On the Development of Tools for the Study of Colloidal Self-Assembly

by

Brandon Butler

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Chemical Engineering and Scientific Computing)
in the University of Michigan
2024

Brandon Butler

butlerbr@umich.edu

ORCID iD: 0000-0001-7739-7796

DEDICATION

I dedicate this thesis to my wife Kristen for her continual support and encouragement.

ACKNOWLEDGEMENTS

As in any large project, the task was not the isolated work of one man or woman. Instead, I have been guided, aided, encouraged and strengthened by those around me. And while fully enumerating such a list would be an impossible task, I will attempt to convey my gratitude below.

First, I thank my parents for raising me with love and instilling in me an unabating curiosity. The hours of answering my incessant cries of why, the 1,000's of dollars in books and the encouragement to not settle academically established the foundation of knowledge and thinking on which I now stand. I thank my teachers throughout my education for giving their lives to the often thankless teaching of youths. In particular, I thank Prof. Seth Oppenhiemer and Prof. Albert Bisson at Mississippi State University for challenging me to think in critical ways.

I am grateful to Prof. Sharon Glotzer for letting me join her lab and giving me the freedom to discover my love for methods/algorithms and scientific development. Thank you for also guiding and improving my scientific writing. To Dr. Bradley Dice, Dr. Simon Adorf, Dr. Vyas Ramasubramani and Dr. Joshua Anderson, I grateful for teaching me the gold standard of scientific software development. Similarly, gratitude is due to Dr. Allen LaCour, Dr. Tim Moore and Dr. Domagoj Fijan for guiding my research and serving as scientific mentors. Karen Coulter thank you for ensuring things did not fall through the cracks and always keeping the lab and us individually afloat.

Thank you to my friends and family I made along the way. The connection you provided gave color to my days in Ann Arbor. To my wife, Kristen, you have always been a sure

refuge through any of life's storms. You have been my most prized companion on this journey. Finally, I thank my Creator from whom all these blessings have flowed.

Financial Acknowledgements

I wish to acknowledge the National Science Foundation which through a number of grants provided support for this research: a Computational and Data-enabled Science and Engineering Award # DMR 1808342 & DMR 2302470 for algorithm and performance development (PI Glotzer); Office of Advanced Cyberinfrastructure Award # OAC 1835612 for HOOMD-blue pythonic architecture for MoSDeF integration (PI Glotzer); and a competitive Seed Fellowship from the Molecular Science Software Institute (MolSSI) (NSF grant OAC-1547580) for development of HOOMD-blue version 3.0.

Additionally I received significant support in the form of computational resources and services provided by Advanced Research Computing at the University of Michigan, Ann Arbor, and the National Science Foundation grant number ACI-1548562; / XSEDE award DMR 140129; and specialized hardware provided by NVIDIA Corp.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PROGRAMS

LIST OF APPENDICES

ABSTRACT

Self-assembly is the process by which a material organizes itself without the need for external stimuli. This process spans a broad range of phenomena, from the crystallization of solids from atoms and molecules to the formation of micelles and cell membranes by amphiphilic molecules to the organization of colloidal crystals from nanoparticles. Understanding these phase transitions is essential to the intelligent development of new materials and structures and in particular for controlling the thermodynamic and kinetic pathways for assembly. Enabling such control will allow for currently impossible to access phases, precision in defect sizes and amounts and control of other *knobs* of phase design at various length scales.

In Chapter 1, we describe the phenomena and challenges to detailed investigation of assembly pathways that this dissertation addresses.

In Chapter 2, we outline two methods used throughout the dissertation.

In Chapter 3, we present a six-step pipeline and Python package, *dupin*, we developed for detecting events from particle trajectories. The detection of transitions in particle-based (e.g. molecular) simulations is typically handled in an *ad hoc* way, while *dupin* provides a generalized detection scheme that maintains interpretability and permits comparison among disparate systems and pathways. Furthermore, by automating the detection of events associated with phase transformations, *dupin* enables self-assembly studies at larger length and time scales than previously feasible by removing the operator from the data-processing loop. We conclude Chapter 3 with example applications of dupin to the study of self-assembly.

In Chapter 4, we outline and discuss a new order parameter that quantifies the symmetry of local particle environments. During the formation of crystals, particles organize themselves locally into motifs with new symmetries that may or may not be present in the fluid or the

final crystal structure. The "Point Group Order Parameter", PGOP, identifies the point group symmetry of an individual particle's local environment. By identifying these local motifs and how they change over time, we can learn how a fluid chooses a particular kinetic pathway to follow. We compare PGOP to other commonly used local order parameters and, through examples, show how it provides a useful level of description not accessible to other order parameters. The chapter begins with an outline of the algorithm that is quickly followed by various demonstrations of PGOP's ability to detect and quantify local order in noisy crystalline systems with/without defects and even amorphous phases.

In Chapter 5, we present another new order parameter we developed for studying phase transformations. This new order parameter consists of a group of functions that form a vector of continuous local coordination numbers, $CN_{\vec{V}}$, in a system of particles. The well-known local coordination number, CN, is defined for particle-based systems as the number of particles that are first nearest neighbors to a given particle. Consequently, CN takes on only discrete integer values, which can be problematic when used as a local order parameter due to fluctuations from thermal effects. $CN_{\vec{V}}$ smooths CN into a continuous value – essentially a dimensionless local density – making it useful in self-assembly studies in which thermal noise and other forces can cause discontinuous changes in CN. To do this, $CN_{\vec{V}}$ uses the area of facets in Voronoi tessellation polytopes to weigh neighbor contributions to a coordination shell. We provide a detailed description of $CN_{\vec{V}}$ and demonstrate its usefulness in noisy systems and in combination with PGOP.

In Chapter 6, we discuss our software development contributions to HOOMD-blue, our group's open source Python simulation toolkit, for its version 3 release. This release included a complete redesign of the application programming interface, various ways to extend molecular dynamics and Monte Carlo simulations of particle-based systems in Python and direct access to HOOMD-blue's internal data buffers. These advancements facilitated numerous new simulation protocols and methodologies while reducing the human capital necessary for the design of new simulation techniques.

We conclude my dissertation in Chapter 7 with a summary of the preceding chapters and provide a forward-looking perspective regarding further work and the potential new applications of our work in the field of self-assembly.

CHAPTER 1

Introduction

Self-assembly is the process by which a system without external guidance or force arranges into some ordered phase such as a crystal. This transition to an ordered state is of interest in various communities such as the colloidal community. In the colloidal community, we have discovered particles that assembly computationally and/or experimentally into various complex phases: quasi-crystals [67, 45], clathrates [82], hierarchical assemblies [110, 119], etc. These assemblies can be used for applications in optics [72], catalysis [83] and sensors [60] among other use-cases.

Studies in these self-assemblies abound, but notable holes in capabilities exist. Most studies fit into one of two types: a deep-dive of one or a few system types [81] or a broad study on multiple systems with less depth [29]. A "Pareto front" between the study scope and depth exists. Scientists need tools to improve simultaneously scope and depth to expand upon our understanding of self-assembly and increase researcher capacities. These tools need to be interpretable by experts and must automate tasks currently necessitating human intervention. In my dissertation, we present a body of work aimed at creating and developing such tools to push the bounds of what questions scientists studying self-assembly can ask.

We begin in Chapter 2 by describing two methods, Voronoi tessellation and molecular dynamics, which are often used or referenced across this dissertation.

Following the overview of methods, We start discussing our original contributions in Chapter 3, outlining a six-step pipeline implemented in the form of a software package `dupin` which can detect autonomously the transition points within a pathway. We define the path in configuration space through the dynamics of the arrangement of particles going from one state to another state as a pathway. When studying pathways, methods like transition path sampling [30], aimless shooting [96] or forward flux sampling [7] are used. However, these require *a priori* knowledge of the final phase and an order parameter to bias the simulation and inherently preclude other phases or states. When studying unbiased pathways, system-specific approaches are used to study the system, which prevents comparison across systems. `dupin` enables automated, comparable and generic event detection for molecular systems without the need to know the "correct" order parameters or phases. The approach enables larger scope studies that are currently infeasible due to the human time investment. Likewise, `dupin` enables the curation of transition states for machine learning applications.

Continuing in Chapter 4, we derive and showcase a new order parameter that preserves information lacking in similar order parameters. In studying the self-assembly of crystals, current bond-orientational (symmetries in a local particle's neighborhood) approaches lack information about the exact symmetries formed [122, 92]. This lack of information leads to an interpretability problem when using the order parameters. Our order parameter, Point Group Order Parameter (PGOP), detects the formation of point group symmetries around a central particle and thus the symmetries formed locally are known. This formation or degradation of local point group ordering is well suited to study crystals and quasicrystals and can provide novel insights into the self-assembly process.

Chapter 5 is similar in approach to Chapter 4; we derive a family of interpretable continuous coordination numbers. In the previous chapter, we developed an order parameter for identifying local bond-orientational ordering while necessarily discarding local density information. To obtain a complementary measure of local density to analyze self-assembly in tandem with PGOP, we need to find a continuous and dimensionless order parameter. The order parameter must be dimensionless to be comparable across systems, and it must be continuous and resilient to thermal noise to be useful for machine learning applications. By extending a continuous coordination number developed in a previous work [24], we provide such an order parameter. The method uses Voronoi tessellation and approaches the discrete coordination number in the limit of identical neighbors. We maintain the interpretability of a local coordination number with the utility of continuous order parameters. In this chapter, we also showcase its utility demonstrating it's performance in simple machine learning tasks with PGOP.

For Chapter 6, we describe and outline the changes we led in HOOMD-blue (a molecular simulation engine) for its version 3 release. Scientific software often focuses, rightly, on computational efficiency. However, this can come at the expense of software extensibility and usability. In this chapter, we highlight changes that increase the software's usability, extensibility, flexibility and interoperability particularly within the Python ecosystem. By optimizing for human use, computational researchers can save time that would be spent scripting or debugging to instead address higher order questions/problems. Such optimization requires careful thought as to the nature of the problem being solved and determining idiomatic use patterns while not limiting software capabilities. In addition, these changes make possible the use of advanced simulation controls and techniques such as selective data storage upon crystal nucleation.

We conclude the thesis with an overview of the chapters and a future outlook on the techniques outlined.

CHAPTER 2

Methods

2.1 Voronoi Tessellation

Voronoi tessellation [134, 133] partitions space into N subsets, where N is the number of points in the tessellation. Voronoi tessellation serves as the dual of the Delaunay triangulation. Formally, each Voronoi cell (subset) is defined for point k as,

$$S_k = \{x \in X : d(x, L(k)) \leq d(x, L(i)) \forall i \in [1, N],\ i \neq k\}, \tag{2.1}$$

where d is a distance metric and L is the locator function (i.e. $L(i)$ returns the location of i). The common definition of Voronoi tessellation comes directly from Equation 2.1. Each Voronoi cell partitions the space closest to one of the N points. These sets can be shown to be convex and are also known as Voronoi polytopes or polyhedra (in 3D). An example tessellation can be seen in Figure 2.1, which depicts a Voronoi tessellation of a slightly perturbed square lattice.

Voronoi tessellation is used often in the self-assembly community as a means of finding a particle's first neighbor shell. Neighbors are those that share a face in the tessellation. A prominent example of this is the Minkowski Structure Metrics [92], which use Voronoi tessellation neighbors to compute and weight Steinhardt order parameters [122].

One popular implementation of Voronoi tessellation is Voro++ [109], which is used by `freud` [104, 33] and is the implementation used throughout this dissertation.

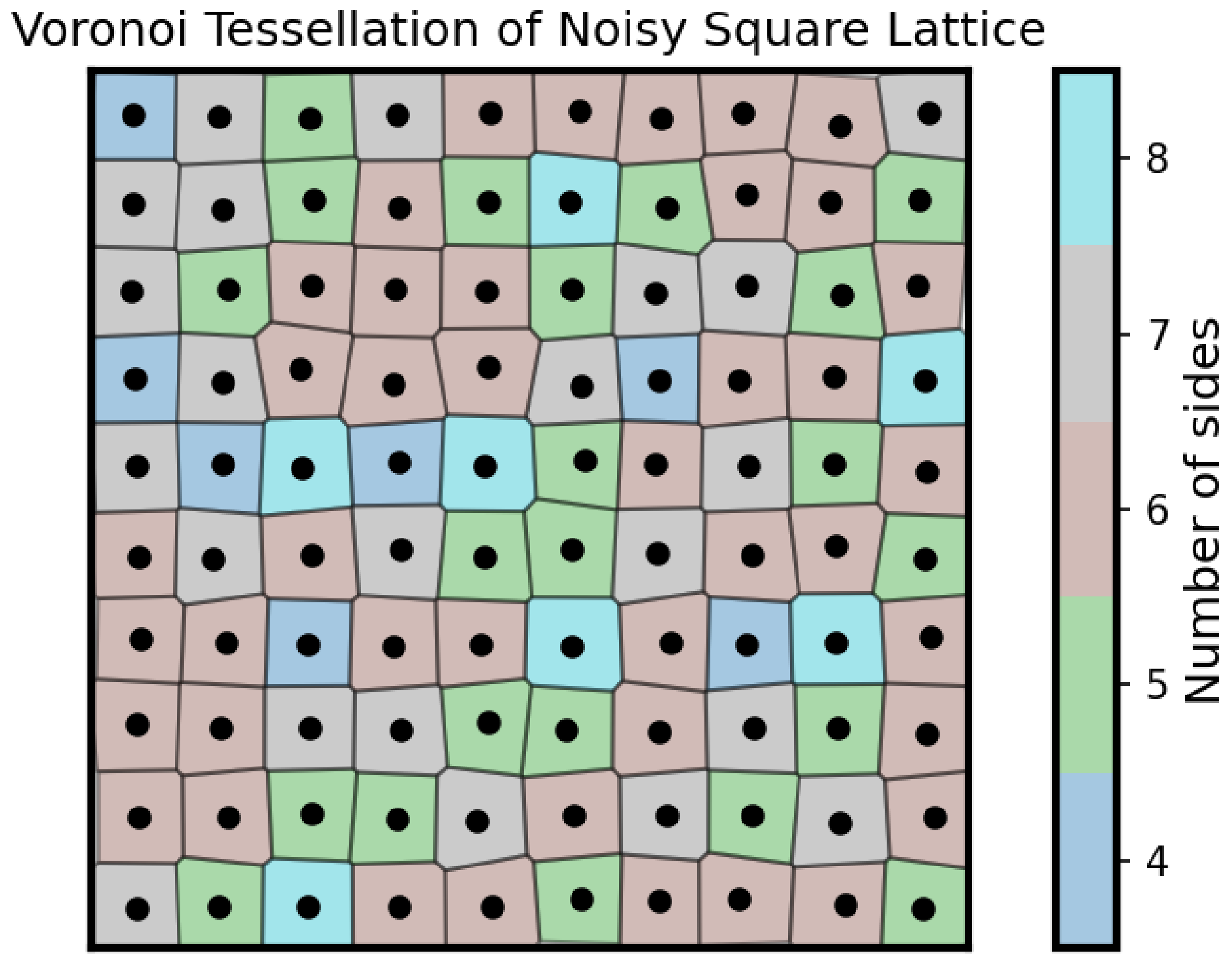

Figure 2.1: A colored Voronoi tessellation of a perturbed square lattice. The Voronoi cells are colored by the number of sides in the polygon.

2.2 Molecular Dynamics

Molecular dynamic uses Newton's equations of motion to simulate the evolution of atoms, molecules and/or particles into time. To do so, molecular dynamics integrates $F = ma$ given a molecular force field. As a tool, molecular dynamics was invented in the 1950's as computers were becoming more prevalent in the academic field [42, 6], though the mathematical framework for molecular dynamics was discovered centuries earlier. Originally molecular dynamics algorithms simulated the microcanonical ensemble, keeping the number of atoms, system volume and energy constant using the velocity verlet algorithm; however, extensions such as thermostats [59, 58] were created to allow studying other statistical mechanical ensembles. Molecular dynamics complements Monte Carlo simulations in studying phase space at a particle level. Since their inception these tools have been used to study numerous systems; examples include colloids [29], metallic glasses [41] and proteins [35], among many others.

Many implementations for molecular dynamics exist: LAMMPS [102], GROMACS [18, 3], OpenMM [37], ESPResSo [135, 52] and Amber [111, 25], etc. In this work, we will use simulations generated by HOOMD-blue [11, 10, 22].

CHAPTER 3

Change Point Detection of Events in Molecular Simulations Using `dupin`

This chapter is adapted from a manuscript currently under review authored by Brandon L. Butler, Domogoj Fijan and Sharon C. Glotzer to at a peer-reviewed journal.

3.1 Introduction

Computer simulations of molecular systems from the atomic to colloidal particle scale are a cornerstone of modern materials research. Given ongoing improvements in algorithms and processor speed, newer studies may make routine use of thousands of simulations or more [127, 129]. As a result, much work has been done to automate, streamline or otherwise simplify the management of large-scale simulation studies from software that manages data and workflows [4, 62] to packages used in data pipelines [104, 33, 87, 91, 51, 124].

We continue this trend by developing a software package that detects events (transitions) within point cloud data, the kind of data produced in molecular simulations. Event detection in point cloud data from particle trajectories is difficult because *a priori* information on the nature of the transition, identifying features or precursors of the transition (if any) is often missing. For studies involving multiple transitions or pathways, as well as large systematic studies, detection is further complicated by the need to automate the task. This paper addresses this problem using approaches collectively known as change point detection

(CPD) [9].

The techniques of CPD, which are commonly used in signal processing, have yet to infiltrate materials research [9]. Change points are defined by abrupt changes beyond expected fluctuations in a time-resolved dataset. In this paper, we will use particle trajectory data as our starting dataset. Molecular dynamics simulation as well as experiments using dynamic (e.g. liquid phase) transmission electron microscopy [84, 98] or confocal laser scanning microscopy [108, 2] — in conjunction with particle tracking software — can produce data consisting of time resolved particle positions and orientations.

Many algorithms [9] have been introduced to detect events associated with change points in data for the purpose of monitoring human activity [9], determining useful telemetry in data centers [8] or predicting machine degradation [117]. Change point detection approaches can be categorized as supervised [43, 54, 105] or unsupervised [14, 15, 69, 70, 71] as well as offline or online. Supervised methods require external labelling of events to *train* on, while unsupervised do not. Offline methods require the entire dataset as input. In contrast, online methods analyze the data as a stream while data is generated so that signals are screened in real time. The approach we take casts event detection as an optimization problem where the objective is minimized over change point locations [20].

Current practices in molecular simulations for event detection involve system or particle level order parameters (e.g. Minkowski structure metrics (MSM) [92], nematic order parameter and local density, to name a few). These local order parameters can be used to classify particles into environments using machine learning (ML) [31, 19, 114, 5, 34, 120]. After computing these parameters or environment labels, events are detected using system-specific or problem-specific detection schemes or by visual inspection. Automating the detection of meaningful events would significantly improve current workflows, facilitate "big data" studies in materials research and provide a consistent approach across studies.

In this paper, we present a new, open-source Python package `dupin` (named after Edgar Allen Poe's detective C. Auguste Dupin) for generic, autonomous event detection in particle trajectories with local or system-wide transitions. We show how `dupin` can partition a system's trajectory into regions of transition and stable (or metastable) states through the use of generic order parameters. In the following section, we outline `dupin`'s multi-stage procedure for detecting a set of change points from a system trajectory. We discuss options available to the user at each stage of the process. We then present two example applications using `dupin` and show its utility in determining the temporal bounds of system-wide structural transitions and particle-level events. We conclude with a general discussion of `dupin` and potential extensions. `dupin` is available on GitHub and distributed through conda-forge and the Python Package Index (PyPI).

3.2 Results

3.2.1 Detection Scheme

`dupin`'s detection scheme is based on CPD [9, 130]. CPD seeks to assign a set of points, K, where a signal, $S : \{s_0, s_1, \ldots, s_N\}$, undergoes a change. We denote a single point in a signal (e.g. s_i) as a frame. We define two operators - indexing and slicing - that act on the signal. The indexing operator returns a single value such that $S[i] = s_i$. The slicing operator produces a sub-signal using a semi-closed interval $S[i, j) : \{s_i, s_{i+1}, \ldots, s_{j-1}\}$. Using the slicing operation, we can see that the set of points K encompass $|K| + 1$ sub-signals between $[k_i, k_{i+1})$, where $k_0 = 0$ and $k_{|K|}$ is the last frame in the signal. The first and last change points, k_0 and $k_{|K|}$ are trivial, so the number of change points is often written as $|K| - 2$ instead of $|K|$. We adopt this convention from this point on in the paper.

As an example of CPD, imagine a 100-frame trajectory of a protein with two conformers, A and B, and a signal S comprising data representing the conformation of the protein in each frame. If the protein changes conformation at frames 40 and 60, then the signal S can be sliced into three sub-signals (Figure 3.1): $S[0, 40), S[40, 60)$, and $S[60, 100)$. In this case, there are two change points $K = \{40, 60\}$.

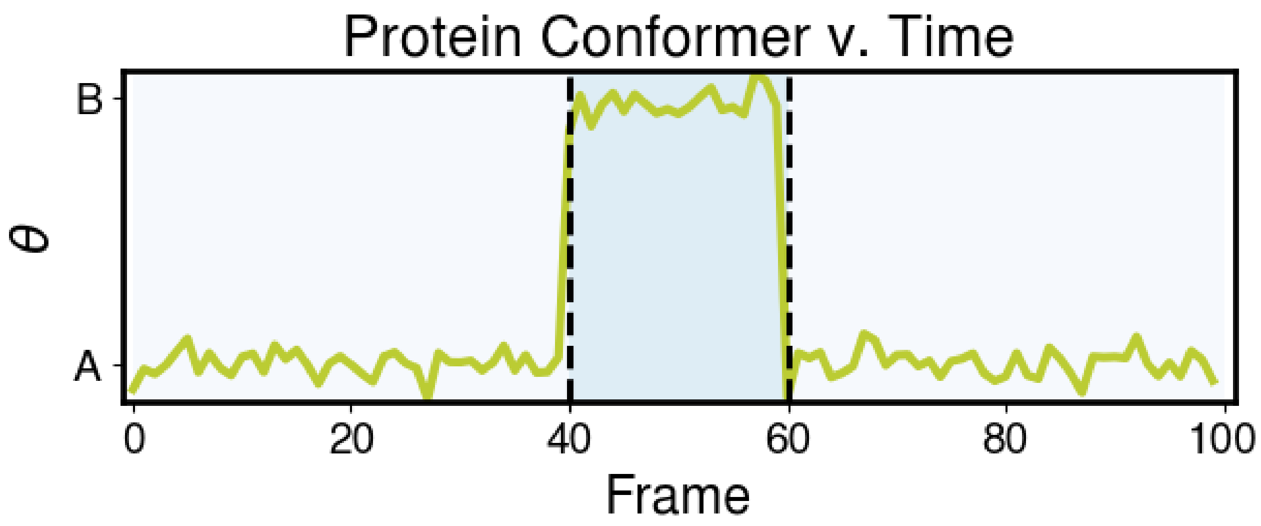

Figure 3.1: An example detection of a protein transitioning between conformers A and B. At frame 40, the protein goes from conformer A to B and back to A at frame 60. The background colors and dashed black line indicate these two change points, and the corresponding three sub-signals are $S[0, 40), S[40, 60), S[60, 100)$. θ represents an exemplar order parameter along which a structural change happens.

We employ a class of CPD algorithms that use the paradigm of optimization and loss/cost functions to select change points from S [130]. A loss or cost function penalizes some notion of error, which is then optimized to minimize the *cost*. We find a set of change points K of size n, such that choice of each single change point $k \in K$ minimizes the cost function C,

$$K = \arg\min_{k} \sum_{i=1}^{n+1} C(S[k_{i-1}, k_i)), \tag{3.1}$$

where $S[k_{i-1}, k_i)$ is the partition of the signal S between potential change points k_{i-1} and k_i. Notice that the number of change points expected, n, in the signal must be known in advance. A more detailed exploration of this issue is provided in the subsequent discussion. Further elaboration of this class of CPD algorithms is given in A.1.

3.2.1.1 Overview of Method

To develop a generic protocol for event detection in molecular trajectories, we must answer three fundamental questions (i) how to generate a signal from a trajectory, (ii) what cost function(s) best partition the trajectory into sub-signals and (iii) how to determine the correct number of change points in a signal to ensure detection of all events. To address these problems, `dupin` uses an approach in which different steps are grouped into stages (see Figure 3.2). We group the steps into three separate stages: data collection, data augmentation and detection. The data collection stage includes the *generate*, *map*, *reduce* and *aggregate* steps. The data augmentation stage includes the *transform* step. The detection stage includes the *detect* step. The *generate, aggregate* and *detect* steps within the data generation and detection stages are always required. The *reduce* steps can be required or optional depending on the data generated while the *map* and *transform* steps are always optional. The schematic of a typical pipeline for event detection in `dupin` is presented in Figure 3.2.

3.2.1.2 Data Collection

We begin by constructing feature vectors for every frame up to a total number of frames N_{frames}. A feature vector is a set of n features used to describe a data point in an n-dimensional space. Order parameters (MSM, local density, etc.) are examples of features. For our case, we refer to features that describe aspects of a given point/time in a trajectory. We combine the feature vectors into a signal S, which is a matrix of size $N_{\text{frames}} \times N_{\text{features}}$. Each matrix element s_{ij} in S contains the value of one of the features in one of the frames of the trajectory. We assume that any change in a molecular system can be adequately described by a such a signal. Changes in the system are thus indicated by changes in the feature vectors over time/frames.

The *generate* step requires us to choose the class of features that will be computed for every frame. Once the feature data is generated, non-scalar or vector features (e.g. per-particle quantities such as Steinhardt order parameters [122]) must first be reduced. Reduction takes a vector feature and converts it through a variety of reducers (functions) to a finite number of scalar features representative of the distribution. Scalar features are never reduced (they are already scalar). The *reduce* step cannot be used on global properties (scalar features) such

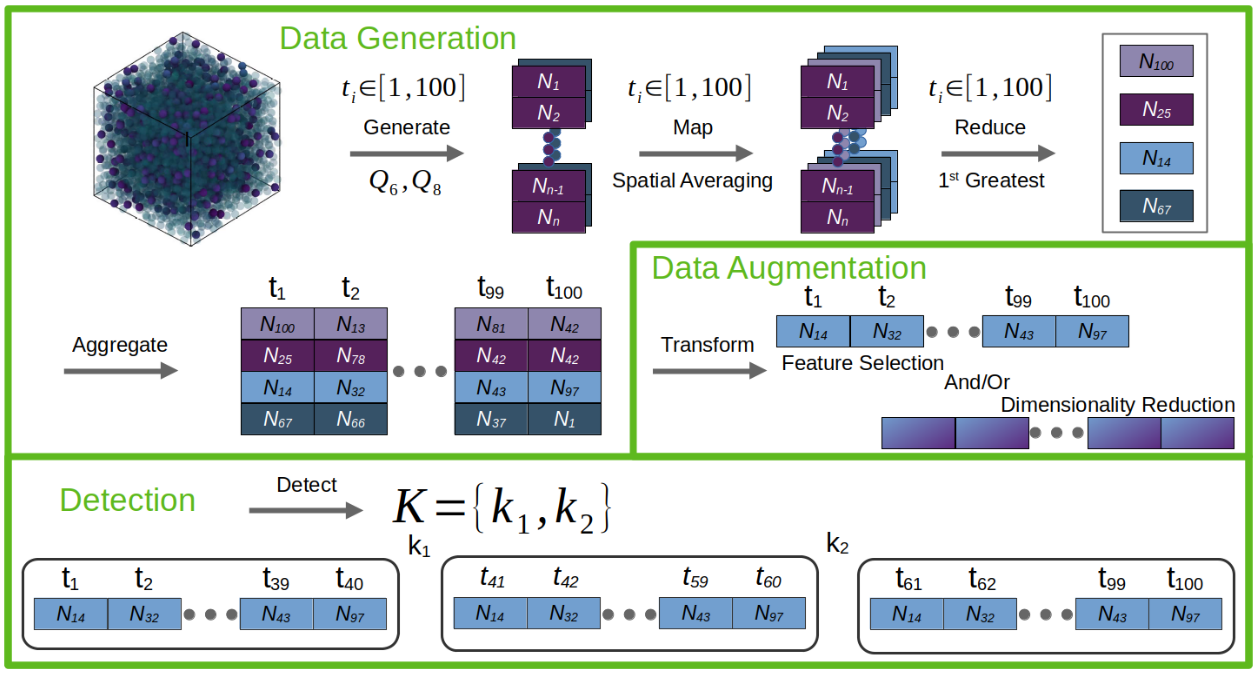

Figure 3.2: An illustration of the typical pipeline for event detection in a particle trajectory. We use the same numbers as the protein conformer example (change points are 40 and 60 and the total frames are 100). Arrows indicate steps and small colored rectangles represent individual per-particle feature values generated from the trajectory. Green boxes separate steps into three general stages: data collection, data augmentation and detection. Data is *generated* from the frame and *mapped* to a new distribution and to itself (note the replication of the original features after mapping). The distributions are then *reduced* into one scalar feature each. For the *generate*, *map* and *reduce* steps underneath the arrows, we provide potential functions/applications of the step. The features are then *aggregated* across frames and *transformed* with either feature selection or dimensionality reduction. Finally, the change points as well as the number of change points are *detected*. The *transform* step is not required, and the *aggregate* step could immediately precede the final step, *detect*. The figure assumes per-particle features only. For global features, the *generate* step immediately precedes the *aggregate* step — they could also optionally be *mapped* first.

as system potential energy but is required for per-particle or high dimensional properties (vector features) like the per-particle potential energy. Examples of scalar features resulting from a *reduce* step include the maximum value, minimum value, mean, mode, median, range, n-th greatest, n-th least, etc. Reducers that operate on the extrema of multi-dimensional features are often an appropriate choice because transitions often occur at extrema (minima or maxima) of a feature.

As an example, consider the utility of the n-th greatest and n-th least reducers in the context of a study of crystallization. The process of homogeneous nucleation and growth of a crystal from a metastable fluid phase starts when local fluctuations cause a group of neighboring particles to form an ordered cluster. If this cluster reaches or exceeds a critical size, it begins to grow, leading to the production of the crystal phase [68]. This process is known as classical nucleation theory. Ideally, CPD would have access to information on the initial fluctuations conspiring to produce the cluster as well as the cluster's subsequent growth. In the following analysis, we assume (i) we have already constructed or defined features that adequately distinguish the solid and fluid phases and (ii) the initial fluctuation is small compared to the system size. To detect the nucleation event, we must use reducers closer to the extrema such as the 1st and 10th greatest or least values (e.g. for local densities). Selecting near the extrema is necessary as most particles are fluid when a nucleus forms and selecting reductions that average over the distribution or select near the mean will not register a nucleation event. On the other hand, as the nucleus grows to 10 then 20 then 50 then 100 particles, more particles assume the approximate feature vector of the crystal. Thus, the 100th greatest or least reducers would capture the growth of a cluster up to a size of 100 particles.

While never required, vector features can be mapped to other distributions before aggregating. An example of a useful mapping is spatial averaging of a feature over its neighbors as is sometimes done with Steinhardt order parameters [77].

After a set of scalar features describing a single frame of the trajectory is generated, the process repeats across all trajectory frames and the results are *aggregated* into a single multi-dimensional, time-dependent signal. This signal can be sent directly to the *detect* step or to the *transform* step. The *transformation* step can involve any combination of three tasks: feature selection, dimensionality reduction and signal filtering. Next, we describe each of these data augmentation tasks as they pertain to `dupin`.

3.2.1.3 Data Augmentation

After features have been reduced, the number of features may easily be in the hundreds. In particle trajectories, each feature incurs noise due to thermal fluctuations — the same fluctuations involved in, e.g., nucleation and growth events in crystallization. As a result, this thermal noise increases the chance of spurious or undetected events for two related reasons. First, one or more features may fluctuate enough to be mistaken as an event. If we assume a baseline chance for such a fluctuation to be 1%, then a signal of 200 features has an 86.6% chance of recording such a spurious event. Second, given n features, if the event appears in only one feature, then as $n \to \infty$ the relative reduction in cost to fitting to that one feature decreases drastically. Assuming comparable noise in each feature, then for n features each feature contributes $1/n$ to the cost. For 200 features, the reduction in cost for fitting to a single feature's change point is at most 0.5% percent of the original cost.

To prevent the noise in high feature dimensions from leading to poor event detection, data augmentation through feature selection or dimensionality reduction can improve performance. The amount of improvement depends on the dimension of the feature set, noise level and other characteristics of the signal. Furthermore, data augmentation drastically improves computational performance of the event detection scheme without diminishing detection performance. The cost of the detection algorithm used in this paper is linear in the dimensionality of the feature set; that is, doubling the number of features reduces the performance by half.

`dupin` currently implements two different approaches to feature selection: (a) "mean-shift" filtering and (b) feature correlation. Mean-shift filtering requires an event-signifying shift in the mean value of the feature distribution and is determined by examining the beginning and end of a signal. We compare their means and standard deviation to each other and filter features based on the likelihood that either end's distribution would produce the mean of the other. Thus, the mean shift filtering is different from a mean-shift cost function detection. Here, we are making a *yes/no* decision to include a feature rather than targeting the exact location of any change.

Feature selection via feature correlation is done through spectral clustering [13], where the similarity matrix is computed based on the correlation matrix of the signal. A pre-determined number of features from each cluster is taken based on a provided feature importance score or is randomly selected from a cluster. More information on these methods for feature selection can be found in A.3. Other feature selection methods such as forward selection or backward selection seamlessly interoperate with `dupin` and can be used as well. Both methods can be found in scikit-learn [100].

For dimensionality reduction, `dupin` implements a machine learning (ML) classifier to reduce the signal to one dimension based on local signal similarity; this approach is a slight variation from the one found in Reference [56]. Signal reduction is accomplished by taking a sliding window across the original signal and using an ensemble (collection) of weak learners to determine the similarity of the window's left and right halves. A weak learner is one with limited ability to discriminate between classes (e.g. a decision stump [64] that classifies based on a single *yes/no* condition). Thus, weak learners cannot distinguish window halves based solely on noise due to their limited discrimination ability. This limitation is a desirable property as we only want the classifier to discriminate on *significant* differences between window halves. Each learner in the ensemble is trained and tested on different data within the window via a stratified shuffle split (from scikit-learn), which reduces the chance that a bad data partition will decrease the test loss. To determine the local signal similarity, we label each frame in the left window half with the class label zero and each frame in the right window half with class label one. The classifiers within the ensemble are then trained on a subset of the data and tested on the remaining data. The testing loss (e.g. the zero-one loss) is used as a metric of dissimilarity. When there is no event, the classifiers should have nothing but random noise to train on, resulting in the classifier being wrong on test data $\sim$50% of the time. However, when an event occurs within the window, a classifier can train on differences between the window halves and accuracy tends towards 100% (i.e. 0.0 loss).

For each window position, we take the average loss as our one-dimensional signal (here, the zero-one loss). Event detection can then be performed along this single dimension, using, for example, a simple mean-shift (see A.1) or linear cost function. Figure 3.3 shows a graphical depiction of the dimensionality reduction implemented in `dupin`. Other schemes and algorithms for dimensionality reduction such as principal component analysis (PCA), uniform manifold approximation and projection (UMAP) [88], etc. can be used to reduce the number of features into a few information-dense dimensions as well.

Signal filtering is commonly used for smoothing a signal and is similar to the *mapping* step in the data generation stage. Here, the completed signal is transformed along the temporal dimension into a new signal. Signal filtering is commonly used for smoothing a signal. `dupin` has a rolling mean signal filter, which smooths noise by averaging the signal across neighboring frames. Other signal filters are available in packages like SciPy and can be used easily with `dupin`.

3.2.1.4 Detection

The final stage in `dupin`'s pipeline is event *detection*. To detect local events, two cost functions are available for use in `dupin`. Both are based on piecewise linear fits of time versus signal features, and both have increased cost when this fit has a higher summed p-norm error $|y - f(x)|_p$. In the examples in Section 3.3, we use the summed and square rooted 2–norm.

The first cost function, C_1, in `dupin` computes the p-norm loss of the piecewise, least-

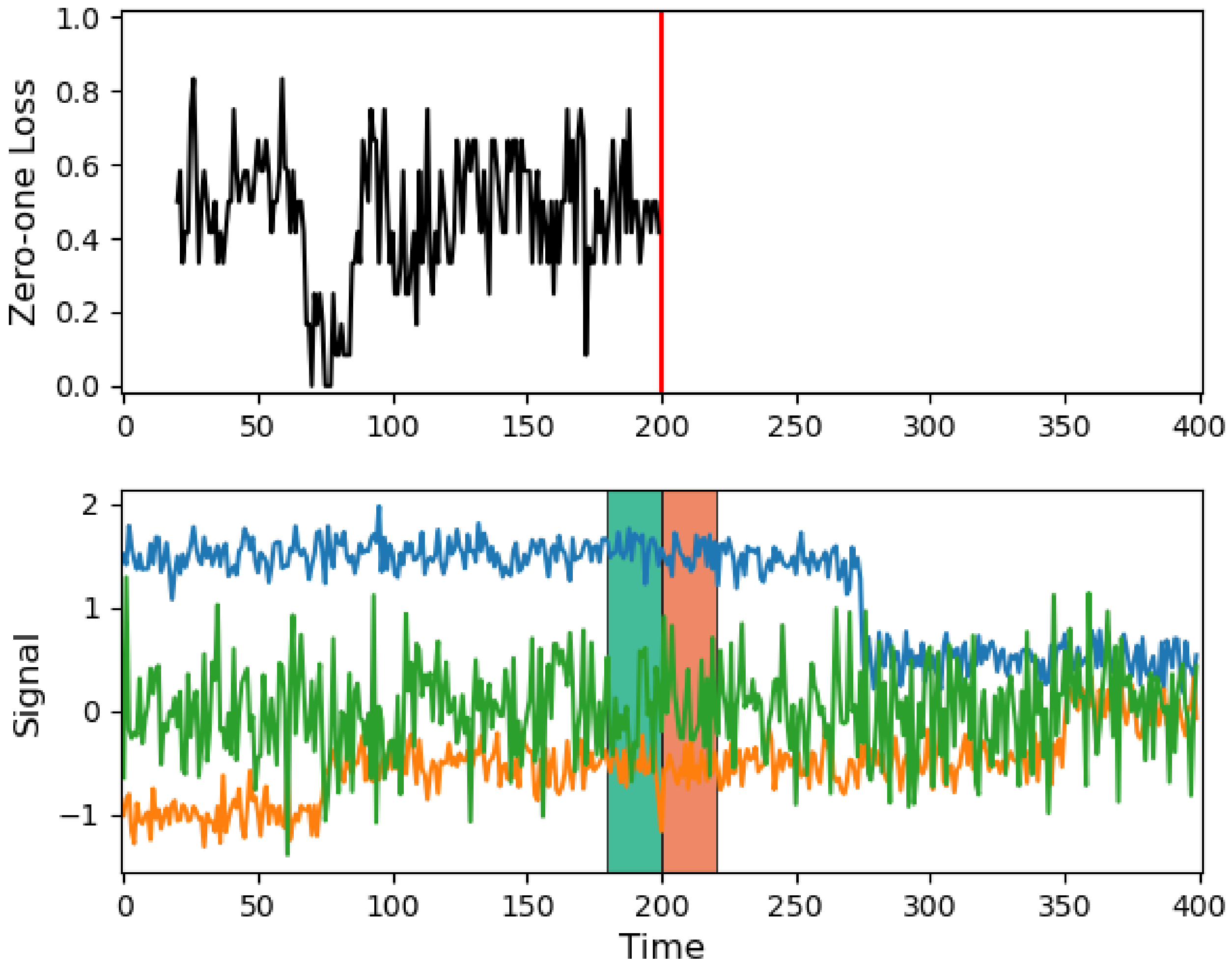

Figure 3.3: An example highlighting the use of classifiers to reduce a three-dimensional signal to one dimension. The process was stopped with a window center frame of 200 for instructive purposes. (a) A plot of the three-dimensional signal. Change points are located at 75, 275 and 350. The box centered at frame 200 represents the two window halves: $[190, 200)$ and $[200, 210)$. (b) The zero-one loss from the start of the signal to the current window. Notice the drop towards zero loss near frame 75, which corresponds to a mean-shift in (a) (blue line).

squared linear fit of each feature as a function of time. This procedure minimizes the p-norm via a least-squared fit on the slope m and intercept b for the given sub-signal $S[i, j)$. To prevent units or feature magnitudes from contributing to faulty detection, we map all features to the unit square independently so that the range for each feature is $[0, 1]$. After mapping to the unit square, C_1 is given by:

$$C_1(S[i, j)) = \min_{m,b} \sum_{x=i}^{j} |S[x] - (mx + b)|_p, \tag{3.2}$$

where $||_p$ is the p-norm, $S[x]$ is the signal value at frame x, and $mx + b$ is the linear fit of the signal where m is the slope of the linear fit and b is the y-intercept.

The second cost function, C_2, is similar to the first, but the linear fit is determined simply by drawing a line from the beginning point to the end point of the sub-signal, without any fitting. As a result, this cost function is more sensitive to sudden shifts and changes in slope in the signal compared to C_1. The C_2 cost function is given by:

$$\begin{aligned} C_2(S[i, j)) &= \sum_{x=i}^{j} |S[x] - (mx + b)|_p \\ m &= \frac{S[j] - S[i]}{j - i} \\ b &= S[i] - mi, \end{aligned} \tag{3.3}$$

The only optimizable parameters in Equation 3.3 are the locations of the change points. In cases where greater sensitivity is desired, C_2 is a viable alternative.

Any detection algorithm can be used in `dupin`. For this work we use the Python package `ruptures` [130], which implements various algorithms for solving the optimization problem already posed (see A.1 for more information). `dupin` provides the implemented cost functions above to the detection class from `ruptures` (we use the dynamic programming solver in this work). Using `ruptures`, we find the optimal change points for a given number of change points.

After detection, we still have one more problem to solve: finding the optimum number of change points. To do this, we find an elbow in the total cost function as a function of the number of change points n. The elbow is defined as the point of maximum curvature but can be expanded to discrete points. `dupin` can use any elbow detection method that works with discrete points. We choose for this work the kneedle algorithm [113] found in the `kneed` Python package; kneedle behaves well with the test systems studied and provides a hyperparameter allowing for sensitivity control in elbow detection. The sensitivity parameter also means the algorithm can return no elbow, allowing `dupin` to select zero change points as the correct CPD. More details on the kneedle algorithm and its usage in `dupin` can be found in A.2.

3.2.2 Online Detection

The above outlined event detection approach can also be used for online detection (i.e. while the particle trajectories are still being generated). In online detection, the *detection* step is applied on-the-fly to the set of frames generated up until the moment of detection. This set of frames can extend to the beginning of the trajectory or be a sliding window of the last N generated frames. To use `dupin` online, a predefined set of features is calculated on-the-fly at the desired frequency leading to a continuous application of *generate* to *aggregate* steps. If using a sliding window rather than the whole trajectory, the order parameters are placed into a “first-in first-out” (FIFO) queue, so that the sliding window moves with the time evolution of the data. Sliding window is the preferred approach for online detection, due to improved performance and easier CPD setup for detection. Using the queue approach allows CPD to be run on only part of the trajectory, which dramatically speeds up detection, making it more viable for online use. Each time a new frame is added to the signal, CPD can be run on the current data in the queue. The algorithm works best when the window size is commensurate with the size of a single event. Either way, `dupin` should be run with the assumption that we are expecting to detect up to one event. However, in practice we run CPD up to a change point set size of approximately four as any elbow detection scheme will need some points beyond the elbow to detect it. To detect multiple events, the queue should be cleared after detecting an event. If this is not done that event will continue to be detected in consecutive runs, leading to undesirable behavior.

3.3 Example Applications

We now demonstrate `dupin`'s event detection scheme step-by-step using `dupin` for two example systems to highlight its usefulness and versatility. All trajectory data was produced by molecular dynamics (MD) simulations run using HOOMD-blue [11, 10, 22]. Feature vector construction (*generation*) was carried out using `freud` [104, 33] in conjunction with `dupin`. The data was organized and managed using the `signac` framework [4, 103, 32]. System visualization was performed using OVITO [124].

3.3.1 Nucleation and growth of a binary crystal of particles

Our first example is a binary system of point particles interacting via the Mie potential [93] ($n = 50$, $m = 25$) with a size ratio of 0.55. The simulation was run using MD for 36.8 million time steps in the isothermal isobaric ensemble ($T = 0.35$, $P = 0.052$, $N_p = 27,000$, in reduced units) to simulate the solidification of a liquid into a crystal by homogeneous nucleation and growth. The simulation was initialized in a NaCl lattice with random placement of the two species, which quickly dissolves upon thermalizing at slightly higher temperature prior to a quench to the target T.

To *generate* the signal we compute the Voronoi polyhedron volume [134, 133] and MSM [92] for spherical harmonics $l = 2, 4, 6, 8, 10, 12$ for each particle. For each feature, we *map* to two distributions — itself (no transformation) and the Voronoi tessellation neighbor average [77]. We then *reduce* each distribution (raw and averaged) to six features: the

1st, 10th and 100th greatest and least values. After *reducing*, the MSM for each l produces twelve features, six from the raw distribution and six from spatial averaging. Following *aggregation*, we *transform* the signal via feature selection through a mean-shift filter with sensitivity of $10e{-}4$. After the mean shift filter, we perform two different analyses. First, we take the filtered signal and *detect* the change points using C_1 with `rupture`'s dynamic programming algorithm, using `kneed` for elbow detection with a sensitivity of 1, for $|K| \in [1, 10]$. For the second analysis, we follow filtering with the ML classifier dimensionality reduction (window size 80) introduced in Subsection 3.2.1.3. We used 200 decision stumps (decision trees of depth one) on features selected by the mean-shift filter to classify windows halves. The zero-one loss is then smoothed over the neighboring three errors on each side with the mean signal filter. We then *detect* events using an L_1 mean shift cost function [66].

The first detection scheme (without the ML classifier) detects two change points — see Figure 3.4 (e, f). The linear detection was dominated by a region of sharp change in several properties ≈160–200 during crystallization. We note that the continuing shift of some of the Voronoi polyhedra volumes or MSMs at the end of the simulation is roughly linear, meaning C_1 does not penalize grouping them into a single sub-signal.

On the other hand, the classifier approach results in a much larger transition event window (i.e. the change points are farther apart). Such a partitioning is expected as the dimensionality reduction scheme picks up on *any* deviation across the window. As a consequence, `dupin` detected change points at frames 90 and 210, in contrast to the linear cost function approach. We note that the reason for the change point locations can be seen in Figure 3.4 (h) where the smoothed average zero-one loss is plotted. The system is still undergoing structural changes at frame 210, indicated by the value of average loss which while higher than 0.0 is lower than the expected value outside a transition of 0.5. This behavior is explained by the observed trends in some Voronoi polyhedra volumes and MSMs, which are still not in equilibrium (flat) after frame 210. The final slice of the trajectory (frame 210 till the end) is thus associated with a new phase of the ongoing transition, which is not finished by the end of the trajectory as indicated by the average loss value.

These same schemes can be followed to apply `dupin` to simulations of molecules, nanoparticles, colloids, polymers or other "particle"-based systems that generate particle positions (and possibly orientations) as a function of time, making `dupin` highly extensible and generalizable.

3.3.2 Collapse of polymer in poor solvent

Our second example demonstrating the use of `dupin` to detect transition events involves detecting the collapse of an isolated polymer chain in an implicit solvent following a change from good to poor solvent. We simulated a polymer comprised of 5000 connected polymer beads interacting via a Lennard-Jones pairwise interaction, with bonds between beads modeled by a simple harmonic potential $U = -0.5k(r - r_0)$. The MD simulation was performed in the canonical ensemble ($N = 5,000, V = 10^6, T = 1$). Following equilibration (1.2 million steps) in good solvent, the simulation was run for 2.5 million steps, where the ε of the Lennard-Jones potential was increased to mimic a change in the Flory χ parameter to a poor solvent. Over the 2.5 million steps, ε was increased from 0.1 to 1.25 over 50 intervals of 30,000 steps, running at the final ε for 1 million steps. Increasing the ε parameter causes the polymer beads to start aggregating. Several stages of aggregation are observed in which the number of lobes decreases in stages until final metastable configuration of two lobes is obtained.

We *generate* the signal using the polymer end-to-end distance, the local density defined as Voronoi polytope volume and the number of clusters of neighboring beads using `freud`'s clustering algorithm ($r = 1.2$). Because Voronoi polytope volume is the only non-scalar feature, we *reduce* only it. We *reduce* the volumes to six features: the 10st, 100th and 1,000th greatest and least values. We do not *transform* the data due to low dimensionality. Finally, for offline detection, we *detect* the change points using C_1 with `rupture`'s dynamic programming algorithm, using `kneed` for elbow detection with a sensitivity of 1, for $|K| \in$

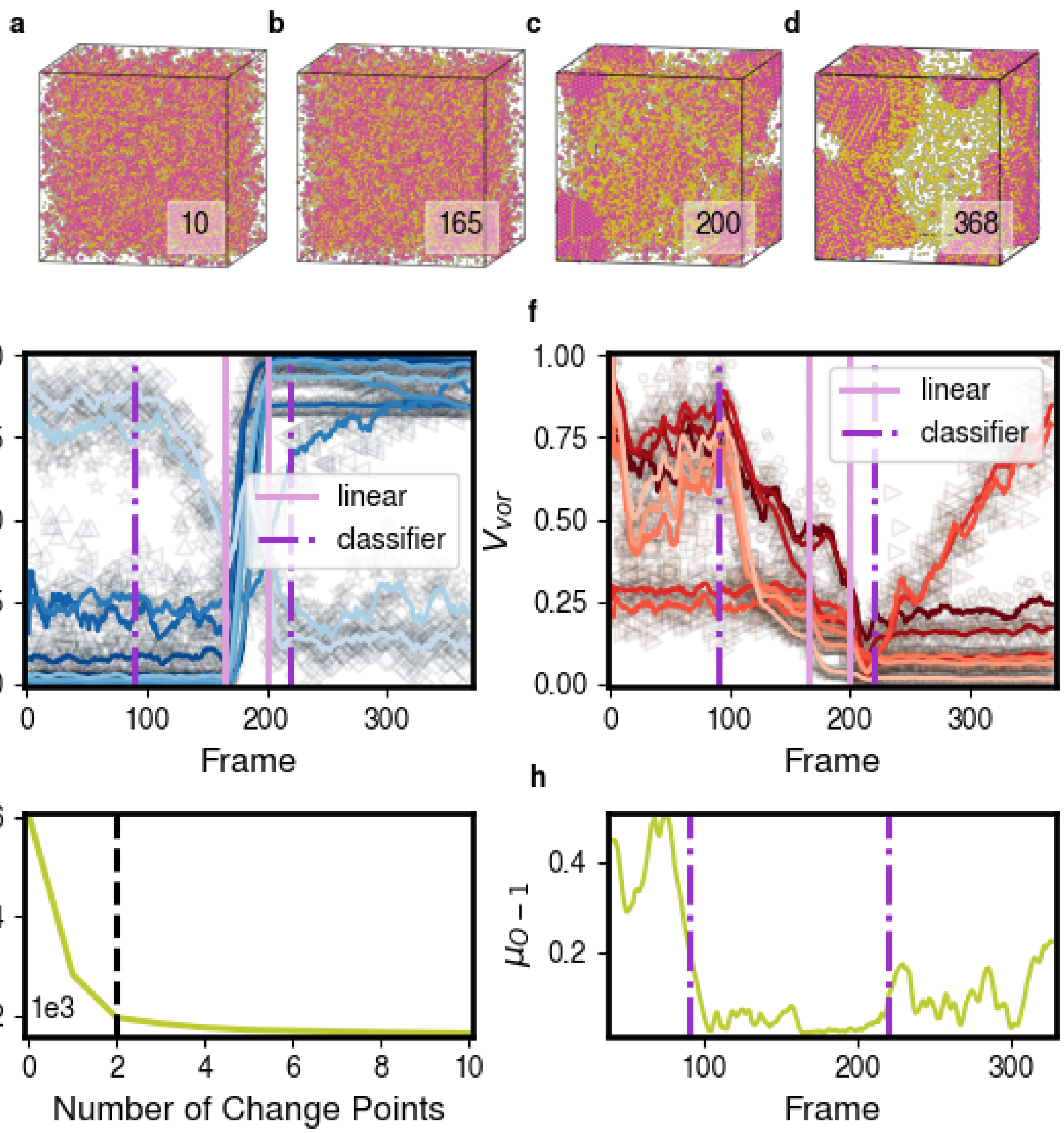

Figure 3.4: Change point detection applied to binary system of particles interacting via the Mie potential. (a-d) Images corresponding to the beginning of the simulation, the locations of the change points for the linear detector and the end of the simulation. Particles are colored according to their type. (e-f) Plot of all MSM $l = 12$ (e) and Voronoi polyhedra volumes (f) that pass the mean-shift filter (sensitivity $10e^{-4}$). Solid lines are the (rolling mean) smoothed features and the translucent dots are the actual data points. The smoothing is purely for visualization and all calculations were done using the actual data points. The detected change points for the linear cost function are the dashed light purple lines and the change points for the ML method (Section 3.2.1.3) are dotted-dashed dark purple. An L_1 mean-shift cost function was used for detection in the ML approach. (g) Plot of the cost associated with the optimum n change points computed using features that pass through a mean-shift filter with the linear cost function. (h) Plot of the smoothed (averaged over three neighbors on each side) mean zero-one loss function as well as the change points from the classifier approach (dotted-dashed dark purple vertical lines).

$[1, 10]$. We also perform online *detection* using the sliding window approach with a window size of 50. All steps up to *detection* are identical though the signal is fed to the detector as a stream rather than simultaneously. The detector used for the online *detection* uses the same cost function, algorithm and elbow detection but only goes to $|K| = 6$. We clear the window whenever an event is detected.

Figure 3.5 shows the CPD analysis for this system. The collapse of the polymer due to poor solvent can be seen in Figure 3.5 (a-e). The collapse *looks* visually continuous, although the images show a collapse mediated by multiple intermediate, discrete steps and the shape of the order parameter with time is sigmoidal. The offline detection scheme only detects the transition to the final snapshot of the simulation because fitting to the individual sections of the collapse does not sufficiently decrease the cost function. On the other hand, the "ramp up", "growth" and "slow down" behavior of the sigmoid leads to three events detected with online detection at frames 65, 95 and 140. This granularity results from the max signal length of 50, which increases the relative reduction in cost for fitting to the three sections of the polymer collapse. Processing the entire trajectory for online detection took 820 milliseconds ($\pm 3.12\mu s$) on a single core of a 3.0 GHz Intel Xeon Gold 6154. This speed is sufficiently fast to use in real time applications such as autonomously triggering simulation protocols during a simulation. Figure 3.5 (f-g) shows `dupin`'s ability to detect the collapse and its component parts in the case of online detection. Figure 3.5 (h) shows the cost associated with the choice of optimum n (number of change points) for offline detection. Likewise Figure 3.5 (i) shows the attempt to detect an elbow in the online case with a proxy for the cost plot, which otherwise would need to be shown for every frame.

3.4 Discussion and Conclusions

We've demonstrated how the procedure described in Section 3.2.1 allows for detection of transition points within a simulation trajectory with a high degree of accuracy. The obvious

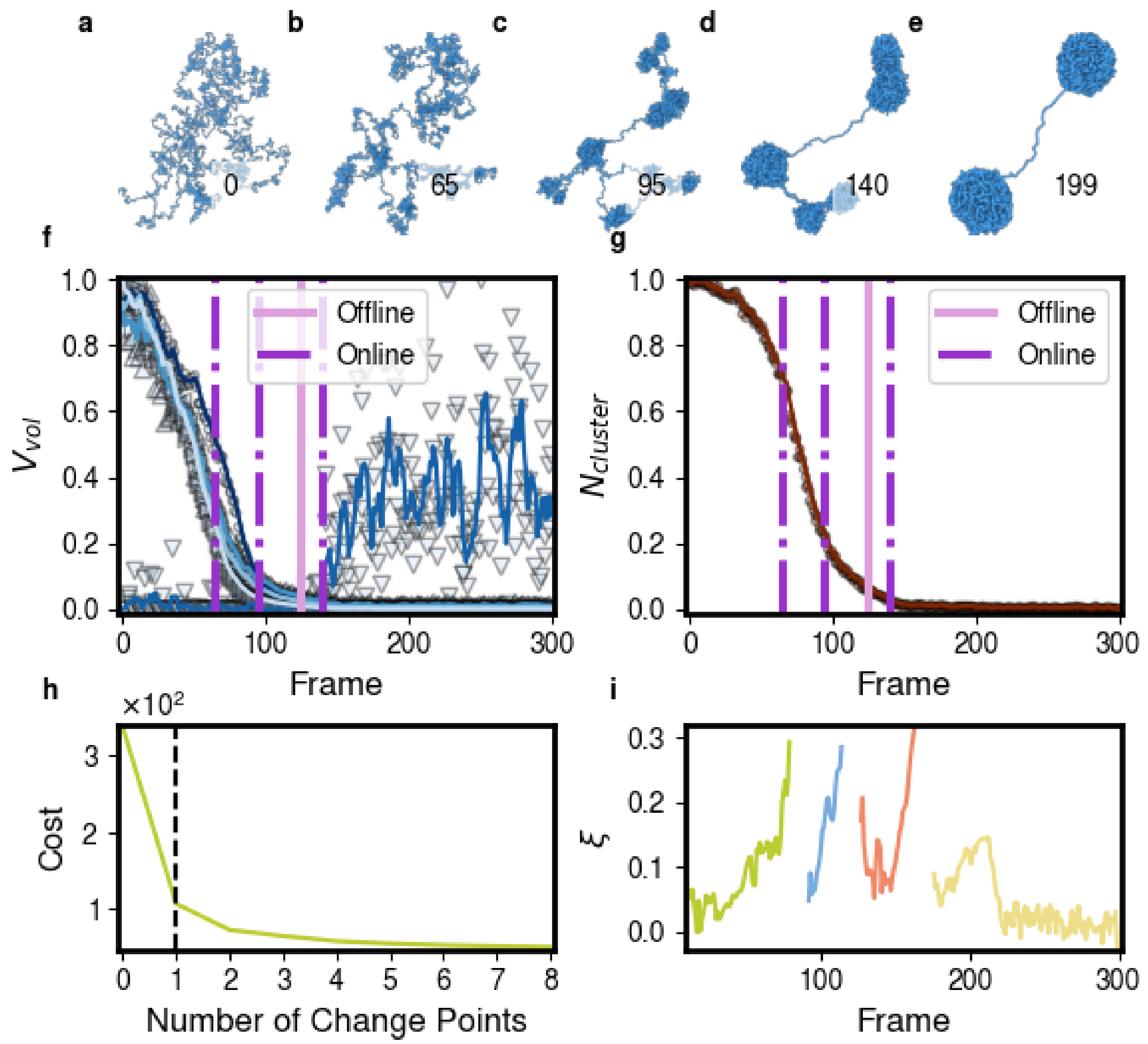

Figure 3.5: Change point detection (offline and online) applied to polymer trajectory. (a-e) Images corresponding to the beginning of the simulation, change points and end of the simulation. (b-c) Online-only change points. (d) Both offline (125) and online (140) have very similar values of change points and only frame 140 is shown. (f-g) Plot of Voronoi polytope volumes (f) and the number of clusters in the simulation according to `freud`'s clustering algorithm with a 1.2σ cutoff (g). Solid lines are the (rolling mean) smoothed features and the translucent dots are the actual data points. The smoothing is purely for visualization and all calculations were done using the actual data points. The light purple solid line is the change point determined by `dupin`'s offline detection. The dark purple dashed dotted lines are the change points determined by `dupin`'s online detection. (h) Plot of the cost associated with the optimum n change points computed on all features for offline detection. (i) Plot of relative improvement of costs from adding a second change point in online detection. The formula used is $\xi = (c_1 - c_2)/c_0$ where c_i is the cost for selecting i change points. Individual curves represent independent detections (i.e. after the window is cleared).

benefit from this approach is the automation of structural transition detection within a study. In studies with hundreds and thousands of simulations the dividends of this approach increase exponentially. Our method does, however, require informative descriptors for the transition. This requirement can be met by selecting a wide variety of descriptors and applying feature selection tools to the signal afterwards.

From the examples presented, we conclude that the classifier approach results in a larger partitioning of the detected events compared to results produced using the linear cost function approach. However, the classifier approach falls short in cases where multiple events are overlapping or in cases where there are abrupt changes at the end of the signal. For instance, if we were to apply the ML method to a system with only one frame into a transition, no event would be detected. The linear approach is better suited for detection in such scenarios and in scenarios where multiple events are expected as showcased in the online polymer example.

`dupin` opens the doors for new ML applications to phenomena such as crystallization pathways, defect formation and active matter, all of which involve structural transitions. By curating the transition data for ML applications, large scale studies that would have been prohibitively costly in terms of human hours are now accessible. Furthermore, leveraging `dupin` for online event detection holds promise to lessen data storage and processing demands and provide a powerful avenue for real-time control over simulations and experiments.

The source code can be found at GitHub at `https://github.com/glotzerlab/dupin`. The documentation hosted by Read the Docs, `https://dupin.readthedocs.io`, also contains three additional examples of event detection: one for a simple simple of WCA spheres forming FCC, one of hard truncated tetrahedra forming cF432 and one of ionic beryllium and chlorine forming a nematic then crystalline phase. The analysis code and data will be available on on Deep Blue Documents following publication.

CHAPTER 4

PGOP: A Point group Order Parameter for Analyzing Local Crystal Symmetry

This chapter is adapted from a manuscript authored by Brandon L. Butler, Maria W. Rashidi and Sharon C. Glotzer to be submitted to a peer-reviewed journal.

4.1 Introduction

Molecular simulations can provide a rich collection of information: particle positions, momenta, forces, etc. However, the shear volume of information is a double-edged sword necessitating the meticulous sifting of data to extract the pertinent details. This "sifting" is conducted using order parameters. Order parameters distill multiple degrees of freedom into one or a few numbers that measure ordering or the departure from randomness in a system. OPs are particularly useful for analyzing crystals [76, 16, 26, 92, 122].

A crystal is defined as a solid that forms a lattice when considering particle positions. The fundamental unit of a crystal is the unit cell consisting of two items: (1) the lattice vectors that are the n directions and lengths of translational symmetry and (2) the basis positions that define occupied points within the unit cell created by the lattice vectors. The unit cell exhibits various symmetries, the set of which is called a space group. Unit cells also have a point group, which accounts for a subset of symmetries of a space group, removing screw axes and glide planes in three dimensions. Symmetry is also a feature of a crystal's one or more Wykoff site(s). A Wykoff site is a subset of positions from the basis positions that are invariant under specific rotations, reflections and/or inversion. Each basis position occupies one Wykoff site within a crystal. The fixing of basis positions across unit cell translations and intra-unit cell Wykoff site symmetries leads to the local bond orientational ordering of a crystal. Discerning the point group and space group directly from simulations is an important step in identifying a crystal.

The most popular OP that analyzes bond-orientational symmetry is the Steinhardt OP [122], Q_ℓ, where ℓ is the spherical harmonic number. The Steinhardt OP depends on the bond order diagram (BOD) of a particle. A BOD is a projection of neighbor bonds, vectors pointing from a central particle to a neighbor, onto the unit sphere. The spherical harmonic expansion of the BOD is taken, and the coefficients are combined in a rotationally invariant way. Therefore, the Steinhardt OP is translationally invariant due to the BOD and rotationally invariant due to the specific combination of spherical harmonic expansion coefficients. However, due to this combination of coefficients, the Steinhardt OPs cannot specify what symmetry is forming — only that one compatible with a given spherical harmonic number ℓ does. Another problem comes from discontinuous changes in OP values when the neighbor finding method changes. An extension to the Steinhardt OPs, Minkowski Structure Metrics (MSM), address this specific issue by (1) computing neighbors via Voronoi tessellation [134, 133] and (2) weighting the BOD by the polytope facet area.

We improve upon this class of OPs by developing one that can detect the local formation of specific point group symmetries. To create the OP, we modify an OP developed by Michael Engel [39] that quantifies at global ordering. Our algorithm, the Point Group Order Parameter, successfully handles noisy local environments while providing local per-particle information on the exact kinds of symmetry formed, unlike MSM or Steinhardt OPs. The PGOP preserves the full information of the bond order diagram by avoiding the coefficient combinations of Steinhardt or MSM. The combining of coefficients removes the point group specific information, and in preserving them, the PGOP can quantify specific point group ordering. We developed the OP to study nucleation events and what precedes them with a particular interest towards which polymorphs exist in the liquid prior to crystallization.

We begin by presenting a full derivation of the PGOP. From there, we proceed to compare the PGOP's behavior to MSM [92]. We continue by looking at the PGOP behavior in a complicated crystal (γ-brass); a system of particles interacting via the oscillating pair potential [94] forming A15; a binary system of LJ-interacting particles forming a glass; and a system of LJ-interacting particles that crystallize into a defected FCC. Finally, we conclude with some closing remarks about PGOP and its potential utility.

4.2 Results

4.2.1 Method Derivation

We begin with a derivation of the PGOP algorithm — for a derivation of the global OP inspiring this work, see [39].

The first step of the algorithm is to compute a weighted bond order diagram (BOD), an extension of a BOD is where a weight is assigned to all neighbor bonds. The weighted bond order diagram is defined as,

$$\mathrm{BOD}(\theta, \phi) = \frac{4\pi}{\sin(\theta)} \sum_{i=1}^{N} \frac{w_i}{w} \delta_{\theta_i,\phi_i}(\theta, \phi), \tag{4.1}$$

where w_i are optional weights per neighbor bond and $w = \sum^N w_i$ and δ is the dirac-delta function. A prefactor $4\pi/\sin(\theta)$ is added to the weighted BOD to simplify the coefficients of a spherical harmonic expansion: The coefficients with this prefactor work out to,

$$Q_{lm} = \sum_i \frac{w_i}{w} Y_{lm}(\theta_i, \phi_i). \tag{4.2}$$

In this work we ignore the prefactor as it has no bearing on the final order parameter because we ultimately only compare correlations BOD where the prefactor would always be identical between BOD.

The choice of neighbors can lead to very different results, so we must carefully choose the method for neighbor finding. With systems of known and constant (among basis positions) coordination number, the n nearest neighbors neighbor-list will suffice. However, when dealing with crystals of multiple Wykoff sites and coordination numbers, a more general approach is warranted. For this purpose, we adopt the Voronoi tessellation [134, 133] as it is parameterless and has equivalent behavior in simple cases like FCC and SC to the informed n nearest neighbors approach. Also, weighting the Voronoi tessellation neighbor list by the area of the Voronoi polytope facets reduces sensitivity to thermal noise because points far away from the central particle have small weights.

However, even with a well-thought-out neighbor definition, the BOD would be highly susceptible to thermal noise due to its use of delta functions. Thus, we further modify the BOD by changing the delta function to a broader distribution. One example distribution and the one used for this work is the von-Mises Fisher [46] distribution, the analog of a Gaussian on the surface of the sphere:

$$p(x) = \frac{\kappa}{2\pi(e^{\kappa} - e^{-\kappa})} e^{\kappa \mu^T x}, \tag{4.3}$$

where κ is known as the concentration parameter and μ is the unit vector pointing to the center of the distribution. The concentration parameter functions something like an inverse standard deviation and large values serve to narrow the distribution. Because the von-Mises Fisher distribution is spread out, thermal effects will be muted. The effect of using the Fisher distribution can be seen in Figure 4.1. We would expect a flat line in Figure 4.1 (a-b) if the broadness of the distribution didn't matter ($\kappa \to \infty$ recovers the delta function). However, we can clearly see that the ability to detect order deteriorates past an intermediate κ. Figure 4.1 (a) has a sensitivity to κ due to optimization being more difficult with highly peaked distributions.

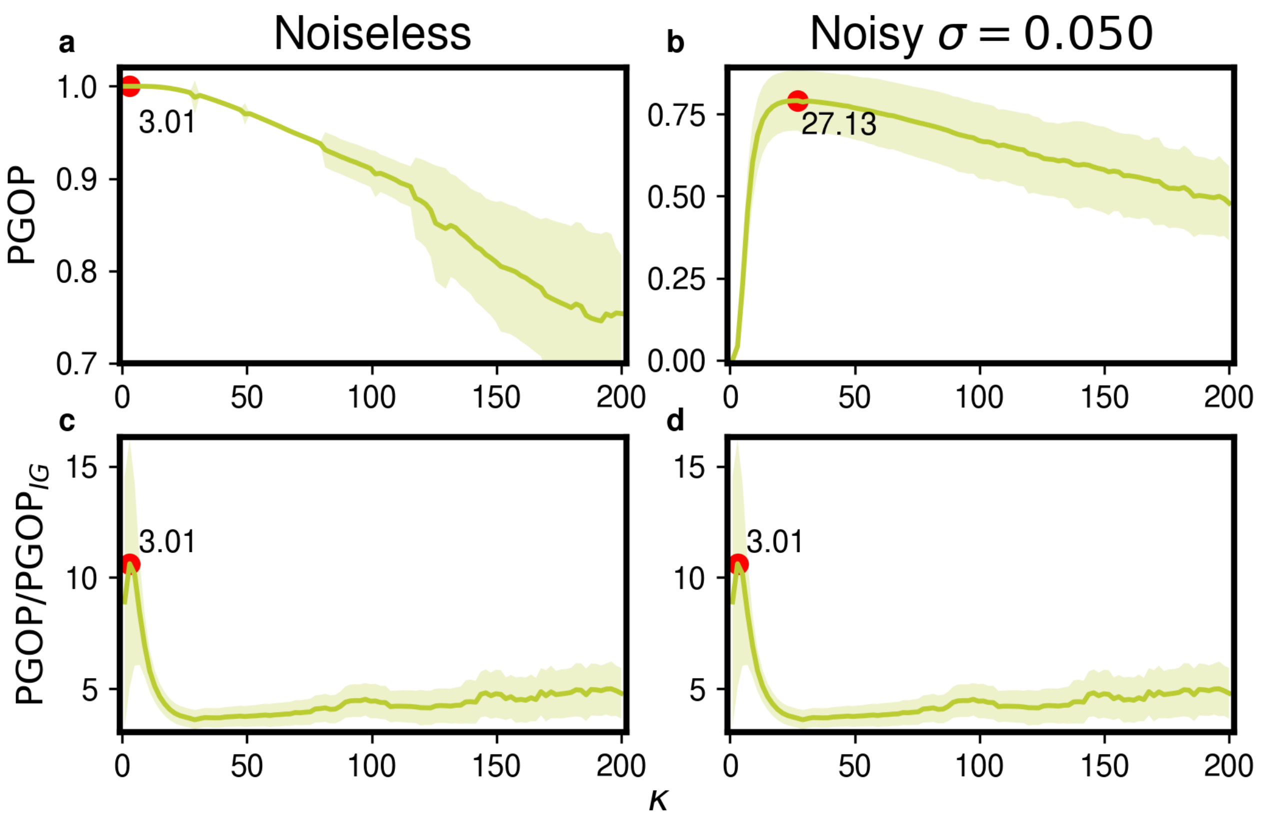

Figure 4.1: Plot of κ's effect on the evaluation of icosahedrally ordered environments, both perfect and noisy. Each line is the average over 25 optimizations at various random rotations. The shaded region above and below the line is one standard deviation. (a) Plot of the PGOP of a perfectly icosahedral neighbor shell with various κ. (b) Plot of the PGOP of noisy icosahedral neighbor shells with various κ. (c-d) Plots of the PGOP divided by the mean PGOP of 25 ideal gas samples for perfect (c) and noisy (d) neighborhoods. These plots (c-d) emphasize the distinctiveness of ideal gas and ordered environments.

With the modified BOD, we proceed to take a spherical harmonic expansion like Steinhardt OPs or Engel's OP. Spherical harmonics expansions are summations of Y_ℓ^m (i.e. $f(\theta,\phi) = \sum_{l=1}^{\infty} Q_\ell^m Y_\ell^m$) and are the equivalent of a Fourier expansion in Euclidean space. Just like Fourier expansions, the coefficients must be computed. The coefficients are defined through an inner product of the spherical harmonics, Y_ℓ^m, and the function being expanded. For the BOD the expansion coefficients are

$$Q_\ell^m = \frac{1}{4\pi} \int_{\theta=0}^{\pi} \int_{\phi=0}^{2pi} \mathrm{BOD}(\theta,\phi) Y_\ell^m(\theta,\phi)^* \sin(\theta) \mathrm{d}\phi \mathrm{d}\theta. \tag{4.4}$$

We solve the integral by a Gaussian-Legendre quadrature with fineness m resulting in $2m^2$ integrand evaluations. Here we note that the use of distributions over delta functions in the BOD also allows for more accurate expansions for a fewer numbers of neighbors.

With the spherical harmonic expansion of the BOD, we can symmetrize the BOD with respect to a point group. To accomplish the symmetrization, we take the Wigner D-matrix [136] for a symmetry group, G, of interest and apply it to the BOD via a specific formula (the semidirect product),

$$\mathrm{BOD}_{sym} = \sum_{\ell=1}^{\infty} \sum_{m'=-l}^{\ell} \sum_{m=-\ell}^{l} D_{\ell}^{m',m}(G) Q_{\ell}^{m*}. \tag{4.5}$$

This results in a symmetrized BOD that has the desired point group symmetry. A Wigner D-matrix is an encoding of one or more symmetry operations in $O(3)$ as finite matrix; thus, all point groups have associated Wigner D-matrices. Importantly, if the original BOD started with the desired symmetry, then the symmetrized BOD will have a normalized Pearson correlation of 1 with the original BOD [39, 136].

We can then compare BODs through a normalized functional inner product between the original and symmetrized BOD. The inner product results in an OP that is 1 if the original BOD exhibits perfect symmetry with respect to a chosen point group and 0 if there is no semblance of a point group in the BOD. We can calculate the inner product (due to the spherical harmonic expansion) as,

$$K(f_i, f_j) = \sum_{\ell}^{\infty} \sum_{m=-\ell}^{m=\ell} Q_{\ell,i}^{m} Q_{\ell,j}^{m}. \tag{4.6}$$

To normalize the inner product, we compute

$$\hat{K}(f_i, f_j) = \frac{K(f_i, f_j)}{\sqrt{K(f_i, f_i) K(f_j, f_j)}}, \tag{4.7}$$

which also gives the Pearson correlation between the two functions.

Thus, the degree of point group ordering for group G is $\hat{K}(\mathrm{BOD}, \mathrm{BOD}_{sym})$. We note here that a value of 0 just as surprising as 1 is; even the ideal gas has a non-zero value for finite numbers of neighbors (see the paper [39] for a more complete explanation). To better ascertain the deviation of PG ordering from a random distribution of points, we can normalize the PGOP by using its value in the ideal gas,

$$\mathrm{PGOP}_{norm} = \frac{\mathrm{PGOP} - \langle \mathrm{PGOP}_{IG} \rangle}{1 - \langle \mathrm{PGOP}_{IG} \rangle}, \tag{4.8}$$

where PGOP is the PGOP for the system of interest and $\langle \mathrm{PGOP}_{IG} \rangle$ is the mean or expected value of PGOP for the ideal gas with the current neighbor finding method. Due to the use of the ideal gas to normalize other system's PGOP, PGOP_{norm} can go below 0.0.

The algorithm does not conclude here; point groups have defined axes of rotation and reflection and, thus, are not rotationally invariant like their Steinhardt [122] cousins. Hence, we have to align optimally the rotation of the original BOD with the axes chosen in the calculation of the Wigner D-matrix. As the optimization over $SO(3)$ (we only need to consider the optimal rotation and not inversions/reflections) is in the general case very bumpy (non-convex), we perform a two-step optimization: a global search over $SO(3)$ and a local one-dimensional gradient descent into a local minimum.

Our global search consists of uniform rotations in $SO(3)$ which, without prior information, is the most unbiased choice for a global search. To obtain uniform rotations, we perform two steps. First, we find uniformly distant points on the sphere for rotational axes. To find uniform points on the sphere, we take the numerical solutions to the Tammes problem [125, 101] in three dimensions for n points. We then take m angles according to the Haar measure [53], which amounts to selecting angles that are equidistant along $1/\pi(\theta - \sin\theta)$. The Haar measure accounts for nonlinearities between rotation angles. Combining equidistant angles of rotation along the Haar measure with the Tammes problem solution results in uniform rotations in $SO(3)$ of $n \cdot m$ points [95].

Following this global search, we take the current optimal rotation and perform three gradient descents in series over the three dimensions of rotation. In developing the algorithm and implementation, we found three one-dimensional optimization to be more stable and to result in higher PGOP values than a single three-dimensional optimization. We represent the rotation by a three-dimensional vector $\vec{\alpha}$ which can be converted to the axis-angle representation ($\vec{x}$ and θ) via

$$\begin{aligned}\theta &= ||\vec{\alpha}|| \\ \vec{x} &= \frac{\vec{\alpha}}{||\vec{\alpha}||}.\end{aligned}$$

These conversions show that the representation is continuous, allowing for continuous optimization schemes. We cycle through these three one-dimensional optimizations of α indices until we converge to a solution.

This solution is not the symmetry of the Wykoff site of a particle as this is a function of the unit cell. Similarly, the output is not the symmetry of the lattice itself as in [39]. The final output is an OP that detects of point group order within local arrangements of particles.

4.2.2 Testing

We show a simple test of the behavior of the PGOP against MSM. We first look at configurations of perfect FCC and BCC with various levels of added Gaussian noise. We use Gaussian noise because it is uncorrelated between particles and therefore is more destructive on average than thermal noise. The behavior of the PGOP by itself would not be particularly illuminating, so we also perform all these calculations with MSM [92], computed using the Python package `freud` [33, 104], as well. In seeing how the PGOP compares to a standard and well-used OP, we can better evaluate the performance of the PGOP in perfect and noisy conditions.

First we look at the values for MSM and the PGOP on random configurations (i.e. ideal gases). For neighbor-finding, we use Voronoi tessellation [134, 133], which is required in MSM but optional for the PGOP. The distributions can be found in Appendix B.1. After deriving the baseline values for various PGOP and MSM, we then take the three crystals and create uncorrelated perfect and noisy configurations at various levels of Gaussian noise and compute the PGOP and MSM. For the PGOP, we use the O_h point group, which is the point group symmetry for the local FCC and BCC — the Voronoi neighbors for FCC form a cuboctahedron and BCC a truncated octahedron both of which have octahedral symmetry as their names suggest. For MSM, we use $l = 6$, which has generally been used to detect crystallization for FCC and BCC [126, 44] (for other l see Appendix B.2). In Figure 4.2, we can see that generally the PGOP detects the original ordering for higher levels of noise that MSM. Thus, the PGOP can still discern order even in high noise environments whereas for MSM the average particle is indistinguishable from an ideal gas particle. Note that in Figure 4.2, rather than plotting the PGOP or MSM directly, we plot the value normalized by the median of the PGOP for an ideal gas via,

$$Y = \frac{\text{PGOP} - \text{PGOP}_{IG,med}}{\text{PGOP}_{IG,med}}, \tag{4.9}$$

which is a slight variant of Equation 4.8. We use this variant to better access how likely a value of PGOP could be from the ideal gas versus an ordered structure.

4.2.3 Application of PGOP to Example systems

Having shown the general behavior of PGOP and MSM with respect to simple crystals, we now look exclusively at the PGOP of various systems crystalline and otherwise.

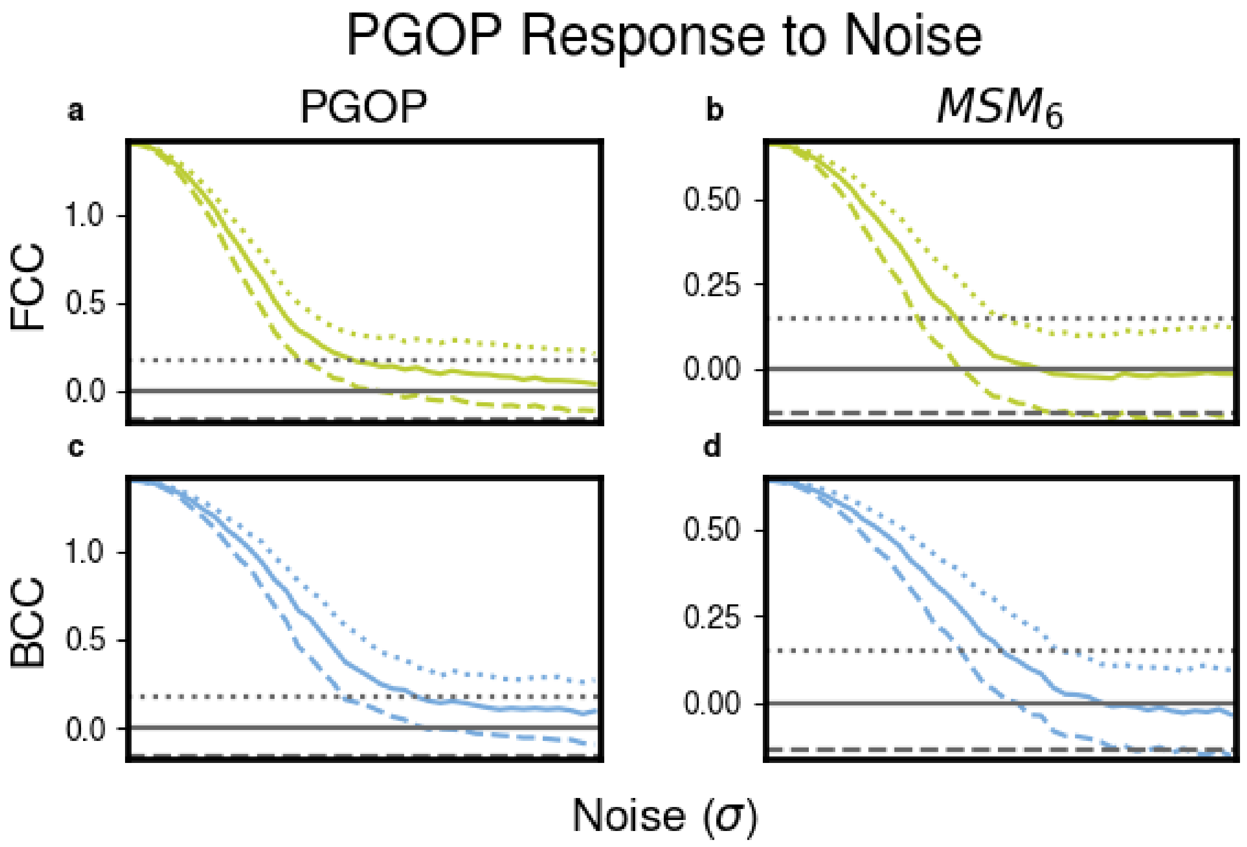

Figure 4.2: (a-d) Plots of the PGOP and MSM_6 for two different crystals (FCC and BCC). Plots show the 0.25, 0.5 and 0.75 quantiles in dashed, solid and dotted lines, respectively. The PGOP values have been normalized by the median of ideal gas, so 0.0 is the median PGOP of the ideal gas. Gray lines correspond to the 0.25, 0.5 and 0.75 quantiles of the ideal gas with the same line style designations as the crystal lines. Crystals have a unit cell of length 1 and go from a standard deviation of 0 to 0.2.

4.2.3.1 γ-brass

Next we examine a configuration of noisy (Gaussian noise with standard deviation 0.0025 with a lattice vectors length one) γ-brass (CuZn). The crystal has four Wykoff sites, two of which have the same symmetry (one occupied by Cu and another by Zn). We show that PGOP can linearly separates the distinct sites, providing insight into the local symmetries of particles. Figure 4.3 shows a Gaussian mixture model (GMM) clustering [106, 27] of the raw data as well as a corresponding plot showing points colored by Wykoff site. For the clustering, we varied the number of clusters from two to seven and chose the clustering that had the highest silhouette score [107]. The plot uses the first two dimensions of a linear discriminant analysis (LDA) projection [55, 78] given the Wykoff sites as class labels. LDA was chosen as it finds a linear projection, which tends to make clusters spherical and separate them among the defined axes, aiding visualization. We also note that LDA cannot *manufacture* difference; the separation of the clusters in (a-b) is due to the Wykoff sites being linearly separable in PGOP space. Regardless, the projection is done purely for visualization because we performed the GMM clustering on the raw data. The LDA projection and GMM were calculated using scikit-learn [100, 21]. Figure 4.3 part (a) shows a clear ability to distinguish between three of the four Wykoff sites in the original space. The two combined Wykoff sites are those that possess the same site symmetry and have nearly identical local environments.

4.2.3.2 Crystallization of A15

To further demonstrate the utility of our new OP, we look at two simulations of particles interacting via the oscillating pair potential [94]. This pair potential exhibits a diverse phase behavior and can be used to study the self-assembly of complex crystals, including even quasi-crystals [36, 40]. Here, we focus on a set of parameters that in molecular dynamics (MD) simulations lead to A15 (Cr_3Si Pearson symbol cP8). In A15, two Wykoff sites exist; one corresponding to each species in the metallic specification, Cr and Si. The simulation is run in MD and scripted using HOOMD-blue [11, 10, 22] and is either slowly-cooled or

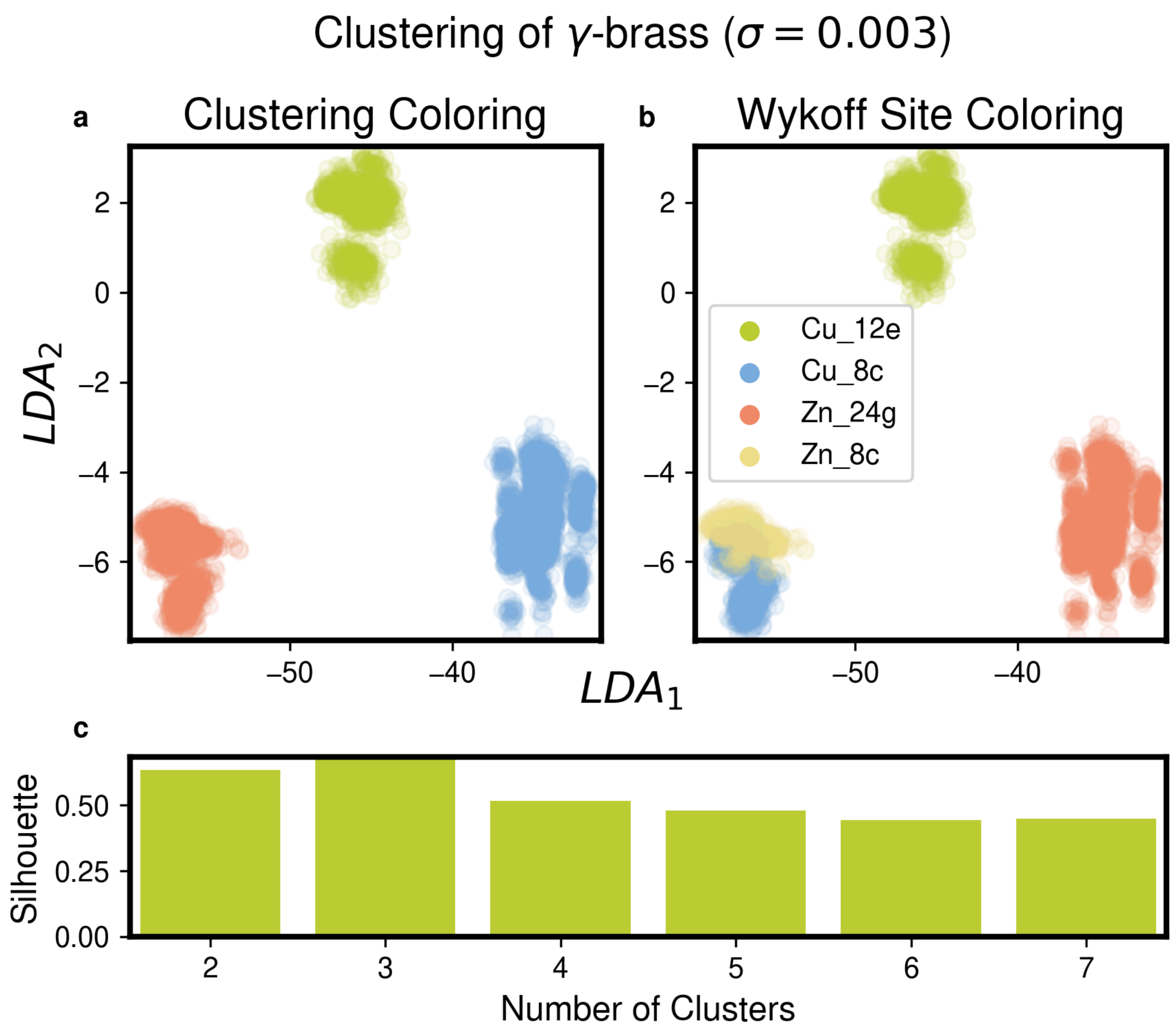

Figure 4.3: (a-b) The particle's PGOP values projected into the first two dimensions of a LDA projection using Wykoff sites as class labels. The PGOP is computed for O_h, T_h, I_h, D_3, D_4, D_5, D_6, D_7 and C_i. (a) Particles are colored by cluster labels assigned by GMM clustering. (b) Particles are colored by the Wykoff site they belong to. (c) The silhouette score given by the scikit-learn [100, 21] is plotted over number of clusters.

quenched. The analyses of the systems is in Figure 4.4.

We find that the PGOP for I_h (full icosahedral symmetry) can clearly detect the onset location of crystallization because the Si environment is perfectly icosahedral. The Cr environment has incomplete icosahedral ordering. Thus, in Figure 4.4 (c-d), we can observe a sharp increase in icosahedral ordering upon crystallization. We also see that in both the quenched and slow-cooled system, crystallization happens very quickly (with respect to the period between analyzed microstates). However, the inset of Figure 4.4 (d) suggests that the transition is similarly “quick” even when viewed at shorter time frames. Furthermore, the quenched system results in a crystal with large swaths of defects, whereas the slow cool has very good assembly with only two grains and two grain boundaries. Interestingly, in both the quenched and slow-cooled cases we can see from the lower quantiles that the icosahedral ordering of the entire system begins at the same time.

4.2.3.3 Formation of Binary Lennard-Jones Glass

We now move to an amorphous phase of matter glass. Specifically, we will use the PGOP to analyze the formation of a binary LJ glass from a linear temperature quench. It is known that binary systems of LJ-interacting particles within certain parameters are good glass formers [61]. We take the protocol from [61] and simulate particles with a size ratio of 0.97, a cohesive energy of -0.8 and a mixing energy of 2.0 dimensionless quantities defined in Equation 4.10. The cohesive energy and mixing energy are defined in terms of the LJ interaction parameters,

$$E_{co} = \frac{(\epsilon_{BB} - \epsilon_{AA})}{(\epsilon_{BB} + \epsilon_{AA})} \tag{4.10}$$

$$E_{mix} = \frac{2\epsilon_{AB}}{(\epsilon_{BB} + \epsilon_{AA})}. \tag{4.11}$$

We use a cooling rate of 1e-6 kT per step from $kT = 5$ to $kT = 0.1$ at a constant pressure of 10 with the box constrained to remain cubic. We verified that the system does transition

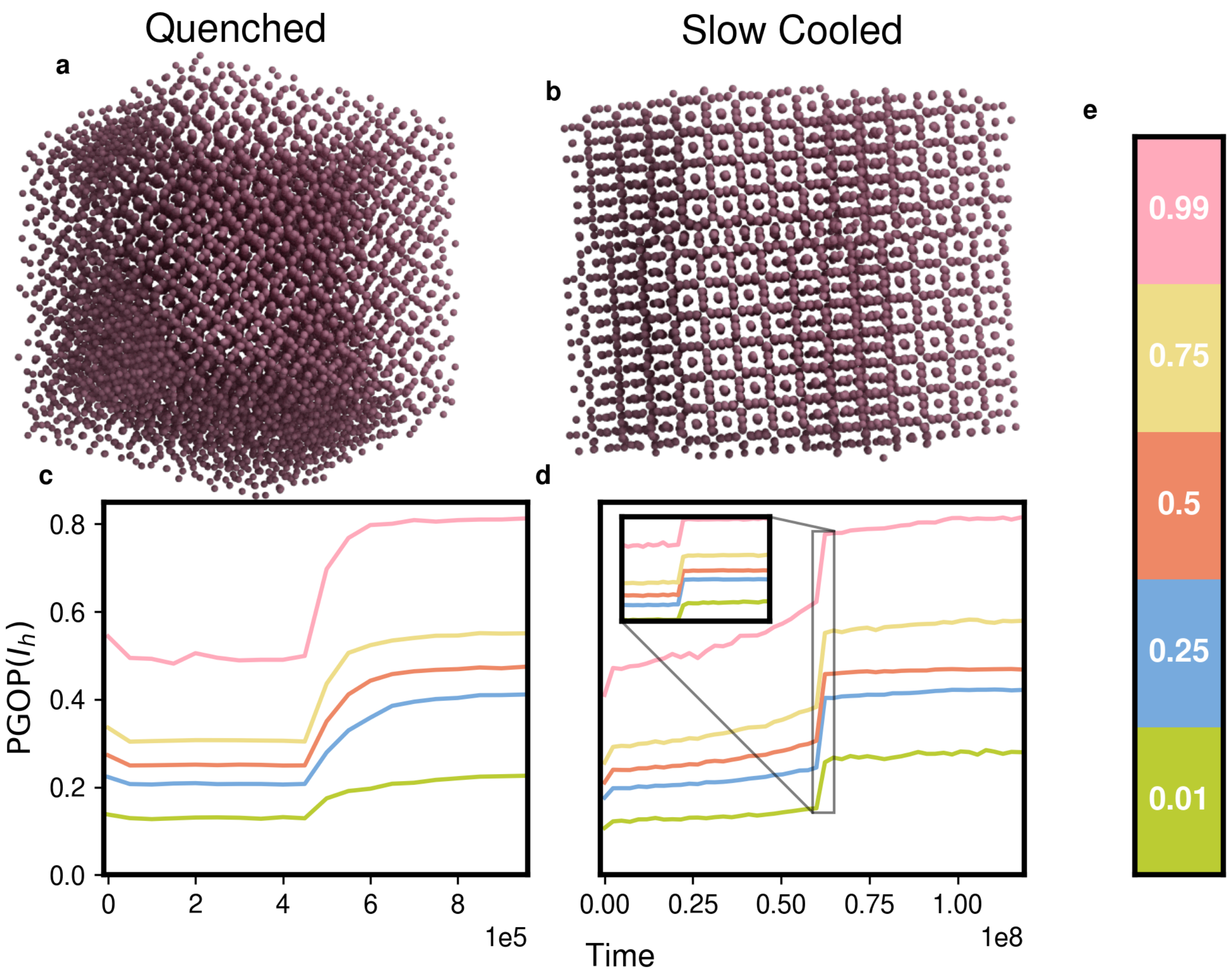

Figure 4.4: (a-b) Visual snapshots of the two systems at the end of the simulation. (a) Visual snapshot of the end of the quenched system. (b) Visual snapshot of the end of the slow-cooled system. (c-d) Plots of various quantiles of full icosahedral ordering across the simulation. (c) Plot of the quenched system. (d) Plot of the slow-cooled system. The inset which zooms in on the transition (black boxed region) and is plotted and colored the same as the main plot. (e) A color legend of which color represents which quantile. The figures clearly shows the utility of the PGOP at analyzing soft matter self-assembly.

into a glass by looking at the mean squared displacement. The results confirming the glassy confinement can be seen in Appendix B.3.

We compute all cyclic and dihedral point groups from order three to twelve (except 11) and I, I_h, O, O_h, T and T_h for all frames in the trajectory. To evaluate system evolution and the final glass we use the PGOP as the input to an agglomerative hierarchical clustering algorithm (AHC), using Ward's linkage, from scikit-learn [100, 21] on the final frame. Ward's linkage defines distance as the increase in intra-cluster variance. We determine the final number of clusters to use by recursively applying the KNEEDLE [113] algorithm for elbow finding on the distance between clusters according to AHC. After each elbow detection, the distance vector is truncated to the last detected elbow and KNEEDLE is rerun until it fails to detect an elbow. Elbow detection resulted in three options for the number of clusters three, ten and fifty-two; we chose to analyze to three clusters. To analyze the development of environments, we train a gradient boosted classifier [86] with 40 trees from scikit-learn on the cluster labels from the cluster algorithm. We achieved an accuracy score of 0.95 on a test-set size of 10% of the data.

In Figure 4.5, we show the results of this analysis. As we can see in (a, d-e), the icosahedral ordering of the system in general and individual particles in particular increases as the system cools. Specifically in (a), we note the clustering of icosahedrally ordered particles into elongated clusters as has been reported in glasses before [79, 118] as the system cools. We also note the development of icosahedral ordering as can be seen in (d). To further analyze this, we look at the size of the largest icosahedrally ordered particle cluster in (e). We determine clusters via `freud`'s [33, 104] clustering algorithm, which finds all connected components with a set minimum edge length ($r = 1.15\sigma$ in this case). From (e) then, we see that the highest icosahedrally ordered environment produces larger and larger clusters over time before peaking and decreasing. The decrease in cluster size may be due a to relaxation processes in the glass.

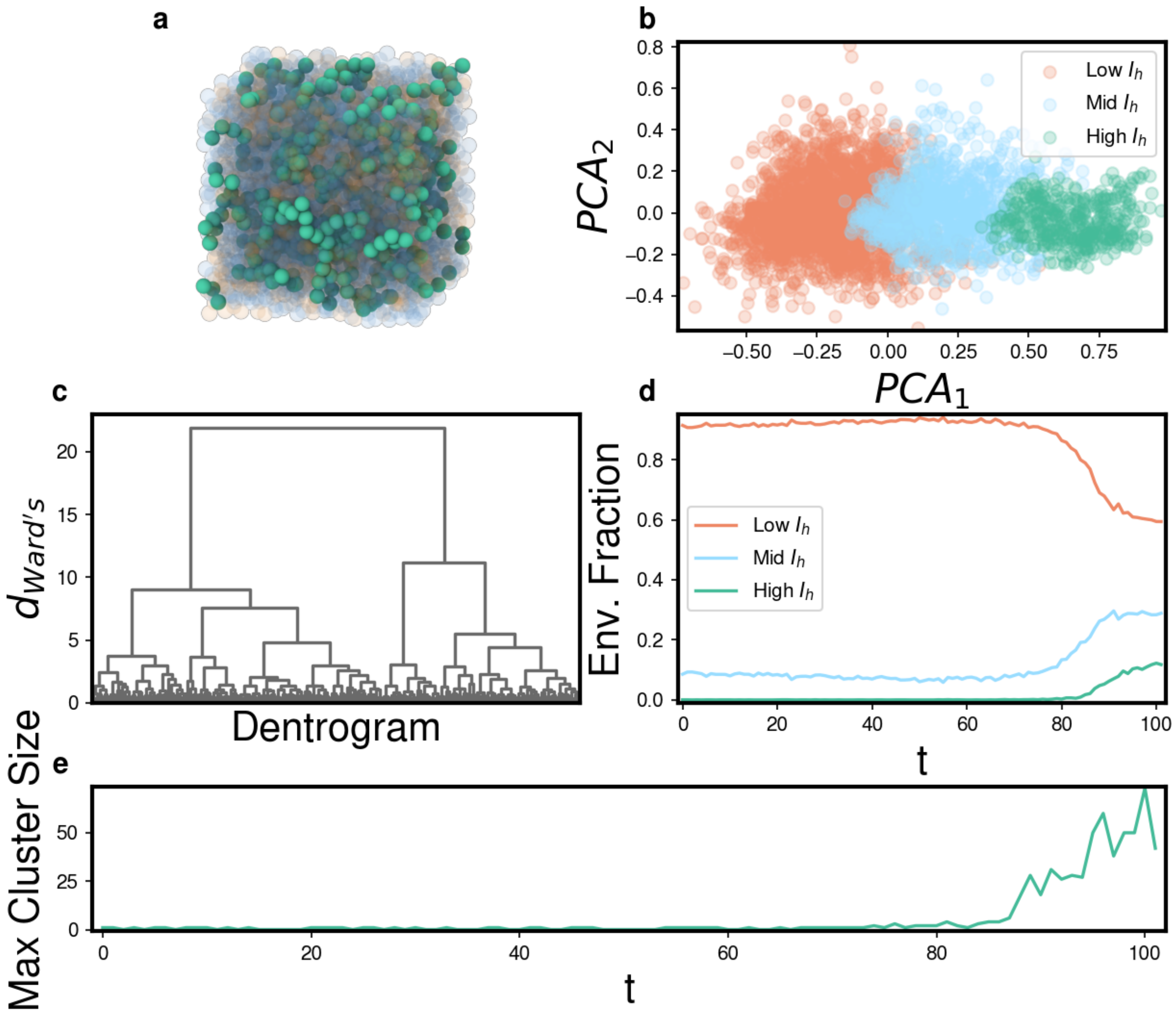

Figure 4.5: (a) Image of the last frame in the simulation with the icosahedrally ordered environment colored dark teal and other particles are translucent blue and orange. (b) Plot of particles in the last frame in the first two PCA dimensions from the PGOP analysis. The clusters corresponds their color in (a). (c) The dendrogram of the AHC with the Ward's linkage distance on the y-axis. (d) Plot of the evolution of glassy environments in the last frame over the simulation. The line colors corresponds to the colors in (a-b). (e) Plot of the size of the largest contiguous cluster of icosahedrally ordered particles.

4.2.3.4 Crystalline Defects

For our final example, we look at the crystallization of LJ-interacting particles into a FCC crystal with HCP stacking faults and other defects. Our purpose in this analysis is to show the effectiveness of the PGOP in detecting deviations in local ordering consistent with defects in a crystal. We ran the system in the NPT ensemble under a pressure of 0.1, a temperature ramp from 5.0 to 0.1 with a cooling rate of 1e-6 kT per step and 3375 particles. The system after cooling was run for an additional 500,000 steps.

As in Subsection 4.2.3.3, we compute all cyclic and dihedral point groups from order three to twelve (except 11) and I, I_h, O, O_h, T and T_h for all frames in the trajectory. We also perform the same clustering method (AHC) resulting in three or twenty clusters as the reasonable options. For our analysis we use three clusters (see Figure 4.6).

We display the results of the analysis in Figure 4.6. We begin by coloring the final frame using the labels assigned by the clustering algorithm (a). We label the clusters FCC, HCP and Other to aid with the comparison (to be made) to polyhedral template analysis (PTA) [76]. "Plates" of the HCP environment can clearly be see in (a) with the bulk consisting of FCC.

To verify the clustering produced meaningful distinctions, we computed the labels of PTA with a RMSD (root mean squared deviation) of 0.15. We also limited the possible environments to FCC, HCP, BCC and Other; although for the analysis, we moved all BCC particles into the Other label. Figure 4.6 (c-d) show the comparison between the automatically chosen clusters and PTA. We chose the optimal mapping from the PGOP determined clusters to the PTA ones and then looked at the confusion matrix between the two labellings (c). This optimal mapping is how we label the clusters in Figure 4.6. We note great agreement between the PGOP clustering's FCC label and the PTA FCC label; however, PTA labels many particles FCC that the PGOP clustering labels as Other. To investigate if this is due to more disorder in these particles, we look at the number of disordered neighbors for particles labeled FCC by PTA that our clustering clustered as the FCC or Other label. We quantity this by counting the number of neighbors labeled Other by PTA. We compute the neighbors using Voronoi tessellation. As hypothesized, the PGOP Other particles have on average many more PTA Other neighbors, suggesting they are more disordered themselves. Thus, we achieve a meaningful partitioning of the system into environments without needing to know the structures to search for unlike PTA, and we could easily determine the nature of the environments by looking at which point group symmetry an environment most possessed. Finally, we emphasize that the clustering was done without PTA in mind or PTA labels and all comparison was done *post hoc* to illustrate the effectiveness of PGOP. We would not expect the clusters to be identical as the two methods are different, and the RMSD hyperparameter was not tuned to best match the AHC clustering.

4.3 Conclusion

In this paper we show that the PGOP can detect the presence and beginnings of crystalline order by analyzing local point group ordering. Through comparisons with MSM, we emphasize the robustness of the PGOP and its ability to distinguish local environments. The PGOP promises increased ability to analyze the nucleation and formation of crystals, defects and even amorphous structures. In Subsection 4.2.3.2, we present the use of the PGOP in an ongoing research project. Such pathways studies are needed to facilitate pathway engineering that enables micro-control of macroscopic systems. However, PGOP and other bond-orientational order parameters remove information on local density. We address this deficient in the next chapter. An optimized CPU software package implementing this method written in C++ and Python is available on GitHub (`https://www.github.com/glotzerlab/pgop`).

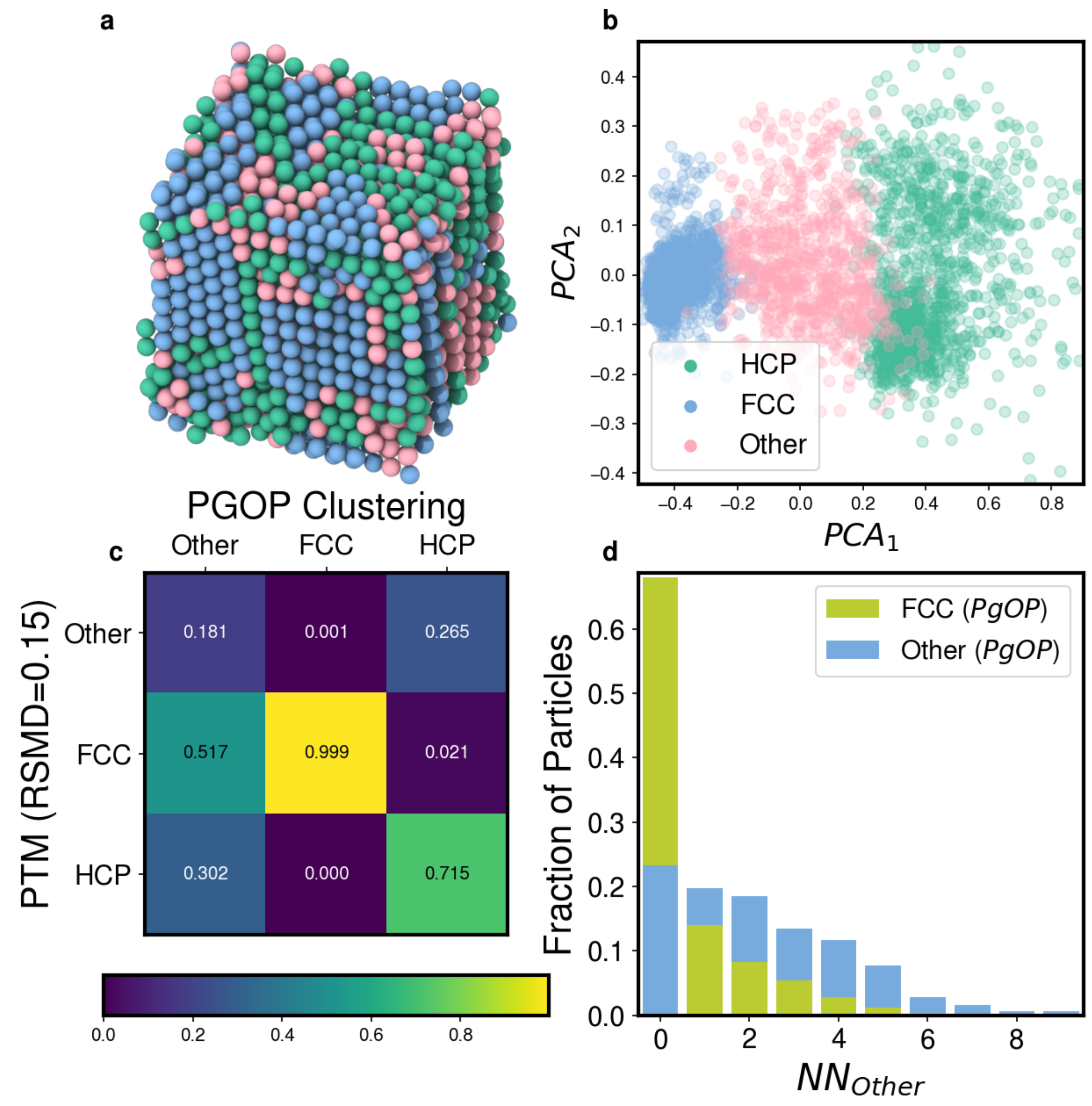

Figure 4.6: (a) Image of the last frame in the simulation colored by the PGOP clustering labels. (b) Plot of particles in the last frame in the first two PCA dimensions from the PGOP analysis. The colors correspond to (a). (c) Confusion matrix of PTA and PGOP AHC labels. Squares are normalized by column (PGOP label). Color bar is shown for the plot underneath. (d) Bar graph of the fraction of particles in the category with N neighbors labeled Other by PTA.

CHAPTER 5

New Continuous Coordination Number

5.1 Introduction

One weakness of PGOP, discussed in the previous chapter, is that distance information is discarded when creating the BOD. Order parameters that capture this information are quantities like local density and coordination number. Local densities have the difficulty in that they are unit sensitive and, thus, require normalization when comparing across disparate systems. However, local coordination numbers are inherently dimensionless quantities, allowing for comparison across systems and phases. The local coordination number is the number of particles a central particle is in contact with or "bonded" to. The difficulty in using coordination number in all but perfect or high density structures, though, is that small perturbations in particle positions can lead to jumps or dips in the coordination number given its discrete nature. The jumps and dips make the relationship between the instantaneous coordination number and a given local environment weaker. This discrete nature makes coordination numbers more difficult to use in machine learning applications as well. The natural solution would be to modify the local coordination number to something more well-behaved (i.e. smoother). One such approach would be to relax the integer coordination number to the reals, $\mathbb{R}$, and allow non-integers such as 8.5. In this chapter, we take one such approach and generalize it to a family of continuous coordination numbers. We begin by deriving the family of coordination numbers. To analyze their effectiveness, we compare our approach to the conventional coordination number from Voronoi tessellation as well as the coordination number algorithm inspiring this work from [24] in classifying particles into simple crystals like FCC. We also highlight PGOP's and $CN_{\vec{V}}$'s improved combined effectiveness in the same classification task.

5.2 Results

5.2.1 Derivation

The new coordination number is a generalization of a coordination number developed in 1978 by Forest L. Carter [24]. The original formula is,

$$CN_2 = \left[\sum^{N} \left(\frac{V_i}{V} \right)^2 \right]^{-1}, \tag{5.1}$$

where V is the volume of a particle's Voronoi polytope and V_i is the volume of the pyramid formed by taking the face between two particles and creating lines to the central particle from the vertices. In systems where the local environment has N equivalent particles $(V_i/V) \to (1/N)$, and Equation 5.1 becomes simply N where N is the number of Voronoi neighbors. We calculate the CN_2 of some simple monoatomic crystals without noise in Table 5.1 and with noise in Table 5.2. We also show the effectiveness of the method compared to the more straight-forward direct counting of neighbors through Voronoi tessellation in Figure 5.1. As Figure 5.1 indicates, CN_2 is much more concentrated (narrow) than CN_0 (counting Voronoi neighbors), as expected.

We make two generalizations of Equation 5.1 to arrive at a family of coordination numbers that can readily delineate the differences in local environments with appreciable noise. The first generalization comes from generalizing the power of the expression (V_i/V) to m. This

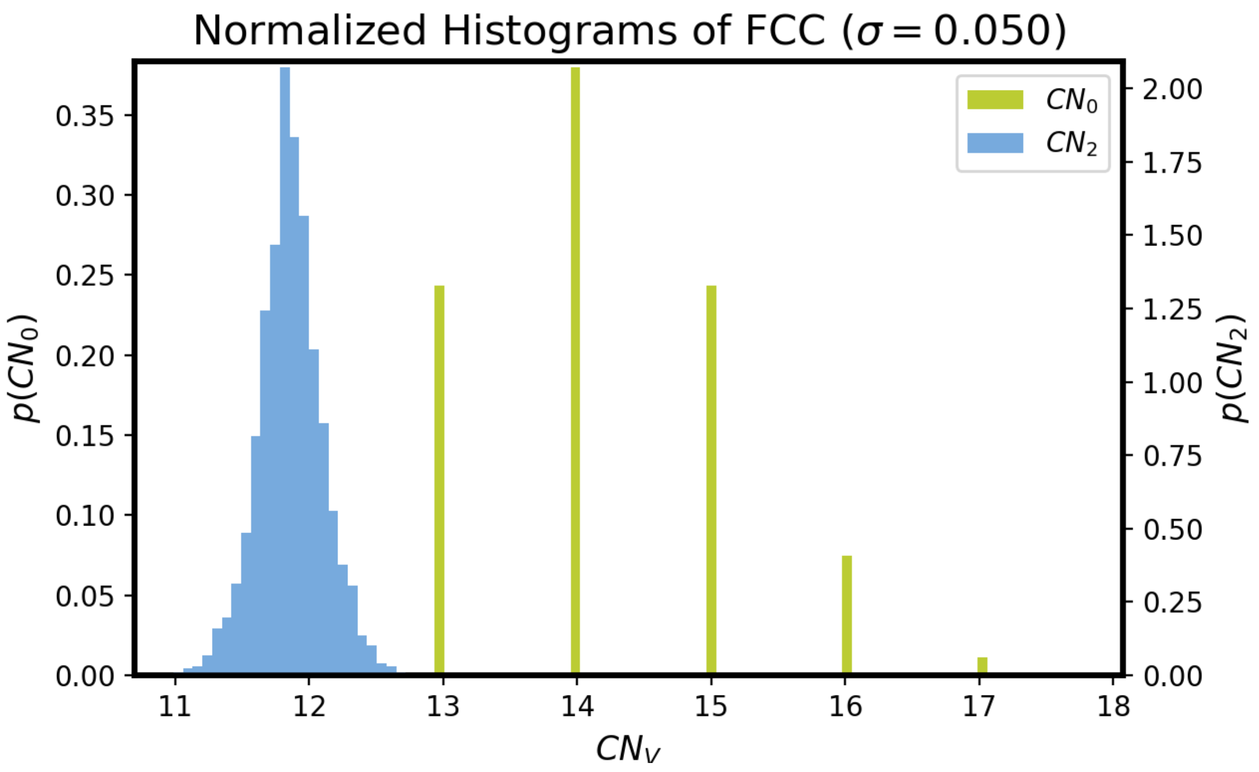

Figure 5.1: Normalized histogram of CN_0 and CN_2. CN_0 is simply counting Voronoi neighbors, while CN_2 is from Equation 5.1. The left y-axis represent CN_0 and the right represents CN_2. The data is from a snapshot of perfect FCC with 4,000 particles perturbed by Gaussian noise with standard deviation of 0.05. We can see that the data is much narrower for CN_2 than for CN_0. The x-axis is shared between the two plots to allow for an easier comparison. The histogram for CN_2 does not span the entire axis.

results in (once accounting for normalization),

$$CN_m = N^{2-m}\left[\sum^{N}\left(\frac{V_i}{V}\right)^m\right]^{-1}. \tag{5.2}$$

Our generalized coordination number is given for various powers of m for perfect crystals in the Table 5.1 and noisy crystals ($\sigma = 0.025$) in Table 5.2. Notice crystals with identical Voronoi neighbors (FCC, SC) maintain identical CN across m for the perfect crystals while for BCC and the ideal gas, IG, go to zero due to non-uniform first neighbor shells. This trend towards zero becomes universal in the presence of thermal noise (Table 5.2).

Table 5.1: Table of the CN_m values for various perfect crystals and m.

System / m	0.0	0.5	1.0	2.0	4.0	8.0	16.0
FCC	12.00	12.00	12.00	12.00	12.00	12.00	12.00
BCC	14.00	14.26	14.00	12.39	8.02	2.78	0.32
SC	6.00	6.00	6.00	6.00	6.00	6.00	6.00
IG	15.46	17.37	15.46	9.36	2.36	0.13	0.00

Table 5.2: Table of the E[CN_m] values for various crystals (and ideal gas) with noise added across m. Each expected value was computed from a system of 4,000 particles with Gaussian noise with 0 mean and 0.025 standard deviation.

System / m	0.0	0.5	1.0	2.0	4.0	8.0	16.0
FCC	14.00	14.91	14.00	11.97	8.64	4.43	1.18
BCC	14.00	14.27	14.00	12.34	7.94	2.69	0.27
SC	15.88	21.15	15.88	6.71	1.13	0.04	0.00
IG	15.46	17.39	15.46	9.35	2.37	0.13	0.00

The second generalization comes in recognizing $(V_i\ V)^m$ as one of many possible functions and generalizing Equation 5.2 to,

$$CN_f = \alpha_{f,g}\ g\left(\sum^{N} f\left(\frac{V_i}{V}\right)\right), \tag{5.3}$$

where f operates on every neighbor, g operates on the sum of f overall neighbors and $\alpha_{f,g}$ handles any normalization. This form by itself is useless, but opens up the possibilities for creating new algorithms that have the expected behavior in the limit of a uniform neighbor shell. One extension is $f(x) = \log(x)$:

$$\begin{aligned} CN_{\log} &= \frac{-1}{\log N} \sum^{N} \log(V_i/V) \\ &= \frac{-1}{\log N} \left(\log \prod^{N} V_i - N \log V \right). \end{aligned} \tag{5.4}$$

Another extension is $f(x) = \exp(x)$:

$$CN_{\exp} = \sum^{N} \exp\left[\left(\frac{V_i}{\sigma V} - \frac{1}{N} \right) \right]. \tag{5.5}$$

These extensions open the possibility for using functions like the Huber loss which is quadratic around 0 but linear past a point determined by the user. The advantage of such an approach would be to assign higher weight to neighbors farther away than $CN_m,\ m \geq 2$ while still going to zero as distance increases.

5.2.2 Analysis

Having looked at the different forms of CN we introduce, we next analyze the information that they provide. We consider a simple classification task where the classifier is given features for a single particle belonging to a noisy FCC, BCC, SC or IG system and returns the predicted phase. We train a linear discriminant analysis (LDA) model using scikit-learn [100, 21]. We chose LDA because its creates linear decision boundaries, preventing overfitting. For features, we first pass three sets of CN as seen in Figure 5.2. We note that $CN_{\vec{V}}$ refers to a vector of coordination numbers given by the two generalizations in Equations 5.2 and 5.3. Here the data is 8,000 particles with 2,000 for each label with $\sigma = 0.05$ Gaussian noise added to the crystals. LDA is trained on 6,400 points and tested on 1,600 (400 from each label). As we can see in Figure 5.2 (a-c), the addition of the higher order m and log and exp in $CN_{\vec{V}}$ leads to a much better fit than simply CN_0 or CN_2.

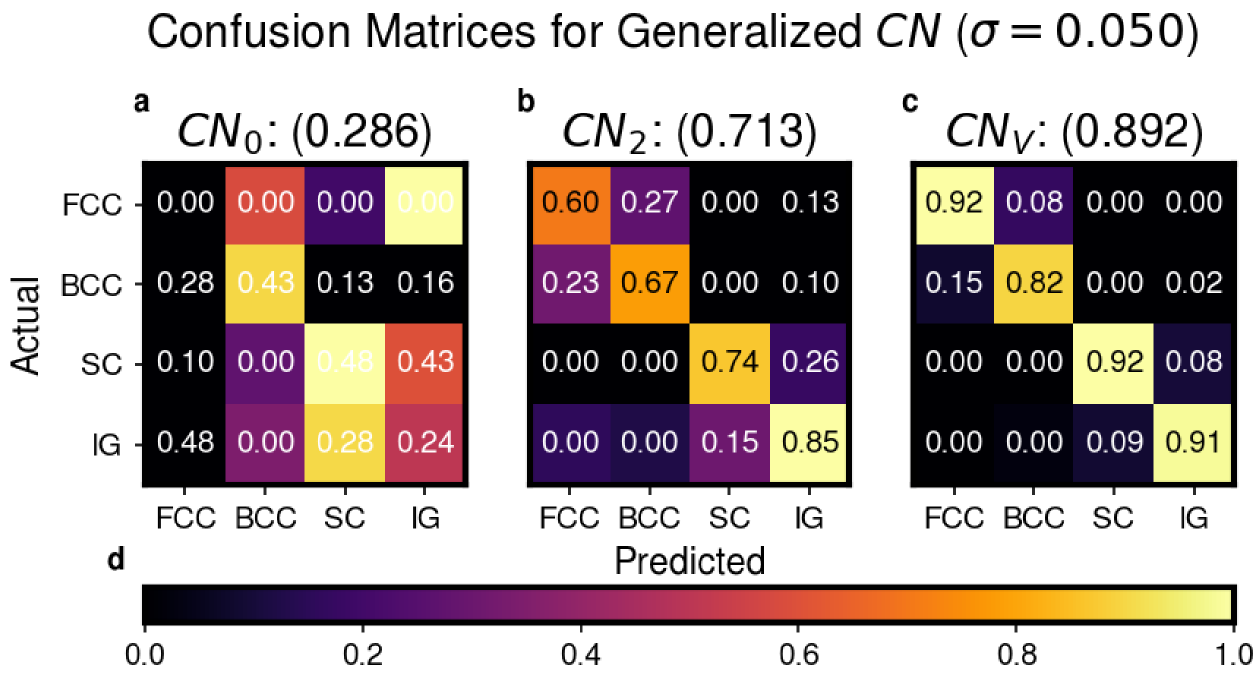

Figure 5.2: (a-c) Confusion matrices for three different definitions of CN. The y-axis represents the actual label of a sample while the x-axis is the predicted label by the model. For example, the value on the FCC row and SC column represents the fraction of FCC test samples predicted to be SC. Values in the confusion matrix are labeled and colored by their occupancy. For example, a perfect classification would be a yellowish-white diagonal and black off-diagonals. (a) Confusion matrix for CN_0. (b) Confusion matrix for CN_2. (c) Confusion matrix for $CN_{\vec{V}}$ for $\vec{V} = [2.0, 4.0, 6.0, 8.0, 12.0, 16.0, \log(), \exp_\sigma(100.0)]$. (d) Color bar for (a-c).

Our next comparison looks at $CN_{\vec{V}}$ and PGOP (see Section 5.1) combined. The dataset here has 4,000 particles with 1,000 of each class at a higher noise level (standard deviation of 0.1) than the previous analysis. Training was on 3,200 points and testing on 800. We trained a random forest classifier [57] on $CN_{\vec{V}}$ and/or PGOP (see Figure 5.3 (a,b)) because as here we are not concerned with overfitting since we have fewer data. Furthermore, we limit leaf nodes in the tree to at least 4 points and only use 100 trees. We used Voronoi tessellation for the PGOP's neighbor list. Figure 5.3 (c) shows that the combination of $CN_{\vec{V}}$ with PGOP significantly improves the classification accuracy of the random forest model compared to each feature independently. This improvement in performance signifies orthogonal information present in the two OPs.

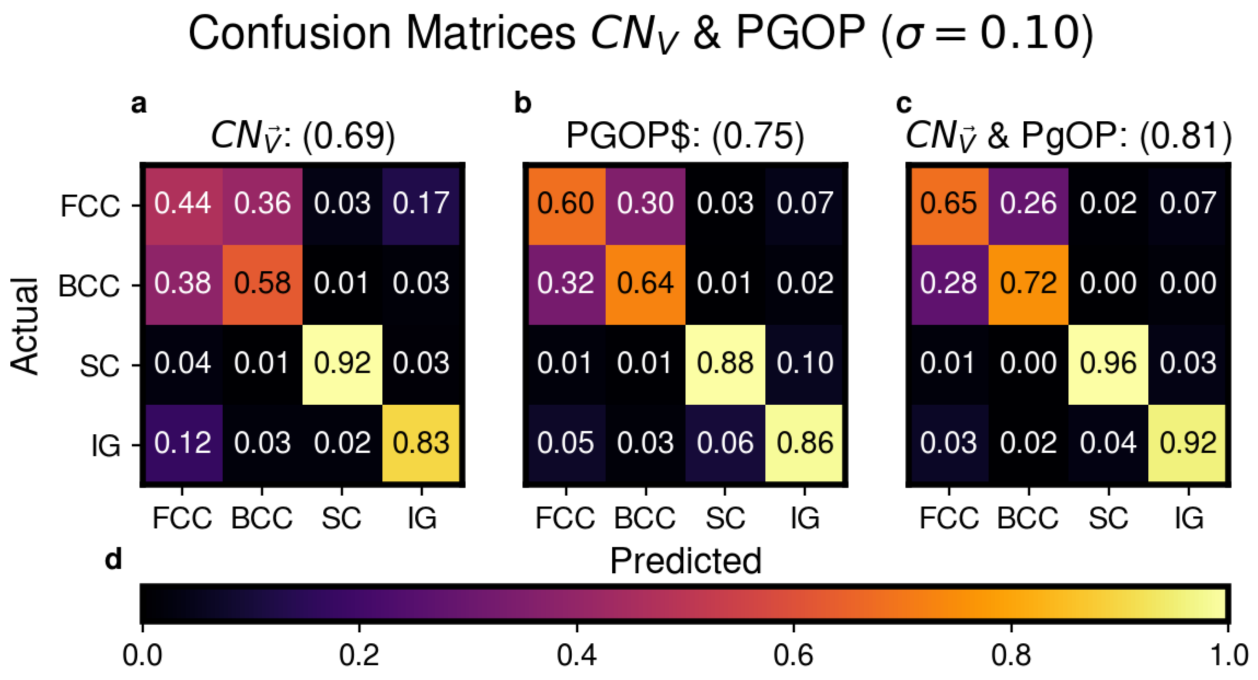

Figure 5.3: (a-c) Confusion matrices for $CN_{\vec{V}}$ and/or PGOP. The y-axis represents the actual label of a sample while the x-axis is the predicted label by the model. For example, the value on the FCC row and SC column represents the fraction of FCC test samples predicted to be SC. Values in the confusion matrix are labeled and colored by their occupancy. For example, a perfect classification would be a yellowish-white diagonal and black off-diagonals. (a) Confusion matrix for $CN_{\vec{V}}$ for $V = [2.0, 4.0, 6.0, 8.0, 12.0, 16.0, \log(), \exp_\sigma(100.0)]$. (b) Confusion matrix for PGOP for point groups $[I_h, O_h, T_h, T, D_5, D_6, D_7, D_8, D_{10}, C_i]$. (c) Confusion matrix for classifier trained on $CN_{\vec{V}}$ and PGOP from (a-b). (d) Color bar for (a-c).

5.3 Conclusion

We have shown our generalized coordination number (specifically $CN_{\vec{V}}$) to be a viable addition to PGOP in analyzing local order in a generic way while maintaining interpretability. Furthermore, the $CN_{\vec{V}}$ vector is significantly more effective at classifying particles into crystal structures than CN_0 or CN_2 by themselves. Finally, we show that $CN_{\vec{V}}$ presents distinct particle environment information than PGOP in Figure 5.3. By their design, PGOP captures information on the arrangement of neighbors, and $CN_{\vec{V}}$ the effective number of neighbors.

Thus, combined they are able interpretability capture bond-orientational (PGOP) and local density information in a dimensionless manner. The combination will serve as useful tools to use in automated but in-depth studies of self-assembly.

This feature will be added to the open-source analysis package `freud` [33, 104] to allow for use in the field. With PGOP and `freud` installed, both analyses can be computed on a given system.

CHAPTER 6

HOOMD-blue Version 3.0 A Modern, Extensible, Flexible, Object-Oriented API for Molecular Simulations

This chapter is adapted from Ref: "HOOMD-blue version 3.0 A Modern, Extensible, Flexible, Object-Oriented API for Molecular Simulations", Brandon L. Butler, Vyas Ramasubramani, Joshua A. Anderson, Sharon C. Glotzer, Proceedings of the 19th Python in Science Conference (2020) [22]

6.1 Introduction

Molecular simulation has been an important technique for studying the equilibrium properties of molecular systems since the 1950s. The two most common methods for this purpose are molecular dynamics and Monte Carlo simulations [90, 6]. Molecular dynamics (MD) is the application of Newton's laws of motion to molecular system, while Monte Carlo (MC) methods employ a Markov chain to sample from equilibrium configurations. Since their inception these tools have been used to study numerous systems, examples include colloids [29], metallic glasses [41] and proteins [35], among others.

Today many software packages exist for these purposes. LAMMPS [102], GROMACS [18, 3], OpenMM [37], ESPResSo [135, 52] and Amber [111, 25] are a few examples of popular MD packages, while Cassandra [115] and MCCCS Towhee [85] provide MC simulation capabilities. Implementations on high performance GPUs [121], parallel architectures [97] and the greater accessibility of computational power have tremendously improved the length [23] and time [116] scales of simulations from those conducted in the mid 1900s. The flexibility and generality of such tools has dramatically increased the usage of molecular simulations, which has in turn led to demands for even more customizable software packages that can be tailored to very specific simulation requirements. Different tools have taken different approaches to enabling this, such as the text-file scripting in LAMMPS, the command line interface provided by GROMACS and the Python, C++, C and Fortran bindings of OpenMM. Recently, programs that have used other interfaces have also added Python bindings such as LAMMPS and GROMACS.

In the development of these tools, the requirements for the software to enable good science became more obvious. Having computational research that is Transferable, Reproducible, Usable (by others) and Extensible (TRUE) [128] is necessary for fully realizing the potential of computational molecular science. HOOMD-blue is part of the MoSDeF project which seeks to bring these traits to the wider computational molecular science community through packages like `mbuild` [73] and `foyer` [74] which are Python packages that generalize generating initial particle configurations and force fields respectively across a variety of simulation back ends [28, 128]. This effort in increased TRUEness is one of many motivating factors for HOOMD-blue version 3.0.

HOOMD-blue [12, 48, 11], an MD and MC simulations engine with a C++ back end, provides to use a Python API facilitated through pybind11. The package is open-source under the 3-clause BSD license, and the code is hosted on GitHub (https://github.com/glotzerlab/hoomd-blue). HOOMD-blue was initially released in 2008 as the first fully GPU-enabled MD simulation engine using NVIDIA GPUs through CUDA. Since its initial release, HOOMD-blue has remained under active development, adding numerous features over the years that have increased its range of applicability, including adding support for domain decomposition (dividing the simulation box among MPI ranks) in 2014 and recent developments that enable support for AMD in addition to NVIDIA GPUs.

Despite its great flexibility, the package's API still has certain key limitations. In particular, since its inception HOOMD-blue has been designed around some maintenance of global state. The original releases of HOOMD-blue provided Python scripting capabilities based on an imperative programming model, but it required that these scripts be run through HOOMD-blue's modified interpreter that was responsible for managing this global state. Version 2.0 relaxed this restriction, allowing the use of HOOMD-blue within ordinary Python scripts and introducing the `SimulationContext` object to encapsulate the global state to some degree, thereby allowing multiple largely independent simulations to coexist in a single script. However, this object remained largely opaque to the user, in many ways still behaving like a pseudo-global state, and version 2.0 otherwise made minimal modifications to the HOOMD-blue Python API, which was largely inspired by and reminiscent of the structure of other simulation software, particularly LAMMPS.

In this paper, we describe the upcoming 3.0 release of HOOMD-blue, which is a complete redesign of the API from the ground up to present a more transparent and Pythonic interface for users. Version 3.0 aspires to match the intuitive APIs provided by other Python packages like SciPy [132], NumPy [131], scikit-learn [100] and `matplotlib` [63], while simultaneously adding seamless interfaces by which such packages may be integrated into simulation scripts using HOOMD-blue. Global state has been completely removed, instead replaced by a highly object-oriented model that gives users explicit and complete control over all aspects of simulation configuration. Where possible, the new version also provides performant, Pythonic interfaces to data stored by the C++ back end. Over the next few sections, we will use examples of HOOMD-blue's version 3.0 API (which is still in development at the time of writing) to highlight the improved extensibility, flexibility and ease of use of the new HOOMD-blue API.

6.2 General API Design

Rather than beginning with abstract descriptions, we will introduce the new API by example. The script below illustrates a standard MD simulation of a Lennard-Jones fluid using the version 3.0 API. Each of the elements of this script is introduced throughout the rest of this section. We also show a rendering of the particle configuration in Figure 6.1.

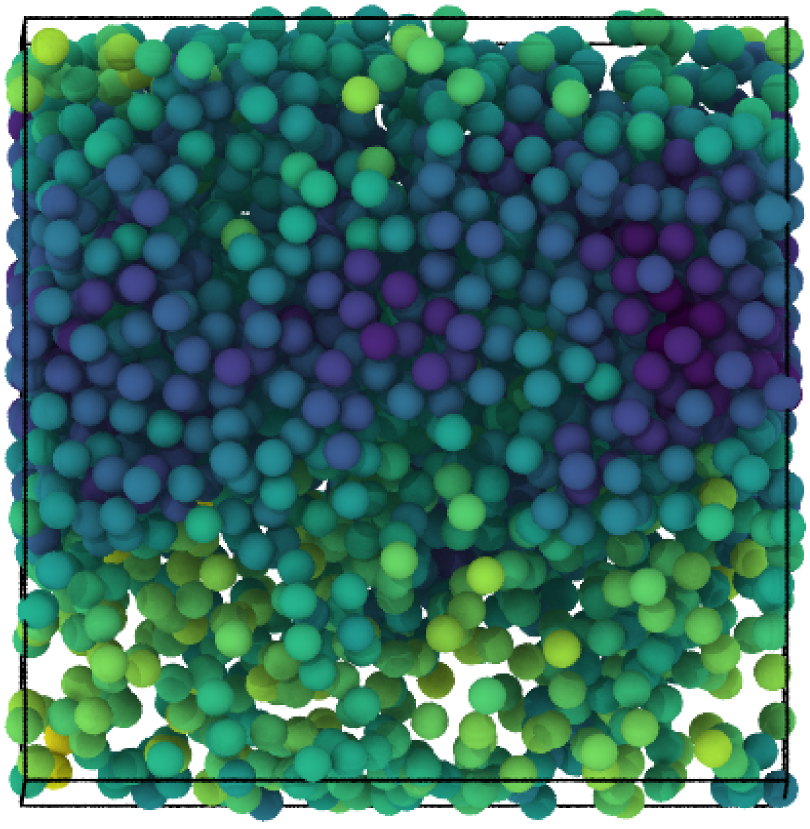

Figure 6.1: A rendering of the Lennard-Jones fluid simulation script output. Particles are colored by the Lennard-Jones potential energy that is logged using the HOOMD-blue `Logger` and `GSD` class objects. Figure is rendered in OVITO [124] using the Tachyon [123] renderer.

```python
import hoomd
import hoomd.md
import numpy as np

device = hoomd.device.Auto()
sim = hoomd.Simulation(device)

# Place particles on simple cubic lattice.
N_per_side = 14
N = N_per_side ** 3
L = 20
xs = np.linspace(0, 0.9, N_per_side)
x, y, z = np.meshgrid(xs, xs, xs)
coords = np.array((x.ravel(), y.ravel(), z.ravel())).T

# One way to define an initial system state is
# by defining a snapshot and using it to
# initialize the system state.
snap = hoomd.Snapshot()
snap.particles.N = N
snap.configuration.box = hoomd.Box.cube(L)
snap.particles.position[:] = (coords - 0.5) * L
snap.particles.types = ['A']

sim.create_state_from_snapshot(snap)

# Create integrator and forces
integrator = hoomd.md.Integrator(dt=0.005)
langevin = hoomd.md.methods.Langevin(hoomd.filter.All(), kT=1., seed
=42)
integrator.methods.append(langevin)

nlist = hoomd.md.nlist.Cell()
lj = hoomd.md.pair.LJ(nlist, r_cut=2.5)
lj.params[('A', 'A')] = {"sigma": 1.0, "epsilon": 1.0}
integrator.forces.append(lj)

# Set up output
gsd = hoomd.output.GSD('trajectory.gsd', trigger=100)
log = hoomd.logging.Logger()
log += lj
gsd.log = log

sim.operations.integrator = integrator
sim.operations.analyzers.append(gsd)
sim.run(100000)

```

Program 6.1: Full script from initialization to run of a Lennard-Jones particle MD simulation.

6.2.1 Simulation, Device, State, Operations

Each simulation in HOOMD-blue is now controlled through three main objects which are joined together by the `Simulation` class: the `Device`, `State` and `Operations` classes. Figure 6.2 shows this relationship with some core attributes/methods for each class. Each `Simulation` object holds the requisite information to run a full molecular dynamics or Monte Carlo simulation, thereby circumventing any need for global state information. The `Device` class denotes whether a simulation should be run on CPUs or GPUs and the number of cores/GPUs it should run on. In addition, the device manages custom memory tracebacks, profiler configurations and the MPI communicator among other things.

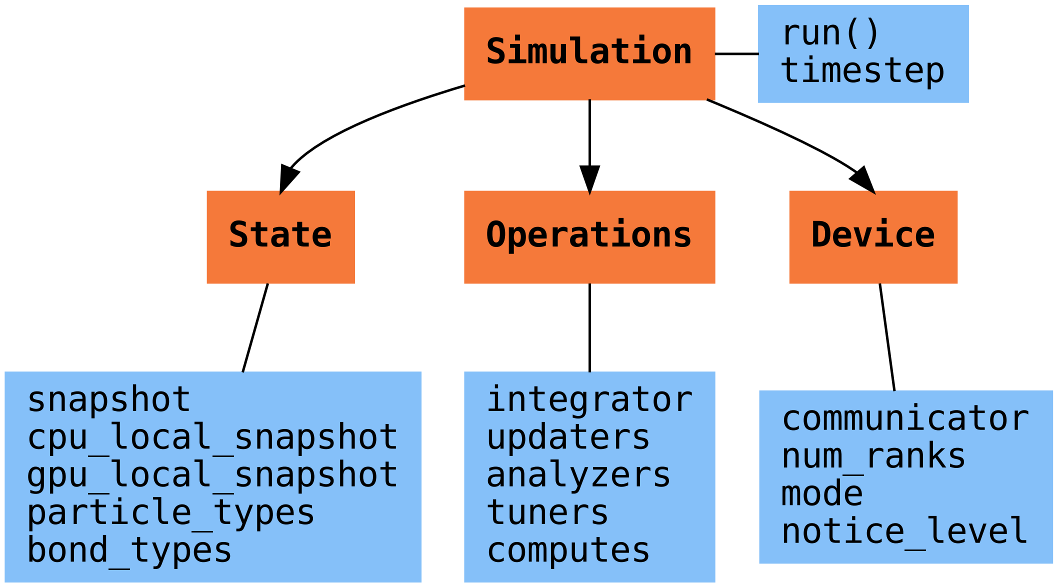

Figure 6.2: Diagram of core objects with some attributes and methods. Classes are in bold and orange; attributes and methods are blue. Figure is made using Graphviz [38, 47].

The `State` class stores the system data (e.g. particle positions, orientations, velocities, the system box). As shown in our example, the state can be initialized from a snapshot, after which the data can be accessed and modified in two ways. One option is for users to

```python
        snap = sim.state.snapshot
        # snapshot only stores data on rank 0
        if snap.exists:
            # set all z positions to 0
            snap.particles.position[:, 2] = 0
        sim.state.snapshot = snap

```

Program 6.2: Example of the global snapshot in use.

operate on a new `Snapshot` object, which exposes NumPy arrays that store a copy of the system data. To construct a snapshot, all system data distributed across MPI ranks must be gathered and combined by the root rank. Setting the state using the snapshot API requires assigning a modified snapshot to the system state (i.e. all system data is reset upon setting). The advantages to this approach come from the ease of use of working with a single object containing the complete description of the state. The following snippet showcases how this approach can be used to set the z position of all particles to zero.

The other API for accessing `State` data is via a zero-copy, rank-local access to the state's data on either the GPU or CPU. On the CPU, we expose the buffers as `numpy.ndarray`-like objects through provided hooks such as `__array_ufunc__` and `__array_interface__`. Similarly, on the GPU we mock much of the CuPy [1] `ndarray` class if it is installed; however, at present the CuPy package provides fewer hooks, so our integration is more limited. Whether or not CuPy is installed, we use version 2 of the `__cuda_array_interface__` protocol for GPU access (compatibility with our GPU buffers in Python therefore depends on the support of version 2 of this protocol). This provides support for libraries such as Numba's [75] GPU just-in-time compiler and PyTorch [99]. We chose to mock NumPy-like interfaces rather than expose `ndarray` objects directly out of consideration for memory safety. To ensure data integrity, we restrict the data to only be accessible within a specific context manager. This approach is much faster than using the snapshot API because it uses HOOMD-blue's data buffers directly, but the nature of providing zero-copy access requires that users deal directly with the domain decomposition since only data for a MPI rank's local simulation box is stored by

```python
        with sim.state.cpu_local_snapshot as data:
            data.particles.position[:, 2] = 0

        # assumes CuPy is installed
        with sim.state.gpu_local_snapshot as data:
            data.particles.position[:, 2] = 0

```

Program 6.3: Example of local snapshots in use.

a given rank. The example below modifies the previous example to instead use the zero-copy API.

The last of the three classes, `Operations`, holds the different *operations* that will act on the simulation state. Broadly, these consist of 3 categories: updaters, which modify simulation state; analyzers, which observe system state; and tuners, which tune the hyper-parameters of other operations for performance. Although updaters and analyzers existed in version 2.x (tuners are a version 3.0 split from updaters), these *operations* have undergone a significant API overhaul for version 3.0 to support one of the more far-reaching changes to HOOMD-blue: the deferred initialization model.

Operations in HOOMD-blue are generally implemented as two classes, a user-facing Python object and an internal C++ object which we denote as the *action* of the operation. On creation, these C++ objects typically require a `Device` and a C++ `State` in order to, for instance, initialize appropriately sized arrays. Unfortunately this requirement restricts the order in which objects may be created since devices and states must exist first. This restriction could create potential confusion for users who forget this ordering and would also limit the composability of modular simulation components by preventing, for instance, the creation of a simple force field without the prior existence of a `Device` and a `State`. To circumvent these difficulties, the new API has moved to a deferred initialization model in which C++ objects are not created until the corresponding Python objects are *attached* to a `Simulation`, a model we discuss in greater detail below.

6.2.2 Deferred C++ Initialization

The core logic for the deferred initialization model is implemented in the `Operation` class, which is the base class for all operations in Python. This class contains the machinery for attaching/detaching operations to/from their C++ counterparts, and it defines the user interface for setting and modifying operation-specific parameters while guaranteeing that such parameters are synchronized with attached C++ objects as appropriate. Rather than handling these concerns directly, the `Operation` class manages parameters using specially defined classes that handle the synchronization of attributes between Python and C++: the `ParameterDict` and `TypeParameterDict` classes. In addition to providing transparent `dict`-like APIs for the automatically synchronized setting of parameters, these classes also provide strict validation of input types, ensuring that user inputs are validated regardless of whether or not operations are attached to a simulation.

Each class supports validation of their keys, and they can be used to define the structure and validation of arbitrarily nested dictionaries, lists and tuples. Likewise, both support default values, but to a varying degree due to their differing purposes. `ParameterDict` acts as a dictionary with additional validation logic. However, the `TypeParameterDict` represents a dictionary in which each entry is validated by the entire defined schema. This distinction occurs often in simulation contexts as simulations with multiple types of particles, bonds, angles, etc. must specify certain parameters for each type. In practice this distinction means that the `TypeParameterDict` class supports default specification with arbitrary nesting, while the `ParameterDict` has defaults but these are equivalent to object attribute defaults. An example `TypeParameterDict` initialization and use of both classes can be seen below.

The specification defined above sets defaults for `ignore_statistics` and `orientable` (the purpose of these is outside the scope of the paper), but requires the setting of the `diameter` for each type.

To store lists of operations that must be attached to a simulation, the analogous `SyncedList` class transparently handles attaching of operations.

```python
        # Specification of Sphere's shape TypeParameterDict
        TypeParameterDict(
            diameter=float, ignore_statistics=False,
            orientable=False, len_keys=1)

        from hoomd.hpmc.integrate import Sphere

        sphere = Sphere(seed=42)
        # Set nselect parameter using ParameterDict
        sphere.nselect = 2
        # Set shape for type 'A' using TypeParameterDict
        sphere.shape['A'] = {'diameter': 1.0}
        # Set shape for types 'B', 'C' and 'D'
        sphere.shape[['B', 'C', 'D']] = {'diameter': 0.5}
```

Program 6.4: Examples of creating and using a type parameter for hard particle Monte Carlo with spheres.

```python
        import hoomd

        ops = hoomd.Operations()
        gsd = hoomd.output.GSD('example.gsd')
        # Append to the SyncedList ops.writers
        ops.writers.append(gsd)
```

Program 6.5: Example of using a `SyncedList` object.

These classes also have the ancillary benefit of improving error messaging and handling. An example error message for trying to set `sigma` for *A-A* interactions in the Lennard-Jones pair potential to a string (i.e. `lj.params[('A', 'A')] = {'sigma': 'foo', 'epsilon': 1.}` would provide the error message,

> TypeConversionError: For types [('A', 'A')], error In key sigma: Value foo of type <class 'str'> cannot be converted using OnlyType(float). Raised error: value foo not convertible into type <class 'float'>.

Previously, the equivalent error would be "TypeError: must be real number, not str", the error would not be raised until running the simulation, and the line setting sigma would not be in the stack trace given.

6.3 Logging and Accessing Data

Logging simulation data for analysis is a critical feature of molecular simulation software packages. Up to now, HOOMD-blue has supported logging through an analyzer interface that simply accepted a list of quantities to log, where the set of valid quantities was based on what objects had been created at any point and stored to the global state. The creation of the base `Operation` class has allowed us to simultaneously simplify and increase the flexibility of our logging infrastructure. The `Loggable` metaclass of `Operation` allows all subclasses to expose their loggable quantities by marking Python properties or methods to query.

The actual task of logging data is accomplished by the `Logger` class, which provides an interface for logging most HOOMD-blue objects and custom user quantities. In the example script from the General API Design section above, we show that the `Logger` can add an operation's loggable quantities using the `+=` operator. The utility of this class lies in its intermediate representation of the data. Using the HOOMD-blue namespace as the basis for distinguishing between quantities, the `Logger` maps logged quantities into a nested dictionary. For example, logging the Lennard-Jones pair potentials total energy would produce this dictionary by a `Logger` object `{'md': {'pair': {'LJ': {'energy': (-1.4, 'scalar')}}}}` where `'scalar'` is a flag to make processing the logged output easier. In real use cases, the dictionary would likely be filled with many other quantities.

Version 3.0 of HOOMD-blue uses properties extensively to expose object data such as the total potential energy of all pair potentials, the trial move acceptance rate in MC integrators and thermodynamic variables like temperature or pressure, all of which can be used directly or stored through the logging interface. To support storing these properties, the logging is quite general and supports scalars, strings, arrays and even generic Python objects. By separating the data collection from the writing to files and by providing such a flexible intermediate representation, HOOMD-blue can now support a range of back ends for logging; moreover, it offers users the flexibility to define their own. For instance, while logging data to text files or standard out is supported out of the box, other back ends like MongoDB, Pandas [89] and Python pickles can now be implemented on top of the existing logging infrastructure. Consistent with the new approach to logging, HOOMD-blue version 3.0 makes simulation output an opt-in feature even for common outputs like performance and thermodynamic quantities. In addition to this improved flexibility in storage possibilities, for HOOMD-blue version 3.0 we have exposed more of an object's data than had previously been available through adding new properties to objects. For example, pair potentials now expose *per-particle* potential energies at any given time (this data is used to color Figure 6.1.

In conjunction with the deferred initialization model, the new logging infrastructure also allows us to more easily export an object's state (not to be confused with the simulation state). Due to the switch to deferred initialization, all operation state information is now stored directly in Python, so we have made object state a loggable quantity. All operations also provide a `from_state` factory method that can reconstruct the object from the state, dramatically increasing the restartability of simulations since the state of each object can be saved at the end of a given run and read at the start of the next.

This code block would create a `Sphere` object with the parameters stored from the last

```python
from hoomd.hpmc.integrate import Sphere

sphere = Sphere.from_state('example.gsd', frame=-1)

```

Program 6.6: Code to initialize simulation state from an extant GSD file.

frame of the gsd file `example.gsd`.

6.4 User Customization

A major improvement in HOOMD-blue version 3 is the ease with which users can customize their simulations in previously impossible ways. The changes that enable this improvement generally come in two flavors, the generalization of existing concepts in HOOMD-blue and the introduction of a completely new `Action` class that enables the user to inject arbitrary actions into the simulation loop.In this section, we first discuss how concepts like periods and groups have been generalized from previous iterations of HOOMD-blue and then show how users can inject completely novel routines to actually modify the behavior of simulations.

6.4.1 Triggers

In HOOMD-blue version 2.x, everything that was not run on every timestep had a period and phase associated with it. The timesteps the operation was run on could then be determined by the expression, `timestep \% period - phase == 0`. In our refactoring and development, we recognized that this concept could be made much more general and consequently more flexible. Objects do not have to be run on a periodic timescale; they just need some indication of when to run. In other words, the operations needed to be *triggered*. The `Trigger` class encapsulates this concept, providing a uniform way of specifying when an object should run without limiting options. `Trigger` objects return a Boolean value when called with a timestep (i.e. they are functors). Each operation that requires triggering is now associated with a corresponding `Trigger` instance which informs the simulation when the operation

```python
from hoomd.trigger import Trigger

class CustomTrigger(Trigger):
    def __init__(self, period, phase=0):
        super().__init__()
        self.period = period
        self.phase = phase

    def __call__(self, timestep):
        return timestep % self.period - self.phase == 0

```

Program 6.7: Example of a custom trigger that recreates the `hoomd.trigger.Periodic` class's behavior.

should run. The previous behavior is now available through the `Periodic` class in the `hoomd .trigger` module. However, this approach enables much more sophisticated logic through composition of multiple triggers such as `Before` and `After` which return `True` before or after a given timestep with the `And`, `Or` and `Not` subclasses that function as logical operators on the return value of the composed `Triggers`.

In addition to the flexibility the `Trigger` class provides by abstracting out the concept of triggering an operation, we use pybind11 to easily allow subclasses of the `Trigger` class in Python. This allows users to create their own triggers in pure Python that will execute in HOOMD-blue's C++ back end. An example of such a subclass that reimplements the functionality of HOOMD-blue version 2.x can be seen below.

User-defined subclasses of `Trigger` are not restricted to simple algorithms or even stateless ones; they can implement arbitrarily complex Python code as demonstrated in the Large Examples section's first code snippet.

6.4.2 Variants

`Variant` objects are used in HOOMD-blue to specify quantities like temperature, pressure and box size which can vary over time. Similar to `Trigger`, we generalized our ability to linearly interpolate values across timesteps (`hoomd.variant.linear_interp` in HOOMD-blue

```python
from math import sin
from hoomd.variant import Variant

class SinVariant(Variant):
    def __init__(self, frequency, amplitude, phase=0, center=0):
        super().__init__()
        self.frequency = frequency
        self.amplitude = amplitude
        self.phase = phase
        self.center = center

    def __call__(self, timestep):
        tmp = sin(self.frequency * timestep + self.phase)
        return self.amplitude * tmp + self.center

    def _min(self):
        return self.center - self.amplitude

    def _max(self):
        return self.center + self.amplitude

```

Program 6.8: Example of custom variant which oscillates according to a sin wave.

version 2.x) to a base class `Variant` which generalizes the concept of functions in the semi-infinite domain of timesteps $t \in \mathbb{Z}_0^+$. This allows sinusoidal cycling, non-uniform ramps and other behaviors. Like `Trigger`, `Variant` can be a direct subclass of the C++ class. An example of a sinusoidal cycling variant is shown below.

6.4.3 ParticleFilters

Unlike `Trigger` or `Variant`, `ParticleFilter` is not a generalization of an existing concept but the splitting of one class into two. However, this split is also targeted at increasing flexibility and extensibility. In HOOMD-blue version 2.x, the `ParticleGroup` class and subclasses served to provide a subset of particles within a simulation for file output, application of thermodynamic integrators and other purposes. The class hosted both the logic for storing the subset of particles and filtering them out from the system. After the refactoring, `ParticleGroup` is only responsible for the logic to store and perform some basic operations on a set of particle tags (a means of identifying individual particles), while the new class `ParticleFilter` implements the

```python
from hoomd.filter import CustomParticleFilter

class PositiveCharge(CustomParticleFilter):
    def __init__(self, state):
        super().__init__(state)

    def __hash__(self):
        return hash(self.__class__.__name__)

    def __eq__(self, other):
        return type(self) == type(other)

    def find_tags(self, state):
        with state.cpu_local_snapshot as data:
            return data.particles.tag[data.particles.charge > 0]

```

Program 6.9: Example of a custom particle filter which only selections particles with positive charge.

selection logic. This choice makes `ParticleFilter` objects lightweight and provides a means of implementing a `State` instance-specific cache of `ParticleGroup` objects. The latter ensures that we do not create multiples of the same `ParticleGroup` which can occupy large amounts of memory. The caching also allows the creation of many of the same `ParticleFilter` object without needing to worry about memory constraints.

`ParticleFilter` can be subclassed (like `Trigger` and `Variant`), but only through the `CustomParticleFilter` class. This is necessary to prevent some internal details from leaking to the user. An example of a `CustomParticleFilter` that selects only particles with positive charge is given below.

6.4.4 Custom Actions

In HOOMD-blue, we distinguish between the objects that perform an action on the simulation state (called *Actions*) and their containing objects that deal with setting state and the user interface (called *Operations*). Through composition, HOOMD-blue offers the ability to create custom actions in Python and wrap them in our `_CustomOperation` subclasses (divided on the type of action performed) allowing the execution of the action in the `Simulation`

```python
        import hoomd
        from hoomd.filter import Intersection, All, Type
        from hoomd.custom import Action

        class SwapType(Action):
            def __init__(self, initial_type, final_type, rate, filter=All()):
                self.final_type = final_type
                self.rate = rate
                self.filter = Intersection([Type(initial_type), filter])

            def act(self, timestep):
                state = self._state
                final_type_id = state.particle_types.index(self.final_type)
                tags = self.filter(state)
                with state.cpu_local_snapshot as snap:
                    tags = np.intersect1d(tags, snap.particles.tag, True)
                    part = data.particles
                    filtered_index = part.rtags[tags]
                    N_swaps = int(len(tags) * self.rate)
                    mask = np.random.choice(
                        filtered_index, N_swaps, replace=False)
                    part.typeid[mask] = final_type_id

```

Program 6.10: Example of a custom action which updates particle types across a simulation.

run loop. The feature makes user created actions behave indistinguishably from native C++ actions. Through custom actions, users can modify state, tune hyperparameters for performance or observe parts of the simulation. In addition, we are adding a signal for Actions to send that would stop a `Simulation.run` call. This would allow actions to stop the simulation when they complete, which could be useful for tasks like tuning MC trial move sizes. With respect to performance, with zero copy access to the data on the CPU or GPU, custom actions can also achieve high performance using standard Python libraries like NumPy, SciPy, Numba, CuPy and others. Below we show an example of an `Action` that switches particles of type `initial_type` to type `final_type` with a specified `rate` each time it is run. This action could be refined to implement a reactive MC move reminiscent of [49] or to have a variable switch rate. These exercises are left to the reader.

6.5 Conclusion

With modern simulation analysis packages such as `freud` [104], MDTraj [87] and MDAnalysis [51, 91], initialization tools such as `mbuild` and `foyer` and visualization packages like OVITO and `plato` using Python APIs, HOOMD-blue, built from the ground up with Python in mind, fits in seamlessly. Version 3.0 improves upon this and presents a Pythonic API that encourages customization. Through enabling Python subclassing of C++ classes, introducing custom actions and exposing data in zero-copy arrays/buffers, we allow HOOMD-blue users to utilize the full potential of Python and the scientific Python community.

6.6 Acknowledgements

This research was supported by the National Science Foundation, Division of Materials Research Award # DMR 1808342 (HOOMD-blue algorithm and performance development) and by the National Science Foundation, Office of Advanced Cyberinfrastructure Award # OAC 1835612 (Pythonic architecture for MoSDeF). Hardware provided by NVIDIA Corp. is gratefully acknowledged. This research was supported in part through computational resources and services supported by Advanced Research Computing at the University of Michigan, Ann Arbor.

CHAPTER 7

Conclusions

Conceptually, computational studies in soft matter can answer a near limitless number of questions. However, practically, computational power, current methods, human interaction time and other factors limit the scale and depth of studies. To increase the scope of questions accessible to the computational researcher, in this dissertation we present multiple points of improvement in methods for the analysis of self-assembly simulations in soft matter with potential utility far outside the field.

7.1 Dissertation Summary

In Chapter 3, we developed and tested a pipeline to detect events from the raw data generated by molecular simulations. We highlighted the general pipeline as well as described various options at each step. By applying this pipeline to two simulations of disparate types and physics, we showed that `dupin`'s pipeline can accurately partition simulations into regions of transition.

In Chapter 4, we derived a new order parameter for detecting the formation of point group symmetry in local environments, PGOP. After deriving the parameter, we showed its behavior compared to MSM and in two complex (multiple Wykoff site) crystals. The new order parameter allows one to directly probe the nature of the disorder-order transitions as bond-orientational ordering develops throughout the system.

In Chapter 5, we complement PGOP with a generalized measure of local coordination number $CN_{\vec{V}}$. We show its inspiration and derive our extension to the original order parameter. Following that, we tested the information in $CN_{\vec{V}}$ by training machine learning classifiers to distinguish between noisy FCC, BCC, SC and ideal gas particles. We also compared $CN_{\vec{V}}$ to PGOP and test them both separately and together in the aforementioned machine learning task. Through that test, we confirmed that $CN_{\vec{V}}$ contains distinct information from the information present in PGOP and that when combined, can further elucidate local order.

In Chapter 6, we outlined the changes to the molecular simulation engine and library HOOMD-blue for its version 3.0 release that heavily increased its extensibility, flexibility and interoperability. We outlined the new object-oriented API, extensibility through composition and inheritance and accessibility to HOOMD's data buffers on the GPU and CPU. These changes allow for new simulation protocols and techniques to be developed and deployed in HOOMD-blue more quickly and easily.

7.2 Outlook

The future is bright for soft matter. More building blocks in the form of nanoparticles are being created each year [112, 65]. Progress in manufacturing techniques also continues unabated [80]. The promise of full control over both structure and material looms ever larger, and we are closer than ever to revolutionizing various technologies from electronic to biosensors.

The ideas presented in this dissertation provide new tools for computationalists to further aid experimentalists in the field. We believe the need for more automated analysis and execution of computational studies is vital to help propel research further. By enabling larger studies per researcher hour, we can iterate over more possible materials quicker and achieve results in fractions of the current time studies take.

Furthermore, there is work that can be done to improve upon our contributions. For our detection pipeline `dupin`, work on more complicated cost functions such as cross-entropy [50] should be explored. In addition, adding built-in methods to reduced features based on spatial dependence would further improve `dupin`'s utility in analyzing self-assembly simulations. For PGOP, although the algorithm is developed, the software implementation could utilize GPU computing to make real-time/online analysis possible.

7.3 Final Thoughts

Tool-making made humanity. That fact hasn't stopped even in our modern era. In research, we depend on tools developed by scientists, engineers and others every day. My work has provided very specific tools for the field of computational soft matter to enable heretofore impossible studies. My desire is that this work would serve as a crucial aid to my fellow lab members and scientists around the world as we push the boundaries of what is known.

APPENDIX A

Change Point Detection of Events in Molecular Simulations using dupin: Supplementary Information

This appendix is adapted from a manuscript currently under review authored by Brandon L. Butler, Domogoj Fijan and Sharon C. Glotzer to at a peer-reviewed journal.

A.1 Change Point Detection

The problem of computing the change point locations in the given signal can be cast as the following optimization problem:

$$K = \underset{k \in K, |K|=m}{\arg\min} \sum_{i=1}^{m+1} C(S[k_{i-1}; k_i]), \tag{A.1}$$

where C is the cost function, m is the number of change points, k_0 is 0 and k_{m+1} is the signal length. The Python package ruptures [130] implements various algorithms that either optimally or approximately solve this optimization problem. For this paper we utilize ruptures's optimal dynamic programming algorithm which was first introduced for another problem [17]. The ruptures package uses a single thread/process and the dynamic programming algorithm has a N^2 time complexity in trajectory length.

Another common problem in CPD is detecting a mean shift in random variables. Mean shift cost functions are functions such as

$$C(S[i,j)) = \sum_{l=i}^{j-1} |\mu_{S[i,j)} - S[l]|_p, \tag{A.2}$$

which penalize deviations from the mean with some norm, p. Figure A.1 shows the behavior of the dynamic programming method on a generated mean-shift signal where $|K| = n$ and $|K| \neq n$ where n is the correct number of change points.

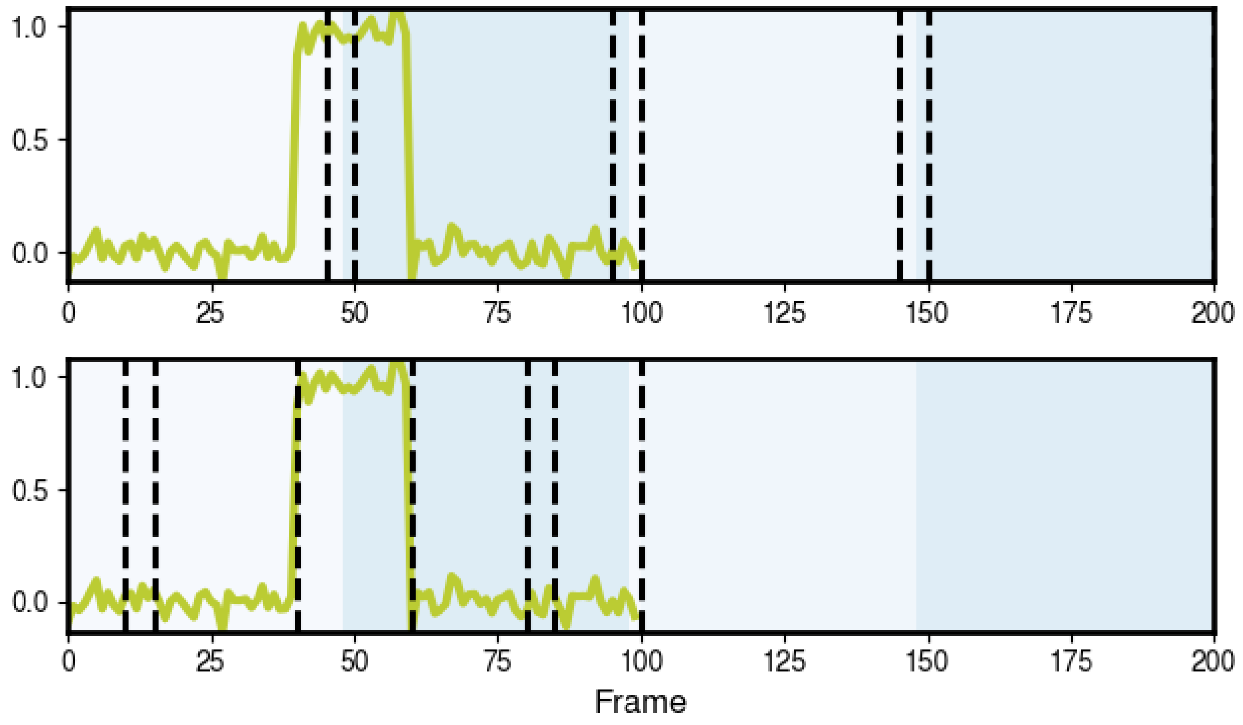

Figure A.1: Two detections of change points on a generated mean-shift signal with three shifts. The alternating blue and green regions represent the correct partitioning of the system. (a) Plot of the best three change points according to the dynamic programming optimizer and L_1 mean-shift cost function. (b) Plot of the best six change points according to the dynamic programming optimizer and L_1 mean-shift cost function. Notice that when $n > 3$, we fit to spurious change points. Any reduction of cost is sufficient to fit to a new point. Thus highlighting the need for method of determining an appropriate n.

A.2 Kneedle Algorithm

To solve the problem of selecting the optimal number of change points n shown in Figure A.1, we apply elbow detection over various change point set cardinalities. We use the kneedle algorithm [113] as implemented in the kneed Python package. First step to detect an elbow is to transform the data $(x_i, y_i) \rightarrow (x_{max} - x_i, y_{max} - y_i)$ which is automatically handled by kneed with appropriate settings. The algorithm starts with fitting the discrete data to a smoothing spline. The spline is then normalized to the unit square to prevent magnitude from determining elbow location. A new set of points D_d is then created where $(x, y) \rightarrow (x, y - x)$. The local maxima (minima for an elbow) is filtered out into $D_{lmx} = (x_{lmx}, y_{lmx})$, where

$$x_{lmx_i} = x_{d_i} \tag{A.3}$$

$$y_{lmx_i} = y_{d_i} \text{ where } y_{d_{i-1}} < y_{d_i},\ y_{d_i} > y_{d_{i+1}} \tag{A.4}$$

For each entry in D_{lmx} a threshold is determined using a sensitivity parameter S, where the threshold equals

$$T_{lmx} = y_{lmx} - S \cdot \frac{\sum_{i=1}^{n-1} x_{d_{i+1}} - x_{d_i}}{n-1} \tag{A.5}$$

A knee is detected if for any maximum y_{lmx}, the value of a consecutive y_d is less than T_{lmx} before a value greater than y_{lmx} is observed.

A.3 Feature Selection

A.3.1 Mean Shift

The mean shift filter determines the mean and standard deviation for all features at the signal's start and finish. Subsequently, these values are employed to calculate the probability that the means of the distributions at either end might be derived from the distribution established at the signal's other end, assuming Gaussian random variables. To do this, we pick a set number of end points at either end, (a, b), which we will use to generate the two distributions. We then compute a's and b's mean and standard deviation and compute the following value

$$n_\sigma = \frac{|\mu_a - \mu_b|}{\min(\sigma_a, \sigma_b)} \tag{A.6}$$

which is the maximum number of standard deviation's of one side's mean from the other's. We finally compute the likelihood this difference would be generated by an iid sampling of the distribution, l and compare this to a sensitivity s, using the error function. If $l < s$, we keep the feature otherwise we discard it.

A.3.2 Feature Correlation

To remove redundant features `dupin` provides a feature selection method that uses the correlation between features to select uncorrelated features. First `dupin` creates a similarity matrix S which uses the absolute value of the Pearson correlation coefficient. This matrix is then fed into the Spectral clustering algorithm from scikit-learn [21, 100]. To compute the optimal number of features, we compute a spectral clustering for all number of clusters up to a specified maximum ($[2, N_{max}]$) and take the clustering that has the highest silhouette score. We then take n features from each cluster. Features are selected from a cluster either randomly or through a provided feature importance score.

APPENDIX B

PGOP

This appendix is adapted from a draft paper authored by Brandon L. Butler, Maria W. Rashidi and Sharon C. Glotzer to be submitted to a peer-reviewed journal.

B.1 Ideal Gas Baselines

In Figure B.1, we give the distributions of PGOP and Minkowski Structure Metrics for the ideal gas distribution with Voronoi tessellation neighbor list. This figure shows the expected range of outputs for a random input to both order parameters and can serve as a basis of normalization and distinguishing a true signal of order from noise.

B.2 Minkowski Structure Metrics Noise Behavior

We look at the behavior of various values of ℓ's in MSM with respect to noise in systems of SC, FCC and BCC (cf. Figure 4.2). As we can see in general MSM behave similarly with respect to ℓ and noise.

B.3 Binary LJ Glass Mean Squared Displacement

To show that we have reached the glass using the mean squared displacement (MSD), we ran the simulation in Section 4.2.3.3 for an additional 3 million steps at the final temperature of

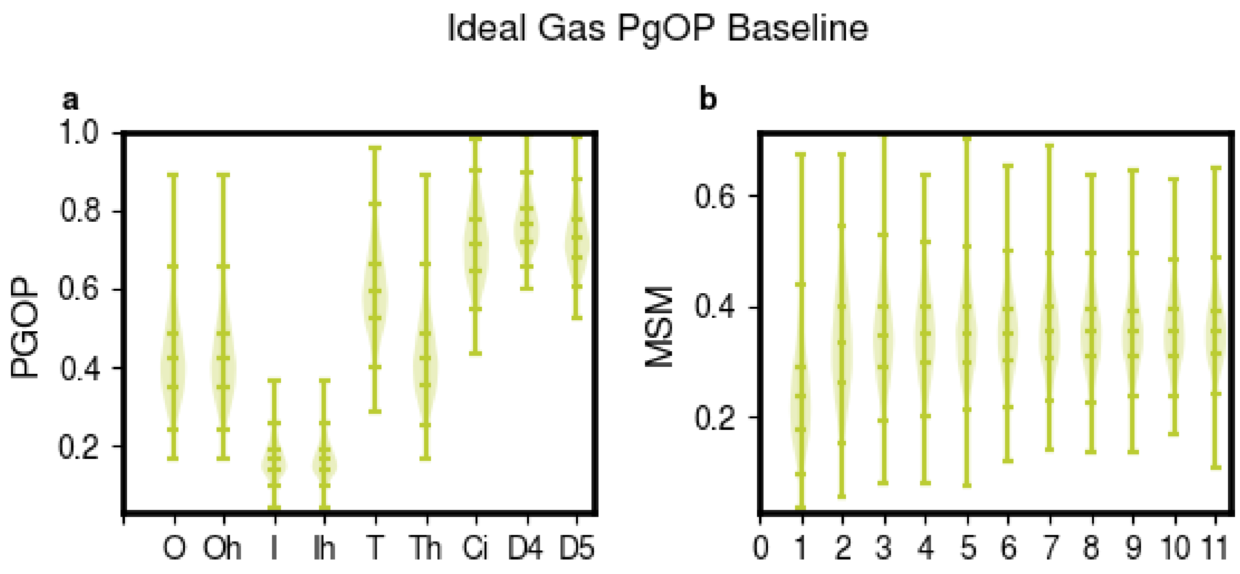

Figure B.1: Violin plots of the ideal gas distributions for (a) PGOP and (b) Minkowski Structure Metrics. The 0.0, 0.2, 0.25, 0.5, 0.75, 0.98, 1.0 quantiles are shown for each (a) point group and (b) ℓ.

0.1 kT. We then computed the MSD with a window increment size of 50,000 timesteps. The results can be seen in Figure B.3. The MSD shows immediate caging in the given timescales indicating the simulation is indeed in the glass.

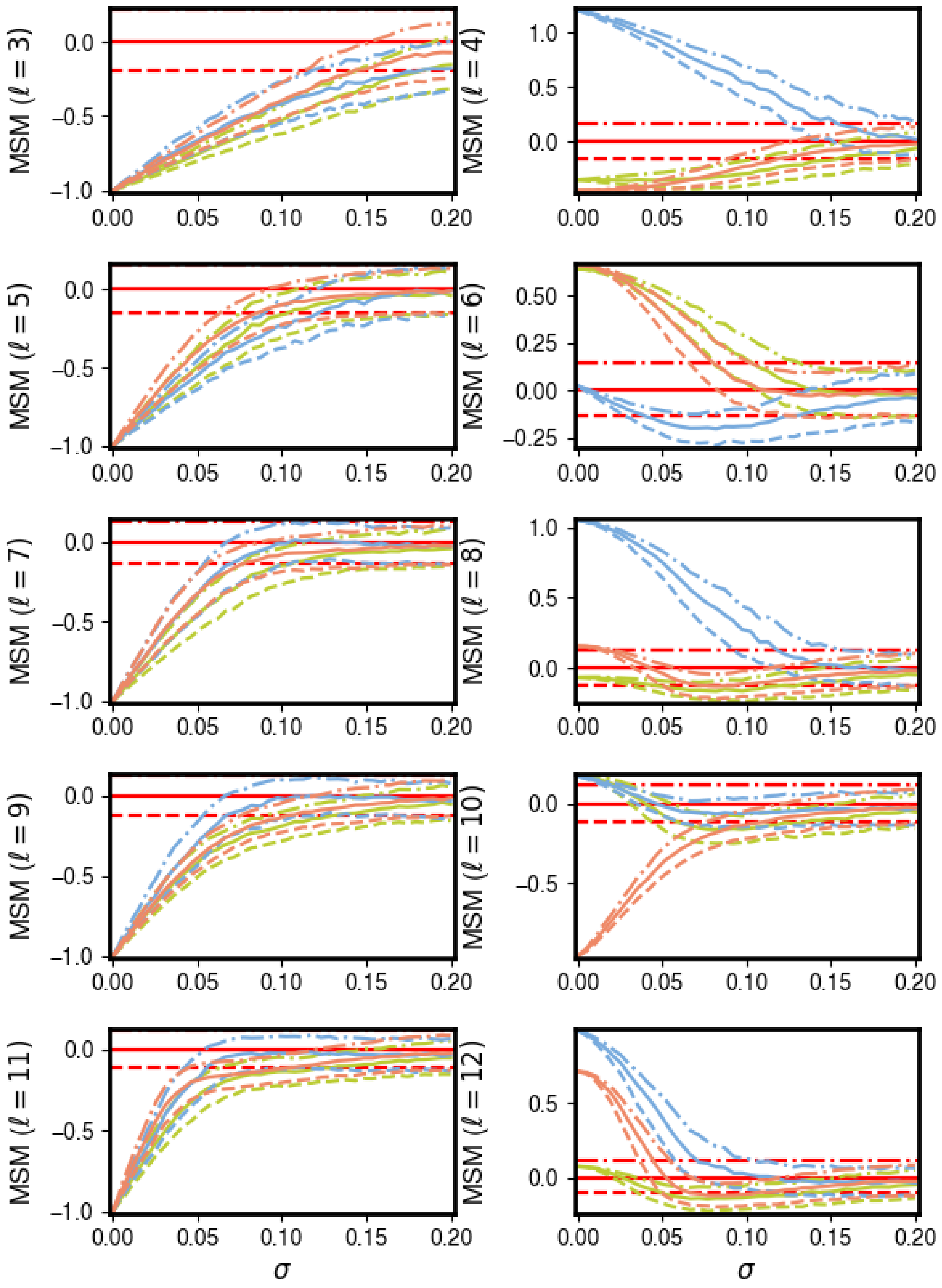

Figure B.2: Plots of MSM behavior with respect to ℓ and noise. Plots show the 0.25, 0.5 and 0.75 quantiles in dashed, solid and dotted lines. A legend for the individual crystals is not shown as the behavior is identical among them and individual lines cannot be discerned. The PGOP values have been normalized by the median of IG, so 0.0 is the median PGOP of the IG. Red lines correspond to the 0.25, 0.5 and 0.75 quantiles with the same line style designations as the crystal lines. Crystals have a unit cell of length 1 and go from a standard deviation of 0 to 0.2.

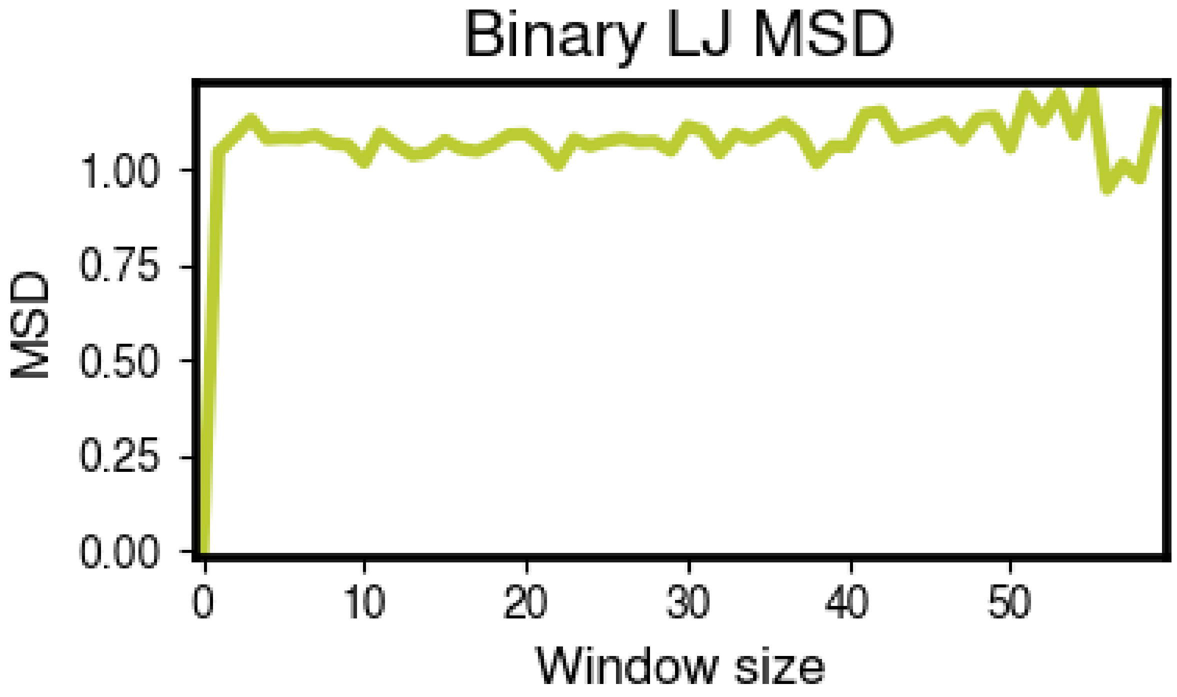

Figure B.3: Plot of the MSD of the binary LJ glass system run for an additional 3 million steps.

APPENDIX C

Further HOOMD Programs

This chapter is adapted from the appendix of Ref: "HOOMD-blue version 3.0 A Modern, Extensible, Flexible, Object-Oriented API for Molecular Simulations", Brandon L. Butler, Vyas Ramasubramani, Joshua A. Anderson, Sharon C. Glotzer, Proceedings of the 19th Python in Science Conference (2020) [22]

In the appendix, we will provide more substantial applications of features new to HOOMD-blue.

C.1 Trigger that detects nucleation

This example demonstrates a `Trigger` that returns true when a threshold Q_6 Steinhardt order parameter [122] (as calculated by freud) is reached. Such a `Trigger` could be used for BCC nucleation detection which could trigger a decrease in cooling rate, a more frequent output of simulation trajectories or any other desired action. Also, in this example we showcase the use of the zero-copy rank-local data access. This example also requires the use of ghost particles, which are a subset of particles bordering a MPI rank's local box. Ghost particles are known by a rank, but the rank is not responsible for updating them. In this case, ghost particles are required for computing the Q_6 value for particles near the edges of the current rank's local simulation box.

```python
import numpy as np
import freud
from mpi4py import MPI
from hoomd.trigger import Trigger

class Q6Trigger(Trigger):
    def __init__(self, simulation, threshold, mpi_comm=None):
        super().__init__()
        self.threshold = threshold
        self.state = simulation.state
        nr = simulation.device.num_ranks
        if nr > 1 and mpi_comm is None:
            raise RuntimeError()
        elif nr > 1:
            self.comm = mpi_comm
        else:
            self.comm = None
        self.q6 = freud.order.Steinhardt(l=6)

    def __call__(self, timestep):
        with self.state.cpu_local_snapshot as data:
            part = data.particles
            box = data.box
            aabb_box = freud.locality.AABBQuery(
                box, part.positions_with_ghosts)
            nlist = aabb_box.query(
                part.position,
                {'num_neighbors': 12, 'exclude_ii': True})
            Q6 = np.nanmean(self.q6.compute(
                (box, part.positions_with_ghosts),
                nlist).particle_order)
            if self.comm:
                return self.comm.allreduce(
                    Q6 >= self.threshold, op=MPI.LOR)
            else:
                return Q6 >= self.threshold

```

Program C.1: A more complicated example of a custom trigger which check the Steinhardt order parameter $l = 6$ and triggers when Q_6 reaches a defined threshold.

C.2 Pandas Logger Back-End

Here we highlight the ability to use the `Logger` class to create a Pandas back end for simulation data. It will store the scalar and string quantities in a single `pandas.DataFrame` object while each array-like object is stored in a separate `DataFrame` object. All `DataFrame` objects are stored in a single dictionary.

```python
import pandas as pd
from hoomd.custom import Action
from hoomd.util import (
    dict_flatten, dict_filter, dict_map)

def is_flag(flags):
    def func(v):
        return v[1] in flags
    return func

def not_none(v):
    return v[0] is not None

def hnd_2D_arrays(v):
    if v[1] in ['scalar', 'string', 'state']:
        return v
    elif len(v[0].shape) == 2:
        return {
            str(i): col for i, col in enumerate(v[0].T)}


class DataFrameBackEnd(Action):
    def __init__(self, logger):
        self.logger = logger

    def act(self, timestep):
        log_dict = self.logger.log()
        is_scalar = is_flag(['scalar', 'string'])
        sc = dict_flatten(dict_map(dict_filter(
            log_dict, lambda x: not_none(x) and is_scalar(x)),
            lambda x: x[0]))
        rem = dict_flatten(dict_map(dict_filter(
            log_dict, lambda x: not_none(x) and not is_scalar(x)),
            hnd_2D_arrays))

        if not hasattr(self, 'data'):
            self.data = {
                'scalar': pd.DataFrame(
                    columns=['.'.join(k) for k in sc]),
                'array': {'.'.join(k): pd.DataFrame()
                        for k in rem}}

        sdf = pd.DataFrame(
            {'.'.join(k): v for k, v in sc.items()},
            index=[timestep])
        rdf = {'.'.join(k): pd.DataFrame(v, columns=[timestep]).T
            for k,v in rem.items()}
        data = self.data
        data['scalar'] = data['scalar'].append(sdf)
        data['array'] = {
            k: v.append(rdf[k]) for k, v in data['array'].items()}

```

Program C.2: Example of a custom action which writes data actively to a `pandas.DataFrame` object.

BIBLIOGRAPHY

[1] CuPy, 2015. https://cupy.chainer.org/.

[2] Salma M. Abdel-Hafez, Rania M. Hathout, and Omaima A. Sammour. Tracking the transdermal penetration pathways of optimized curcumin-loaded chitosan nanoparticles via confocal laser scanning microscopy. *International Journal of Biological Macromolecules*, 108:753–764, 03 2018. https://www.sciencedirect.com/science/article/pii/S0141813017329100.

[3] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1-2:19–25, 09 2015. http://www.sciencedirect.com/science/article/pii/S2352711015000059.

[4] Carl S. Adorf, Paul M. Dodd, Vyas Ramasubramani, and Sharon C. Glotzer. Simple data and workflow management with the signac framework. *Computational Materials Science*, 146:220–229, 04 2018. http://www.sciencedirect.com/science/article/pii/S0927025618300429.

[5] Carl S. Adorf, Timothy C. Moore, Yannah J. U. Melle, and Sharon C. Glotzer. Analysis of Self-Assembly Pathways with Unsupervised Machine Learning Algorithms. *J. Phys. Chem. B*, 124(1):69–78, 01 2020. https://doi.org/10.1021/acs.jpcb.9b09621.

[6] B. J. Alder and T. E. Wainwright. Studies in Molecular Dynamics. I. General Method. *J. Chem. Phys.*, 31(2):459–466, 08 1959. https://aip.scitation.org/doi/abs/10.1063/1.1730376.

[7] Rosalind J. Allen, Chantal Valeriani, and Pieter Rein ten Wolde. Forward flux sampling for rare event simulations. *Journal of Physics: Condensed Matter*, 21(46):463102, October 2009.

[8] Daniel Alves, Katia Obraczka, and Rick Lindberg. Identifying relevant data center telemetry using change point detection. In *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, pages 1–4, 2020.

[9] Samaneh Aminikhanghahi and Diane J. Cook. A survey of methods for time series change point detection. *Knowl Inf Syst*, 51(2):339–367, 05 2017. https://doi.org/10.1007/s10115-016-0987-z.

[10] Joshua A. Anderson, M. Eric Irrgang, and Sharon C. Glotzer. Scalable Metropolis Monte Carlo for simulation of hard shapes. *Computer Physics Communications*, 204:21–30, 07 2016. https://www.sciencedirect.com/science/article/pii/S001046551630039X.

[11] Joshua A. Anderson, Jens Glaser, and Sharon C. Glotzer. HOOMD-blue: A Python package for high-performance molecular dynamics and hard particle Monte Carlo simulations. *Computational Materials Science*, 173:109363, 02 2020. http://www.sciencedirect.com/science/article/pii/S0927025619306627.

[12] Joshua A. Anderson, Chris D. Lorenz, and A. Travesset. General purpose molecular dynamics simulations fully implemented on graphics processing units. *Journal of Computational Physics*, 227(10):5342–5359, 05 2008. http://www.sciencedirect.com/science/article/pii/S0021999108000818.

[13] Andrew Y. Ng, Micheal I. Jorden, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856, 2002.

[14] Alexander Aue, Siegfried Hörmann, Lajos Horváth, and Matthew Reimherr. Break detection in the covariance structure of multivariate time series models. *The Annals of Statistics*, 37(6B):4046 – 4087, 2009. https://doi.org/10.1214/09-AOS707.

[15] Daniel Barry and J. A. Hartigan. A bayesian analysis for change point problems. *Journal of the American Statistical Association*, 88(421):309–319, 1993.

[16] Albert P. Bartók, Risi Kondor, and Gábor Csányi. On representing chemical environments. *Phys. Rev. B*, 87(18):184115, 05 2013. https://link.aps.org/doi/10.1103/PhysRevB.87.184115.

[17] Richard Bellman. On a routing problem. *Quart. Appl. Math.*, 16(1):87–90, 1958. https://www.ams.org/qam/1958-16-01/S0033-569X-1958-0102435-2/.

[18] H. J. C. Berendsen, D. van der Spoel, and R. van Drunen. GROMACS: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications*, 91(1):43–56, 09 1995. http://www.sciencedirect.com/science/article/pii/001046559500042E.

[19] Emanuele Boattini, Marjolein Dijkstra, and Laura Filion. Unsupervised learning for local structure detection in colloidal systems. *The Journal of Chemical Physics*, 151(15):154901, 10 2019. http://aip.scitation.org/doi/abs/10.1063/1.5118867.

[20] Marcel Bosc, Fabrice Heitz, Jean-Paul Armspach, Izzie Namer, Daniel Gounot, and Lucien Rumbach. Automatic change detection in multimodal serial mri: application to multiple sclerosis lesion evolution. *NeuroImage*, 20(2):643–656, 2003.

[21] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake Vanderplas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: Experiences from the scikit-learn project. http://arxiv.org/abs/1309.0238, 09 2013.

[22] Brandon L. Butler, Vyas Ramasubramani, Joshua A. Anderson, and Sharon C. Glotzer. Hoomd-blue version 3.0 a modern, extensible, flexible, object-oriented api for molecular simulations. In Meghann Agarwal, Chris Calloway, Dillon Niederhut, and David Shupe, editors, *Proceedings of the 19th Python in Science Conference*, pages 24 – 31, 7 2020.

[23] Surendra Byna, Jerry Chou, Oliver Rubel, Prabhat, Homa Karimabadi, William S. Daughter, Vadim Roytershteyn, E. Wes Bethel, Mark Howison, Ke-Jou Hsu, Kuan-Wu Lin, Arie Shoshani, Andrew Uselton, and Kesheng Wu. Parallel I/O, analysis, and visualization of a trillion particle simulation. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 11 2012.

[24] Forest L. Carter. Quantifying the concept of coordination number. *Acta Crystallographica Section*, 34(10):2962–2966, Oct 1978. https://doi.org/10.1107/S0567740878009838.

[25] David A. Case, Thomas E. Cheatham, Tom Darden, Holger Gohlke, Ray Luo, Kenneth M. Merz, Alexey Onufriev, Carlos Simmerling, Bing Wang, and Robert J. Woods. The Amber biomolecular simulation programs. *Journal of Computational Chemistry*, 26(16):1668–1688, 2005. https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.20290.

[26] Heejung W. Chung, Rodrigo Freitas, Gowoon Cheon, and Evan J. Reed. Data-centric framework for crystal structure identification in atomistic simulations using machine learning. *Phys. Rev. Mater.*, 6(4):043801, 04 2022. https://link.aps.org/doi/10.1103/PhysRevMaterials.6.043801.

[27] E. Clark and A. Quinn. A data-driven Bayesian sampling scheme for unsupervised image segmentation. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, volume 6, pages 3497–3500 vol.6, 03 1999.

[28] Peter T Cummings and Justin B Gilmer. Open-source molecular modeling software in chemical engineering. *Current Opinion in Chemical Engineering*, 23:99–105, 03 2019. http://www.sciencedirect.com/science/article/pii/S2211339819300073.

[29] Pablo F. Damasceno, Michael Engel, and Sharon C. Glotzer. Predictive Self-Assembly of Polyhedra into Complex Structures. *Science*, 337(6093):453–457, 07 2012. https://science.sciencemag.org/content/337/6093/453.

[30] Christoph Dellago, Peter G. Bolhuis, Félix S. Csajka, and David Chandler. Transition path sampling and the calculation of rate constants. *The Journal of Chemical Physics*, 108(5):1964–1977, February 1998.

[31] Bradley Dice. *Complex Crystallization Pathways Analyzed in a Continuous Feature Space*. Thesis, University of Michigan, 2021. http://deepblue.lib.umich.edu/handle/2027.42/170058.

[32] Bradley D. Dice, Brandon L. Butler, Vyas Ramasubramani, Alyssa Travitz, Michael M. Henry, Hardik Ojha, Kelly L. Wang, Carl S. Adorf, Eric Jankowski, and Sharon C. Glotzer. Signac: Data Management and Workflows for Computational Researchers. *Proceedings of the 20th Python in Science Conference*, pages 23–32, 2021. https://conference.scipy.org/proceedings/scipy2021/bradley_dice.html.

[33] Bradley D. Dice, Vyas Ramasubramani, Eric S. Harper, Matthew P. Spellings, Joshua A. Anderson, and Sharon C. Glotzer. Analyzing Particle Systems for Machine Learning and Data Visualization with freud. *Proceedings of the 18th Python in Science Conference*, pages 27–33, 2019. http://conference.scipy.org/proceedings/scipy2019/bradley_dice.html.

[34] C. Dietz, T. Kretz, and M. H. Thoma. Machine-learning approach for local classification of crystalline structures in multiphase systems. *Phys. Rev. E*, 96(1):011301, 07 2017. https://link.aps.org/doi/10.1103/PhysRevE.96.011301.

[35] Gregory L. Dignon, Wenwei Zheng, Young C. Kim, Robert B. Best, and Jeetain Mittal. Sequence determinants of protein phase behavior from a coarse-grained model. *PLOS Computational Biology*, 14(1):e1005941, 01 2018. https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005941.

[36] Julia Dshemuchadse, Pablo F. Damasceno, Carolyn L. Phillips, Michael Engel, and Sharon C. Glotzer. Moving beyond the constraints of chemistry via crystal structure discovery with isotropic multiwell pair potentials. *Proceedings of the National Academy of Sciences*, 118(21):e2024034118, 05 2021. https://www.pnas.org/doi/abs/10.1073/pnas.2024034118.

[37] Peter Eastman, Jason Swails, John D. Chodera, Robert T. McGibbon, Yutong Zhao, Kyle A. Beauchamp, Lee-Ping Wang, Andrew C. Simmonett, Matthew P. Harrigan, Chaya D. Stern, Rafal P. Wiewiora, Bernard R. Brooks, and Vijay S. Pande. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. *PLOS Computational Biology*, 13(7):e1005659, 07 2017. https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005659.

[38] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz and dynagraph - static and dynamic graph drawing tools. In *Graph Drawing Software*, pages 127–148. Springer-Verlag, 2003.

[39] Michael Engel. Point Group Analysis in Particle Simulation Data. *arxiv (cond-mat)*, 06 2021. http://arxiv.org/abs/2106.14846.

[40] Michael Engel, Pablo F. Damasceno, Carolyn L. Phillips, and Sharon C. Glotzer. Computational self-assembly of a one-component icosahedral quasicrystal. *Nature Mater*, 14(1):109–116, 01 2015. https://www.nature.com/articles/nmat4152.

[41] Yue Fan, Takuya Iwashita, and Takeshi Egami. How thermally activated deformation starts in metallic glass. *Nat Commun*, 5(1):1–7, 09 2014. https://www.nature.com/articles/ncomms6083.

[42] Enrico Fermi, P Pasta, Stanislaw Ulam, and Mary Tsingou. Studies of the nonlinear problems, 1955.

[43] Kyle D. Feuz, Diane J. Cook, Cody Rosasco, Kayela Robertson, and Maureen Schmitter-Edgecombe. Automated detection of activity transitions for prompting. *IEEE Transactions on Human-Machine Systems*, 45(5):575–585, 2015.

[44] L. Filion, M. Hermes, R. Ni, and M. Dijkstra. Crystal nucleation of hard spheres using molecular dynamics, umbrella sampling, and forward flux sampling: A comparison of simulation techniques. *The Journal of Chemical Physics*, 133(24):244115, 12 2010. https://doi.org/10.1063/1.3506838.

[45] Steffen Fischer, Alexander Exner, Kathrin Zielske, Jan Perlich, Sofia Deloudi, Walter Steurer, Peter Lindner, and Stephan Förster. Colloidal quasicrystals with 12-fold and 18-fold diffraction symmetry. *Proceedings of the National Academy of Sciences*, 108(5):1810–1814, February 2011.

[46] Ronald Aylmer Fisher. Dispersion on a sphere. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 217(1130):295–305, January 1997. https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1953.0064.

[47] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-phong Vo. A Technique for Drawing Directed Graphs. *Ieee Transactions on Software Engineering*, 19(3):214–230, 1993.

[48] Jens Glaser, Trung Dac Nguyen, Joshua A. Anderson, Pak Lui, Filippo Spiga, Jaime A. Millan, David C. Morse, and Sharon C. Glotzer. Strong scaling of general-purpose molecular dynamics simulations on GPUs. *Computer Physics Communications*, 192:97–107, 07 2015. http://www.sciencedirect.com/science/article/pii/S0010465515000867.

[49] Sharon C. Glotzer, Dietrich Stauffer, and Naeem Jan. Monte Carlo simulations of phase separation in chemically reactive binary mixtures. *Phys. Rev. Lett.*, 72(26):4109–4112, 06 1994. https://link.aps.org/doi/10.1103/PhysRevLett.72.4109.

[50] I. J. Good. Maximum entropy for hypothesis formulation, especially for multidimensional contingency tables. *The Annals of Mathematical Statistics*, 34(3):911–934, 1963. http://www.jstor.org/stable/2238473.

[51] Richard J. Gowers, Max Linke, Jonathan Barnoud, Tyler J. E. Reddy, Manuel N. Melo, Sean L. Seyler, Jan Domański, David L. Dotson, Sébastien Buchoux, Ian M. Kenney, and Oliver Beckstein. MDAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations. *Proceedings of the 15th Python in Science Conference*, pages 98–105, 2016. https://conference.scipy.org/proceedings/scipy2016/oliver_beckstein.html.

[52] Horacio V. Guzman, Nikita Tretyakov, Hideki Kobayashi, Aoife C. Fogarty, Karsten Kreis, Jakub Krajniak, Christoph Junghans, Kurt Kremer, and Torsten Stuehn. ESPResSo++ 2.0: Advanced methods for multiscale molecular simulation. *Computer Physics Communications*, 238:66–76, 05 2019. http://www.sciencedirect.com/science/article/pii/S0010465518304399.

[53] Alfred Haar. Der Massbegriff in der Theorie der Kontinuierlichen Gruppen. *Annals of Mathematics*, 34(1):147–169, 1933. https://www.jstor.org/stable/1968346.

[54] Manhyung Han, La The Vinh, Young-Koo Lee, and Sungyoung Lee. Comprehensive context recognizer based on multimodal sensors in a smartphone. *Sensors*, 12(9):12588–12605, 9 2012. http://dx.doi.org/10.3390/s120912588.

[55] Trevor Hastie, Jerome Friedman, and Robert Tibshirani. Linear Methods for Classification. In Trevor Hastie, Jerome Friedman, and Robert Tibshirani, editors, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Series in Statistics, pages 79–113. Springer, 2001. https://doi.org/10.1007/978-0-387-21606-5_4.

[56] Shohei Hido, Tsuyoshi Idé, Hisashi Kashima, Harunobu Kubo, and Hirofumi Matsuzawa. Unsupervised Change Analysis Using Supervised Learning. In Takashi Washio, Einoshin Suzuki, Kai Ming Ting, and Akihiro Inokuchi, editors, *Advances in Knowledge Discovery and Data Mining*, Lecture Notes in Computer Science, pages 148–159. Springer, 2008.

[57] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1995.

[58] William G. Hoover. Atomistic nonequilibrium computer simulations. *Physica A: Statistical Mechanics and its Applications*, 118(1):111–122, 1983. https://www.sciencedirect.com/science/article/pii/0378437183901802.

[59] William G. Hoover, Anthony J. C. Ladd, and Bill Moran. High-strain-rate plastic flow studied via nonequilibrium molecular dynamics. *Phys. Rev. Lett.*, 48:1818–1820, Jun 1982. https://link.aps.org/doi/10.1103/PhysRevLett.48.1818.

[60] Philip D. Howes, Rona Chandrawati, and Molly M. Stevens. Colloidal nanoparticles as advanced biological sensors. *Science*, 346(6205):1247390, October 2014.

[61] Yuan-Chao Hu, Weiwei Jin, Jan Schroers, Mark D. Shattuck, and Corey S. O'Hern. Glass-forming ability of binary Lennard-Jones systems. *Physical Review Materials*, 6(7):075601, July 2022.

[62] Sebastiaan P. Huber, Spyros Zoupanos, Martin Uhrin, Leopold Talirz, Leonid Kahle, Rico Häuselmann, Dominik Gresch, Tiziano Müller, Aliaksandr V. Yakutovich, Casper W. Andersen, Francisco F. Ramirez, Carl S. Adorf, Fernando Gargiulo, Snehal Kumbhar, Elsa Passaro, Conrad Johnston, Andrius Merkys, Andrea Cepellotti, Nicolas Mounet, Nicola Marzari, Boris Kozinsky, and Giovanni Pizzi. AiiDA 1.0, a scalable computational infrastructure for automated reproducible workflows and data provenance. *Sci Data*, 7(1):300, 09 2020. http://www.nature.com/articles/s41597-020-00638-4.

[63] John D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3):90–95, 05 2007.

[64] Wayne Iba and Pat Langley. Induction of one-level decision trees. In Derek Sleeman and Peter Edwards, editors, *Machine Learning Proceedings 1992*, pages 233–240, San Francisco (CA), 1992. Morgan Kaufmann. https://www.sciencedirect.com/science/article/pii/B9781558602472500358.

[65] Prasad Govindrao Jamkhande, Namrata W. Ghule, Abdul Haque Bamer, and Mohan G. Kalaskar. Metal nanoparticles synthesis: An overview on methods of preparation, advantages and disadvantages, and applications. *Journal of Drug Delivery Science and Technology*, 53:101174, 2019. https://www.sciencedirect.com/science/article/pii/S1773224718308189.

[66] Venkata Jandhyala, Stergios Fotopoulos, Ian MacNeill, and Pengyu Liu. Inference for single and multiple change-points in time series. *Journal of Time Series Analysis*, 34(4):423–446, 2013. http://onlinelibrary.wiley.com/doi/abs/10.1111/jtsa.12035.

[67] Kwanghwi Je, Sangmin Lee, Erin G. Teich, Michael Engel, and Sharon C. Glotzer. Entropic formation of a thermodynamically stable colloidal quasicrystal with negligible phason strain. *Proceedings of the National Academy of Sciences*, 118(7):e2011799118, February 2021.

[68] S. Karthika, T. K. Radhakrishnan, and P. Kalaichelvi. A Review of Classical and Nonclassical Nucleation Theories. *Crystal Growth & Design*, 16(11):6663–6681, 11 2016. https://doi.org/10.1021/acs.cgd.6b00794.

[69] Yoshinobu Kawahara and Masashi Sugiyama. Sequential change-point detection based on direct density-ratio estimation. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(2):114–127, 2012.

[70] Yoshinobu Kawahara, Takehisa Yairi, and Kazuo Machida. Change-point detection in time-series data based on subspace identification. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 559–564, 2007.

[71] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 289–296, 2001.

[72] Shin-Hyun Kim, Su Yeon Lee, Seung-Man Yang, and Gi-Ra Yi. Self-assembled colloidal structures for photonics. *NPG Asia Materials*, 3(1):25–33, January 2011.

[73] Christoph Klein, János Sallai, Trevor J. Jones, Christopher R. Iacovella, Clare McCabe, and Peter T. Cummings. A Hierarchical, Component Based Approach to Screening Properties of Soft Matter. In Randall Q Snurr, Claire S. Adjiman, and David A. Kofke, editors, *Foundations of Molecular Modeling and Simulation: Select Papers from FOMMS 2015*, Molecular Modeling and Simulation, pages 79–92. Springer, 2016. https://doi.org/10.1007/978-981-10-1128-3_5.

[74] Christoph Klein, Andrew Z. Summers, Matthew W. Thompson, Justin B. Gilmer, Clare McCabe, Peter T. Cummings, Janos Sallai, and Christopher R. Iacovella. Formalizing atom-typing and the dissemination of force fields with foyer. *Computational Materials Science*, 167:215–227, 09 2019. http://www.sciencedirect.com/science/article/pii/S0927025619303040.

[75] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A LLVM-based Python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, pages 1–6. Association for Computing Machinery, 11 2015. https://doi.org/10.1145/2833157.2833162.

[76] Peter Mahler Larsen, Søren Schmidt, and Jakob Schiøtz. Robust structural identification via polyhedral template matching. *Modelling Simul. Mater. Sci. Eng.*, 24(5):055007, 05 2016. https://dx.doi.org/10.1088/0965-0393/24/5/055007.

[77] Wolfgang Lechner and Christoph Dellago. Accurate determination of crystal structures based on averaged local bond order parameters. *J. Chem. Phys.*, 129(11):114707, 09 2008. https://aip.scitation.org/doi/full/10.1063/1.2977970.

[78] Olivier Ledoit and Michael Wolf. Honey, I Shrunk the Sample Covariance Matrix. *UPF Economics and Business Working Paper Working Paper No. 691*, (433840), 06 2003. https://papers.ssrn.com/abstract=433840.

[79] Mirim Lee, Chang-Myeon Lee, Kwang-Ryeol Lee, Evan Ma, and Jae-Chul Lee. Networked interpenetrating connections of icosahedra: Effects on shear transformations in metallic glass. *Acta Materialia*, 59(1):159–170, January 2011.

[80] Qi Li, John Kulikowski, David Doan, Ottman A. Tertuliano, Charles J. Zeman, Melody M. Wang, George C. Schatz, and X. Wendy Gu. Mechanical nanolattices printed using nanocluster-based photoresists. *Science*, 378(6621):768–773, 2022.

[81] Yein Lim, Sangmin Lee, and Sharon C. Glotzer. Engineering the Thermodynamic Stability and Metastability of Mesophases of Colloidal Bipyramids through Shape Entropy. *ACS Nano*, 17(5):4287–4295, March 2023.

[82] Haixin Lin, Sangmin Lee, Lin Sun, Matthew Spellings, Michael Engel, Sharon C. Glotzer, and Chad A. Mirkin. Clathrate colloidal crystals. *Science*, 355(6328):931–935, March 2017.

[83] Pit Losch, Weixin Huang, Emmett D. Goodman, Cody J. Wrasman, Alexander Holm, Andrew R. Riscoe, Jay A. Schwalbe, and Matteo Cargnello. Colloidal nanocrystals for heterogeneous catalysis. *Nano Today*, 24:15–47, February 2019.

[84] Binbin Luo, John W. Smith, Zihao Ou, and Qian Chen. Quantifying the Self-Assembly Behavior of Anisotropic Nanoparticles Using Liquid-Phase Transmission Electron Microscopy. *Acc. Chem. Res.*, 50(5):1125–1133, 05 2017. https://doi.org/10.1021/acs.accounts.7b00048.

[85] Marcus G. Martin. MCCCS Towhee: A tool for Monte Carlo molecular simulation. *Molecular Simulation*, 39(14-15):1212–1222, 12 2013. https://doi.org/10.1080/08927022.2013.828208.

[86] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Boosting Algorithms as Gradient Descent. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. https://proceedings.neurips.cc/paper_files/paper/1999/file/96a93ba89a5b5c6c226e49b88973f46e-Paper.pdf.

[87] Robert T. McGibbon, Kyle A. Beauchamp, Matthew P. Harrigan, Christoph Klein, Jason M. Swails, Carlos X. Hernández, Christian R. Schwantes, Lee-Ping Wang, Thomas J. Lane, and Vijay S. Pande. MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories. *Biophysical Journal*, 109(8):1528–1532, 10 2015. http://www.sciencedirect.com/science/article/pii/S0006349515008267.

[88] Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. http://arxiv.org/abs/1802.03426, 09 2020.

[89] Wes McKinney. Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*, pages 56–61, 2010. https://conference.scipy.org/proceedings/scipy2010/mckinney.html.

[90] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.*, 21(6):1087–1092, 06 1953. https://aip.scitation.org/doi/abs/10.1063/1.1699114.

[91] Naveen Michaud-Agrawal, Elizabeth J. Denning, Thomas B. Woolf, and Oliver Beckstein. MDAnalysis: A toolkit for the analysis of molecular dynamics simulations. *Journal of Computational Chemistry*, 32(10):2319–2327, 2011. https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.21787.

[92] Walter Mickel, Sebastian C. Kapfer, Gerd E. Schröder-Turk, and Klaus Mecke. Shortcomings of the bond orientational order parameters for the analysis of disordered particulate matter. *J. Chem. Phys.*, 138(4):044501, 01 2013. http://aip.scitation.org/doi/full/10.1063/1.4774084.

[93] Gustav Mie. Zur kinetischen Theorie der einatomigen Körper. *Annalen der Physik*, 316(8):657–697, 1903. http://onlinelibrary.wiley.com/doi/abs/10.1002/andp.19033160802.

[94] Marek Mihalkovič and C. L. Henley. Empirical oscillating potentials for alloys from ab initio fits and the prediction of quasicrystal-related structures in the Al-Cu-Sc system. *Phys. Rev. B*, 85(9):092102, 03 2012. https://link.aps.org/doi/10.1103/PhysRevB.85.092102.

[95] R. E. Miles. On Random Rotations in R^3. *Biometrika*, 52(3/4):636–639, 1965. https://www.jstor.org/stable/2333716.

[96] Ryan Gotchy Mullen, Joan-Emma Shea, and Baron Peters. Easy Transition Path Sampling Methods: Flexible-Length Aimless Shooting and Permutation Shooting. *Journal of Chemical Theory and Computation*, 11(6):2421–2428, June 2015.

[97] Christoph Niethammer, Stefan Becker, Martin Bernreuther, Martin Buchholz, Wolfgang Eckhardt, Alexander Heinecke, Stephan Werth, Hans-Joachim Bungartz, Colin W. Glass, Hans Hasse, Jadran Vrabec, and Martin Horsch. Ls1 mardyn: The Massively Parallel Molecular Dynamics Code for Large Systems. *J. Chem. Theory Comput.*, 10(10):4455–4464, 10 2014. https://doi.org/10.1021/ct500169q.

[98] Zihao Ou, Ziwei Wang, Binbin Luo, Erik Luijten, and Qian Chen. Kinetic pathways of crystallization at the nanoscale. *Nat. Mater.*, 19(4):450–455, 04 2020. http://www.nature.com/articles/s41563-019-0514-1.

[99] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8026–8037. Curran Associates, Inc., 2019. http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[100] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. http://jmlr.org/papers/v12/pedregosa11a.html.

[101] Bor Plestenjak and Vladimir Batagelj. Optimal arrangements of n points on a sphere and in a circle. *Proceedings of the 6th International Symposium on Operational Research*, pages 83–87, 2001.

[102] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Sandia National Labs., Albuquerque, NM (United States)*, (SAND-91-1144), 05 1993. https://www.osti.gov/biblio/10176421.

[103] Vyas Ramasubramani, Carl S. Adorf, Paul M. Dodd, Bradley D. Dice, and Sharon C. Glotzer. Signac: A Python framework for data and workflow management. *Proceedings of the 17th Python in Science Conference*, pages 152–159, 2018. https://conference.scipy.org/proceedings/scipy2018/vyas_ramasubramani.html.

[104] Vyas Ramasubramani, Bradley D. Dice, Eric S. Harper, Matthew P. Spellings, Joshua A. Anderson, and Sharon C. Glotzer. Freud: A software suite for high throughput analysis of particle simulation data. *Computer Physics Communications*, page 107275, 03 2020. http://www.sciencedirect.com/science/article/pii/S0010465520300916.

[105] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Using mobile phones to determine transportation modes. *ACM Trans. Sen. Netw.*, 6(2), 3 2010. https://doi.org/10.1145/1689239.1689243.

[106] Sylvia. Richardson and Peter J. Green. On Bayesian Analysis of Mixtures with an Unknown Number of Components (with discussion). *Journal of the Royal Statistical Society: Series B (Methodological)*, 59(4):731–792, 11 1997. https://doi.org/10.1111/1467-9868.00095.

[107] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 11 1987. https://www.sciencedirect.com/science/article/pii/0377042787901257.

[108] C. Patrick Royall, Ard A. Louis, and Hajime Tanaka. Measuring colloidal interactions with confocal microscopy. *J. Chem. Phys.*, 127(4):044507, 07 2007. http://aip.scitation.org/doi/full/10.1063/1.2755962.

[109] Chris H. Rycroft. VORO++: A three-dimensional Voronoi cell library in C++. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 19(4):041111, 10 2009. https://doi.org/10.1063/1.3215722.

[110] Eric S. Harper, Brendon Waters, and Sharon C. Glotzer. Hierarchical self-assembly of hard cube derivatives. *Soft Matter*, 15(18):3733–3739, 2019.

[111] Romelia Salomon-Ferrer, David A. Case, and Ross C. Walker. An overview of the Amber biomolecular simulation package. *WIREs Computational Molecular Science*, 3(2):198–210, 2013. https://onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1121.

[112] Devleena Samanta, Wenjie Zhou, Sasha B. Ebrahimi, Sarah Hurst Petrosko, and Chad A. Mirkin. Programmable matter: The nanoparticle atom and DNA bond. *Advanced Materials*, 34(12):2107875, 2022. https://onlinelibrary.wiley.com/doi/abs/10.1002/adma.202107875.

[113] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "Kneedle" in a Haystack: Detecting Knee Points in System Behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 166–171, 06 2011.

[114] S. S. Schoenholz, E. D. Cubuk, D. M. Sussman, E. Kaxiras, and A. J. Liu. A structural approach to relaxation in glassy liquids. *Nature Phys*, 12(5):469–471, 05 2016. https://www.nature.com/articles/nphys3644.

[115] Jindal K. Shah, Eliseo Marin-Rimoldi, Ryan Gotchy Mullen, Brian P. Keene, Sandip Khan, Andrew S. Paluch, Neeraj Rai, Lucienne L. Romanielo, Thomas W. Rosch, Brian Yoo, and Edward J. Maginn. Cassandra: An open source Monte Carlo package for molecular simulation. *Journal of Computational Chemistry*, 38(19):1727–1739, 2017. https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.24807.

[116] David E. Shaw, Ron O. Dror, John K. Salmon, J. P. Grossman, Kenneth M. Mackenzie, Joseph A. Bank, Cliff Young, Martin M. Deneroff, Brannon Batson, Kevin J. Bowers, Edmond Chow, Michael P. Eastwood, Douglas J. Ierardi, John L. Klepeis, Jeffrey S. Kuskin, Richard H. Larson, Kresten Lindorff-Larsen, Paul Maragakis, Mark A. Moraes, Stefano Piana, Yibing Shan, and Brian Towles. Millisecond-scale molecular dynamics simulations on Anton. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 1–11. Association for Computing Machinery, 11 2009. https://doi.org/10.1145/1654059.1654126.

[117] Zunya Shi and Abdallah Chehade. A dual-lstm framework combining change point detection and remaining useful life prediction. *Reliability Engineering & System Safety*, 205:107257, 2021. https://www.sciencedirect.com/science/article/pii/S0951832020307572.

[118] Masato Shimono and Hidehiro Onodera. Dynamics and Geometry of Icosahedral Order in Liquid and Glassy Phases of Metallic Glasses. *Metals*, 5(3):1163–1187, September 2015.

[119] Aldo Spatafora-Salazar, Dana M Lobmeyer, Lucas HP Cunha, Kedar Joshi, and Sibani Lisa Biswal. Hierarchical assemblies of superparamagnetic colloids in time-varying magnetic fields. *Soft Matter*, 17(5):1120–1155, 2021.

[120] Matthew Spellings and Sharon C. Glotzer. Machine learning for crystal identification and discovery. *AIChE Journal*, 64(6):2198–2206, 2018. https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.16157.

[121] Matthew Spellings, Ryan L. Marson, Joshua A. Anderson, and Sharon C. Glotzer. GPU accelerated Discrete Element Method (DEM) molecular dynamics for conservative, faceted particle simulations. *Journal of Computational Physics*, 334:460–467, 04 2017. http://www.sciencedirect.com/science/article/pii/S0021999117300244.

[122] Paul J. Steinhardt, David R. Nelson, and Marco Ronchetti. Bond-orientational order in liquids and glasses. *Phys. Rev. B*, 28(2):784–805, 07 1983. https://link.aps.org/doi/10.1103/PhysRevB.28.784.

[123] John Edward Stone. An efficient library for parallel ray tracing and animation. *University of Missouri, Thesis*, 1998. http://jedi.ks.uiuc.edu/~johns/tachyon/papers/thesis.pdf.

[124] Alexander Stukowski. Visualization and analysis of atomistic simulation data with OVITO-the Open Visualization Tool. *Modelling Simul. Mater. Sci. Eng.*, 18(1):015012, 12 2009. https://doi.org/10.1088%2F0965-0393%2F18%2F1%2F015012.

[125] P. M. L. Tammes. On the origin of number and arrangement of the places of exit on the surface of pollen-grains. *Recueil des travaux botaniques néerlandais*, 27(1):1–84, 01 1930. https://natuurtijdschriften.nl/pub/552640.

[126] Pieter Rein ten Wolde, Maria J. Ruiz-Montero, and Daan Frenkel. Numerical Evidence for bcc Ordering at the Surface of a Critical fcc Nucleus. *Phys. Rev. Lett.*, 75(14):2714–2717, 10 1995. https://link.aps.org/doi/10.1103/PhysRevLett.75.2714.

[127] Stephen Thomas, Monet Alberts, Michael M Henry, Carla E Estridge, and Eric Jankowski. Routine million-particle simulations of epoxy curing with dissipative particle dynamics. *J. Theor. Comput. Chem.*, 17(03):1840005, 05 2018. https://www.worldscientific.com/doi/abs/10.1142/S0219633618400059.

[128] Matthew W. Thompson, Justin B. Gilmer, Ray A. Matsumoto, Co D. Quach, Parashara Shamaprasad, Alexander H. Yang, Christopher R. Iacovella, Clare McCabe, and Peter T. Cummings. Towards molecular simulations that are transparent, reproducible, usable by others, and extensible (TRUE). *Molecular Physics*, 118(9-10):e1742938, 06 2020. https://doi.org/10.1080/00268976.2020.1742938.

[129] Matthew W. Thompson, Ray Matsumoto, Robert L. Sacci, Nicolette C. Sanders, and Peter T. Cummings. Scalable Screening of Soft Matter: A Case Study of Mixtures of Ionic Liquids and Organic Solvents. *J. Phys. Chem. B*, 123(6):1340–1347, 02 2019. https://doi.org/10.1021/acs.jpcb.8b11527.

[130] Charles Truong, Laurent Oudre, and Nicolas Vayatis. Selective review of offline change point detection methods. *Signal Processing*, 167:107299, 2020. https://www.sciencedirect.com/science/article/pii/S0165168419303494.

[131] Stefan van der Walt, S. Chris Colbert, and Gael Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering*, 13(2):22–30, 03 2011.

[132] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J.

Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nat Methods*, 17(3):261–272, 03 2020. https://www.nature.com/articles/s41592-019-0686-2.

[133] Georges Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les parallélloèdres primitifs. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1908(134):198–287, 07 1908. https://www.degruyter.com/document/doi/10.1515/crll.1908.134.198/html.

[134] Georges Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. premier mémoire. sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1908(133):97–102, 01 1908. https://www.degruyter.com/document/doi/10.1515/crll.1908.133.97/html.

[135] Florian Weik, Rudolf Weeber, Kai Szuttor, Konrad Breitsprecher, Joost de Graaf, Michael Kuron, Jonas Landsgesell, Henri Menke, David Sean, and Christian Holm. ESPResSo 4.0 - an extensible software package for simulating soft matter systems. *Eur. Phys. J. Spec. Top.*, 227(14):1789–1816, 03 2019. https://doi.org/10.1140/epjst/e2019-800186-9.

[136] Eugene P. Wigner. *Group Theory: And Its Application to the Quantum Mechanics of Atomic Spectra*. Elsevier, September 2013. Google-Books-ID: UITNCgAAQBAJ.