computational approaches have become indispensable for mapping the configurational landscape of interfaces and guiding materials design.[12−15] Modeling an interface under working conditions is essentially a non-equilibrium surface problem whose dynamics depend on the poorly constrained, synthesis-specific starting structure.[16,17] To address this, researchers typically begin by establishing an equilibrium thermodynamic baseline,[18,19] which entails a series of decisions: the level of energy evaluation (first-principles versus classical force fields)[20,21] and the depth of configurational sampling required to obtain entropies and free energies (harmonic, quasi-harmonic, or fully anharmonic treatments; atomistic versus lattice representations).[22−25]

Most realistic surface problems are too complex for exhaustive first-principles treatment, forcing researchers to navigate an accuracy−sampling trade-off.[18,26] Ab initio thermodynamics, the workhorse for predicting surface phase diagrams, typically evaluates only tens to thousands of candidate structures within a practical computing budget, with selections often guided by chemical intuition.[18,19] Structure-search algorithms such as particle-swarm,[27] Bayesian,[28,29] and genetic[30,31] optimizers expand the candidate pool but still assume a sharply truncated configuration space in which atoms vibrate harmonically about fixed lattice sites.[22,24] This harmonic perspective understates the structural and compositional agility of catalytic surfaces; in many cases, comprehensive sampling with well-parametrized classical potentials can reveal low-energy motifs that sparse, high-accuracy DFT surveys may miss.[20,21,32,33]

Although progress has been made toward systematically improvable thermodynamic sampling of surface phases,[32,34,35]

Received: August 14, 2025
Revised: September 25, 2025
Accepted: September 26, 2025
Published: October 20, 2025

https://doi.org/10.1021/acs.jctc.5c01348
J. Chem. Theory Comput. 2025, 21, 10765−10779

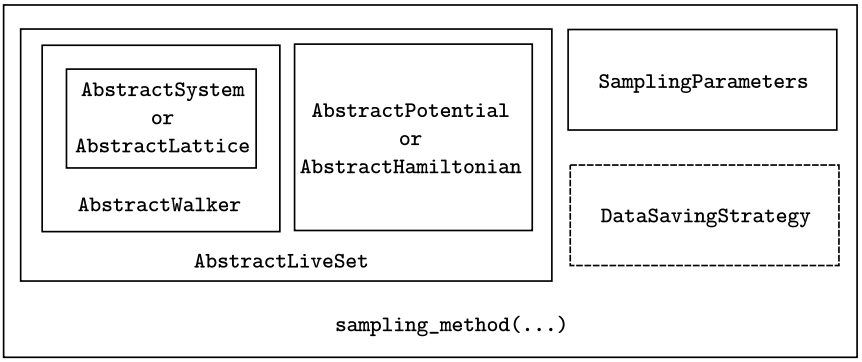


Figure 1. Schematic overview of the `FreeBird.jl` data structure. To perform a sampling calculation, the required settings are specified in `SamplingParameters`. Typically, either a live set or a single initial walker, containing the embedded system description, is used. The system is represented by an `AbstractSystem` or `AbstractLattice`, wrapped by an `AbstractWalker`. Energies are evaluated using an `AbstractPotential` or `AbstractHamiltonian`, which is attached to the live set or walker. Optional data output behavior is configured via a `DataSavingStrategy`.

the field still lacks a fast, reproducible benchmarking platform dedicated to interfacial structure prediction, applicable to solid−vacuum/gas systems as well as defects at solid−solid and solid−liquid boundaries. We therefore introduce `FreeBird.jl`, an open-source Julia package featuring a highly parallel, modular architecture that allows users to interchange system resolutions, energy models, and Monte Carlo−style algorithms without rewriting code. Distinct from existing packages,[36−44] it integrates nested sampling, Wang−Landau, Metropolis Monte Carlo, and (for suitable lattice systems) exact enumeration into a single extensible framework. This framework spans both atomistic and lattice representations with interchangeable energy calculators and move sets, enabling direct, reproducible benchmarking and comparison of interfacial phase equilibria across methods. `FreeBird.jl` is an active project; planned updates include superposition-enhanced nested sampling,[45] parallel tempering,[46] transition-matrix Monte Carlo,[47] geometry-aware coarse-graining (e.g., Gay−Berne[48,49] and patchy-particle[50] models), and full GPU acceleration. Our long-term vision is to position `FreeBird.jl` at the core of autonomous, multifidelity workflows (integrating Bayesian experimental design, interface-aware machine-learning potentials, and a surface-structure database) to accelerate the discovery and design of functional interfaces.

This paper is organized as follows: Section 2 outlines the overall structure of the `FreeBird.jl` package, describing its capabilities for handling both continuous atomistic and discretized lattice systems, and briefly reviewing the sampling algorithms it implements. We then present demonstrations and comparisons using various sampling schemes in `FreeBird.jl` to identify phase transitions in several atomistic Lennard-Jones systems (a pure cluster, a binary cluster, and a surface−adsorbate system) as well as in discretized lattice models in two and three dimensions. Section 4 summarizes how `FreeBird.jl` serves as a versatile platform for constructing and modeling interfacial systems, and how its functionality is expected to expand to support additional models, positioning it as a focal point for future surface-related modeling and computational materials design. This work is not intended as a comprehensive documentation of the `FreeBird.jl` package (which is provided separately) but rather as a pedagogical blueprint of its composition, intended to convey the underlying software design philosophy and scientific vision to future users and developers.

2. IMPLEMENTATIONS

In this section, we outline the design philosophy and structure of `FreeBird.jl`, along with the data types it implements to enable efficient execution. We then describe how these data types support the construction of computational models for both continuous atomistic and discretized lattice systems. Finally, we summarize the configurational sampling methods currently available in `FreeBird.jl`, providing a brief introduction to technical details relevant to our demonstrations. All results reported here were obtained using `FreeBird.jl` release version 0.2.1.

2.1. Code. Here, we provide an overview of `FreeBird.jl`'s structure and key functionalities. The code is designed for readability and maintainability through a highly modular, functional architecture, and achieves performance by introducing specialized data types and enforcing type stability within functions. We also enhance sustainability and reproducibility by following established research software engineering best practices. This section does not attempt to detail all aspects of code usage; comprehensive documentation, including examples and tutorials, is available at https://wexlergroup.github.io/FreeBird.jl/, which should be consulted for the most up-to-date information.

2.1.1. Overview of the Code. The `FreeBird.jl` source code is organized into several modules, each containing functions related to a specific theme or topic. These modules fall into a few broad categories, many of which define Julia *abstract types* to establish the conceptual hierarchy of data types. The first category comprises modules for system construction, where new data types are introduced.

- `AbstractWalkers`: Defines the abstract concept of walkers, the central data structure in `FreeBird.jl`, which stores system information such as atomic or lattice configuration, number of components, and energy.
- `AbstractLiveSets`: Defines the abstract concept of live sets (from nested sampling terminology), which group a collection of walkers with an associated energy-evaluating function. More generally, a live set can be viewed as a data type holding both a list of walkers and an energy calculator.

https://doi.org/10.1021/acs.jctc.5c01348
J. Chem. Theory Comput. 2025, 21, 10765−10779

- `AbstractPotentials`: Defines abstract types for interatomic potentials, such as the Lennard-Jones (LJ) potential.
- `AbstractHamiltonians`: Defines abstract types for discrete Hamiltonians, designed for efficient energy evaluation of lattice systems.

The next category comprises functional modules, code that operates on the data types described above to perform sampling or enumeration.

- `SamplingSchemes`: Defines sampling methods, including nested sampling, Wang−Landau sampling, Monte Carlo sampling, and exact enumeration.
- `MonteCarloMoves`: Provides definitions of Monte Carlo moves, such as random walks and swap moves, for use within the sampling methods.
- `EnergyEval`: Specifies how energies are computed by combining a walker with an abstract potential or Hamiltonian.

The penultimate category includes a dedicated module for input/output handling.

- `FreeBirdIO`: Provides functions for converting between atomic structures stored in files and the walker system defined in `FreeBird.jl`. It also defines `DataSavingStrategy` types, which specify how and when output data are saved during a sampling run, as well as the format of the saved data.

Lastly, we include a module providing auxiliary tools.

- `AnalysisTools`: Offers convenience functions for postcalculation analyses. This module is optional, and the code can operate without it.

Figure 1 provides an overview of how the data types defined in the modules are integrated to perform a sampling calculation. Further details about the data types themselves are presented in the following section.

2.1.2. Data Structure and Type Systems. As a just-in-time (JIT) compiled language, Julia can generate specialized, optimized machine code for fast runtime execution when variable data types are explicitly specified in functions. Julia *functions* can also perform different tasks depending on the data types of their inputs, a paradigm known as *multiple dispatch*.

`AbstractWalker` is the core data type in `FreeBird.jl`, with two current subtypes: `AtomWalker{C}` and `LatticeWalker{C}`. An `AbstractWalker` object stores both the system's structure or configuration and its energy, along with supplementary data such as iteration number and particle constraints. Both `AtomWalker{C}` and `LatticeWalker{C}` are parametrized by `C`, the number of components in the system, enabling specialized function implementations. For example, single-component systems are simpler, allowing dedicated, faster methods for `AtomWalker{1}` that avoid the component loops required when $C > 1$.

`AbstractPotential` and `AbstractHamiltonian` are abstract types for defining interactions. For example, `LennardJonesParameterSets` is a subtype of `AbstractPotential` that stores one or more parameter sets for defining an LJ potential. Similarly, `GenericLatticeHamiltonian` is a subtype of `AbstractHamiltonian` that specifies an on-site interaction energy and a list of n-th nearest-neighbor interactions.

`AbstractLiveSet` is a data type that holds a list of `AbstractWalkers` together with an `AbstractPotential` or `AbstractHamiltonian`. When these elements are combined, the energies of the `AbstractWalkers` are automatically calculated and updated. This type is central to performing nested sampling and also serves as a convenient structure for organizing inputs and outputs from other sampling methods.

Type stability is critical for the performance of Julia code. It means that a function's output type depends on the *types* of its inputs rather than their *values*. This property enables the Julia compiler to determine the output type at compile time, allowing efficient memory allocation at runtime. If the return type cannot be inferred, the machine code must accommodate all possible outcomes, which significantly slows execution. Nearly all functions in `FreeBird.jl` are designed to be type stable, except for those in auxiliary modules that are not used during calculations.

2.1.3. Research Software Development Best Practices. Continuous integration (CI) is a central component of `FreeBird.jl`'s development process. Ensuring research software is maintainable and its development sustainable requires deliberate investment in and enforcement of good practices. In this section, we outline the software development practices adopted in `FreeBird.jl` and describe how they are automated.

We use GitHub for version control, enforcing strict branch policies. New features are developed in a `feature/*` branch and merged into `dev` only after passing tests and receiving reviewer approval via a pull request. The `dev` branch is subsequently merged into `main`, typically in conjunction with a new release. `hotfix/*` branches are reserved for bug fixes and are merged promptly into both `dev` and `main`. `docs/*` branches are used exclusively for documentation-related changes, including updates to the dedicated `docs` directory and function docstrings in the source code.

`Documenter.jl`[51] is used to automatically generate and deploy `FreeBird.jl`'s documentation to GitHub Pages. It collects and renders method docstrings for inclusion in the documentation, ensuring that function descriptions remain synchronized with the source code. All exported methods are required to have docstrings describing their functionality, input arguments, and return values (if any). Documentation for any function can also be accessed directly from the Julia REPL. `Documenter.jl` builds separate pages for stable, development, and release versions of the code, and generates a preview when a pull request is opened. The `FreeBird.jl` documentation also includes tutorials and examples to support new users and developers.

Unit testing is a critical component of CI. We use Julia's built-in testing framework to create test suites for each module of `FreeBird.jl` and employ *Coveralls* to track and report code coverage. Maintaining equal or higher coverage is a requirement for pull request approval.

Other automated workflows include the TagBot, which automatically creates tags, releases, and changelogs when a new version of the code is registered in the Julia General registry. CompatHelper periodically updates `FreeBird.jl`'s dependencies, generating an automated pull request whenever a new version of a dependency package becomes available.

All CI components are automated using GitHub Actions. The workflow files, located in the `.github/workflows` directory of the `FreeBird.jl` repository, can be readily

https://doi.org/10.1021/acs.jctc.5c01348
J. Chem. Theory Comput. 2025, 21, 10765−10779

incorporated into any Julia-based project. To ensure readability and consistency, we follow the BlueStyle coding conventions when developing FreeBird.jl. Overall, Julia offers a mature and comprehensive CI ecosystem, well integrated with GitHub and other platforms, that supports the sustainable development of FreeBird.jl.

2.1.4. Parallelization. Another aspect of FreeBird.jl's flexibility is its ability to perform fast, lightweight computations with relatively inexpensive energy evaluators on limited computing resources, while also scaling to high-performance computing (HPC) platforms for intensive, large-scale campaigns. Currently, FreeBird.jl uses two main forms of parallelism (Threads for multithreading and Distributed for multiprocessing), both native to the Julia standard library, to accelerate computations across available hardware. Threads are used for embarrassingly parallel loops, while Distributed is applied to sampling steps that can be executed concurrently (primarily in nested sampling; see Section 2.3.3) by spawning multiple workers, typically one per CPU core. Both approaches can be combined by assigning multiple threads to each worker to further increase CPU utilization, as most FreeBird.jl functions are threaded.

Julia supports parallelization via the Message Passing Interface (MPI) protocol through external packages such as MPI.jl.[52] GPU computing is also enabled by hardware-specific packages, including CUDA.jl[53−55] for NVIDIA, AMDGPU.jl[56] for AMD, oneAPI.jl[57] for Intel, and Metal.jl[58] for Apple Silicon GPUs. Additionally, vendor-neutral GPU programming models such as OpenCL.jl[59] and KernelAbstractions.jl[60] allow Julia code to run across multiple platforms from different hardware vendors.

2.2. Models. In FreeBird.jl, physical models are classified into two categories: (1) atomistic systems, composed of particles with positions that vary continuously in real space (typically $\mathbb{R}^3$), where the total energy is defined by a Hamiltonian derived from an interatomic potential; and (2) lattice systems, in which particles occupy a discrete set of lattice sites and the energy is defined by a lattice Hamiltonian. Both categories can accommodate any number of chemical species and arbitrary cell shapes, with or without periodic boundaries, either in all spatial dimensions or in a user-selected subset of dimensions. FreeBird.jl also provides numerous convenience functions to construct these systems (see examples later in this section).

Although the present work highlights applications to monatomic solids, forthcoming releases will extend FreeBird.jl to molecular systems by integrating ASE calculators[61] for classical[62−64] and machine-learning[65−67] force fields and coupling with Molly.jl[68] to enable molecular dynamics−based sampling of bonds, angles, and dihedrals.

We provide code examples to illustrate the typical Julia syntax used in FreeBird.jl, but these listings are entirely optional. Skipping them will not affect the overall comprehensibility of the work.

2.2.1. Atomistic Systems with Interatomic Potentials. We use the term *atomistic* to describe a system composed of individual particles whose positions vary continuously in real space, confined within a simulation cell that may employ periodic boundary conditions. Configuration space is explored using *single-particle displacement moves*, trial translations with directions chosen uniformly at random and magnitudes drawn from a prescribed distribution. The sequence of accepted moves forms a random walk through configuration space, which is the default sampling scheme in FreeBird.jl (among others). Figure 2 illustrates how to generate a set of random atomistic configurations and store them in an AtomWalker{C} data type.

```
1  # Load FreeBird
2  julia> using FreeBird
3
4  # Generate 3 random configurations of 5 hydrogen
   ↪  atoms in a cubic box with 10.0 Å^3 volume per
   ↪  atom
5  julia> hydrogens = generate_initial_configs(3,
   ↪  10.0, 5)
6  3-element Vector{AtomsBase.FastSystem{...}}:
7   FastSystem(H5, periodicity = FFF)
8   FastSystem(H5, periodicity = FFF)
9   FastSystem(H5, periodicity = FFF)
10
11 # Draw an ASCII diagram of the first configuration
12 julia> view_structure(hydrogens[1])
13     .--------.
14    /|        |
15   * |  H     |
16   H |        |
17   | |     H  |
18   | .--H----H.
19   |/        /
20   *--------*
21
22 # Wrap the first configuration in a two-component
   ↪  walker; split the 5 atoms into components of 2
   ↪  and 3 atoms, and freeze the first component
23 julia> wk = AtomWalker{2}(hydrogens[1];
   ↪  list_num_par=[2,3], frozen=[true,false])
24 AtomWalker{2}(
25     configuration      : FastSystem(H5, periodicity
   ↪    = FFF)
26     energy             : 0.0 eV
27     iter               : 0
28     list_num_par       : [2, 3]
29     frozen             : Bool[1, 0]
30     energy_frozen_part : 0.0 eV)
31
32 # Use broadcasting to wrap all configurations in
   ↪  `hydrogens` into single-component walkers, with
   ↪  all atoms free
33 julia> wks = AtomWalker.(hydrogens)
34 3-element Vector{AtomWalker{1}}:
35  AtomWalker{1}(
36     configuration      : FastSystem(H5, periodicity
   ↪    = FFF)
37     energy             : 0.0 eV
38     iter               : 0
39     list_num_par       : [5]
40     frozen             : Bool[0]
41     energy_frozen_part : 0.0 eV)
42 ...
```

Figure 2. Julia code demonstrating the generation of three AtomWalker objects for systems containing five hydrogen atoms in a cubic box, including visualization, multicomponent splitting with partial freezing, and batch wrapping via broadcasting. Some output is truncated.

2.2.1.1. Construction of Atomistic Configurations. As noted earlier, in AtomWalker{C}, C is a positive integer specifying the number of components (e.g., distinct chemical species or rigid subgroups) represented by the walker. In Figure 2, line 23 demonstrates how a system of five particles in a cubic box can be divided into two components, with the first

https://doi.org/10.1021/acs.jctc.5c01348
J. Chem. Theory Comput. 2025, 21, 10765−10779

component frozen. Frozen particles are excluded from subsequent single-particle displacement moves; their energy is computed once and stored, while interactions with mobile particles are still calculated on the fly, thereby reducing overall computational cost.

In an atomistic system, components are defined as groups of particles sharing a common characteristic (for example, distinct chemical species, rigid fragments, or subsets subject to identical move or interaction parameters). Many functions in `FreeBird.jl` operate natively on such multicomponent systems; for instance, the total potential energy is computed as the sum of inter- and intracomponent contributions. This design provides flexibility for modeling diverse systems, including mixtures, host−guest complexes, and interfaces.

2.2.1.2. Definition of Atomistic Potentials. Particles in an atomistic system interact through interatomic potentials, implemented as subtypes of `AbstractPotential`, currently classified as single- or multicomponent and as either pairwise or many-body. For example, in Figure 3, line 2, the `LJParameters` (a single-component, pairwise potential) is constructed using keyword arguments. In this example, specifying a potential cutoff at 4σ automatically shifts the potential energies so they approach 0 eV at the cutoff. This shift can be disabled by supplying the additional argument `shift = false`. For multicomponent systems, `CompositeParameterSets` can be used to define interactions. This data structure, a subtype of `MultiComponentPotential`, stores a matrix of `LJParameters`.

```
1  # Construct a Lennard-Jones potential
2  julia> lj = LJParameters(epsilon=0.1, sigma=2.5,
   ↪  cutoff=4.0)
3  LJParameters(0.1 eV, 2.5 Å, 4.0,
   ↪  -9.763240814208984e-5 eV)
4
5  # Combine the walkers and Lennard-Jones potential
   ↪  into a live set
6  julia> ls = LJAtomWalkers(wks, lj)
7  LJAtomWalkers(AtomWalker{1}, LJParameters):
8  [1] AtomWalker{1}(
9      configuration      : FastSystem(H5, periodicity
       ↪  = FFF)
10     energy             : 496.83218514161376 eV
11     iter               : 0
12     list_num_par       : [5]
13     frozen             : Bool[0]
14     energy_frozen_part : 0.0 eV)
15
16 [2] AtomWalker{1}(
17     configuration      : FastSystem(H5, periodicity
       ↪  = FFF)
18     energy             : 75.15053441600284 eV
19     iter               : 0
20     list_num_par       : [5]
21     frozen             : Bool[0]
22     energy_frozen_part : 0.0 eV)
23
24 [3] AtomWalker{1}(
25     configuration      : FastSystem(H5, periodicity
       ↪  = FFF)
26     energy             : 5016.408030136223 eV
27     iter               : 0
28     list_num_par       : [5]
29     frozen             : Bool[0]
30     energy_frozen_part : 0.0 eV)
31
32 LJParameters(0.1 eV, 2.5 Å, 4.0,
   ↪  -9.763240814208984e-5 eV)
```

Figure 3. Julia code for constructing a Lennard-Jones potential and combining it with the walkers from Figure 2 to create a live set (`LJAtomWalkers`). Energies correspond to the initial configurations; output is shown in full.

Lastly, Figure 3 illustrates how to construct a live set by combining a set of walkers with an LJ potential. `LJAtomWalkers`, a subtype of `AtomWalkers` (itself a subtype of `AbstractLiveSet`), contains `AtomWalker` configurations of particles interacting via LJ potentials. By default, when walkers and a potential are passed to `LJAtomWalkers`, their energies are automatically computed and updated. For example, the walkers in `wks` in line 33 of Figure 2 initially have no assigned energies; invoking `LJAtomWalkers(wks, lj)` in line 6 of Figure 3 evaluates those energies using the potential `lj` defined in line 2. This design allows any set of walkers to be paired with any potential while keeping energies current. Supplying the keyword argument `assign_energy = false` disables this feature, leaving energies unchanged. Many functions in `FreeBird.jl` follow this pattern, where expected default behavior can be overridden through additional user-provided arguments. This approach supports robust code execution and reliable user interaction while enabling developers to explore unconventional settings.

2.2.2. Lattice Systems with Discrete Hamiltonians. We implement lattice models with two primary objectives: (1) to serve as coarse-grained alternatives to continuous atomistic models, and (2) to enable the study of systems whose exact partition function can be computed under certain simplifying conditions (e.g., small lattices or Ising-like systems), thereby providing accuracy benchmarks for the sampling methods.

2.2.2.1. Construction of Lattice Configurations. Lattice configurations are represented by the built-in, parametrized data type `MLattice{C,G}`, where `C` specifies the number of components (in the chemical thermodynamic sense) and `G` denotes the lattice geometry. `MLattice{C,G}` is a subtype of `AbstractLattice`. Figure 4 provides two examples of constructing two-dimensional lattice systems in `FreeBird.jl`: (1) a single-component square lattice (`C = 1` by default, `G = SquareLattice`) created using `SLattice{G}`, an alias for `MLattice{1,G}`, and (2) a two-component triangular lattice (`C = 2`, `G = TriangularLattice`). In both cases, only the keyword argument `components` is passed to the constructor, with all other fields (lattice vectors, basis, and supercell dimensions) set to their default values but modifiable by the user. To model adsorption processes, a subset of lattice grid points can be designated as adsorption sites. Particles occupying these sites experience an on-site energy in addition to their neighboring interaction energies.

2.2.2.2. Definition of Lattice Hamiltonians. The energy of a lattice can be efficiently evaluated using a discrete Hamiltonian, which specifies only the on-site energy and as many n-th nearest-neighbor interactions as needed. Because a lattice is defined on a discretized grid rather than by continuous distances, a lattice gas model is considerably more efficient than an interatomic potential, where energy depends explicitly on distance. For example, Figure 5 shows the construction of a generic lattice Hamiltonian, specifying the magnitudes of the on-site, first-nearest, and second-nearest neighbor interactions, along with the units of energy. The lattice Hamiltonian is given by

https://doi.org/10.1021/acs.jctc.5c01348
J. Chem. Theory Comput. 2025, 21, 10765−10779

```
# Construct a single-component lattice with 4 particles occupying the first 4 lattice sites using SLattice (alias for MLattice with one component)
julia> sl = SLattice{SquareLattice}(components=[[1,2,3,4]])
SLattice{SquareLattice}
    lattice_vectors      : [1.0 0.0 0.0; 0.0 1.0 0.0; 0.0 0.0 1.0]
    positions            : 16 grid points
    supercell_dimensions : (4, 4, 1)
    basis                : [(0.0, 0.0, 0.0)]
    periodicity          : (true, true, false)
    cutoff radii         : 2 nearest neighbors cutoffs [1.1, 1.5]
    occupations          :
      ● ● ● ●
      ○ ○ ○ ○
      ○ ○ ○ ○
      ○ ○ ○ ○
    adsorptions          : full adsorption

# Construct a two-component lattice (MLattice requires the number of components as a type parameter). Split the 4 particles into two components: [1,2] and [3,4]
julia> ml = MLattice{2,TriangularLattice}(components=[[1,2],[3,4]])
MLattice{2, TriangularLattice}
    lattice_vectors      : [1.0 0.0 0.0; 0.0 1.7320508075688772 0.0; 0.0 0.0 1.0]
    positions            : 16 grid points
    supercell_dimensions : (4, 2, 1)
    basis                : [(0.0, 0.0, 0.0), (0.5, 0.8660254037844386, 0.0)]
    periodicity          : (true, true, false)
    cutoff radii         : 2 nearest neighbors cutoffs [1.1, 1.5]
    occupations          :
      ❶ ❷ ○ ○
       ❶ ❷ ○ ○
      ○ ○ ○ ○
       ○ ○ ○ ○
    adsorptions          : full adsorption
```

Figure 4. Julia code for constructing a single-component square lattice (`sl`) and a two-component triangular lattice (`ml`) with four occupied sites. The two-component system assigns particles [1,2] to the first component and [3,4] to the second.

```
# Define the Hamiltonian:
# on-site energy = -0.04 eV
# nearest-neighbor energy = -0.01 eV
# next-nearest-neighbor energy = -0.0025 eV
julia> h = GenericLatticeHamiltonian(-0.04, [-0.01, -0.0025], u"eV")
GenericLatticeHamiltonian{2,Quantity{Float64, L^2 MT^-2, Unitful.FreeUnits{(eV,), L^2 MT^-2, nothing}}}:
    on_site_interaction:      -0.04 eV
    nth_neighbor_interactions: [-0.01, -0.0025] eV
```

Figure 5. Julia code for constructing a `GenericLatticeHamiltonian` with specified on-site, nearest-neighbor, and next-nearest-neighbor interaction energies.

$$H = N\varepsilon_0 + \sum_{\mathrm{nn}} \varepsilon_{\mathrm{nn}} + \sum_{\mathrm{nnn}} \varepsilon_{\mathrm{nnn}} + \cdots \tag{1}$$

where N is the number of occupied sites, ε_0 is the energy of an occupied site, $\varepsilon_{\mathrm{nn}}$ is the nearest-neighbor interaction energy, and $\varepsilon_{\mathrm{nnn}}$ is the next-nearest-neighbor interaction energy. We pass these interaction values into a `GenericLatticeHamiltonian{N,U}`, as shown in Figure 5, where N is the number of n-th nearest-neighbor interaction terms included (two in this example) and `U` specifies the energy units. The Hamiltonian is independent of specific lattice geometry and can therefore be applied to any lattice type. For multicomponent systems, a matrix of `GenericLatticeHamiltonian` objects can be defined for inter- and intracomponent interactions using `MLatticeHamiltonian{C,N,U}`, where `C` is the number of components.

2.3. Methods. In this section, we provide brief introductions to several sampling algorithms currently implemented in `FreeBird.jl`, including Metropolis sampling, Wang–Landau sampling, nested sampling, and exact enumeration for finite lattices. Serving as a toolbox, `FreeBird.jl` is continually expanding its range of sampling approaches; here, we describe only those currently available. Since these algorithms are regularly optimized and updated, we focus here on outlining their underlying fundamental principles rather than implementation-specific details.

2.3.1. Metropolis Sampling. The Monte Carlo (MC) sampling in this study follows the Metropolis algorithm.[69] This method samples from the canonical ensemble at fixed temperature by generating a random walk through configuration space. The probability of transitioning from state 1 to state 2 is given by

$$P(1 \rightarrow 2) = \min[1, e^{-\beta(E_2 - E_1)}] \tag{2}$$

where E_1 and E_2 are the energies of states 1 and 2, respectively; k_B is the Boltzmann constant; T is the temperature; and $\beta = 1/(k_B T)$ is the inverse temperature. The algorithm proceeds as follows:

1. Initialize the configuration.
2. Select a particle at random and propose a move:
   - *Atomistic*: Displace by a random vector.
   - *Lattice*: Swap with a randomly chosen empty site.
3. Compute the energy difference $\Delta E = E_2 - E_1$.
4. Accept the move with probability $P(1 \rightarrow 2)$, as given in eq 2.
5. Repeat steps 2–4 for many iterations.

Thermodynamic properties, such as the constant-volume heat capacity $C_V(T)$, can be obtained by analyzing the resulting ensemble. In `FreeBird.jl`, the only required inputs are the number of iterations and, for atomistic systems, the step size. The package also supports temperature sweeps, where the final configuration at one temperature is used as the initial configuration for the next, facilitating efficient re-equilibration.

2.3.2. Wang–Landau Sampling. The Wang–Landau (WL) algorithm is an MC method that achieves uniform sampling across a system's energy range by iteratively refining an estimate of the density of states, $g(E)$.[70] This adaptive strategy enables thorough exploration of configuration space, including rugged or frustrated energy landscapes where conventional Metropolis algorithms may become trapped in metastable states. The algorithm proceeds as follows:

1. Initialize the configuration, $g(E)$, energy histogram $H(E)$, and modification factor f.

https://doi.org/10.1021/acs.jctc.5c01348
J. Chem. Theory Comput. 2025, 21, 10765−10779

```julia
# Generate an initial set of 120 walkers and an LJ potential
walkers = AtomWalker.(generate_initial_configs(120, 562.5, 6))
lj = LJParameters(epsilon=0.1, sigma=2.5, cutoff=4.0)

# Nested sampling
ns_energies, ns_ls, ns_params = nested_sampling(LJAtomWalkers(walkers, lj),
    NestedSamplingParameters(mc_steps=200, step_size=0.1), 30_000, MCRandomWalkClone(), SaveEveryN(n_traj=10))

# Metropolis Monte Carlo using a temperature grid from 1000 K down to 50 K, in 50 K steps
mc_energies, mc_ls, mc_cvs, acceptance_rates = monte_carlo_sampling(walkers[1], lj,
    MetropolisMCParameters(collect(1000.0:-50:50), equilibrium_steps=100_000, sampling_steps=100_000,
    step_size=0.1))

# Wang-Landau sampling
wl_energies, wl_ls, wl_params, S, H = wang_landau(walkers[1], lj, WangLandauParameters(num_steps=10_000,
    energy_min=-1.26, energy_max=0.0, num_energy_bins=1_000, step_size=1.0, f_min=1.00001))
```

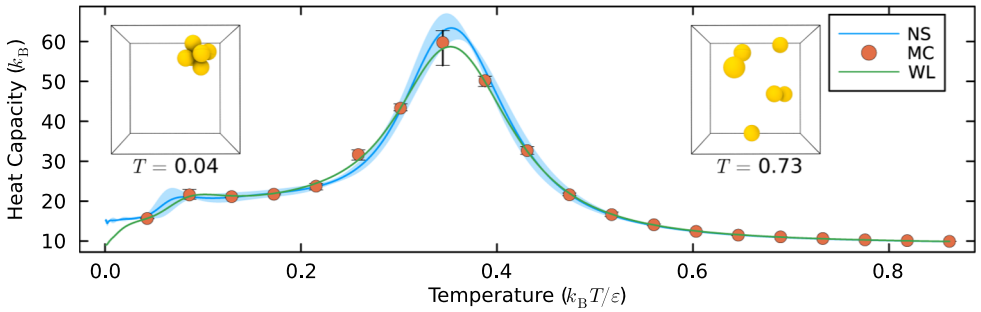


Figure 6. Julia code and results for the constant-volume heat capacity of the LJ_6 cluster computed using nested sampling (NS), Metropolis Monte Carlo (MC), and Wang−Landau (WL) sampling. Each method was repeated three times; NS and WL are shown as lines with shaded ranges (min−max), and MC as markers with error bars (min−max). Insets show representative structures from high (0.73 $k_B T/\varepsilon$) to low (0.04 $k_B T/\varepsilon$) temperature, illustrating the gas-to-cluster transition. Structures are taken from NS and verified with MC at corresponding temperatures.

2. Select a particle at random and propose a Metropolis-style move.
3. Compute the ratio $\eta = g(E_1)/g(E_2)$.
4. Accept the move with $P(1 \rightarrow 2) = \min[1, \eta]$.
5. Update g and H at the visited energy E_v:
   - $g(E_v) \leftarrow g(E_v) \times f$.
   - $H(E_v) \leftarrow H(E_v) + 1$.
6. If $H(E)$ is sufficiently flat, reset $H(E)$ and reduce f.
7. Repeat steps 2−6 until f meets the convergence criterion.

This method provides direct estimates of $g(E)$, from which thermodynamic quantities such as entropy and free energy can be derived. In `FreeBird.jl`, additional required inputs beyond those for Metropolis sampling include the energy range and resolution, as well as convergence criteria for $H(E)$ flatness and f reduction.

2.3.3. Nested Sampling. The nested sampling (NS) algorithm is an MC method in which the accessible phase space contracts stochastically, producing approximately uniform steps in $\ln \Gamma$, the logarithm of the remaining phase-space volume.[71,72] This transformation converts the high-dimensional canonical partition function integral into a tractable one-dimensional sum.[23,24,73] The algorithm proceeds as follows.

1. Initialize the *live set* of K *walkers*, each representing a configuration sampled uniformly from the prior distribution over allowed configurations.
2. Select the C highest-energy walkers.
3. Update the energy limit to $E_{lim} \leftarrow E_{highest}$.
4. Compute the expected cumulative configuration-space contraction after i iterations as $\langle \Gamma_i \rangle = [K/(K + C)]^i$.
5. Replace the C selected walkers with new configurations drawn uniformly from the region where $E_{new} \leq E_{lim}$.
6. Repeat steps 2−5 for many iterations.

The procedure continues until sufficiently low-energy regions of configuration space have been sampled. From the sequence of replaced walkers and their associated energies, the canonical partition function is estimated as

$$Z(\beta) = \sum_i w_i e^{-\beta E_i} \tag{3}$$

where $w_i = \Gamma_{i-1} - \Gamma_i$ is the weight for the i-th iteration. Because the sampling steps are temperature-independent, any value of β can be substituted into eq 3 during postprocessing, allowing thermodynamic properties to be computed over a broad temperature range from a single simulation. In `FreeBird.jl`, additional required inputs beyond those for Metropolis sampling include K walkers, C, and the total number of NS iterations.

2.3.4. Exact Enumeration for Lattice Systems. Exact enumeration (also referred to as *complete*, *direct*, or *exhaustive* enumeration) is the process of explicitly generating all possible configurations (i.e., all arrangements of particles and their species) on a finite, discretized system such as a lattice with a fixed number of sites and particles. Each unique microstate is identified and its energy evaluated to compute the exact partition function of the system. The method is straightforward to implement, typically by generating all unique permutations of an integer array, for example

$$\underbrace{[1, 1, 1, 2, 2, 3, 3, \ldots, 0, 0, 0]}_{\text{number of lattice sites}}$$

https://doi.org/10.1021/acs.jctc.5c01348
J. Chem. Theory Comput. 2025, 21, 10765−10779

```julia
# Generate an initial set of 960 walkers with two components
walkers = AtomWalker.(generate_initial_configs(960, 281.25, [1,12]; particle_types=[:H,:He]))

# Define the LJ interactions and construct a CompositeParameterSets object
lj11 = LJParameters(epsilon=0.1, sigma=2.5, cutoff=4.0)
lj22 = LJParameters(epsilon=0.05, sigma=2.5, cutoff=4.0)
lj12 = LJParameters(epsilon=sqrt(0.1*0.05), sigma=2.5, cutoff=4.0)
lj = CompositeParameterSets(2, [lj11, lj12, lj22])

# Nested sampling
ns_energies, ns_ls, ns_params = nested_sampling(LJAtomWalkers(walkers, lj),
↪  NestedSamplingParameters(mc_steps=400, step_size=0.1), 500_000, MCRandomWalkClone(), SaveEveryN(n_traj=10))

# Metropolis Monte Carlo using a temperature grid from 400 K down to 100 K, in 25 K steps
mc_energies, mc_ls, mc_cvs, acceptance_rates = monte_carlo_sampling(walkers[1], lj,
↪  MetropolisMCParameters(collect(400.0:-25:100), equilibrium_steps=10_000_000, sampling_steps=5_000_000,
↪  step_size=1.0))

# Wang-Landau sampling
wl_energies, wl_ls, wl_params, S, H = wang_landau(walkers[1], lj, WangLandauParameters(num_steps=100_000,
↪  energy_min=-2.44, energy_max=0.0, num_energy_bins=1_000, step_size=1.0, f_min=1.00001))
```

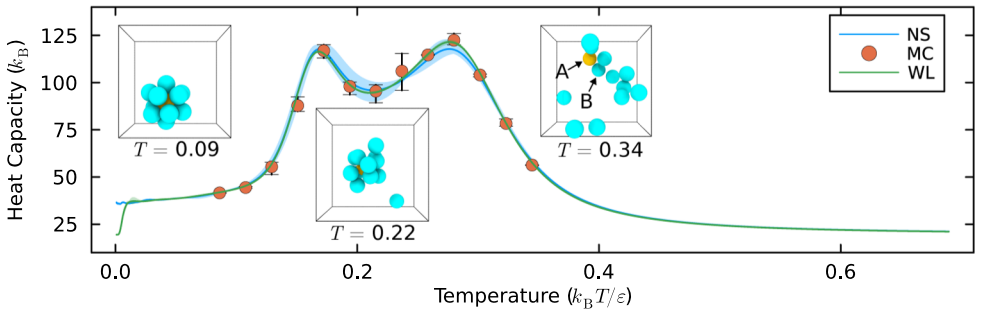


Figure 7. Julia code and results for the constant-volume heat capacity of the LJ A_1B_{12} cluster computed using nested sampling (NS), Metropolis Monte Carlo (MC), and Wang−Landau (WL) sampling. Each method was repeated three times; NS and WL are shown as lines with shaded ranges (min−max), and MC as markers with error bars (min−max). Insets show the structural change from high to low temperature, where the B-type particles condense around the single A-type particle to form a cluster. Structures are taken from NS and corroborated by MC at corresponding temperatures.

where nonzero entries label (chemical) species and zeros denote unoccupied sites. Because it enumerates the full microstate space, this approach serves as a reference standard for benchmarking approximate sampling methods. Its applicability, however, is limited to small lattice systems, as the number of configurations grows exponentially with the total number of particles and sites.

3. DEMONSTRATIONS

In this section, we demonstrate how `FreeBird.jl` can be used to identify the number, temperature, and nature of phase transitions across diverse systems. These span single- and multicomponent compositions, spatial resolutions from continuous atomistic models to discretized lattice models, and dimensionalities in both two and three dimensions. We also benchmark three sampling methods (NS, MC, and WL) to illustrate `FreeBird.jl`'s versatility and to validate its predictions. The complete set of simulation parameter values is provided in the Julia code associated with each demonstration in Figures 6−9. For clarity and reproducibility, we focus on relatively simple model systems. In each case, we compare constant-volume heat capacity (C_V) curves as functions of temperature, identifying phase transitions from the peak positions, which are determined using the `findmaxima()` function in the Julia package `Peaks.jl`. For MC simulations, because runs were performed at discrete temperatures that may not coincide exactly with a transition temperature, the reported MC transition temperature corresponds to the simulated point nearest each identified phase transition. Each sampling method is executed three times to estimate the uncertainty in the resulting C_V profiles. The reported values are the means of three runs, with the minimum and maximum at each temperature providing the lower and upper bounds of the uncertainty (hereafter referred to as min−max). Although these systems contain only a small number of particles, they provide minimal yet chemically and physically meaningful representations.

3.1. Atomistic Systems.

To illustrate `FreeBird.jl`'s applicability to models with continuously varying particle positions, we examine three representative atomistic systems: (1) a six-particle LJ cluster (LJ_6), (2) a 13-particle binary LJ cluster (A_1B_{12}), and (3) an LJ(111) surface slab with a frozen substrate and mobile adsorbates.

3.1.1. LJ_6 Cluster. We begin with the LJ_6 cluster, with benchmark results available from NS.[73] We use the shifted, truncated LJ potential parameters from ref 32 (ϵ = 0.1 eV, σ = 2.5 Å, cutoff $r_c = 4\sigma$). The system consists of six free LJ particles in a nonperiodic cubic box with an edge length of 15 Å, corresponding to a number density of $2.78 \times 10^{-2}\ \sigma^{-3}$. We performed NS simulations with 120 walkers and a constant walker-removal rate of $C = 1$ (Section 2.3.3), using the `MCRandomWalkClone()` algorithm to generate new

https://doi.org/10.1021/acs.jctc.5c01348
J. Chem. Theory Comput. 2025, 21, 10765−10779

```julia
# Read walkers from a file; periodic boundary conditions set to (true, true, false)
ats = read_walkers("slab.extxyz", pbc="TTF") # 320 configurations stored in this file -> 320 walkers

# Split each walker into two components: 80 frozen particles (surface) and 4 free particles (adsorbates)
walkers = [AtomWalker{2}(at.configuration; list_num_par=[80,4], frozen=Bool[1,0]) for at in ats]

# Create an LJ potential
lj = LJParameters(epsilon=0.1, sigma=2.5, cutoff=4.0, shift=true)

# Nested sampling
ns_energies, ns_ls, ns_params = nested_sampling(LJAtomWalkers(walkers, lj),
    NestedSamplingParameters(mc_steps=200, step_size=0.1), 50_000, MCRandomWalkClone(), SaveEveryN(n_traj=10))

# Metropolis Monte Carlo using a temperature grid from 2000 K down to 100 K, in 100 K steps
mc_energies, mc_ls, mc_cvs, acceptance_rates = monte_carlo_sampling(walkers[1], lj,
    MetropolisMCParameters(collect(2000.0:-100:100), equilibrium_steps=5_000_000, sampling_steps=5_000_000,
    step_size=1.0))

# Wang-Landau sampling
wl_energies, wl_ls, wl_params, S, H = wang_landau(walkers[1], lj, WangLandauParameters(num_steps=10_000,
    energy_min = -57.06, energy_max=-50.0, num_energy_bins=1_000, step_size=1.0, f_min=1.00001))
```

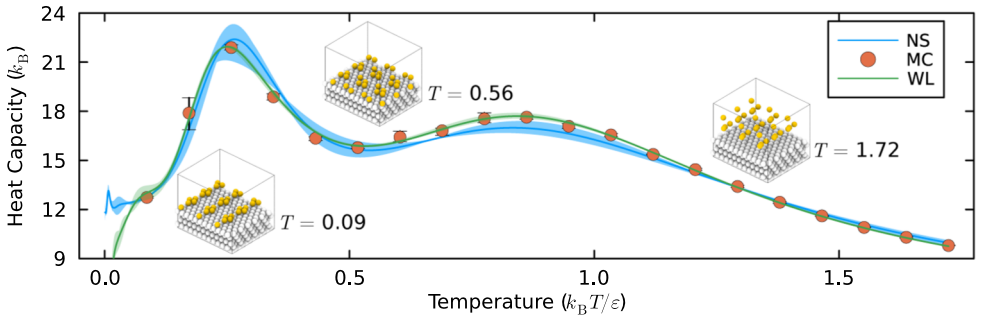


Figure 8. Julia code and results for an LJ(111) surface with quarter coverage ($\theta = 1/4$), showing the constant-volume heat capacity computed using nested sampling (NS), Metropolis Monte Carlo (MC), and Wang–Landau (WL) sampling. Each method was repeated three times; NS and WL are shown as lines with shaded ranges (min–max), and MC as markers with error bars (min–max). Insets illustrate temperature-dependent structural transitions, with adsorbates reorganizing on the surface. Representative structures are taken from NS and corroborated with MC; for visualization, the 4 × 4 per-monolayer unit cell is replicated twice in the lateral dimensions.

configurations by cloning an existing walker and subsequently decorrelating it via random walks. For MC simulations, the temperature was decreased in 50 K increments from 1000 to 50 K. For WL sampling, we employed 1000 bins spanning the energy range [-1.26,0] eV (slightly above the ground-state energy of −1.27 eV), applied an 80% flatness criterion to the energy histogram, and used an f-schedule of $f \leftarrow \sqrt{f}$.

Previous studies on a lower-density LJ_6 system (2.31×10^{-3} σ^{-3}) reported two phase transitions: a gas–solid transition at 0.31 k_BT/ε and a lower-temperature solid–solid transition at 0.10 k_BT/ε, with the latter favoring an octahedral ground state.[73] Figure 6 shows that all three methods recover these transitions, with consistent transition temperatures in units of k_BT/ε: 0.353 (NS), 0.345 (MC), and 0.352 (WL) for the gas–solid transition; and 0.082 (NS), 0.086 (MC), and 0.101 (WL) for the solid–solid transition. This low-temperature transition has also been described in previous studies of small Lennard-Jones clusters as a melting from a solid to a liquid-like state, with reported transition temperatures in good agreement.[73−75]

3.1.2. Binary LJ Cluster. Next, we examine the binary A_1B_{12} cluster, for which geometry-optimized structures are available as benchmarks.[76] We reuse the shifted, truncated LJ potential from Section 3.1.1, assigning distinct well depths to represent the two components: $\varepsilon_{AA} = 0.1$ eV for A–A interactions, $\varepsilon_{BB} = 0.05$ eV for B–B interactions, and $\varepsilon_{AB} = \sqrt{\varepsilon_{AA}\varepsilon_{BB}} \approx 0.07$ eV (Berthelot rule) for A–B interactions. To isolate the effect of interaction-energy asymmetry on the C_V curve, we set the particle sizes equal ($\sigma_A = \sigma_B = 2.5$ Å). The particles are placed in a nonperiodic cubic box of edge length 15.41 Å, corresponding to a number density of 5.56×10^{-2} σ^{-3}. Simulations were run as follows: NS simulations used 960 walkers with 400 decorrelation steps, a significant increase over LJ_6 to account for the more than doubled degrees of freedom from 13 particles; MC was cooled from 400 to 100 K in 25 K increments; and WL sampled energies from 0 to −2.44 eV, slightly above the ground-state energy (−2.45 eV).

As with LJ_6, the A_1B_{12} cluster exhibits a gas–solid transition at higher temperatures and a solid–solid transition at lower temperatures. Unlike LJ_6's octahedral ground state, however, the low-temperature structure adopts an A–B icosahedral core–shell configuration.[76] Figure 7 shows that all three methods reproduce these features, with closely matching transition temperatures in units of k_BT/ε: 0.276 (NS), 0.280 (MC), and 0.275 (WL) for the gas–solid transition, and 0.169 (NS), 0.172 (MC), and 0.167 (WL) for the solid–solid transition. Compared to LJ_6, the A_1B_{12} system shows a slightly lower gas–solid transition temperature and a markedly higher, sharper solid–solid transition. The ground-state structure corresponds to the minimum-energy configuration in the binary Lennard-Jones database,[76] and the double-peaked C_V curve and associated phase changes are consistent with

https://doi.org/10.1021/acs.jctc.5c01348
J. Chem. Theory Comput. 2025, 21, 10765−10779

```julia
# 2D lattice example
# Initial single-component square lattice (4×4×1 by default) with sites [1,2,3,4] occupied
initial_lattice = SLattice{SquareLattice}(components=[[1,2,3,4]])

# Hamiltonian: adsorption energy = -0.04 eV, nearest-neighbor energy = -0.01 eV, next-nearest-neighbor energy = -0.0025 eV
h = GenericLatticeHamiltonian(-0.04, [-0.01, -0.0025], u"eV")

# Exact enumeration
exact_energies, exact_configs = exact_enumeration(initial_lattice, h)

# Nested sampling
walkers = [LatticeWalker(generate_random_new_lattice_sample!(initial_lattice)) for _ in 1:1000]
ns_energies, ns_ls, ns_params = nested_sampling(LatticeGasWalkers(walkers, h), LatticeNestedSamplingParameters(), 10_000, MCRejectionSampling(), SaveEveryN())

# Metropolis Monte Carlo using a temperature grid from 200 K down to 10 K, in 10 K steps
mc_energies, mc_configs, mc_cvs, acceptance_rates = monte_carlo_sampling(initial_lattice, h, MetropolisMCParameters(collect(200.0:-10:10), equilibrium_steps=25_000, sampling_steps=25_000))

# Wang-Landau sampling
wl_energies, wl_configs, wl_params, S, H = wang_landau(initial_lattice, h, WangLandauParameters(energy_min=-0.20625, energy_max=-0.15875))

# A 3D lattice with three layers can be constructed by setting supercell_dimensions=(4,4,3). The sampling lines above can then be rerun (with more NS iterations/MC steps) to obtain results:
initial_lattice = SLattice{SquareLattice}(components=[[1,2,3,4]], supercell_dimensions=(4,4,3), adsorptions=collect(1:16))
```

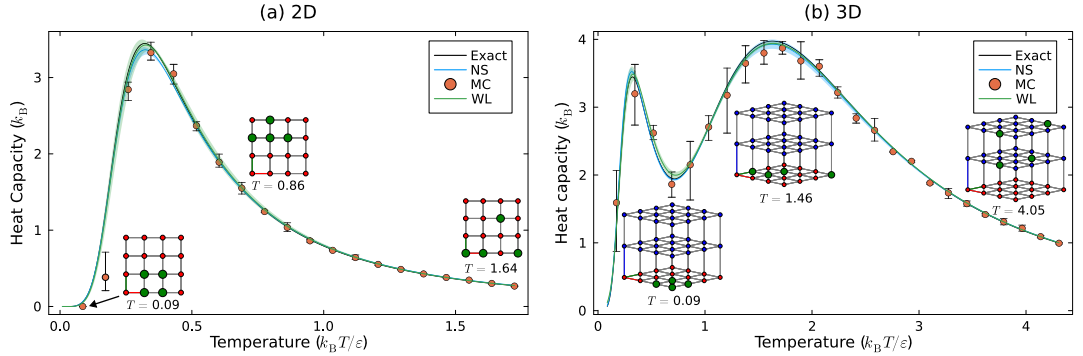


Figure 9. Julia code and results for lattice models with discrete occupancy: (a) a two-dimensional 4 × 4 × 1 lattice and (b) a three-dimensional 4 × 4 × 3 lattice, both with fractional coverage $\theta = 4/16$. The constant-volume heat capacity is computed using nested sampling (NS), Metropolis Monte Carlo (MC), and Wang−Landau (WL) sampling. Each method was repeated three times; NS and WL are shown as lines with shaded ranges (min−max), and MC as markers with error bars (min−max). Exact results were obtained by full enumeration of all configurations. Insets show the configurational change from high to low temperature, with blue circles indicating lattice sites, red circles indicating adsorption sites, and green circles indicating occupied sites.

descriptions from early work using argon- and neon-like Lennard-Jones parameters.[77]

3.1.3. LJ Surfaces. Finally, we examine the (111) surface of a face-centered cubic LJ solid, building on previous NS benchmarks.[32,33] We model a five-layer slab containing 80 fixed particles (16 per layer) and four mobile adsorbates, corresponding to a maximum coverage of $\theta = 1/4$. The simulation box has periodic boundaries in the in-plane directions (a = 11.2 Å, b = 9.73 Å) and a nonperiodic upper boundary at c = 29.15 Å where simple reflection is applied if a particle moves beyond this limit. Sampling parameters are as follows: NS ran for 50,000 steps with 320 walkers; MC cooled from 2000 to 100 K in 100 K increments; and WL sampled energies from −50.0 to −57.06 eV, slightly above the ground-state energy (−57.07 eV) (Figure 8).

This LJ(111) quarter-coverage (q.c.) system is expected to display a broad C_V peak at 0.86 k_BT/ε (surface condensation) and a sharper peak at 0.25 k_BT/ε (two-dimensional adsorbate ordering).[32] Figure 8 shows that all three methods yield consistent transition temperatures in units of k_BT/ε: 0.833 (NS), 0.862 (MC), and 0.849 (WL) for the condensation transition; and 0.265 (NS), 0.259 (MC), and 0.247 (WL) for the ordering transition. These findings are in exact agreement with our earlier study employing an independent nested sampling implementation.[32]

3.1.4. Discussion of Atomistic Demonstrations. For the simple atomistic systems examined, NS, MC, and WL produced consistent configurational statistics and thermodynamic properties. The agreement among the three methods, and with published benchmarks,[32,73,75,76] confirms that they are correctly implemented within `FreeBird.jl`. This unified framework allows verification of sampling outputs by cross-checking across methods using a consistent call structure, making it straightforward for users to apply multiple algorithms to their systems of interest.

Because all sampling methods use the same random-walk and energy-evaluation routines, the total number of energy evaluations provides a consistent probe of computational cost

https://doi.org/10.1021/acs.jctc.5c01348
J. Chem. Theory Comput. 2025, 21, 10765−10779

(Table 1). The reported values range from a few million to several billion evaluations, illustrating the order-of-magnitude differences in computational effort between algorithms. Note that these costs vary with both system complexity and algorithm choice. Table 1 is not intended as a direct performance comparison, since the methods differ in their intrinsic overheads (e.g., number of walkers for NS, temperature grids for MC, and number of bins plus flatness criterion for WL), and each can be further tuned to balance accuracy and cost. The reported values serve only as references for results obtained using straightforward, representative parameter settings for each algorithm.

Table 1. Number of Energy Evaluations Used in Each of Nested Sampling (NS), Metropolis Monte Carlo (MC), and Wang–Landau (WL) Sampling, in Millions[a]

system	NS	MC (at each T)	WL
LJ_6	5.0	0.2	247.9
LJ A_1B_{12}	200.0	15.0	6735.8
LJ(111) q.c.	12.5	10.0	330.5

[a]For MC, this value applies to each temperature individually.

In LJ_6, NS required 25,000 iterations with 200 decorrelation steps each, totaling 5 million evaluations. MC ran 4 million evaluations across 20 temperatures. WL, which flattened a 1000-bin histogram to $f < 1.00001$ over 16 iterations using 24,787 batches of 10,000 walks, required nearly 248 million evaluations, which is an order of magnitude higher than NS or MC, and produced a well-converged, low-variance heat-capacity trace.

In the binary LJ A_1B_{12} cluster, configurational complexity increases due to the larger number of particles and the presence of a second species, introducing compositional disorder. NS used 960 walkers and nearly 500,000 iterations (at 400 steps each), totaling 200 million evaluations. MC performed 15 million evaluations per temperature, while WL required over 6.7 billion evaluations to flatten a 1000-bin histogram to $f < 1.00001$.

In the LJ(111) slab with adsorbates, the number of mobile particles is small, but the fixed substrate creates a complex PES with many near-degenerate configurations. NS required 320 walkers (more than double that of LJ_6) and over twice the number of energy evaluations. MC sampling increased 50-fold (from 0.2 to 10 million evaluations per temperature), especially near phase transitions. WL required a similar effort as for LJ_6 but required significantly more energy evaluations than the other two sampling methods.

Although direct comparisons are challenging due to differing objectives and strategies, the sampling methods exhibit distinct trade-offs. NS is well-suited for generating configurations that span a wide energy range, from gas-like states to ground-state structures, making it effective for identifying candidate transition regions. Its stepwise nature limits parallelism compared with approaches such as parallel tempering,[46] but its walker-removal rate can be tuned to emphasize either exploratory breadth or resolution.[45] This flexibility makes NS a strong first-pass strategy for locating regions of interest for subsequent refinement with other algorithms. MC is advantageous when properties at specific temperatures are of primary interest. WL, though computationally demanding for continuous systems,[78] can yield low-variance estimates once convergence is reached, due to its comprehensive sampling of configurations finely spaced in energy. In practice, the methods can complement each other: NS can identify temperature ranges for MC and guide energy bounds for WL, while MC energy histograms can inform WL binning resolution within those bounds. As a modular toolbox, `FreeBird.jl` supports not only method selection but also coordination across algorithms. The three demonstrations underscore the framework's flexibility, accommodating varying numbers of components, interaction schemes, and physical constraints. As the codebase evolves, it will continue to support increasingly complex and realistic systems.

3.2. Lattice Systems. To demonstrate `FreeBird.jl`'s applicability to models with discretely positioned particles, we consider two representative lattice systems: (1) a two-dimensional square lattice with partially occupied sites, and (2) a three-dimensional primitive cubic lattice that models adsorption and desorption across layers.

3.2.1. Two-Dimensional Lattice Models. We begin with a two-dimensional 4×4 square lattice containing four occupied sites, corresponding to a coverage of $\theta = 1/4$. This system serves as a coarse-grained analog of LJ(100) q.c. and admits exact thermodynamic results via enumeration. The lattice Hamiltonian is defined as

$$H = N\varepsilon_{ads} + \sum_{nn} \varepsilon_{nn} + \sum_{nnn} \varepsilon_{nnn} + \cdots \tag{4}$$

where N is the number of occupied sites; ε_{ads} is the adsorption energy per occupied site; and ε_{nn} and ε_{nnn} are the interaction energies between nearest-neighbor (nn) and next-nearest-neighbor (nnn) pairs of occupied sites, respectively. We set $\varepsilon_{nn} = -0.01$ eV to model exothermic pairing between nearest neighbors and $\varepsilon_{ads} = 4\varepsilon_{nn} = -0.04$ eV to account for adsorption at the centers of four implicit surface particles. To mimic the distance decay of an LJ potential, we use $\varepsilon_{nnn} = \varepsilon_{nn}/4 = -0.0025$ eV, reflecting the approximate quartering of LJ interaction strength from $r_{min} = 2^{1/6}\sigma$ to $\sqrt{2}r_{min}$. We perform NS with 1,000 walkers and a constant walker-removal rate $C = 1$, using `MCRejectionSampling()` to generate new configurations and progressing only when a lattice configuration of equal or lower energy is found; MC with a temperature increment of 10 K (from 200 to 10 K), and WL sampling using 100 energy bins spanning [-0.20625,-0.15875] eV. This range extends slightly beyond the exact bounds, minimum energy −0.205 eV and maximum −0.16 eV, by half the smallest interaction magnitude ($|\varepsilon_{nnn}|/2 = 0.00125$ eV). Exact enumeration of all 1,820 configurations reveals a single order–disorder transition at 0.321 k_BT/ε, favoring a square ground state. As shown in Figure 9, all three methods detect this transition, yielding consistent transition temperatures in units of k_BT/ε: 0.323 (NS), 0.345 (MC), and 0.325 (WL).

3.2.2. Three-Dimensional Lattice Models. We next examine a three-dimensional $4 \times 4 \times 3$ primitive cubic lattice with four occupied sites. The bottom layer represents an adsorbent surface, while the upper two layers act as a fluid phase into which particles may desorb, emulating surface condensation. The same lattice Hamiltonian is used as in Section 3.2.1, except that $\varepsilon_{ads} = 0$ eV is assigned to occupied sites in the fluid layers to isolate the effect of fluid-phase introduction. Only nn and nnn interactions are considered. Simulation parameters match those of the two-dimensional model. Exact enumeration of the full configuration space (194,580 configurations) again reveals an order–disorder transition at lower temperatures

https://doi.org/10.1021/acs.jctc.5c01348
J. Chem. Theory Comput. 2025, 21, 10765−10779

(0.319 k_BT/ε, nearly identical to the 0.321 k_BT/ε transition in the 2D case), favoring the square ground state. In contrast to the two-dimensional system, however, the three-layer model also shows a broad C_V peak at higher temperatures (1.629 k_BT/ε), corresponding to a surface condensation transition. As shown in Figure 9, all three methods capture both features with closely matching transition temperatures: 0.319 (NS), 0.259 (MC), and 0.327 (WL) for the order−disorder transition; and 1.629 (NS), 1.637 (MC), and 1.637 (WL) for the surface condensation transition. Compared to the 2D case, the computational effort for all methods was increased to ensure convergence. The number of NS iterations was raised from 10,000 to 30,000; MC equilibration and sampling steps at each temperature were increased from 25,000 to 100,000; and WL retained the same number of energy bins (since the potential-energy landscape is unchanged), but the number of MC proposals per attempt to reduce f was increased from 100 to 1,000 to account for the larger number of microstates. These settings were used to generate the results shown in Figure 9(b).

3.2.3. Discussion of Lattice Demonstrations. Lattice models can greatly reduce computational cost while still capturing the essential surface phase transitions observed in atomistic systems. For small lattices, exact enumeration is tractable: the 4 × 4 2D model comprises 1,820 configurations, which can be exhaustively sampled in under 1 s on a single CPU core with a single thread, whereas the 3D model with three layers and four adsorbates contains 194,580 configurations and can be enumerated in about 30 s under the same conditions. WL sampling performs particularly well on these discrete systems, benefiting from the finite number of energy states and their known spacing. MC results are generally consistent with WL but show greater uncertainty near phase transitions, especially at low temperatures, where dense phases dominate and single-particle MC moves struggle to interconvert between competing low-energy configurations.

NS poses additional challenges. In lattice models, discrete energy degeneracies produce plateaus in the NS likelihood−energy profile.[79−81] To break these degeneracies, we follow the approach of Murray et al. and recent `BraWl` implementations, adding to each configuration's energy a uniformly distributed perturbation much smaller than the typical level spacing (e.g., $<10^{-12}$ eV).[79,82] With single-site random-walk moves, lattice NS shows sharply reduced acceptance rates as the energy ceiling approaches the ground-state energy. As shown in Figure 9, 10,000 NS samples are required for the 2D lattice to establish the correct distribution for constructing an accurate partition function, exceeding the total number of microstates in the system. Introducing cluster or collective moves, or increasing the live-set size, can restore sampling efficiency.[83] Even so, WL remains the most efficient method for these systems.

Collectively, these demonstrations underscore `FreeBird.jl`'s versatility in implementing and contrasting diverse sampling strategies within a single lattice system, thereby facilitating both performance benchmarking and thermodynamic cross-validation.

4. CONCLUSIONS AND OUTLOOK

In this work, we introduced `FreeBird.jl`, a comprehensive, flexible, and extensible toolbox for modeling solid interfaces, implemented in the modern programming language Julia. `FreeBird.jl` provides multiple sampling methods (Metropolis sampling, Wang−Landau sampling, and nested sampling) within a unified framework that employs a common structure-handling system and energy calculators across all methods. Its flexibility arises from the use of abstract data types, which can be easily customized through convenient constructor functions. These data types also facilitate extending the code to new classes of systems beyond the already general atomistic continuous systems and discretized lattices. `FreeBird.jl`'s extensibility further enables the incorporation of additional sampling methods (e.g., replica exchange) using the existing data structures, as well as the implementation of alternative interaction models beyond simple pairwise potentials.

Julia, as a high-level, just-in-time compiled language, significantly lowers the barrier to developing complete, high-performance computational chemistry packages.[84] The availability of high-quality standard libraries and community packages in the Julia ecosystem, particularly those from JuliaMolSim, has been instrumental in the rapid development of `FreeBird.jl`. Examples include `AtomsBase.jl`[85] for handling atomic and cell properties, `AtomsIO.jl`[86] and `ExtXYZ.jl`[87] for reading and writing atomic structures, and the Julia standard libraries `Threads` and `Distributed` for straightforward parallelization, which substantially improves `FreeBird.jl`'s performance and enables execution on HPC systems. Contributors to this open-source package span a range of career stages, from undergraduate students to established computational scientists, underscoring its accessibility to both users and developers. Finally, `FreeBird.jl` exemplifies the adoption of modern software engineering practices to enhance productivity, sustainability, and reproducibility, serving as a template for the development of contemporary scientific software.

Future developments of `FreeBird.jl` can be categorized into algorithmic and implementation enhancements. Algorithmically, integrating the existing sampling methods with ensembles beyond the canonical, such as the semigrand-canonical ensemble (to allow composition changes) and the grand-canonical ensemble (for chemical potential control), will expand the range of systems and properties relevant to materials discovery. Additionally, while the current move sets are adequate for many systems, more sophisticated proposals (e.g., cluster or collective moves) are often necessary near critical points or in systems with slow dynamics to ensure efficient sampling.[83,88] From an implementation perspective, enabling `FreeBird.jl` for efficient GPU computing is essential for scaling to exascale platforms. This capability can be combined with machine-learning interatomic potentials for energy evaluations, which naturally exploit GPU acceleration. Overall, `FreeBird.jl` functions not only as a performant tool for sampling computations, but also as an integrative interface linking diverse components of molecular simulation, and as a versatile platform for advancing computational chemistry and materials science.

■ ASSOCIATED CONTENT

Data Availability Statement

The computational package `FreeBird.jl` is an open-source project publicly available at https://github.com/wexlergroup/FreeBird.jl. The data that support the findings of this study, as well as the `FreeBird.jl` input files for generating the data, are openly available at https://github.com/wexlergroup/FreeBird.jl-paper-data-2025.

https://doi.org/10.1021/acs.jctc.5c01348
J. Chem. Theory Comput. 2025, 21, 10765−10779

AUTHOR INFORMATION

Corresponding Author

Robert B. Wexler − *Department of Chemistry and Institute of Materials Science and Engineering, Washington University in St. Louis, St. Louis, Missouri 63130, United States;* orcid.org/0000-0002-6861-6421; Email: wexler@wustl.edu

Authors

Ray Yang − *Department of Chemistry and Institute of Materials Science and Engineering and Department of Computer Science and Engineering, McKelvey School of Engineering, Washington University in St. Louis, St. Louis, Missouri 63130, United States;* orcid.org/0000-0001-6628-5166

Junchi Chen − *Department of Chemistry and Institute of Materials Science and Engineering, Washington University in St. Louis, St. Louis, Missouri 63130, United States;* orcid.org/0000-0001-6008-0899

Douglas Thibodeaux − *Department of Chemistry and Institute of Materials Science and Engineering, Washington University in St. Louis, St. Louis, Missouri 63130, United States*

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jctc.5c01348

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

R.B.W. acknowledges support from the National Science Foundation under Grant No. 2305155. A portion of this research was conducted as part of a user project at the Center for Nanophase Materials Sciences (CNMS), which is a US Department of Energy, Office of Science User Facility at Oak Ridge National Laboratory. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a Department of Energy User Facility.

REFERENCES

(1) Soderstedt, C. J.; Yuan, Y.; Vigil, S. A.; Ford, H. H.; Fratarcangeli, M.; Lin, Z.; Chen, J. G.; Moreno-Hernandez, I. A. Oxidized Overlayers of Ruthenium and Iridium as Electrocatalysts for Anodic Reactions. *J. Am. Chem. Soc.* **2025**, 5c04767.

(2) Liu, F.; Mao, R.; Liu, Z.; Du, J.; Gao, P. Probing phonon transport dynamics across an interface by electron microscopy. *Nature* **2025**, *642*, 941−946.

(3) Souza, J. G. S.; Bertolini, M.; Liu, J.; Nagay, B. E.; Martins, R.; Costa, R. C.; Brunson, J. C.; Shibli, J.; Figueiredo, L. C.; Dongari-Bagtzoglou, A.; Feres, M.; Barão, V. A. R.; Bor, B. Exploring the Impact of Biotic and Abiotic Surfaces on Protein Binding Modulation and Bacteria Attachment: Integrating Biological and Mathematical Approaches. *ACS Nano* **2025**, *19*, 23393−23413.

(4) Somorjai, G. A. Low energy electron diffraction and auger electron spectroscopy studies of the structure of adsorbed gases on solid surfaces. *Surf. Sci.* **1973**, *34*, 156−176.

(5) Somjit, V.; Yildiz, B. Atomic and Electronic Structure of the Al_2O_3/Al Interface during Oxide Propagation Probed by Ab Initio Grand Canonical Monte Carlo. *ACS Appl. Mater. Interfaces* **2022**, *14*, 42613−42627.

(6) May, J. W. *Discovery of Surface Phases by Low Energy Electron Diffraction (LEED)*; Elsevier, 1970; Vol. *21*, pp 151−280..

(7) Hütner, J. I.; Conti, A.; Kugler, D.; Mittendorfer, F.; Kresse, G.; Schmid, M.; Diebold, U.; Balajka, J. Stoichiometric reconstruction of the Al_2O_3(0001) surface. *Science* **2024**, *385*, 1241−1244.

(8) Lander, J. Low-energy electron diffraction and surface structural chemistry. *Prog. Solid State Chem.* **1965**, *2*, 26−116.

(9) Rupprechter, G.; Weilach, C. Spectroscopic studies of surface−gas interactions and catalyst restructuring at ambient pressure: mind the gap. *J. Phys.: Condens. Matter* **2008**, *20*, 184019.

(10) Akbashev, A. R.; Roddatis, V.; Baeumer, C.; Liu, T.; Mefford, J. T.; Chueh, W. C. Probing the stability of $SrIrO_3$ during active water electrolysis *via operando* atomic force microscopy. *Energy Environ. Sci.* **2023**, *16*, 513−522.

(11) Schröder, J.; Zamora Zeledón, J. A.; Kamat, G. A.; Kreider, M. E.; Wei, L.; Mule, A. S.; Torres, A.; Yap, K.; Sokaras, D.; Gallo, A.; Stevens, M. B.; Jaramillo, T. F. Tracking the Dynamics of a Ag-MnO_x Oxygen Reduction Catalyst Using In Situ and Operando X-ray Absorption Near-Edge Spectroscopy. *ACS Energy Lett.* **2023**, *8*, 2962−2969.

(12) Zhang, Z.; Zandkarimi, B.; Alexandrova, A. N. Ensembles of Metastable States Govern Heterogeneous Catalysis on Dynamic Interfaces. *Acc. Chem. Res.* **2020**, *53*, 447−458.

(13) Du, X.; Damewood, J. K.; Lunger, J. R.; Millan, R.; Yildiz, B.; Li, L.; Gómez-Bombarelli, R. Machine-learning-accelerated simulations to enable automatic surface reconstruction. *Nat. Comput. Sci.* **2023**, *3*, 1034−1044.

(14) Hörmann, L.; Stark, W. G.; Maurer, R. J. Machine learning and data-driven methods in computational surface and interface science. *npj Comput. Mater.* **2025**, *11*, 196.

(15) Chaudhuri, S.; Maurer, R. J. Challenges in the Theory and Atomistic Simulation of Metal Electrodeposition. *ACS Electrochem.* **2025**, *1*, 1014−1032.

(16) Phan, T. H.; Banjac, K.; Cometto, F. P.; Dattila, F.; García-Muelas, R.; Raaijman, S. J.; Ye, C.; Koper, M. T. M.; López, N.; Lingenfelder, M. Emergence of Potential-Controlled Cu-Nanocuboids and Graphene-Covered Cu-Nanocuboids under *Operando* CO_2 Electroreduction. *Nano Lett.* **2021**, *21*, 2059−2065.

(17) Zhou, C.; Ngan, H. T.; Lim, J. S.; Darbari, Z.; Lewandowski, A.; Stacchiola, D. J.; Kozinsky, B.; Sautet, P.; Boscoboinik, J. A. Dynamical Study of Adsorbate-Induced Restructuring Kinetics in Bimetallic Catalysts Using the PdAu(111) Model System. *J. Am. Chem. Soc.* **2022**, *144*, 15132−15142.

(18) Reuter, K.; Scheffler, M. Composition, structure, and stability of RuO_2(110) as a function of oxygen pressure. *Phys. Rev. B* **2001**, *65*, 035406.

(19) Wexler, R. B.; Qiu, T.; Rappe, A. M. Automatic Prediction of Surface Phase Diagrams Using Ab Initio Grand Canonical Monte Carlo. *J. Phys. Chem. C* **2019**, *123*, 2321−2328.

(20) Rosenbrock, C. W.; Gubaev, K.; Shapeev, A. V.; Pártay, L. B.; Bernstein, N.; Csányi, G.; Hart, G. L. W. Machine-learned interatomic potentials for alloys and alloy phase diagrams. *npj Comput. Mater.* **2021**, *7*, 24.

(21) Vandermause, J.; Xie, Y.; Lim, J. S.; Owen, C. J.; Kozinsky, B. Active learning of reactive Bayesian force fields applied to heterogeneous catalysis dynamics of H/Pt. *Nat. Commun.* **2022**, *13*, 5183.

(22) Campbell, C. T.; Sprowl, L. H.; Árnadóttir, L. Equilibrium Constants and Rate Constants for Adsorbates: Two-Dimensional (2D) Ideal Gas, 2D Ideal Lattice Gas, and Ideal Hindered Translator Models. *J. Phys. Chem. C* **2016**, *120*, 10283−10297.

(23) Pártay, L. B.; Csányi, G.; Bernstein, N. Nested sampling for materials. *Eur. Phys. J. B* **2021**, *94*, 159.

(24) Ashton, G.; Bernstein, N.; Buchner, J.; Chen, X.; Csányi, G.; Fowlie, A.; Feroz, F.; Griffiths, M.; Handley, W.; Habeck, M.; Higson, E.; Hobson, M.; Lasenby, A.; Parkinson, D.; Pártay, L. B.; Pitkin, M.; Schneider, D.; Speagle, J. S.; South, L.; Veitch, J.; Wacker, P.; Wales, D. J.; Yallup, D. Nested sampling for physical scientists. *Nat. Rev. Methods Primers* **2022**, *2*, 39.

(25) Saunders, B.; Hörmann, L.; Maurer, R. J. Comprehensive Structure Exploration and Thermodynamics of Heteroatom Doped Graphene Superstructures. *arXiv* **2025**.

(26) Reuter, K.; Stampf, C.; Scheffler, M. *Handbook of Materials Modeling*; Springer Netherlands: Dordrecht, 2005; pp 149−194.

(27) Lu, S.; Wang, Y.; Liu, H.; Miao, M.-s.; Ma, Y. Self-assembled ultrathin nanotubes on diamond (100) surface. *Nat. Commun.* **2014**, *5*, 3666.
(28) Ulissi, Z. W.; Singh, A. R.; Tsai, C.; Nørskov, J. K. Automated Discovery and Construction of Surface Phase Diagrams Using Machine Learning. *J. Phys. Chem. Lett.* **2016**, *7*, 3931−3935.
(29) Carr, S.; Garnett, R.; Lo, C. B. A. S. C. Applying bayesian optimization to the search for global minima on potential energy surfaces *Proceedings of the 33rd International Conference on Machine Learning*: New York, New York, USA, 2016, pp 898−907.
(30) Bauer, M. N.; Probert, M. I. J.; Panosetti, C. Systematic Comparison of Genetic Algorithm and Basin Hopping Approaches to the Global Optimization of Si(111) Surface Reconstructions. *J. Phys. Chem. A* **2022**, *126*, 3043−3056.
(31) Han, Y.; Wang, J.; Ding, C.; Gao, H.; Pan, S.; Jia, Q.; Sun, J. Prediction of surface reconstructions using MAGUS. *J. Chem. Phys.* **2023**, *158*, 174109.
(32) Yang, M.; Pártay, L. B.; Wexler, R. B. Surface phase diagrams from nested sampling. *Phys. Chem. Chem. Phys.* **2024**, *26*, 13862−13874.
(33) Chatbipho, T.; Yang, R.; Wexler, R. B.; Pártay, L. B. Adsorbate phase transitions on nanoclusters from nested sampling. *arXiv* **2025**.
(34) Zhou, Y.; Scheffler, M.; Ghiringhelli, L. M. Determining surface phase diagrams including anharmonic effects. *Phys. Rev. B* **2019**, *100*, 174106.
(35) Mambretti, F.; Raucci, U.; Yang, M.; Parrinello, M. How Does Structural Disorder Impact Heterogeneous Catalysts? The Case of Ammonia Decomposition on Non-stoichiometric Lithium Imide. *ACS Catal.* **2024**, *14*, 1252−1256.
(36) Van De Walle, A.; Asta, M.; Ceder, G. The alloy theoretic automated toolkit: A user guide. *Calphad* **2002**, *26*, 539−553.
(37) Martiniani, S.; Stevenson, J. nested_sampling. 2013; https://github.com/js850/nested_sampling.
(38) Martiniani, S., Stevenson, J. sens 2013. https://github.com/smcantab/sens
(39) Martiniani, S., Suryadevara, P., Stevenson, J., Collanton, R., Schrenk, J. mcpele. 2014. https://github.com/martiniani-lab/mcpele
(40) Partay, L. B., Bernstein, N., Schlegel, M., Baldock, R. J. N., Csányi, G., Havens, S., Stenczel, T., Kermode, J., Fletcher, V., Daff, T., Yang, R., Bartók, A. P. pymatnest 2015https://github.com/libAtoms/pymatnest
(41) Shah, J. K.; Marin-Rimoldi, E.; Mullen, R. G.; Keene, B. P.; Khan, S.; Paluch, A. S.; Rai, N.; Romanielo, L. L.; Rosch, T. W.; Yoo, B.; Maginn, E. J. Cassandra: An open source Monte Carlo package for molecular simulation. *J. Comput. Chem.* **2017**, *38*, 1727−1739.
(42) Ångqvist, M.; Muñoz, W. A.; Rahm, J. M.; Fransson, E.; Durniak, C.; Rozyczko, P.; Rod, T. H.; Erhart, P. ICET − A Python Library for Constructing and Sampling Alloy Cluster Expansions. *Adv. Theory Simul.* **2019**, *2*, 1900015.
(43) Barroso-Luque, L.; Yang, J. H.; Xie, F.; Chen, T.; Kam, R. L.; Jadidi, Z.; Zhong, P.; Ceder, G. smol: A Python package for cluster expansions andbeyond. *J. Open Source Softw.* **2022**, *7*, 4504.
(44) Bernstein, N. pymatnext. 2023; https://github.com/libAtoms/pymatnext.
(45) Martiniani, S.; Stevenson, J. D.; Wales, D. J.; Frenkel, D. Superposition Enhanced Nested Sampling. *Phys. Rev. X* **2014**, *4*, 031034.
(46) Swendsen, R. H.; Wang, J.-S. Replica Monte Carlo Simulation of Spin-Glasses. *Phys. Rev. Lett.* **1986**, *57*, 2607−2609.
(47) Wang, J.-S.; Tay, T. K.; Swendsen, R. H. Transition Matrix Monte Carlo Reweighting and Dynamics. *Phys. Rev. Lett.* **1999**, *82*, 476−479.
(48) Berne, B. J.; Pechukas, P. Gaussian Model Potentials for Molecular Interactions. *J. Chem. Phys.* **1972**, *56*, 4213−4216.
(49) Gay, J. G.; Berne, B. J. Modification of the overlap potential to mimic a linear site−site potential. *J. Chem. Phys.* **1981**, *74*, 3316−3319.
(50) Zhang, Z.; Glotzer, S. C. Self-Assembly of Patchy Particles. *Nano Lett.* **2004**, *4*, 1407−1413.
(51) JuliaDocs/Documenter.Jl. JuliaDocs, 2025; https://github.com/JuliaDocs/Documenter.jl.
(52) JuliaParallel/MPI.Jl. JuliaParallel, 2025; https://github.com/JuliaParallel/MPI.jl.
(53) Besard, T.; Foket, C.; De Sutter, B. Effective Extensible Programming: Unleashing Julia on GPUs. *IEEE Trans. Parellel Distributed Syst.* **2019**, *30*, 827−841.
(54) Besard, T.; Churavy, V.; Edelman, A.; Sutter, B. D. Rapid software prototyping for heterogeneous and distributed platforms. *Adv. Eng. Softw.* **2019**, *132*, 29−46.
(55) JuliaGPU/CUDA.Jl. JuliaGPU, 2025; https://github.com/JuliaGPU/CUDA.jl.
(56) JuliaGPU/AMDGPU.Jl. JuliaGPU, 2025; https://github.com/JuliaGPU/AMDGPU.jl.
(57) Besard, T. oneAPI.Jl. 2022; https://github.com/JuliaGPU/oneAPI.jl.
(58) Besard, T.; Hawkins, M. Metal.Jl. 2022; https://github.com/JuliaGPU/Metal.jl.
(59) JuliaGPU/OpenCL.Jl. JuliaGPU, 2025; https://github.com/JuliaGPU/OpenCL.jl.
(60) Churavy, V. KernelAbstractions.jl. 2025; https://github.com/JuliaGPU/KernelAbstractions.jl.
(61) Herbst, M. F. Mfherbst/ASEconvert.Jl. 2025; https://github.com/mfherbst/ASEconvert.jl.
(62) Jorgensen, W. L.; Tirado-Rives, J. The OPLS [optimized potentials for liquid simulations] potential functions for proteins, energy minimizations for crystals of cyclic peptides and crambin. *J. Am. Chem. Soc.* **1988**, *110*, 1657−1666.
(63) Cornell, W. D.; Cieplak, P.; Bayly, C. I.; Gould, I. R.; Merz, K. M.; Ferguson, D. M.; Spellmeyer, D. C.; Fox, T.; Caldwell, J. W.; Kollman, P. A. A second generation force field for the simulation of proteins, nucleic acids, and organic molecules. *J. Am. Chem. Soc.* **1995**, *117*, 5179−5197.
(64) MacKerell, A. D., Jr; Bashford, D.; Bellott, M.; Dunbrack, R. L., Jr; Evanseck, J. D.; Field, M. J.; Fischer, S.; Gao, J.; Guo, H.; Ha, S.; et al. All-Atom Empirical Potential for Molecular Modeling and Dynamics Studies of Proteins. *J. Phys. Chem. B* **1998**, *102*, 3586−3616.
(65) Zhang, S.; Zubatyuk, R.; Yang, Y.; Roitberg, A.; Isayev, O. ANI-1xBB: An ANI-based reactive potential for small organic molecules. *J. Chem. Theory Comput.* **2025**, *21*, 4365−4374.
(66) Anstine, D. M.; Zubatyuk, R.; Isayev, O. AIMNet2: a neural network potential to meet your neutral, charged, organic, and elemental-organic needs. *Chem. Sci.* **2025**, *16*, 10228−10244.
(67) Wood, B. M.; Dzamba, M.; Fu, X.; Gao, M.; Shuaibi, M.; Barroso-Luque, L.; Abdelmaqsoud, K.; Gharakhanyan, V.; Kitchin, J. R.; Levine, D. S. others UMA: A Family of Universal Models for Atoms. *arXiv* **2025**.
(68) Greener, J. G. Differentiable simulation to develop molecular dynamics force fields for disordered proteins. *Chem. Sci.* **2024**, *15*, 4897−4909.
(69) Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A. H.; Teller, E. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.* **1953**, *21*, 1087−1092.
(70) Wang, F.; Landau, D. P. Efficient, Multiple-Range Random Walk Algorithm to Calculate the Density of States. *Phys. Rev. Lett.* **2001**, *86*, 2050−2053.
(71) Skilling, J. Nested Sampling. *AIP Conf. Proc.* **2004**, *735*, 395−405.
(72) Skilling, J. Nested sampling for general Bayesian computation. *Bay. Anal.* **2006**, *1*, 833−859.
(73) Pártay, L. B.; Bartók, A. P.; Csányi, G. Efficient Sampling of Atomic Configurational Spaces. *J. Phys. Chem. B* **2010**, *114*, 10502−10512.
(74) Wales, D. J. Atomic clusters with addressable complexity. *J. Chem. Phys.* **2017**, *146*, 054306.
(75) Maillard, L.; Finocchi, F.; Godinho, C.; Trassinelli, M. Nested Sampling for Exploring Lennard-Jones Clusters. 2025; http://arxiv.org/abs/2501.11370.

(76) Mravlak, M.; Kister, T.; Kraus, T.; Schilling, T. Structure diagram of binary Lennard-Jones clusters. *J. Chem. Phys.* **2016**, *145*, 024302.
(77) Sabo, D.; Predescu, C.; Doll, J.; Freeman, D. L. Phase changes in selected Lennard-Jones X 13- n Y n clusters. *J. Chem. Phys.* **2004**, *121*, 856−867.
(78) Zhou, C.; Bhatt, R. N. Understanding and improving the Wang-Landau algorithm. *Phys. Rev. E* **2005**, *72*, 025701.
(79) Murray, I.; MacKay, D.; Ghahramani, Z.; Skilling, J. Nested sampling for Potts models. *Adv. Neural Inf. Process. Syst.* **2005**.
(80) Schittenhelm, D.; Wacker, P. Nested Sampling And Likelihood Plateaus. 2021; http://arxiv.org/abs/2005.08602.
(81) Fowlie, A.; Handley, W.; Su, L. Nested sampling with plateaus. *Mon. Not. R. Astron. Soc.* **2021**, *503*, 1199−1205.
(82) Naguszewski, H. J.; Pártay, L. B.; Quigley, D.; Woodgate, C. D. BraWl: Simulating the thermodynamics and phase stability of multicomponent alloys using conventional and enhanced sampling techniques. 2025; http://arxiv.org/abs/2505.05393.
(83) Swendsen, R. H.; Wang, J.-S. Nonuniversal critical dynamics in Monte Carlo simulations. *Phys. Rev. Lett.* **1987**, *58*, 86−88.
(84) Pulsipher, J. L.; Cole, D. L.; Jalving, J.; Zavala, V. M. *Introduction to Software for Chemical Engineers*; CRC Press, 2025; pp 774−806.
(85) Herbst, M.; Kurchin, R. AtomsBase.Jl. 2023; https://github.com/JuliaMolSim/AtomsBase.jl.
(86) Herbst, M. F. Mfherbst/AtomsIO.Jl. 2025; https://github.com/mfherbst/AtomsIO.jl.
(87) libAtoms/ExtXYZ.Jl: Extended XYZ Read/Write Support for Julia. https://github.com/libAtoms/ExtXYZ.jl, 2025
(88) Wolff, U. Collective Monte Carlo Updating for Spin Systems. *Phys. Rev. Lett.* **1989**, *62*, 361−364.

https://doi.org/10.1021/acs.jctc.5c01348
J. Chem. Theory Comput. 2025, 21, 10765−10779