Complex Crystallization Pathways Analyzed in a Continuous Feature Space

by

Bradley Dice

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Physics and Scientific Computing)
in The University of Michigan
2021

Bradley Dice

bdice@umich.edu

ORCID iD: 0000-0002-9983-0770

Dedicated to my family,
David, Pam, Jacob, Carman, my grandparents,
and the many educators and mentors who have supported me
in finding my way to (and through) graduate school.

ACKNOWLEDGEMENTS

During the course of performing research and writing this dissertation, I have received immeasurable personal and professional support from my advisor Sharon C. Glotzer, colleagues, mentors, friends, and family. I offer my sincere thanks to all who have helped me take each step in this journey.

First, I would like to recognize my advisor Sharon C. Glotzer. Your scientific vision and enthusiasm for discovery have inspired me to pursue hard questions and push boundaries in all of my endeavors. It has been a true privilege to work with you and share in the multitude of ideas that have shaped both my research and personal growth through my doctoral studies. I additionally thank the thesis committee, Bryan Goldsmith, Xiaoming Mao, Ashwin Shahani, and Kai Sun, for their time and support in reviewing this dissertation.

I have been fortunate to have amazing colleagues and mentors within the Glotzer group. I would like to thank Carl Simon Adorf, Vyas Ramasubramani, Rose Cersonsky, Matthew Spellings, James Antonaglia, Brandon Butler, Chrisy Du, Kelly Wang, Corwin Kerr, Tommy Waltmann, and Tobias Dwyer for your support as peers, co-authors, and friends. To Shannon Moran, I offer special appreciation for your extensive mentorship in scientific communication and many career chats along the way. To Joshua Anderson, Jens Glaser, Timothy Moore, Julia Dshemuchadse, and Domagoj Fijan, I thank you for your leadership as research scientists and post-doctoral researchers, and the numerous contributions you made to my development as a computational scientist. Karen Coulter, your support keeps our lab afloat, and I have been thankful for your guidance along every part of my journey at U-M. Thanks also to Yezhi Jin for her amazing work as an undergraduate mentee in the early stages of my crystallization pathways research.

Special thanks go to the team with whom I have collaborated on the **signac** project. In addition to members of the Glotzer group Carl Simon Adorf, Vyas Ramasubramani, Brandon Butler, Tim Moore, and Pengji Zhou, it has been a privilege to work with Alyssa Travitz, Mike M. Henry, and Google Summer of Code mentee Hardik Ojha as each of you have brought so much of yourselves to the project. It is amazing to work with a team like this, where a common passion has enabled us to build a community of scientists with many talents and a shared commitment to helping one another.

Thanks to the many friendly physicists who I've gotten to know during my (pre-pandemic) time in Ann Arbor – Noah Steinberg, Harry Liu, Elizabeth Drueke, Matthew Day, Alec Kirkley, Dan McCusker, and Torben Purz, it has been a joy to share living spaces, lunchtimes, and band practices with you. Adarash Mishra and Annie Murphey, your friendship made every day in Ann Arbor exciting, and I am thankful to have you in my life. I thank the optiMize staff and fellows, the DJs of MEMCO, the band (Funktorial!/Local Optimist Club), and the REACT team for opportunities to learn, grow, and play within caring communities.

I offer my gratitude to my mentors from William Jewell College: Patrick Bunton, Maggie Sherer, Blane Baker, Jason Morrill, Lori Wetmore, Erin Martin, Azadeh Rafizadeh, and Landon Young, all of whom saw potential in me and helped me to develop as a scientist, scholar, and citizen throughout my undergraduate studies. Finally, I thank Rosemary Camp and Ken Kramme, who sparked in me a passion for the intersection of physics and chemistry that has inspired every one of my academic endeavors since.

During my PhD, I have been supported by two fellowships, first by National Science Foundation Graduate Research Fellowship Grant DGE 1256260 (2016–2019), and subsequently by a Molecular Sciences Software Institute Investment Fellowship (2019–2021), from the National Science Foundation under ACI 1547580, S212: Impl: The Molecular Sciences Software Institute [1, 2]. I thank the staff of the Molecular Sciences Software Institute, particularly Doaa Altarawy, Jessica Nash, and Daniel G. Smith, for their guidance on professionally designing and building community around the open-source scientific software projects **freud** and **signac**. I also thank the University of Michigan's Michigan Institute for Computational Discovery and Engineering (MICDE) for fellowship funding that provided computing resources used in this research.

TABLE OF CONTENTS

DEDICATION . . . . **ii**

ACKNOWLEDGEMENTS . . . . **iii**

LIST OF FIGURES . . . . **ix**

LIST OF APPENDICES . . . . **xvi**

ABSTRACT . . . . **xviii**

CHAPTER

I. Introduction 1

1.1 Why Simulations? 1
1.2 What is a Pathway? 2

II. Crystallization Pathways: Theory and Methods 6

2.1 Structural Descriptors and Order Parameters 10
2.2 Minkowski Structure Metrics 12
2.2.1 Steinhardt Order Parameter q_l, w_l 13
2.2.2 Averaged Steinhardt Order Parameter $\overline{q}_l, \overline{w}_l$ 14
2.2.3 Minkowski Structure Metrics q_l', w_l' 15
2.2.4 Properties of Minkowski Structure Metrics 16
2.3 Order Parameters Using Machine Learning 17
2.3.1 The Descriptor Reduction Paradigm 18
2.3.2 Feature Engineering 19
2.3.3 The Importance of Model Architecture 20
2.3.4 Machine Learning Methods for Structure Detection 20
2.3.5 Choosing Neighbors, Features, and ML Models 24
2.4 The Continuous Topological Order Parameter (CTOP) 27
2.5 UMAP Dimensionality Reduction 29
2.5.1 Data Density on the CTOP Manifold 30
2.5.2 Mathematical Foundations of UMAP 31
2.5.3 Example: UMAP Applied to MNIST Images 33

III. Crystallization Pathways: Results 35

3.1 Self-Assembly of Hard Polyhedra 35
3.1.1 Hard Particle Monte Carlo Simulation Protocol 35

3.1.2 Self-Assembly of Hard Polyhedra into fcc 36
3.1.3 Self-Assembly of Hard Polyhedra into bcc 40
3.1.4 Self-Assembly of Hard Polyhedra into sc 42
3.2 Self-Assembly of Complex Structures 43
3.2.1 A15 Structure ($cP8$-Cr_3Si) 45
3.2.2 Diamond ($cF8$-C) 48
3.2.3 σ Phase ($tP30$-CrFe) 51
3.2.4 Decagonal Quasicrystal 53
3.2.5 β-Mn Structure 55
3.3 Comparisons of Pathways 57
3.3.1 Monte Carlo and Molecular Dynamics Reveal Similar Paths 58
3.3.2 Comparing fcc, bcc, and sc Structures 59
3.3.3 Solid-Solid Transitions 60
3.3.4 Comparing Many Pathways 61
3.4 Outlook and Future Directions 64
3.4.1 Experimental Applications of the CTOP Method 64
3.4.2 Local Symmetry Analysis with Point Groups 65
3.4.3 Applications to 2D Systems 66
3.4.4 Handling Particle Shape with Set Voronoi Constructions 66
3.4.5 Minkowski Tensors and Continuous Morphometric Measures 67
3.4.6 Alternate Unsupervised Machine Learning Approaches 70
3.4.7 $\mathbb{E}^3$ Equivariant Neural Networks 71

IV. Predicting Properties of Photonic Crystals with Machine Learning 72

4.1 Photonic Materials and Machine Learning 72
4.1.1 Electromagnetic Waves in Heterogeneous Macroscopic Media . . . 73
4.1.2 Eigensolver Methods for Computing Photonic Bands 74
4.2 Past Approaches to Photonic Structure Design 74
4.3 Designing Convolutional Neural Networks for Predicting Photonic Properties 75
4.3.1 Data Preprocessing and Augmentation 75
4.3.2 Building a Scalable Model 77
4.3.3 Iterations on CNN Model Architecture 79
4.4 Computing the Photonic Density of States 81
4.4.1 $\mathbb{E}^3$ Equivariant Models with Graph Convolutional Networks 84
4.5 Database of Photonic Crystal Structures 86
4.6 Outlook and Future Directions 87

V. The freud Library for Particle Analysis 89

5.1 Introduction 89
5.2 Library Design and Features 90
5.3 Machine Learning Pipelines 93
5.4 Performance and Real-Time Visualization 93
5.5 Wide-Ranging Applications of Data Analysis in Particle Systems 94
5.6 Outlook and Future Directions 95

VI. The signac Framework for Data Management and Workflow Automation 97

6.1 Introduction 97
6.2 Data Model and Integration with Scientific Applications 98
6.3 HDF5 Data Stores 100
6.4 Improving Performance and Scalability 102
6.5 signac-dashboard: Visualizing Research Data 105

6.5.1 Dashboard Features 105
6.6 Outlook and Future Directions 108

VII. Conclusions 110

APPENDICES **115**

BIBLIOGRAPHY **162**

LIST OF FIGURES

Figure

2.1 Visualization of the Voronoi polyhedra for an fcc system. The Voronoi cells of this noisy fcc structure are slightly distorted rhombic dodecahedra. This rendering uses plato [3] and pythreejs [4]. 15

2.2 Overview of machine learning algorithms. Figure adapted from the documentation of scikit-learn [5]. 17

2.3 Many machine learning approaches for quantifying order in particle systems follow a paradigm where a high-dimensional feature space is constructed through the computation of descriptors for each particle and then reduced into a low-dimensional representation such as a class identifier. 18

2.4 Common neighbor definitions for the particle shown in red. Neighbors are shown in green. Figure from Ref. [6] with permission. **(a)** The Voronoi polygon for the particle is shaded red. The edges in bold red define which particles are neighbors. The blue bonds form the dual of the Voronoi diagram, called the Delaunay graph. **(b)** Neighbors found using the Voronoi diagram. Note that the furthest neighbor has a small edge in the Voronoi diagram in (a), so its neighbor weight is small. **(c)** Neighbors found using a cutoff radius $r_{\max}$. **(d)** Neighbors found using k-nearest neighbors, $k = 6$. 25

2.5 The variants of the Continuous Topological Order Parameter (CTOP). 27

2.6 The CTOP order parameter is a continuous function of particle positions, and can be mapped continuously into a low-dimensional representation using UMAP. . . . 29

2.7 **(a)** Sample 28×28 pixel images of digits 3, 1, 4 from the MNIST data set. **(b)** UMAP embedding of the MNIST data set into two dimensions. **(c)** UMAP embedding of the MNIST data set into three dimensions. Visualization generated using Plotly [7]. 34

3.1 **(a)** A 2D UMAP embedding of the 6-dimensional $CTOP_q$ self-assembly pathway of 15,625 hard icosahedra crystallizing from a fluid into the fcc structure $cF4$-Cu. Each point in the scatter plot corresponds to the $CTOP_q$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from fluid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. **(b)** A 2D UMAP embedding of the 6-dimensional $CTOP_q$ as shown in (a), colored by a reference analysis method, Polyhedral Template Matching, computed by OVITO [8, 9]. This demonstrates that clusters of the $CTOP_q$ embedding correspond to local environments, with the one side of the image predominantly marked as "Other" (fluid particles) while the other side is predominantly marked as fcc (solid particles). The hcp defects, shown in red, cluster in an area slightly away from the fcc part of the manifold, because hcp particles share many (but not all) symmetries with fcc particles. **(c)** The reference analysis method, Polyhedral Template Matching, shows the emergence of particles in an hcp local environment, followed by the formation of a predominantly fcc phase. The black vertical line indicates the step at which the target packing fraction of $\phi = 0.52$ was reached. See (b) for the structure corresponding to each color. **(d)** Snapshot of the final frame of the simulation. Particles are colored by their environment type (fcc-like in green, hcp-like in red, bcc-like in blue). The simulation protocol is described in Section 3.1.1. The system is slowly compressed to a target packing fraction over the course of the simulation. 37

3.2 **(a)** The $CTOP_q$ embedding of icosahedra self-assembling fcc at a target packing fraction of $\phi = 0.54$. **(b)** The $CTOP_{\overline{qw}}$ embedding shows three clusters: two fcc clusters and one hcp cluster. **(c)** The self-assembly progress as a function of time. Note that zero particles show fluid-like order in their second shell in the final frame. This demonstrates how the addition of second-shell order can assist in separating solid and fluid phases in feature space. **(d)** Snapshot of the final frame of the simulation, which shows hcp stacking faults in the fcc crystal. 39

3.3 **(a)** A 2D UMAP embedding of the 6-dimensional $CTOP_q$ self-assembly pathway of 15,625 hard cuboctahedra crystallizing from a fluid into the bcc structure $cI2$-W. Each point in the scatter plot corresponds to the $CTOP_q$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from fluid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. **(b)** UMAP embedding of $CTOP_q$, colored by a reference analysis method, Polyhedral Template Matching, computed by OVITO [8, 9]. Here, the particles identified as fcc and hcp environments persist into the final frame shown in (d). **(c)** The reference analysis method, Polyhedral Template Matching, shows the emergence of particles in an hcp local environment, followed by the formation of a predominantly fcc phase. The black vertical line indicates the step at which the target packing fraction of $\phi = 0.63$ was reached. See (b) for the structure corresponding to each color. **(d)** Snapshot of the final frame of the simulation. See (b) for the structure corresponding to each color. The simulation protocol is described in Section 3.1.1. The system is slowly compressed to a target packing fraction over the course of the simulation. 41

3.4 **(a)** Number of particles in each cluster of the $CTOP_q$ feature space. The trend for environment 0 (blue) closely follows the number of bcc-like particles identified by Polyhedral Template Matching in Figure 3.3(c). **(b)** A rendering of the final simulation frame, colored by clusters in the $CTOP_q$ feature space. The particles in environment 1 (orange) correspond to the same regions as the fcc, hcp, and "other" particles seen in Figure 3.3(d). 42

3.5 Self-assembly of cubes into a simple cubic structure at a packing fraction of 0.61. **(a)** UMAP embedding of $CTOP_q$, colored by Polyhedral Template Matching. **(b)** UMAP embedding of $CTOP_{\overline{qw}}$, colored by Polyhedral Template Matching. **(c)** Counts of all particles in the solid cluster of (b), and counts of the largest contiguous cluster of solid-like particles. See text for details. **(d)** Snapshot of the final frame of the simulation. 43

3.6 **(a)** UMAP embedding of the 9-dimensional $CTOP_{qw}$ self-assembly pathway of 15,625 particles crystallizing from a fluid into the A15 structure $cP8$-Cr_3Si. Each point in the scatter plot corresponds to the $CTOP_{qw}$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from liquid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. Each type of local environment present in the final structure (Cr-like and Si-like) appears as a cluster in the embedding. **(b)** Snapshot of the final frame of the simulation. The inset shows the A15 unit cell [10]. Particles are colored by their environment type cluster, shown in (c) (Cr-like in blue, Si-like in orange). The defect and liquid-like particle environments (colored green) form a grain boundary. **(c)** Clustering by $CTOP_{qw\overline{qw}}$ readily shows the dominant clusters' local environments. The inset shows the $CTOP_{qw\overline{qw}}$ feature space. **(d)** This shows the counts of each cluster in $CTOP_{qw\overline{qw}}$ as a function of time. The Cr and Si environments appear at the same time, and are in roughly a 3:1 proportion as is expected from the chemical formula of the prototype structure. 45

3.7 **(a)** UMAP embedding of the 18-dimensional $CTOP_{qw\overline{qw}}$ self-assembly pathway of 15,625 particles crystallizing from a fluid into the diamond structure $cF8$-C. Each point in the scatter plot corresponds to the $CTOP_{qw\overline{qw}}$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from liquid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. **(b)** Snapshot of the final frame of the simulation. Particles are colored by their environment type cluster, shown in (c). The dominant environment 0 particles have not crystallized to the $cF8$-C local structure, described further in the text. The diamond-like environment 1 is colored orange. The fluid-like particle environment 2 is colored green. **(c)** Clustering by $CTOP_{qw\overline{qw}}$ readily shows the dominant clusters' local environments. The inset shows the $CTOP_{qw\overline{qw}}$ feature space. **(d)** This shows the counts of each cluster in $CTOP_{qw\overline{qw}}$ as a function of time. The Cr and Si environments appear at the same time, and are in roughly a 3:1 proportion as is expected from the chemical formula of the prototype structure. 48

3.8 **(a)** 3D UMAP embedding of the 18-dimensional $CTOP_{qw\overline{qw}}$ self-assembly pathway of 15,625 particles crystallizing from a fluid into the diamond structure $cF8$-C. Each point in the scatter plot corresponds to the $CTOP_{qw\overline{qw}}$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from liquid (purple and dark blue-green) to solid (light green and yellow). Visualization generated using Plotly [7]. **(b)** Another view of the embedding in (a). 50

3.9 **(a)** UMAP embedding of the 9-dimensional $CTOP_q$ self-assembly pathway of 15,625 particles crystallizing from a fluid into the structure $tP30$-CrFe. Each point in the scatter plot corresponds to the $CTOP_q$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from liquid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. **(b)** Snapshot of the final frame of the simulation. Particles are colored by their environment type cluster, shown in (c) and described in the text. **(c)** Clustering by $CTOP_{qw\overline{qw}}$ readily shows the dominant clusters' local environments. The inset shows the $CTOP_{qw\overline{qw}}$ feature space. **(d)** This shows the counts of each cluster in $CTOP_{qw\overline{qw}}$ as a function of time. 52

3.10 **(a)** UMAP embedding of the 12-dimensional $CTOP_{\overline{qw}}$ self-assembly pathway of 15,625 particles crystallizing from a fluid into the structure $tP30$-CrFe. Each point in the scatter plot corresponds to the $CTOP_q$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from liquid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. **(b)** Snapshot of the final frame of the simulation. Particles are colored by their environment cluster. Solid-like particles in environment 0 are blue. Fluid-like and defect particles in environment 1 are orange. **(c)** Clustering by $CTOP_{\overline{qw}}$ distinguishes fluid from solid, but cannot identify the Wyckoff positions or their local symmetries. **(d)** The $CTOP_q$ embedding previously shown in Figure 3.9, colored by clusters in the $CTOP_{\overline{qw}}$ space. **(e)** The counts of each cluster in $CTOP_{\overline{qw}}$ as a function of time, showing the fluid to crystal transition with only a small number of fluid-like particles remaining as defects in the crystal structure shown in (b). 54

3.11 UMAP embedding of a decagonal quasicrystal self-assembling. Simulation data provided by Kelly Wang. 55

3.12 **(a)** Snapshot of the self-assembled β-Mn structure formed by 15,625 truncated dodecahedra. Particles are colored arbitrarily for high contrast. The bond order diagram in the upper left and radial distribution function in the lower left confirm that the obtained structure is β-Mn. **(b)** UMAP embedding of the 18-dimensional $CTOP_{qw\overline{qw}}$ self-assembly pathway of 15,625 particles crystallizing from a fluid into the β-Mn structure. Each point in the scatter plot corresponds to the $CTOP_{qw\overline{qw}}$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from liquid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. **(c)** The $CTOP_{qw\overline{qw}}$ UMAP embedding, colored by Polyhedral Template Matching. 56

3.13 A co-embedding of four different simulations transitioning between fluid and fcc structures. The simulation times are normalized such that the dark purple color is the initial state and light yellow is the final state. **(a)** UMAP of the hard icosahedra compressed to $\phi = 0.52$, previously shown in Figure 3.1. **(b)** UMAP of the hard icosahedra compressed to $\phi = 0.54$, previously shown in Figure 3.2. **(c)** UMAP of WCA spheres at $kT = 0.03$. WCA self-assembly simulation data provided by Allen LaCour. **(d)** A simulation of hard spheres initialized in a close-packed fcc structure and slowly expanded, allowing the solid to melt into a fluid. Note that the color scheme is reversed for this simulation because it begins in an fcc solid (dark purple) and melts to the fluid (yellow). 58

3.14 A co-embedding of three different simulations in the $CTOP_q$ feature space. The simulation times are normalized such that the lighter color is the initial state (fluid) and the darker color is the final state (solid). Colors are chosen arbitrarily to help identify the data in the co-embedding. **(a)** UMAP of the hard icosahedra compressed to $\phi = 0.52$, previously shown in Figure 3.1. **(b)** UMAP of the hard cuboctahedra compressed to $\phi = 0.63$, previously shown in Figure 3.3. **(c)** UMAP of the hard cubes compressed to $\phi = 0.61$, previously shown in Figure 3.5. **(d)** The co-embedding with all simulations shown on the same axes. 59

3.15 A co-embedding of three different simulations in the $CTOP_{qw}$ feature space. Color schemes are chosen arbitrarily to help identify the data in the co-embedding. The average UMAP value for each frame is shown in a series of circles colored by the frame time. **(a)** UMAP of the hard icosahedra compressed to $\phi = 0.52$, previously shown in Figure 3.1. **(b)** UMAP of the hard cuboctahedra compressed to $\phi = 0.63$, previously shown in Figure 3.3. **(c)** UMAP of a solid-solid phase transition (fcc to bcc). Data provided by Chrisy Xiyu Du [11]. **(d)** The co-embedding with all simulations shown on the same axes. The UMAP averages of each trajectory form a triangle for the fluid, fcc, and bcc states. 60

3.16 A co-embedding of 10 different simulations in the $CTOP_q$ feature space. The subfigures show the pathway embeddings of 8 of the simulations (selected as representative trajectories of the whole). Clockwise from top, the embedding shows local environments from the A15 ($cP8$-Cr_3Si), σ-CrFe ($tP30$-CrFe), fcc ($cF4$-Cu) and hcp ($hP2$-Mg), bcc ($cI2$-W), sc ($cP1$-Po), clathrate-I ($cP54$-K_4Si_{23}), and diamond ($cF8$-C) crystal structures. Color schemes are chosen arbitrarily to help identify the data in the co-embedding. Some trajectories shown were provided by Carl Simon Adorf, Sangmin Lee, and Allen LaCour. 62

3.17 Illustration of the linearly independent Minkowski tensors. **(a)** In two dimensions, there are four linearly independent Minkowski tensors. **(b)** In three dimensions there are six. Taken from Ref. [12] with permission. 68

4.1 The flow of data from an input crystal structure to a predicted photonic band structure. The dielectric image $\varepsilon(\mathbf{r})$ is generated from the input crystal structure and processed by a deep neural network (the figure shows a simplified model) to predict the band frequencies $\omega_n(\mathbf{k}, \varepsilon(\mathbf{r}))$ for each band number n in the first 20 bands and each wavevector $\mathbf{k}$ in the band path over the Irreducible Brillouin Zone (IBZ). 76

4.2 Benchmark on Summit (Oak Ridge National Laboratory) for scaling the training of a deep CNN machine learning model. Shown is time to train our model for 1 epoch vs. number of nodes. The strong scaling is nearly ideal, up to the tested limit of 1024 nodes (6144 NVIDIA V100 GPUs in total). 77

4.3 The MPB calculated photonic band structure (left) compared with the band structure prediction of our trial ML model, labeled cnn2 (right). The band structure above was predicted after training the model on the XSEDE Bridges GPU-AI partition for 50 epochs on a dataset consisting of only 127 sample structures using 10 random rotations. The dielectric structure is diamond ($cF8$-C) with particles at the lattice sites having a radius of 0.2 times the length of a unit cell. The trial model learns curvature near the gamma point, but otherwise converges to the average band frequency value over all $\mathbf{k}$ vectors. 79

4.4 Densities of states computed for two crystal structures exhibiting band gaps. **(a)** The band structure and density of states for the diamond structure ($cF8$-C) with dielectric sphere radius $r = 0.24$. **(b)** The band structure and density of states for the β-cristobalite structure ($cF24$-SiO_2) with dielectric sphere radius $r = 0.24$. . . 83

4.5 Histogram of ω_{max} for the 2,400 crystal structures in space group 227 whose densities of states were computed. 83

4.6 Screenshots of the photonics database online, produced to accompany Ref. [13]. . 86

5.1 Common Python tools for simulation analysis at varying length scales. The **freud** library is designed for nanoscale systems, such as colloidal crystals and nanoparticle assemblies. In such systems, interactions are described by coarse-grained models where particles' atomic constituents are often irrelevant and particle anisotropy (non-spherical shape) is common, thus requiring a generalized concept of particle "types" and orientation-sensitive analyses. These features contrast the assumptions of most analysis tools designed for biomolecular simulations and materials science. 91

5.2 Overview of data visualization tools that can be coupled with the **freud** library. **(a)** Interactive visualization of a Lennard-Jones particle system, rendered in a Jupyter notebook using plato [3] with the pythreejs backend [4]. **(b)** Hard tetrahedra colored by local density, path traced with fresnel [14]. **(c)** A crystalline grain identified using **freud**'s `LocalDensity` module and cut out for display using OVITO [9]. The image shows a $tP30$-CrFe structure formed from an isotropic pair potential optimized to generate this structure [15]. 93

5.3 Cluster analysis of buildings in three American cities, revealing different levels of city "crystallinity." Figure reproduced from Ref. [16] under the Creative Commons Attribution International 4.0 License. 94

6.1 Overview of the **signac** framework. Users first create a project, which initializes a workspace directory on disk. Users define state points which are dictionaries that uniquely identify a job. The workspace holds a directory for each job, containing JSON files that store the state point and job document. The job directory name is a hash of the state point's contents. Here, the `init.py` file initializes an empty project and adds one job with state point {`"a": 1`}. Next, users define a workflow using a subclass of **signac-flow**'s `FlowProject`. The workflow shown has three operations (simulate, analyze, visualize) that, when executed, produce two new files `results.txt` and `plot.png` in the job directory. Special thanks to Kelly Wang for contributing the design and concept of this figure. 99

6.2 Aggregation, groups, and bundling allow users to build complex workflows. The features are orthogonal, and can be used in any combination. Aggregation enables one operation or group to act on multiple jobs. Groups allow users to combine multiple operations into one, with dependencies among operations resolved at run time. Bundling helps users efficiently leverage HPC schedulers by submitting multiple commands in the same script, to be executed in serial or parallel. 103

6.3 An example of **signac-dashboard** showing the image viewer modules with data collected for Chapter III. 106

6.4 Additional examples of **signac-dashboard**. **(a)** Custom job titles are shown for each job, generated by a overridden `job_title` function. **(b)** The `StatepointList` module shows the keys and values for the job state point in the upper left. 108

7.1 "Dependency" by Randall Munroe. Alternate text: "Someday ImageMagick will finally break for good and we'll have a long period of scrambling as we try to reassemble civilization from the rubble." Reproduced under a Creative Commons Attribution-NonCommercial 2.5 License. Source: https://xkcd.com/2347/ 112

A.1 Common Python tools for simulation analysis at varying length scales. The **freud** library is designed for nanoscale systems, such as colloidal crystals and nanoparticle assemblies. In such systems, interactions are described by coarse-grained models where particles' atomic constituents are often irrelevant and particle anisotropy (non-spherical shape) is common, thus requiring a generalized concept of particle "types" and orientation-sensitive analyses. These features contrast the assumptions of most analysis tools designed for biomolecular simulations and materials science. 117

A.2 Comparison of runtime for neighbor finding algorithms in **freud** and SciPy for varied system sizes. See text for details. 121

A.3 Histogram of the Steinhardt Q_6 order parameter for 4000 particles in simple cubic, body-centered cubic, and face-centered cubic structures with added Gaussian noise. 124

A.4 UMAP of particle descriptors computed for simple cubic, body-centered cubic, and face-centered cubic structures of 4000 particles with added Gaussian noise. The particle descriptors include Q_l for $l \in \{4, 6, 8, 10, 12\}$. Some noisy configurations of bcc can be confused as fcc and vice versa, which accounts for the small number of errors in the support vector machine's test classification. 127

A.5 Interactive visualization of a Lennard-Jones particle system, rendered in a Jupyter notebook using `plato` with the `pythreejs` backend. 128

A.6 Hard tetrahedra colored by local density, path traced with `fresnel`. 130

A.7 A crystalline grain identified using **freud**'s `LocalDensity` module and cut out for display using OVITO. The image shows a $tP30$-CrFe structure formed from an isotropic pair potential optimized to generate this structure [15]. 132

B.1 Overview of the **signac** framework. Users first create a project, which initializes a workspace directory on disk. Users define state points which are dictionaries that uniquely identify a job. The workspace holds a directory for each job, containing JSON files that store the state point and job document. The job directory name is a hash of the state point's contents. Here, the `init.py` file initializes an empty project and adds one job with state point {`"a": 1`}. Next, users define a workflow using a subclass of **signac-flow**'s `FlowProject`. The workflow shown has three operations (simulate, analyze, visualize) that, when executed, produce two new files `results.txt` and `plot.png` in the job directory. Special thanks to Kelly Wang for contributing the design and concept of this figure. 136

B.2 Aggregation, groups, and bundling allow users to build complex workflows. The features are orthogonal, and can be used in any combination. Aggregation enables one operation or group to act on multiple jobs. Groups allow users to combine multiple operations into one, with dependencies among operations resolved at run time. Bundling helps users efficiently leverage HPC schedulers by submitting multiple commands in the same script, to be executed in serial or parallel. 147

LIST OF APPENDICES

Appendix

A. Analyzing Particle Systems for Machine Learning and Data Visualization with freud 116

A.1 Abstract . . . 116
A.2 Introduction . . . 117
A.2.1 Data Pipelines . . . 119
A.3 Performance and Integrability . . . 120
A.4 Machine Learning . . . 123
A.5 Visualization . . . 126
A.5.1 plato . . . 128
A.5.2 fresnel . . . 129
A.5.3 OVITO . . . 131
A.6 Conclusions . . . 132
A.7 Getting **freud** . . . 133
A.8 Acknowledgments . . . 133

B. signac: Data Management and Workflows for Computational Researchers . . . 135

B.1 Abstract . . . 135
B.2 Introduction . . . 136
B.3 Structure and Implementation . . . 138
B.4 Applications of signac . . . 140
B.5 Overview of New Features . . . 142
B.5.1 Data Archival . . . 142
B.5.2 Improved Data Storage, Retrieval, and Integrations . . . 143
B.5.3 Data Visualization and Integrations . . . 144
B.5.4 Performance Enhancements . . . 145
B.5.5 Improved User Output . . . 146
B.5.6 Enhanced Workflows . . . 146
B.6 Executing Complex Workflows via Groups and Aggregation . . . 147
B.6.1 Groups . . . 149
B.6.2 Aggregation . . . 150
B.6.3 Bundling . . . 151
B.6.4 Cluster Templates . . . 152
B.7 Synced Collections: Backend-Agnostic, Persistent, Mutable Data Structures 153
B.7.1 Motivation . . . 153
B.7.2 Summary of Features . . . 154
B.7.3 Applications of Synced Collections . . . 155
B.8 Project Evolution . . . 157
B.9 Conclusions . . . 159

B.10 Installing signac 159
B.11 Acknowledgments 160
B.12 Author Contributions 160

ABSTRACT

The ability to engineer the kinetic and thermodynamic processes by which nanoparticles and colloids form crystals would open new possibilities for materials design. Simulations, coupled with data science and machine learning, can open new frontiers towards obtaining this kind of control over matter at the nanoscale. We need novel approaches in theory and computation to bridge the perspectives of forward design (predicting the properties of a material from its components and their interactions) and inverse design (predicting components and interactions that produce a desired set of material properties). In Chapter I, I provide background about materials design via self-assembly and outline the questions I address in this dissertation.

In Chapter II, I present mathematical and physical motivation for a new structural descriptor, the Continuous Topological Order Parameter (CTOP), for analyzing crystallization pathways during self-assembly from a microscopic (particle-local) perspective. The CTOP is comprised of Minkowski Structure Metrics and captures continuous deformations in particles' local environments, which we express as a high-dimensional feature space. This space is coupled with the topology-preserving Uniform Manifold Approximation and Projection (UMAP) dimensionality reduction algorithm to produce a continuous mapping from particles' local environments into interpretable self-assembly pathways.

In Chapter III, I apply the CTOP method to a wide range of nanoparticle systems undergoing self-assembly. I directly investigate the CTOP manifold to illuminate aspects of the crystallization process in many types of simple and complex crystals. Hard Particle Monte Carlo and molecular dynamics simulations of pair potentials are analyzed, resulting in a diverse array of self-assembled crystal structures with unit cells ranging from one to 54 particles. I apply unsupervised learning methods to the pathways revealed by the CTOP method, enabling identification of particles' local environments in self-assembling systems including Frank-Kasper phases and decagonal quasicrystals. Next, I show comparisons of pathways with this method, including solid-solid phase transitions. The chapter closes with a discussion of this method's implications for the study of self-assembly, and an outlook on how continuous feature spaces may inspire future analyses and engineering applications.

Chapter IV presents a study of machine learning applied to photonic crystals developed in an effort to accelerate photonic materials design. I discuss the development of convolutional neural networks and equivariant neural networks for predicting photonic properties. I conclude with a discussion of photonic densities of states and equivariance in physics-based machine learning, which may provide further insight on this challenging problem.

Chapter V covers my contributions as a core developer and maintainer of the **freud** library, an open-source software package used for data analysis in this dissertation. I describe the use of the **freud** library in machine learning pipelines and data visualization, and summarize publications using **freud** (33 to date) across the fields of soft matter, statistical mechanics, and particle-based simulation.

In Chapter VI, I discuss my contributions as a core developer and maintainer of the open-source **signac** data management framework, which helps researchers execute reproducible computational studies, scaling from laptops to supercomputers and emphasizing portability and fast prototyping. I describe the framework's data

model, HDF5 data stores for large numerical arrays, enhancements to performance and scalability, and the **signac-dashboard** application for data visualization.

Finally, I conclude with a summary of the work presented in this dissertation and insights for future research in the field of self-assembly pathways, photonics, and software designed for computational researchers.

CHAPTER I

Introduction

1.1 Why Simulations?

The ability of scientists and engineers to create new materials drives advances in many fields. As humanity's scientific understanding of the universe around us has increased, so has our ability to control matter at every length scale. This level of control comes from a fusion of theoretical, computational, and experimental knowledge. Within computational materials science, there are several key frontiers being explored. One such frontier is that of system size, which is key to understanding nuanced statistical behaviors of complex systems and the dynamics of systems where microscopic and macroscopic phenomena are interdependent, such as in large macromolecules. In the space of particle simulations, we have seen computational power progress from Fermi et al. performing molecular dynamics of 64 particles in 1954 [17] to the present day, where massive supercomputers have enabled simulations of a billion atoms [18]. The power of simulation and computational methods has grown by so many orders of magnitude because of combined advances in hardware, software, theory, and algorithms (so much that our ability to generate new data sometimes exceeds our ability to analyze it).

Another frontier is that of computational materials design: we must bridge the gaps between forward design (predicting the properties of a material from its components and their interactions) and inverse design (predicting components and interactions that produce a desired set of material properties). Handling high-dimensional design spaces is challenging, though approaches for inverse design and machine learning have shown promise in this area [19, 15, 20]. For example, there are a huge number of problems in the study of materials that can be phrased in terms of optimization, opening opportunities for emerging tools from data science and machine learning. As a result, machine learning algorithms complement standard tools of the field, with adoption of machine learning rapidly increasing across the physical sciences.

Overall, computational power in the form of massive simulations, data analysis, and machine learning allows us to uncover patterns in the makeup of both everyday and exotic substances. While simulations do have limitations, they also allow scientists to enter a world where the laws of physics can be studied in a pure form – frictionless, idealized, and precise models that allow us to reduce the real world into a frame where the most important effects can be measured, interactions can be tweaked at any time, and the ruling truths can be derived from mathematical foundations.

1.2 What is a Pathway?

The study of phase transformations encompasses some of the broadest questions in physics and materials science. The diversity of ordered crystal phases covers many orders of length scales, with particles ranging from atoms to molecules to macromolecules to nanoparticles to colloids. At the atomic scale, metal alloys freeze into solid crystals. Water molecules typically form hexagonal layers called ice $\mathrm{I_h}$, but under the right temperature and pressure conditions can exist in many other crystalline forms. However, there are many ways to achieve interesting crystal structures without low temperature and high pressure conditions. Consider larger building blocks: for example, proteins and other biomolecules can be crystallized in solution. At the nanometer and micron scale, nanoparticles and colloids can self-organize into a wide range of complex crystalline structures.

The key commonality among all of these phase transformations is that the individual components interact with one another to collectively produce an ordered phase with lower free energy. However, we have yet to uncover a comprehensive microscopic theory of precisely how these building blocks arrange to form ordered structures. Systems of particles seem to coordinate a complex sequence of motions, driven by fluctuations that allow them to locally explore their phase space. Macroscopic theories such as Classical Nucleation Theory (CNT) help to explain some first-order phase transitions, but there are still many phenomena that are not well described by CNT [21]. By zooming in on microscopic processes, we hope to provide new perspectives on the kinetic and thermodynamic factors that influence crystallization, uncovering pathways that can one day be engineered.

In this dissertation, we specifically focus on a few questions pertaining to microscopic understandings of the pathways by which crystalline solids form:

- What happens at the level of individual atoms or nanoparticles when they self-organize from a fluid into a crystal?
- What – and how many – unique local environments (or motifs) emerge along the kinetic pathway from liquid to crystal?
- How can we find the important local motifs in real time without prior knowledge of what to look for?

We will also explore comparisons between crystallization pathways:

- Do atoms and nanoparticles that form the same crystal structure do so in the same way? If not, how many different kinetic pathways are there for a given crystal structure, and in what ways do they differ?
- Do large unit cell crystals, whether atomic or colloidal, follow kinetic pathways that are more "complex" (in some way) than simple crystals' pathways?

We begin to investigate these questions via the analysis of hundreds of particle-level simulations of crystallization pathways, across a wide range of interaction types and crystal structures. We present an approach applying unsupervised machine learning for dimensionality reduction and manifold learning through the use of the Uniform Manifold Approximation and Projection (UMAP) technique [22]. The manifold we explore is a novel structural descriptor developed for this work, called the **Continuous Topological Order Parameter (CTOP)**, which is composed of a set of features called Minkowski Structure Metrics [6] that capture continuous changes in particles' local environments. UMAP produces a low-dimensional embedding of the high-dimensional descriptor data (features) while preserving its topological structure. We show that the CTOP analysis method can be applied in an autonomous and parameter-free way to characterize crystallization in a variety of single-component and multi-component, simple and complex, self-assembling colloidal systems. This method is designed from the ground up with a focus on understanding microscopic dynamics, building on particle-local descriptors commonly used for identifying equilibrium structures. We expand the theoretical and practical utility of past work in analyzing equilibrium structures by emphasizing the use case of analyzing fluid-to-solid phase transitions and crystal formation with physically-motivated design of the high dimensional feature-space and its topology. Finally, we will discuss how the topology and geometry of the structural descriptor's high-dimensional feature space offer insight about generalizing the study of crystal formation and comparing order across crystal structures.

CHAPTER II

Crystallization Pathways: Theory and Methods

In this chapter, I discuss the field of crystallization pathways and motivate the use of a structural descriptor, the Continuous Topological Order Parameter (CTOP), that can be applied in an autonomous and parameter-free way to characterize crystallization.

Designed for use in an unsupervised setting, the CTOP and its UMAP embedding permit the discovery of local environments along the crystallization pathway without having to know what to look for *a priori*, revealing information pertinent to microscopic processes. This is critical since in most cases we do not know a reaction coordinate or an order parameter for the phase transition. As an unsupervised method, UMAP gives us the ability to consider the manifold structure of the features without the need to build and classify a training set of data. This makes the UMAP ML approach fundamentally different from many current uses of supervised ML in the context of crystallization and self-assembly. Our approach reveals how fluctuations change local structures by sampling a continuous, high dimensional manifold and mapping out precisely how local environments change over that manifold. With this information in hand, we have a detailed, microscopic understanding of how a variety of systems – primarily nanoparticles and colloids in the scope of this chapter, but potentially extending to atomic or biomolecular systems – assemble into a crystal from a liquid, across the entire kinetic pathway.

This approach complements and can be used alongside rare-event sampling (advanced sampling) methods used for the calculation of macroscopic quantities relevant for crystallization like nucleation rates and free energy barriers. First, we aim to study crystallization processes particle-by-particle, to discover the time-dependent sequence of microscopic processes that moves a system along the crystallization pathway from disorder to order. We seek the local particle environments (motifs) along the pathway, without assumptions as to what these motifs might look like. Our goal is to identify pathway "fingerprints" unique to the microscopic pathway a given system takes during self-assembly. By comparing fingerprints among many systems forming the same structure via different interactions, we can offer new insight on the fundamental questions enumerated above and hopefully offer hints towards a general, microscopic theory of self-assembly. It is also a necessary step for the rational development of design rules that can guide the synthesis of new nanomaterials by engineering both the nanoparticle building blocks and their kinetic assembly pathway simultaneously. Though there are many possible building blocks that can target a given thermodynamic phase and crystal structure, the ability to engineer the pathway of formation itself would enable a new level of control over self-assembly involving both thermodynamics and kinetics.

Many published studies of crystallization and self-assembly focus primarily on a single system (e.g., water, $CaCO_3$, hard spheres) or a single model, where technical details of the calculations are strongly coupled to the properties of the different phases. We show that unsupervised ML techniques can be used to describe a wide range of particle-based models along their nonequilibrium assembly pathways, independent of the details of the model. Past work has demonstrated pair potentials [15] and particle shapes that self-assemble from fluid phases into a diversity of crystal structures [23, 20, 24], which motivates the choices of systems that we analyze in this chapter. This study is made possible by our group's development of open-source software, particularly the data management software **signac** [25, 26], the **freud** library for analyzing particle simulations [27], and the GPU-optimized molecular simulation code HOOMD-blue [28], as well as other open-source software developed by the scientific Python community.

Previous work [29, 30] has developed unsupervised analysis methods for the analysis of diverse crystal structures and the kinetic pathways by which they form. We will build on these methods, incorporating physical invariants and tools from topological data analysis to develop a ML framework that characterizes the manifold structure of particle local environments. Our proposed framework has three steps:

(1) First, we compute local descriptors that can quantify local order in a continuous way. Current methods for studying crystallization perform structure-specific classification of particles (e.g., into face-centered-cubic-like and liquid-like, whether by supervised ML or a threshold value of some structural descriptor) [29, 30, 31, 32, 33, 34, 8, 35]. Two key issues arise here: first, local descriptors computed using a particle neighborhood defined by a cutoff distance or number of nearest neighbors are typically discontinuous. In a fluid state, particles near the edge of a cutoff distance will enter and exit the neighbor shell, which usually causes jumps in the local descriptor's value. Second, classification (or thresholding values, such as choosing particles with a Steinhardt order parameter $q_6 > 0.4$) usually results in a sharp line drawn between fluid-like and solid-like environments, even though those distinctions are not always clear.[1] Avoiding such jumps and arbitrary thresholds in local structural descriptors would be beneficial for structural analysis. Therefore, we leverage structure-agnostic, continuous local structure descriptors, a nontrivial and different approach that is more generalizable, more capable of distinguishing subtle differences in local environments, and parameter-free. We will use continuous local descriptors as features, such as Minkowski Structure Metrics (MSMs) [6] that are not only rotationally and translationally invariant, but also scale invariant, generalizable to many crystal structures, and able to quantify local disorder. MSMs can be computed efficiently by using the freud analysis library, an open-source Python package [27]. We expand this argument for continuity, and how Minkowski Structure Metrics resolve it, in Section 2.3.5.

(2) Second, we build a vector of these continuous descriptors to represent particle local environments and distinguish between the local environments of different crystal structures. Concatenating these descriptors into a vector produces a high-dimensional manifold (a topological space that locally resembles Euclidean space) where distances between two points are related to the structure of the corresponding particles' local environments. That is, vectors that are similar in feature space correspond to particle environments that are similar in "real" space. However, making observations about the feature space is difficult because of its high dimensionality. This high dimensionality arises from the fact that each of the N particles in the system is described by an f-dimensional vector, where f is the number of features, or descriptors. Typically $N \gg f$, which is necessary for adequately sampling the manifold.

(3) Finally, we embed the manifold in a low number of dimensions (2D or 3D),

[1]As far back as Lindemann in 1910, the line at which a solid melts into a fluid has been drawn, with homogeneous bulk melting occuring when particle vibrations exceed a threshold [36].

preserving topological structure. We will use the Uniform Manifold Approximation and Projection (UMAP) dimensionality reduction technique, which produces a low-dimensional embedding of the high-dimensional data while preserving its topological structure. The local connectedness of the manifold, derived from the descriptors' continuity and UMAP's topological preservation, is the key to this method's utility. We discuss topological preservation further in Section 2.5.

The UMAP embedding of particle descriptors carries a rich description of the underlying particle environments and crystallization process. The most interesting quality of this embedding space is that the changes in local symmetries caused by particle rearrangements appear as continuous paths in this embedding space. That is, during crystallization, particles traverse the embedded space from a fluid-like cluster[2] to a solid-like cluster, in accordance with changes in their local environment.

By computing UMAP projections of combined data from multiple simulations (which we call co-embeddings), it is possible to directly compare motifs across assembly pathways. Co-embeddings of systems forming different structures offer insight into the manifold structure of local environments and how interparticle interactions make some structures preferable over others. Such co-embeddings allow us to explore the self-assembly of crystal structures in a general manner, e.g., to examine common pathways.

2.1 Structural Descriptors and Order Parameters

We adopt the following definitions from B. Peters' Reaction Rate Theory and Rare Events [37] (emphasis added):

- A *collective variable* is any function of the full phase space coordinates.

[2]Note that here, a "cluster" refers to a cluster of points in the UMAP embedding, not a cluster of particles in the fluid/solid.

- An *order parameter* is a special collective variable that clearly distinguishes reactants from products.
- A *reaction coordinate* is a special scalar order parameter that quantifies dynamical progress along the pathway from reactants to products.
- All degrees of freedom apart from the reaction coordinate comprise the *bath*.

Many structural descriptors have been developed for crystallization in various systems, including (in chronological order of publication) the bond-orientational order parameter q_l from Steinhardt [33], common neighbor analysis (CNA) [35], centrosymmetry parameter analysis [38], bond angle analysis [39], shape matching algorithms from Keys et al. [34], neighbor distance analysis [40], topological cluster classification (TCC) [41], polyhedral template matching [8], and more. Some of these structural descriptors meet the criteria of being an order parameter, depending on the characteristics of the fluid and solid being studied. All of these are meant to identify crystal structures, and typically focus on analyzing "final" structures that are in equilibrium, though some can also be applied to measure the process of crystallization. However, all structure descriptors in the literature have some kinds of drawbacks that make them inapplicable to certain systems. For example, some methods struggle to distinguish between fcc and hcp [39], or mixtures of fcc and bcc structures [42]. Methods relying on the topological properties of a particle's local neighborhood (CNA, TCC) are susceptible to misclassification when the structure has a large amount of positional noise, which led to the development of adaptive CNA [40]. Analyses that are not robust to perturbations of the particle positions often lead to poor accuracy at high temperature. This makes the analysis of systems like plasma crystals extremely difficult, because of their "highly disturbed lattices" [43, 42].

Despite the wealth of order parameters, each with their own advantages and disadvantages, there are key questions left unresolved that the development of new methods may be able to help solve. We identify the following questions that could be addressed by novel approaches to structural descriptors. How could we distinguish between multiple product states? What can we use to detect and characterize competing mechanisms, such as the "identity crisis" posed by Teich et al. [44]? What if we do not know the appropriate structural descriptors *a priori*?

We choose to approach the search for a microscopic theory of self-assembly from the perspective of these questions. In particular, the desire to distinguish multiple product states and identify motifs without prior knowledge of the local orderings in a fluid or crystal inspired us to seek out new descriptors. To distinguish multiple states, we consider approaches that involve vector-based structural descriptors (or simply a concatenation of multiple scalar descriptors into a vector). Combining multiple order parameters is quite common, and many researchers choose 2D projections of Steinhardt order parameters such as the q_4–q_6 plane to show the ordering of multiple ordered phases [45, 46, 47].

2.2 Minkowski Structure Metrics

In this section, we present the Minkowski Structure Metrics [6], whose continuity properties are unusual among structural descriptors and enable us to construct novel unsupervised learning methods for structural analysis. We first describe the calculation of the Steinhardt order parameter q_l and commonly used variants w_l, $\overline{q}_l$, and $\overline{w}_l$, building up to the definition of Minkowski Structure Metrics q'_l and w'_l as well as their neighbor-averaged values $\overline{q}'_l$ and $\overline{w}'_l$. We follow with a discussion of literature using Minkowski Structure Metrics for local structure analysis. In this work, we use Minkowski Structure Metrics to map continuously from the space of particles' positions into the high-dimensional CTOP feature space that characterizes local symmetries. The formulas described in this section are adapted from the implementation and documentation of the **freud** library for particle analysis. The implementation in **freud** owes credit to a number of developers, including Chrisy Xiyu Du, Erin Teich, Matthew Spellings, Vyas Ramasubramani, Bradley Dice, and Brandon Butler.

2.2.1 Steinhardt Order Parameter q_l, w_l

The q_l order parameter described by Steinhardt et al. [33] is a rotationally invariant quantity that describes the correlation between points on the unit sphere with the spherical harmonics of order l. This can be used to obtain a measure of local bond-orientational order for each particle in a system.

First, we describe the computation of $q_l(i)$. For a particle i, we calculate the quantity q_{lm} by summing the spherical harmonics between particle i and its neighbors j in a local region:

$$q_{lm}(i) = \frac{1}{N_b} \sum_{j=1}^{N_b} Y_{lm}(\theta(\mathbf{r}_{ij}), \phi(\mathbf{r}_{ij})) \tag{2.1}$$

Then the q_l order parameter is computed by combining the q_{lm} in a rotationally invariant fashion to remove local orientational order:

$$q_l(i) = \sqrt{\frac{4\pi}{2l+1} \sum_{m=-l}^{l} |q_{lm}(i)|^2} \tag{2.2}$$

Closely related is the third-order rotationally invariant quantity w_l, defined as a weighted average over the $q_{lm}(i)$ values using Wigner 3-j symbols (related to Clebsch-

Gordan coefficients):

$$w_l(i) = \sum_{m_1+m_2+m_3=0} \begin{pmatrix} l & l & l \\ m_1 & m_2 & m_3 \end{pmatrix} q_{lm_1}(i) q_{lm_2}(i) q_{lm_3}(i) \tag{2.3}$$

In this work, we always apply a normalization factor to w_l, redefined as follows:

$$w_l(i) = \frac{\sum_{m_1+m_2+m_3=0} \begin{pmatrix} l & l & l \\ m_1 & m_2 & m_3 \end{pmatrix} q_{lm_1}(i) q_{lm_2}(i) q_{lm_3}(i)}{\left(\sum_{m=-l}^{l} |q_{lm}(i)|^2\right)^{3/2}} \tag{2.4}$$

2.2.2 Averaged Steinhardt Order Parameter $\overline{q}_l, \overline{w}_l$

The "average" variant of this order parameter takes into account q_{lm} values from the first and second shell combined [45]. To compute this parameter, we perform an average over the first neighbor shell of the particle to implicitly include information about the second neighbor shell. This averaging is performed by replacing the value $q_{lm}(i)$ in the original definition by $\overline{q}_{lm}(i)$, the average value of $q_{lm}(k)$ over all the N_b neighbors k of particle i, including particle i itself (treated as index 0 in the summation):

$$\overline{q}_{lm}(i) = \frac{1}{N_b} \sum_{k=0}^{N_b} q_{lm}(k) \tag{2.5}$$

Then $\overline{q}_l$ is computed using these averaged values:

$$\overline{q}_l(i) = \sqrt{\frac{4\pi}{2l+1} \sum_{m=-l}^{l} |\overline{q}_{lm}(i)|^2} \tag{2.6}$$

The average variant $\overline{w}_l$ is defined in a similar way, using $\overline{q}_{lm}(i)$ in place of $q_{lm}(i)$ while summing over the Wigner 3-j symbols:

$$\overline{w}_l(i) = \frac{\sum_{m_1+m_2+m_3=0} \begin{pmatrix} l & l & l \\ m_1 & m_2 & m_3 \end{pmatrix} \overline{q}_{lm_1}(i) \overline{q}_{lm_2}(i) \overline{q}_{lm_3}(i)}{\left(\sum_{m=-l}^{l} |\overline{q}_{lm}(i)|^2\right)^{3/2}} \tag{2.7}$$

2.2.3 Minkowski Structure Metrics q'_l, w'_l

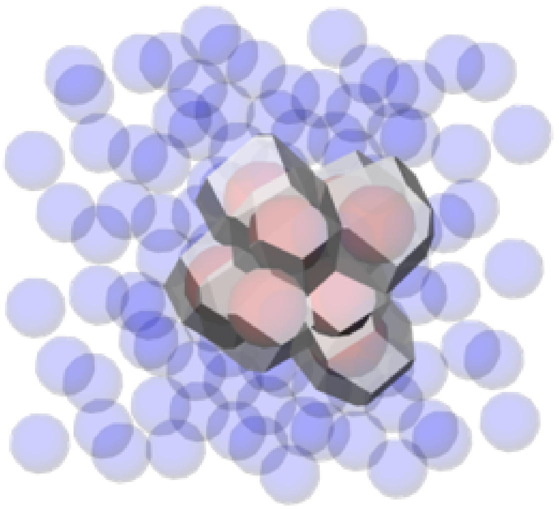


Figure 2.1: Visualization of the Voronoi polyhedra for an fcc system. The Voronoi cells of this noisy fcc structure are slightly distorted rhombic dodecahedra. This rendering uses plato [3] and pythreejs [4].

The "weighted" variant of this order parameter requires a scalar weight indicating the contribution of each neighbor bond. The specific weighted case considered in this section is one where the neighbors and their weights are obtained from Voronoi polyhedra and their facet areas, an example of which is shown in Figure 2.1. Mickel et al. terms this the *morphometric neighborhood*, and using this basis for computing bond-orientational order parameters results in the Minkowski Structure Metrics [6]. We will explain how the Minkowski Structure Metrics are computed, then we will explain why they are useful. We first define the Voronoi neighbors to be particle pairs that share a facet in the Voronoi diagram of the particle system, with the directed weight between two particles w_{ij} defined as

$$w_{ij} = \frac{A_j}{\sum_{k=1}^{N_b} A_k} \tag{2.8}$$

where A_k represents the facet area of neighbor k for the Voronoi polyhedron of particle i. Note that we choose a different convention than Mickel et al. – these weights are not symmetric, i.e., $w_{ij} \neq w_{ji}$, because we normalize by the surface area of the origin particle's Voronoi polyhedron so that $\sum_{j=1}^{N_b} w_{ij} = 1$. Voronoi diagrams are computed using the voro++ software [48], which is called by the **freud** library.

The Steinhardt order parameter formulas are then modified as follows, replacing $q_{lm}(i)$ with the weighted value $q'_{lm}(i)$:

$$q'_{lm}(i) = \frac{1}{\sum_{j=1}^{N_b} w_{ij}} \sum_{j=1}^{N_b} w_{ij} Y_{lm}(\theta(\mathbf{r}_{ij}), \phi(\mathbf{r}_{ij})) \tag{2.9}$$

This results in the following expressions:

$$q'_l(i) = \sqrt{\frac{4\pi}{2l+1} \sum_{m=-l}^{l} |q'_{lm}(i)|^2} \tag{2.10}$$

$$w'_l(i) = \frac{\sum_{m_1+m_2+m_3=0} \begin{pmatrix} l & l & l \\ m_1 & m_2 & m_3 \end{pmatrix} q'_{lm_1}(i) q'_{lm_2}(i) q'_{lm_3}(i)}{\left(\sum_{m=-l}^{l} |q'_{lm}(i)|^2\right)^{3/2}} \tag{2.11}$$

Combining these modifications (average, weighted, and third-order w_l) can be nontrivial. Here, we define the combination of "average" and "weighted" (used to define the features of $CTOP_{\overline{qw}}$) in a way that uses weighted neighbors for the construction of q'_{lm} but does not consider neighbor weights when averaging over the second shell of neighbors (each neighbor has equal weight regardless of the size of the Voronoi facet).

2.2.4 Properties of Minkowski Structure Metrics

The differences between the Minkowski Structure Metrics and traditional Steinhardt order parameters may seem subtle, but in fact have significant implications for robust structural analysis and for this work. The choice of neighborhood greatly affects the magnitudes of q_l values, and the morphometric neighborhood is robust to small changes in neighbors' positions that could remove or replace nearest neighbors under the k-nearest neighbors or cutoff distance definitions [6]. We are specifically interested in Minkowski Structure Metrics based on Voronoi neighborhoods because their values are continuous with respect to particle positions, thereby eliminating the discontinuities associated with changes in particle neighbors. We show this in following sections.

2.3 Order Parameters Using Machine Learning

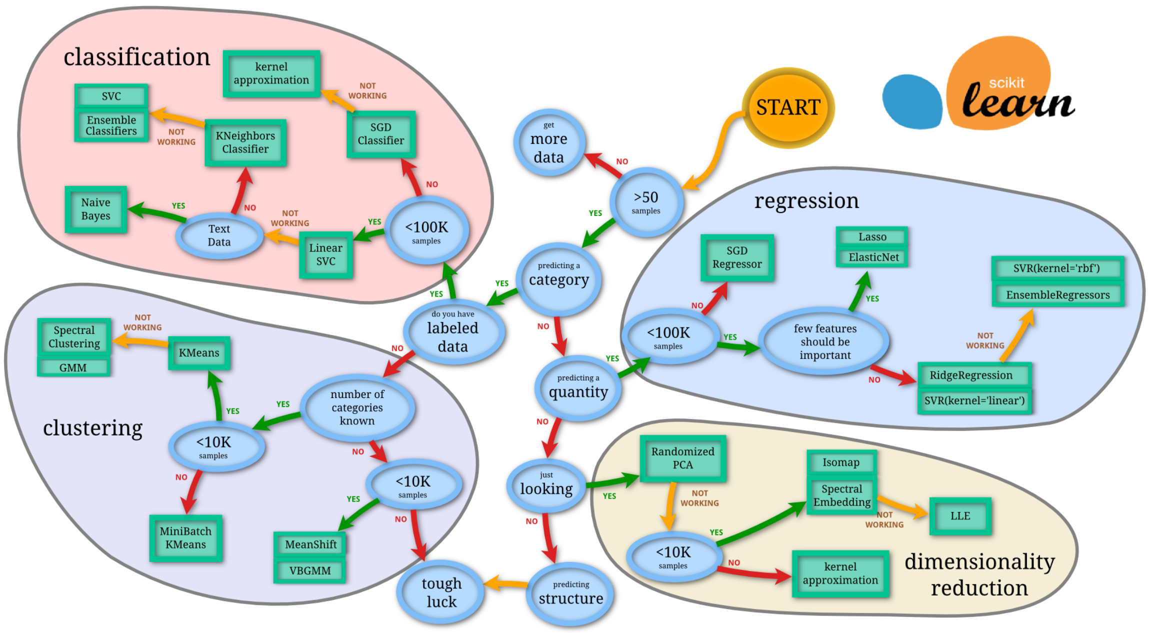
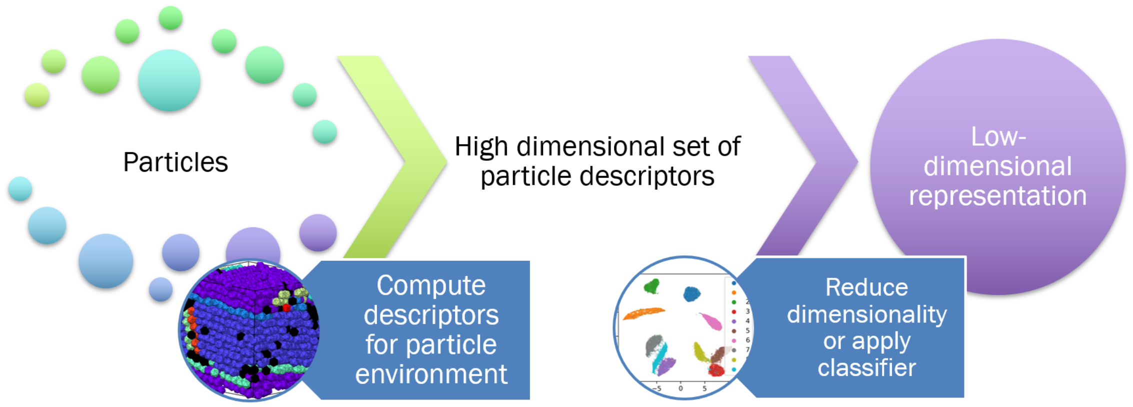


Figure 2.2: Overview of machine learning algorithms. Figure adapted from the documentation of scikit-learn [5].

Molecular simulations enable the collection of long-running trajectories of self-assembling nanoparticles, amassing terabytes of data that show a system traversing a kinetic pathway towards a different phase. The challenge is to identify the relevant features that describe the physics of this pathway and the microscopic processes that comprise it. This is where machine learning can be used to further our understanding: at each point in time, we can construct feature vectors containing dozens or hundreds of values and use them in conjunction with machine learning pipelines. A broad overview of machine learning approaches is shown in Figure 2.2. This study's approaches to machine learning primarily fit into the categories of dimensionality reduction and clustering.

2.3.1 The Descriptor Reduction Paradigm



Particles
High dimensional set of particle descriptors
Low-dimensional representation
Compute descriptors for particle environment
Reduce dimensionality or apply classifier

Figure 2.3: Many machine learning approaches for quantifying order in particle systems follow a paradigm where a high-dimensional feature space is constructed through the computation of descriptors for each particle and then reduced into a low-dimensional representation such as a class identifier.

Most current approaches to machine learning on particle systems follow the paradigm outlined in Figure 2.3. In this general class of models, there are two stages. The first stage is where feature engineering plays a role, when features are computed from the input data (typically particles' positions and a description of a periodic box containing them). This is imperative because particle data is represented as vectors $\mathbf{x}_i \in \mathbb{R}^3$, which neglects periodic boundary conditions, rotational invariance, translational invariance, and a number of other considerations. Finding neighbors and transforming into a set of relative vectors $\mathbf{r}_{ij} \in \mathbb{R}^3$ for each particle respects the periodic boundary conditions and establishes translational invariance, but not rotational invariance. Computing features from a particles' neighbors allows for these additional invariances to be described. The second stage involves the machine learning algorithm of choice, such as a classifier, regressor, or dimensionality reduction algorithm. We emphasize the independence of these stages because we believe that physics-based machine learning involves a combination of expressive and physically-motivated features with adaptable and physically-motivated models. Next, we will describe both feature engineering and model architecture.

2.3.2 Feature Engineering

Feature engineering is the process of selecting data representations, choosing specific transformations or analyses to perform on raw inputs, and empirically refining those choices, resulting in a more useful form of data ready for use in machine learning algorithms or data science applications. Feature engineering is important because the representations of data are fundamentally responsible for determining the aspects of a system that they can adequately describe, and the study of transitions such as complex assembly pathways can involve many representations of data. Quoting AI researcher Andrew Ng, "Coming up with features is difficult, time-consuming, [and] requires expert knowledge. 'Applied machine learning' is basically feature engineering" [49]. It is this process of feature engineering (with appropriate model selection) that leads to domain knowledge (e.g., physical invariants) being embedded in a machine learning application.

For example, nanoparticle simulations can be represented as a series of positions over time [29, 50] or as a movie where the particles' positions are encoded on a discrete grid. Crystal structures can even be represented as a graph [51], where nodes represent particles and edges define particles' neighbors. While all of these representations have been used as inputs for different problems, the ability of a machine learning algorithm to extract meaning from a data set is heavily influenced by the representation of input data.

2.3.3 The Importance of Model Architecture

In addition to feature engineering, the architecture of machine learning models has a significant effect on their utility for a given problem. Several hard problems in artificial intelligence and machine learning have been solved (or at least established an approach that is useful in practice) through finding the right algorithms for a given data representation. For example, deep convolutional neural networks have been found to be extremely successful at classifying images [52]. The architecture of a convolutional neural network enforces translational invariance, with "filters" acting on spatially local grid of pixels. This translation invariance helps the convolutional neural network build meaningful representations for the problem, because identifying a bicycle or cat is fundamentally the same regardless of which part of the image contains those objects. Similarly, recurrent neural networks have been used for text modeling [53], aided by the connection between temporal patterns in natural language and the internal representation of the recurrent neural network. These examples illustrate the broader idea that finding the right representation of data provides a significant benefit to the accuracy and interpretability of machine learning algorithms.

2.3.4 Machine Learning Methods for Structure Detection

Recent literature has used both supervised and unsupervised machine learning methods for structure detection. Here, we focus on the features and model architectures chosen, because we hope to build on these ideas. First, we will describe related works using supervised machine learning methods for complex structures. This study builds on the ideas presented in these works, especially those in the unsupervised category, with added emphasis on the continuity and topology of feature space and expanded applications to new complex structures and quasicrystals not previously studied.

Supervised Methods

Machine learning methods for studying crystals have largely focused on structure identification as a *classification* problem, and most commonly in a supervised setting where reference data can be used to train a model. The classification problem can be understood as determining the phase and crystal structure for all particles in a simulation, and assigning values such as fluid, face-centered cubic, or body-centered cubic to each particle. Some recent examples of this, which follow the paradigm shown in Figure 2.3, can be found in Refs. [29, 54, 50, 46, 55]. For example, Boattini et al. used neural networks for identifying binary crystals [55]. A single layer feed-forward neural network was used with a training set of crystal structures. The features used are Steinhardt second-shell average values, with particle neighbors found using either a cutoff radius or the solid angle nearest neighbor (SANN) method [56]. Coli et al. modified this approach, adding two hidden layers with 72 neurons to the neural network and using a larger number of standard (rather than second-shell average) Steinhardt order parameters in order to retain more information about local environments [54]. We adopt a similar technique in this study, including both first-shell and second-shell averaged Minkowski Structure Metrics in the features.

Looking through the lens of classification is sufficient for some problems, but it can be difficult to use in many cases that are of scientific interest. For example, classification requires training data for all the desired phases and thus requires additional information for systems where the local environments are not known *a priori*. Training data may need to be specific to the particular system and not just the crystal structures or local environments being detected. For example, thermal noise in systems like plasma crystals can lead to severely distorted local environments that are hard to identify but are nonetheless a part of a structure with long-range crystalline order [43, 42]. Even in the domain of colloidal and nanoparticle systems that our group frequently examines, Hard Particle Monte Carlo simulations exhibit significant (athermal) positional noise at lower densities because permitting exploration of the particles' local volume maximizes the system's entropy (particles interacting under pair potentials are typically held closely in place by the potential energy cost of moving away from their equilibrium position).

There are many difficult choices to make when implementing a supervised learning algorithm, such as which structures to include in a training set or the temperatures used for the training data. What if the obtained structure isn't in the training set, or the expected structure is not known? Unsupervised methods, discussed next, can solve some of these challenges, but sometimes also introduce new challenges.

Unsupervised Methods

Spellings and Glotzer used both supervised and unsupervised methods to analyze the phase diagram of an oscillating pair potential parameterized by k and ϕ [29]. The unique feature set of spherical harmonics computed over different numbers of nearest neighbors allows for local environments to be identified by a high-dimensional "fingerprint." This paper describes an unsupervised learning approach where the fingerprints are reduced to 128 dimensions with Principal Components Analysis (PCA) and a Gaussian mixture model is fit to the high-dimensional fingerprints. The paper notes, "in ordered systems with well-defined shells of nearest-neighbor particles, there is often a degeneracy in terms of which nearest-neighbor particles within the shell the algorithm will find – for example, when looking at 5-particle neighborhoods in the $cI2$-W (BCC) structure, there are $\binom{8}{5}$ ways to place five particles in the eight vertices of the cube in the first neighbor shell, many of which are equivalent by symmetry." This degeneracy, driven by the choice of features, has a significant effect on the topology of feature space. While supervised methods can learn which features are important and ignore the irrelevant features, unsupervised methods often do not have that kind of discriminatory capacity.

Boattini et al. used autoencoders to generate 2D embeddings of local environments for several single-component and binary structures [31]. Like the previously mentioned work from Boattini, this paper uses Steinhardt order parameters averaged over the second neighbor shell. The use of an autoencoder results in a neural network that is trained to compress an input vector into a low-dimensional "bottleneck," or projection space, and subsequently project back into the original vector space. This combines an "encoder" and "decoder" which are jointly optimized to perform an identity mapping. We note that traditional autoencoders may not learn continouous mappings into the latent space, which was one of the motivations for the development of variational autoencoders [57].

Adorf et al. showed how unsupervised methods can be used to measure nucleation behavior with clustering of a high dimensional feature space [30]. This paper uses a high-dimensional feature vector to describe particles' local environments based on the bispectrum of the group SO(3). The unsupervised learning algorithm uses PCA to project these vectors into a 20 dimensional space, followed by a UMAP dimensionality reduction into 10 dimensions (and 2 dimensions for visualization). The HDBSCAN clustering algorithm is applied in the 10-dimensional space. The resulting clusters are visualized in the 2D UMAP projection. This method is used to characterize a number of self-assembly simulations from Ref. [15]. Because the bispectrum descriptors are based on a number of nearest neighbors, they are discontinuous with respect to particle positions.

2.3.5 Choosing Neighbors, Features, and ML Models

The literature discussed above as well as the previous discussion of representations and models leads us to ask, what properties should be desired in a representation of particle data useful for understanding phase transitions, nucleation, and self-assembly? Invariance under translation, rotation, and reflection covers the three-dimensional Euclidean group of isometries, $\mathbb{E}^3$. These properties are reasonable to expect – the process of crystallization does not depend on the observer's frame of reference, nor do the physical laws change with respect to any of those symmetries. However, two additional properties are important to consider.

Continuity

The first additional property is continuity with respect to particle positions. Given a system of N particles with positions $\mathbf{x}_i$, we compute a local structural descriptor $f(\mathbf{x}_i, \{\mathbf{x}_j | j \in N_G(\mathbf{x}_i)\})$ that depends on the particle and its neighbors $N_G(\mathbf{x}_i)$ (borrowing the notation from graph theory for an open neighborhood that does not include the particle itself). Specifically, perturbing the particles with a small amount of positional noise δ_i (from thermal noise in a molecular dynamics simulation or just a small Monte Carlo move) should have only a small effect on the value of the local structural descriptor. This is related to the (ϵ, δ) definition of continuity in real analysis. First, there are some common issues arising from the choice of neighborhood when developing structural descriptors.

The most common choices of neighborhood given by N_G use a cutoff distance $r_{\max}$ or a fixed number of nearest neighbors (also called k-nearest neighbors). The cutoff distance is typically chosen by looking at the radial distribution function $g(r)$ in some

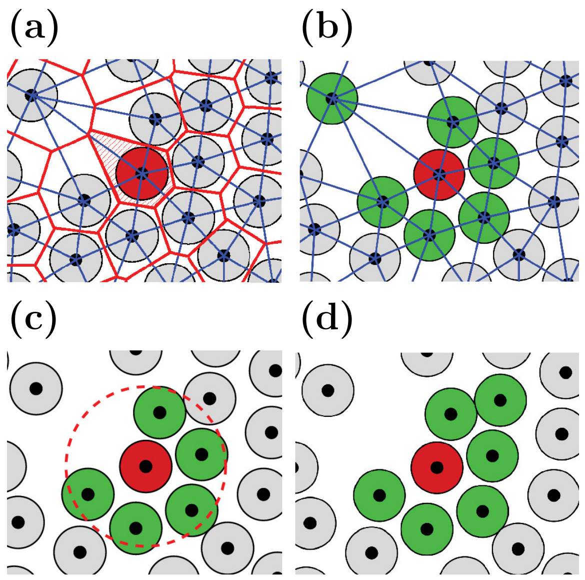


Figure 2.4: Common neighbor definitions for the particle shown in red. Neighbors are shown in green. Figure from Ref. [6] with permission. **(a)** The Voronoi polygon for the particle is shaded red. The edges in bold red define which particles are neighbors. The blue bonds form the dual of the Voronoi diagram, called the Delaunay graph. **(b)** Neighbors found using the Voronoi diagram. Note that the furthest neighbor has a small edge in the Voronoi diagram in (a), so its neighbor weight is small. **(c)** Neighbors found using a cutoff radius r_{max}. **(d)** Neighbors found using k-nearest neighbors, $k = 6$.

state (usually the crystalline solid) and choosing a value that falls in between two peaks. Choosing r_{max} as the "trough" between the first and second peaks corresponds to selecting the first neighbor shell. However, the value of r_{max} may not be so clearly defined if the particles interact under a potential with multiple minima, if the system is near a phase boundary between two crystals with different neighbor spacings, or if the crystal has multiple Wyckoff positions. In those cases, particles may be expected to enter or leave the cutoff radius, resulting in a discontinuous jump in the neighbor subgraph $N_G(\mathbf{x}_i)$. These modes of neighbor finding are shown in Figure 2.4. For common structural descriptors such as the Steinhardt order parameter [33], this results in a jump in the measured values of $q_l(i)$. Similarly, a fixed number of nearest neighbors may not be a good choice if multiple crystal structures can exist in the same system – they cannot be easily compared using the same structural descriptors. Consider analyzing structures such as face-centered cubic, which has 12 nearest neighbors. If analyzed with a neighborhood of $k = 8$ nearest neighbors (the typical choice for body-centered cubic structures), the neighbor subgraph changes dramatically as the 12 particles oscillate around their ideal lattice positions, entering and leaving the set of the closest 8.

However, it is possible to construct neighbor subgraphs that vary continuously with respect to particle positions. A concrete example of this is the Voronoi diagram, which computes a convex polyhedron (or polygon in 2D) for each particle whose facets (or edges in 2D) bisect the line segments between neighboring particles. The resulting Voronoi regions contain the set of points that are closer to their contained particle than any other particle. The neighbor subgraph is then augmented with weights w_j such that each neighbor j has a weight $w_j = \frac{A_j}{\sum_k A_k}$ where A_k is the facet area (or edge length in 2D) of the facet between particle i and particle k. In this definition of neighborhood, neighbor particles enter and exit the neighborhood continuously, with their weights going to zero as the neighbor particle moves away and rising from zero as it gets closer. For example, consider Figure 2.4(a-b). The furthest neighbor in the upper-left contributes only a tiny amount to the Voronoi polygon and thus has a small neighbor weight. If that particle were to drift farther from the central particle in red, its contribution would drop to zero and it would no longer be a neighbor of the central particle. Conversely, if it were to get closer, its weight would grow.

In this work, the Minkowski Structure Metric (defined in Eqn. 2.10) is used with a Voronoi neighborhood with neighbor weights set according to the Voronoi facet areas [6]. Here, the neighborhood selection and local structural descriptor uses the Voronoi weighted information to jointly define a descriptor that is a continuous function of particle positions [6].

Scale Invariance

The second property to discuss is scale invariance. While density fluctuations (which carry an associated length scale) certainly do play a role in crystallization, this can be characterized separately from the orientational ordering found via Minkowski Structure Metrics [58, 59] Furthermore, the choice of using structure descriptors that are scale-invariant means that simulations of vastly different materials with similar local structures can be compared directly. The ordering of different systems, whether colloids with micron length scales or nanoparticles with nanometer length scales, can be equated more fairly if the structural descriptors are only sensitive to the neighborhood shape, rather than the magnitude of the neighbor bond vectors. The characteristic densities and length scales can be found by considering the inverse of the volume of the Voronoi polyhedron (or area of the Voronoi polygon in 2D) or the radial distribution function $g(r)$.

2.4 The Continuous Topological Order Parameter (CTOP)

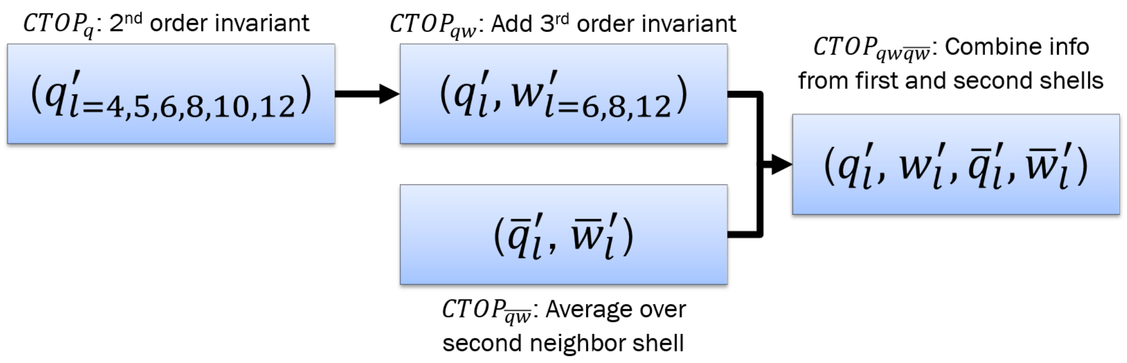

$CTOP_q$: 2nd order invariant
$(q'_{l=4,5,6,8,10,12})$
$CTOP_{qw}$: Add 3rd order invariant
$(q'_l, w'_{l=6,8,12})$
$CTOP_{qw\overline{qw}}$: Combine info from first and second shells
$(q'_l, w'_l, \bar{q}'_l, \bar{w}'_l)$
$(\bar{q}'_l, \bar{w}'_l)$
$CTOP_{\overline{qw}}$: Average over second neighbor shell

Figure 2.5: The variants of the Continuous Topological Order Parameter (CTOP).

Here we will define a new order parameter that we call the Continuous Topological Order Parameter (CTOP). This descriptor and its variants are $\mathbb{E}^3$ invariant, scale-invariant, continuous structural descriptors. The components of the CTOP are Minkowski Structure Metrics, described in Section 2.2 and defined in Eqn. 2.10. The CTOP is a vector-valued quantity, and is parameter-free aside from the choice of local symmetries (l values) to consider.

Minkowski Structure Metrics have been used for a number of recent studies in crystallization, including both simulation and experimental data [60, 47, 46]. We build on these past works using Minkowski Structure Metrics by treating the CTOP feature space of multiple Minkowski Structure Metrics as a manifold and examining its topology and embeddings in two (or three) dimensions as a means of unsupervised local structure analysis. The topology of this manifold is significant because it reveals pathways sampled by individual particles as they explore configurations with different kinds of local symmetry. The CTOP and its variants are defined as:

$$CTOP_q(i) = \{q'_l(i) | l \in \{4, 5, 6, 8, 10, 12\}\} \tag{2.12}$$

$$CTOP_w(i) = \{w'_l(i) | l \in \{6, 8, 12\}\} \tag{2.13}$$

$$CTOP_{\overline{q}} = \{\overline{q}'_l(i) | l \in \{4, 5, 6, 8, 10, 12\}\} \tag{2.14}$$

$$CTOP_{\overline{w}} = \{\overline{w}'_l(i) | l \in \{6, 8, 12\}\} \tag{2.15}$$

$$CTOP_{qw}(i) = CTOP_q(i) \oplus CTOP_w(i) \tag{2.16}$$

$$CTOP_{\overline{qw}}(i) = CTOP_{\overline{q}}(i) \oplus CTOP_{\overline{w}}(i) \tag{2.17}$$

$$CTOP_{qw\overline{qw}}(i) = CTOP_{qw}(i) \oplus CTOP_{\overline{qw}}(i) \tag{2.18}$$

where $\oplus$ represents vector concatenation. This order parameter is implemented using the **freud** library [27], discussed in Chapter V. Figure 2.5 demonstrates these variants and their definitions.

The results shown in this work use a UMAP embedding [22], which reduces the high-dimensional feature space into $d = 2$ dimensions (or any small integer – we choose to consider only $d \leq 3$ for the purposes of data visualization). This algorithm is chosen for several reasons, discussed in Section 2.5. We use the term "feature space" to describe the high-dimensional manifold of CTOP vectors which describe particles' local environments. It is in the mapping from particles' positions into feature space that continuity plays a significant role. Once particles' local environments are continuously mapped into the feature space via CTOP, we use UMAP specifically for its topology-preserving properties, as depicted in Figure 2.6. While the high-dimensional CTOP manifold does not always map cleanly into two dimensions, it gives a good sense for the structure of the data and how particles' local environments evolve during the course of a simulation.

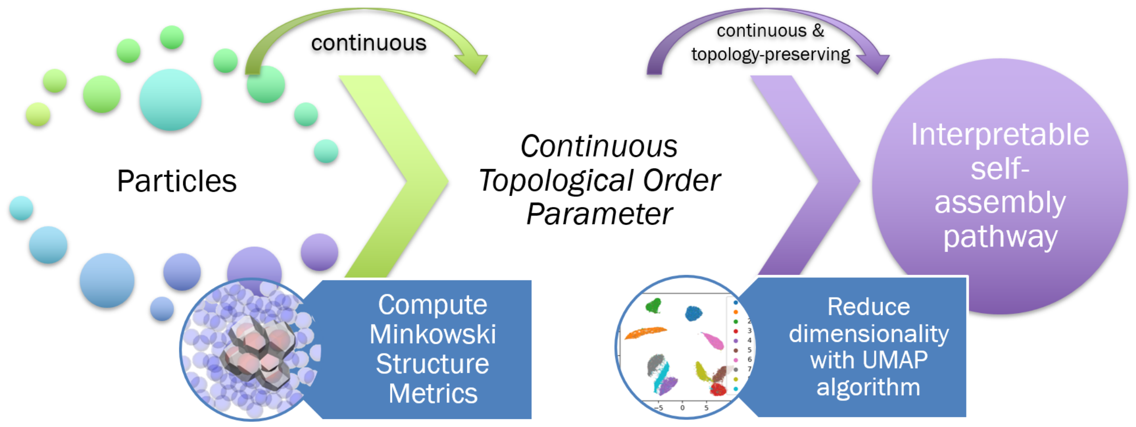

continuous
continuous & topology-preserving
Particles
Continuous Topological Order Parameter
Interpretable self-assembly pathway
Compute Minkowski Structure Metrics
Reduce dimensionality with UMAP algorithm

Figure 2.6: The CTOP order parameter is a continuous function of particle positions, and can be mapped continuously into a low-dimensional representation using UMAP.

2.5 UMAP Dimensionality Reduction

Here, we introduce the Uniform Manifold Approximation and Projection (UMAP) algorithm used for dimensionality reduction of the CTOP feature space. The general goal of dimensionality reduction algorithms is to transform high-dimensional data into a lower number of dimensions, while preserving some essential characteristics of the higher-dimensional data. Dimensionality reduction techniques may be linear or nonlinear. A well-known linear method of dimensionality reduction is Principal Components Analysis, developed by Pearson in 1901 [61]. Nonlinear methods include UMAP and t-SNE [62]. Furthermore, there are distinctions between methods that aim to preserve pairwise distances globally, and those that prioritize the preservation of local structure at the expense of global structure. The goal of the dimensionality reduction dictates what kind of methods should be applied. For instance, PCA gives the minimal squared reconstruction error for a given number of projection dimensions.

Our choice of the UMAP algorithm for dimensionality reduction is motivated by theoretical, empirical, and practical considerations. First, we outline the theoretical motivations for the UMAP algorithm, and explain how these pertain to our problem.

2.5.1 Data Density on the CTOP Manifold

It is common for explorations of a new parameter space (such as the Oscillating Pair Potential studied by Spellings et al. [29] or the space of polyhedra studied by Damasceno et al. [23]) to result in a large number of trajectories where, prior to any analysis, we do not know the final state or if/when a phase transition occurred. In such cases, we also have no prior knowledge about the *data density* on the $CTOP$ manifold (or any other choice of descriptors). That is, we do not know what distribution to expect for the number of samples of particles in disordered or ordered local environments because we do not know how long the trajectory spent in fluid or solid phases. We do know that the data distribution is affected by the number of samples obtained from fluid-like and solid-like particles, and that it is especially sensitive to the sampling rate during the process of crystallization.

Identifying crystal structures obtained from self-assembly typically only requires analyzing the final few frames saved from a simulation. Therefore, many simulations only save one frame for every few hours of simulation (perhaps every few million time steps). This has historically been the norm, because of the cost of storage space and increased processing power required to analyze large trajectory files. However, as we begin to investigate the microscopic processes of crystallization, we demand orders of magnitude more data to capture the fast timescales over which crystallization can occur (especially in supercooled systems). Advanced sampling has a significant role to play in making it easier to capture data about phase transitions, but it remains a challenge to configure and run such simulations for arbitrary systems.[3] For the purposes of studying microscopic dynamics and crystallization, we must run simulations that save frames at a high frequency. This ensures that the resulting trajectory data has adequately sampled the order parameter manifold and allows for reconstruction of phase transitions and analysis of local order emerging at the per-particle level.

2.5.2 Mathematical Foundations of UMAP

Described by McInnes et al. [22], UMAP builds on the fields of Riemannian geometry and algebraic topology. The general process involves finding approximate nearest neighbors in a high-dimensional feature space and finding an embedding into a lower dimensional space that preserves distances to the neighbors in the high dimensional space. UMAP handles nonuniform data distributions in the high-dimensional space, like those described above, through a locally-varying Riemannian metric. We present a heavily simplified summary of the UMAP algorithm and refer the reader to McInnes' UMAP paper for further details [22]. Another description of the algorithm

[3]Advances in open-source software packages such as OpenPathSampling [63, 64] and SSAGES [65] are slowly chipping away at the level of difficulty, but few tools are well-suited for colloidal self-assembly.

can be found in the UMAP online documentation.[4] First, every data point is assigned a custom distance metric based on its nearest neighbors, such that the local density at each point appears to be uniform. The original data may exist in any space, not just $\mathbb{R}^n$, as long as distances can be measured (enabling the determination of nearest neighbors). To reconcile the differences between each point's distance metric, fuzzy simplicial sets are constructed for each point. The construction of this metric ensures that every point's fuzzy simplicial set contains at least one other point, so that their union is connected. The union of these fuzzy simplicial sets contains information about the topology and metric structure of the high-dimensional manifold. Next, the fuzzy topological representation must be embedded into $\mathbb{R}^d$, where d is the embedding dimension d. Finally, the cross-entropy between the high-dimensional and low-dimensional fuzzy topological representation is minimized using stochastic gradient descent, resulting in a low-dimensional embedding whose topology in $\mathbb{R}^d$ is optimized to match that of the high-dimensional data.

Ultimately, this means that a manifold sampled in the high-dimensional space will have a similar local connectivity, and thus a similar topological structure, in the embedded space (assuming that the embedding dimension is high enough to represent the topology of the original manifold). The UMAP algorithm is frequently used to help solve the problem of visualizing high-dimensional data, and has seen widespread usage in several fields, including molecular simulations [30] and single-cell genomics [66].

UMAP is also highly scalable, and produces embeddings faster than most competing nonlinear dimensionality reduction techniques [22]. We use a UMAP implementation from the RAPIDS cuML library [67, 68], leveraging GPU acceleration for

[4] `https://umap-learn.readthedocs.io/en/latest/how_umap_works.html`

the nearest neighbor computations and embedding optimization.

2.5.3 Example: UMAP Applied to MNIST Images

As an example, we can apply the UMAP algorithm to a data set of images containing handwritten digits. This data set, popularly used for supervised classification algorithms, was modified from a National Institute of Standards and Technology data set and is thus known as MNIST. The raw data are available online[5] and can be easily downloaded with the Python package `mnist`.[6] The grayscale images are 28 pixels by 28 pixels, thus there are $28^2 = 784$ pixel values in each image. This can be imagined as a vector in $\mathbb{R}^{784}$, thereby allowing Euclidean distances to be measured between each image. This topological space can then be embedded into a lower number of dimensions. In Figure 2.7, we show embeddings into two and three dimensions. The clusters that are close together in the UMAP embedding are also close in the high-dimensional space. The embeddings in two and three dimensions, shown in Figure 2.7(b) and (c), are able to distinguish these clusters from the 784-dimensional manifold. The digits 8, 3, and 5 are similar in shape, and their clusters are adjacent in 2 and 3 dimensions because the Euclidean distances between these images in the 784-dimensional feature space are relatively small. Similarly, the digits 7, 9, and 4 are clustered together. Isolated clusters appear for the digits 0, 1, 2, and 6. The number of pixels that differ between an 8 and a 5 is small, compared to the number that differ between an 8 and a 7. This results in a smaller distance in the high-dimensional feature space, causing the clusters to be close together in the embedding space. This example demonstrates the workings of UMAP: given a high-dimensional set of features, it finds a projection into a lower number of dimensions that preserves the local structure and topology of the original feature space.

[5] `http://yann.lecun.com/exdb/mnist/`
[6] `https://pypi.org/project/mnist/`

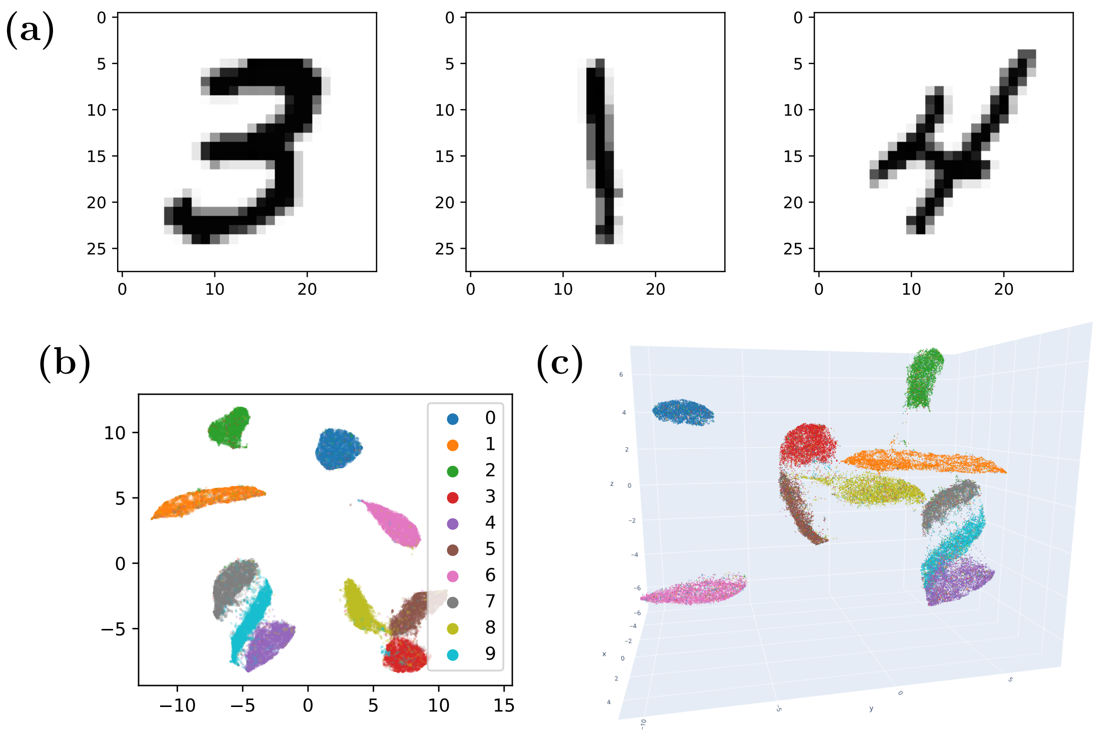


Figure 2.7: **(a)** Sample 28×28 pixel images of digits 3, 1, 4 from the MNIST data set. **(b)** UMAP embedding of the MNIST data set into two dimensions. **(c)** UMAP embedding of the MNIST data set into three dimensions. Visualization generated using Plotly [7].

CHAPTER III

Crystallization Pathways: Results

In this chapter I will demonstrate a wide variety of self-assembling systems that can be described by the CTOP order parameter. The range of applications includes common crystal structures such as face-centered cubic (fcc), body-centered cubic (bcc), hexagonal close-packed (hcp), and simple cubic (sc), as well as complex crystal structures with multiple Wyckoff positions and quasicrystals. Unless otherwise stated, all figures are rendered using Matplotlib [69] and OVITO [9].

3.1 Self-Assembly of Hard Polyhedra

First I will demonstrate the CTOP analysis method for a set of Hard Particle Monte Carlo (HPMC) simulations, inspired by past studies of self-assembly in hard polyhedra [23].

3.1.1 Hard Particle Monte Carlo Simulation Protocol

For this series of simulations, we use the Hard Particle Monte Carlo (HPMC) method implemented in HOOMD-blue [70, 28]. A set of 19 different polyhedra simulated by Damasceno et al. [23] were selected for self-assembly simulations. These polyhedra were generated using the coxeter package [71] and normalized to have unit volume. A total of 169 simulations were run in the NVT ensemble, each with different shapes and target packing fractions. Each simulation started with a configuration of 15,625 (25^3) particles in a dilute gas at a packing fraction of 0.05 in a cubic simulation box. The initial configuration was compressed to a target packing fraction ϕ (typically in the range $\phi \in [0.5, 0.6]$, based on an estimated packing fraction at which self-assembly would occur from past studies). During the quick compression scheme, the box volume and particle positions were first scaled down by a factor of 0.999, while keeping the box in a cubic geometry. If any particle overlaps were created by the rescaling, Monte Carlo moves were attempted until the configuration was relaxed to a state with no overlaps. After each box rescaling in the quick compression, the simulation ran for an additional 100 steps. This process was repeated until the target packing fraction ϕ was reached. In total, each simulation ran for approximately 2.5 million Monte Carlo steps while compressing to the target packing fraction (more steps were needed to relax systems that had lots of overlaps during the rescaling). Finally, the system was allowed to equilibrate at constant volume for an additional 5 million Monte Carlo steps. The simulation configuration was saved every 5,000 steps.

3.1.2 Self-Assembly of Hard Polyhedra into fcc

The fcc structure can be self-assembled by 51 of the 145 polyhedra reported by Damasceno et al. [23]. Many of these shapes are nearly-spherical, with isoperimetric quotients close to 1.[1] The fcc phase is also readily formed by hard spheres and several pair potentials, including the WCA and Lennard-Jones potentials. Comparisons among multiple fcc-forming systems are performed in Section 3.3.1.

In Figure 3.1, we show the results of a simulation of hard icosahedra. This HPMC

[1]The isoperimetric quotient is defined as $IQ = \frac{36\pi V^2}{S^3}$, and compares the volume V and surface area S of an arbitrary shape relative to that of a sphere. A sphere has $IQ = 1$.

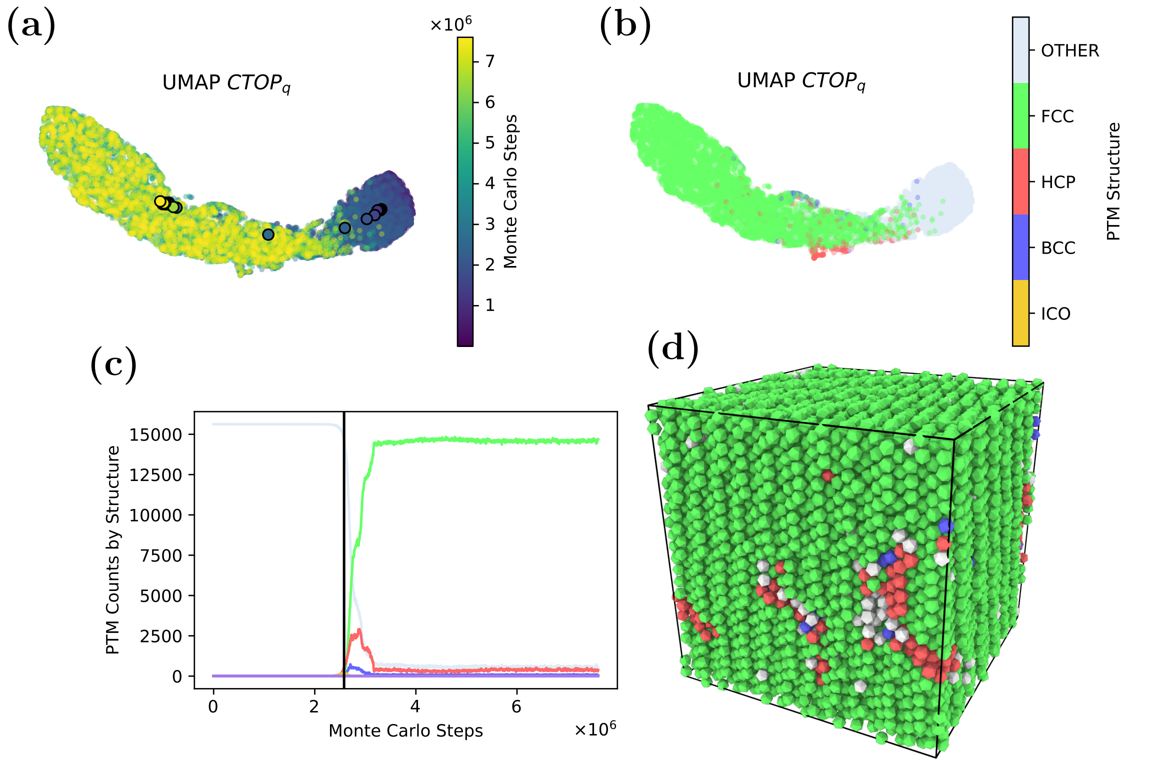


Figure 3.1: **(a)** A 2D UMAP embedding of the 6-dimensional $CTOP_q$ self-assembly pathway of 15,625 hard icosahedra crystallizing from a fluid into the fcc structure $cF4$-Cu. Each point in the scatter plot corresponds to the $CTOP_q$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from fluid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. **(b)** A 2D UMAP embedding of the 6-dimensional $CTOP_q$ as shown in (a), colored by a reference analysis method, Polyhedral Template Matching, computed by OVITO [8, 9]. This demonstrates that clusters of the $CTOP_q$ embedding correspond to local environments, with the one side of the image predominantly marked as "Other" (fluid particles) while the other side is predominantly marked as fcc (solid particles). The hcp defects, shown in red, cluster in an area slightly away from the fcc part of the manifold, because hcp particles share many (but not all) symmetries with fcc particles. **(c)** The reference analysis method, Polyhedral Template Matching, shows the emergence of particles in an hcp local environment, followed by the formation of a predominantly fcc phase. The black vertical line indicates the step at which the target packing fraction of $\phi = 0.52$ was reached. See (b) for the structure corresponding to each color. **(d)** Snapshot of the final frame of the simulation. Particles are colored by their environment type (fcc-like in green, hcp-like in red, bcc-like in blue). The simulation protocol is described in Section 3.1.1. The system is slowly compressed to a target packing fraction over the course of the simulation.

simulation starts in a low density fluid, and is slowly compressed to a target packing fraction of $\phi = 0.52$. As the system is compressed, it begins to crystallize, first showing hcp-like local environments and later a predominantly fcc phase. This system shows how the $CTOP_q$ order parameter captures a continuous transition as the system becomes more ordered, while the Polyhedron Template Matching (PTM) reference method classifies each particle discretely according to a threshold. For the systems shown in this chapter, the PTM threshold is set to 0.15, which represents the root mean squared deviation (RMSD) between the measured local environment and the nearest PTM reference structure (e.g., fcc, bcc, hcp). This PTM threshold value is chosen empirically, to reduce the number of fluid particles misclassified as a solid structure.

We can make a few key observations about the UMAP plots in 3.1, which also hold for the analyses that follow:

1. The embedding shows a continuous manifold. This continuity is preserved by UMAP from the high-dimensional CTOP space. Though some topological features present in the CTOP manifold cannot be easily embedded in 2D, we present the data in a 2D embedding for the sake of interpretability. Some features emerge more readily in 3D embeddings, and will be discussed where relevant. Additionally, the apparent continuity indicates that the CTOP feature space is well-sampled: we cannot expect (and generally do not observe) a continuous manifold if the CTOP only observes a starting frame and an ending frame. This is because the construction of the UMAP embedding relies on nearest neighbors in the CTOP space, which means that the system should be sampled frequently during the phase transition.

2. Both the CTOP values and the reference analysis method (PTM) change from fluid to solid around the same point in time. This gives us good evidence that the CTOP is accurately detecting the onset of local order.

3. Points that are nearby in the UMAP embedding are also nearby in the high-

dimensional CTOP space. Nearby points have similar symmetries in their local environments as measured by the Minkowski Structure Metrics.

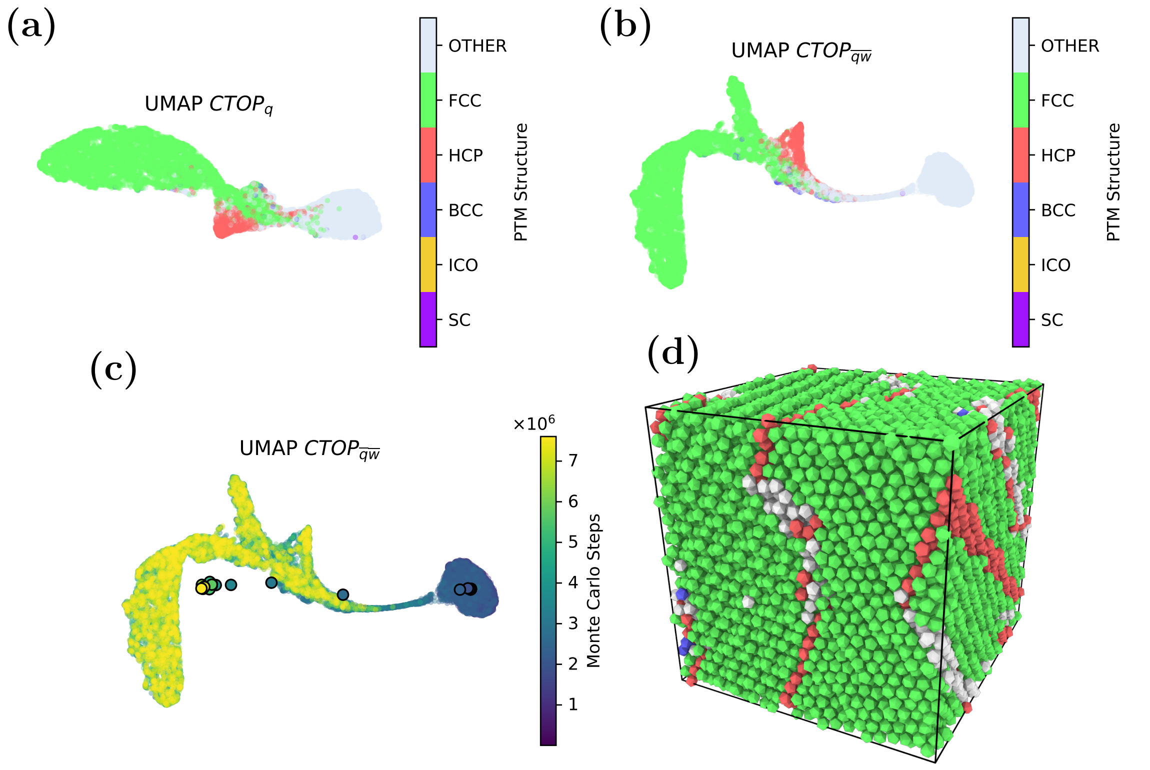


Figure 3.2: **(a)** The $CTOP_q$ embedding of icosahedra self-assembling fcc at a target packing fraction of $\phi = 0.54$. **(b)** The $CTOP_{\overline{qw}}$ embedding shows three clusters: two fcc clusters and one hcp cluster. **(c)** The self-assembly progress as a function of time. Note that zero particles show fluid-like order in their second shell in the final frame. This demonstrates how the addition of second-shell order can assist in separating solid and fluid phases in feature space. **(d)** Snapshot of the final frame of the simulation, which shows hcp stacking faults in the fcc crystal.

Figure 3.2 shows another simulation of icosahedra, this time compressed to a slightly higher packing fraction. In this simulation, a fraction of hcp particles persist as a stacking fault until the final frame of the simulation, which appears as a red cluster. In part (b) of the figure, we embed into a new descriptor space, $CTOP_{\overline{qw}}$. This variant of the order parameter incorporates an average of q_l' values over the second shell of neighbors, as well as averaged values for the w_l' third-order Minkowski Structure Metrics. Part (b) shows that the addition of a second shell average splits the fcc particles into a group of bulk fcc particles and fcc particles that are adjacent to the hcp stacking fault. If the hcp layer and defects were removed, only one cluster would remain in the $CTOP_{\overline{qw}}$ feature space, because every particle would see the same bulk fcc structure in its first and second neighbor shells.

We also note that part (c) of the figure shows no particles in the initial fluid cluster at the end of the simulation. This is because all the particles, even the ones in a local environment that is neither fcc nor hcp, have well-ordered fcc particles in their second neighbor shell. Thus, the averaged order is nonzero when computing the second-shell averaged Minkowski Structure Metrics $\overline{q}'_l$ and $\overline{w}'_l$.

In some simulations, the second-shell order parameter may appear to be discontinuous, with separate fluid and solid clusters in the embedding. This is because the number of particles with partially ordered second neighbor shells is typically small, resulting in incomplete sampling of that portion of the high-dimensional manifold. For those incompletely sampled regions, the UMAP algorithm is sometimes unable to identify that the two components should be connected. Overall, the second-shell averaged information of $CTOP_{\overline{qw}}$ is most useful for distinguishing fluid from solid particles, whereas the first-shell order of $CTOP_{qw}$ is more powerful at describing each particle's local environment. Particles in the fluid phase will have little correlation between their local environments and those of their neighbors, while solid particles often have high correlation with their neighbors. The second-shell average captures these correlations, leading to distinct fluid and solid signatures.

3.1.3 Self-Assembly of Hard Polyhedra into bcc

In Figure 3.3, we show the self-assembly of cuboctahedra into the $cI2$-W (bcc) structure. We show these results in the $CTOP_q$ feature space. Figure 3.3(b) shows how the local structures identified by Polyhedral Template Matching correspond to

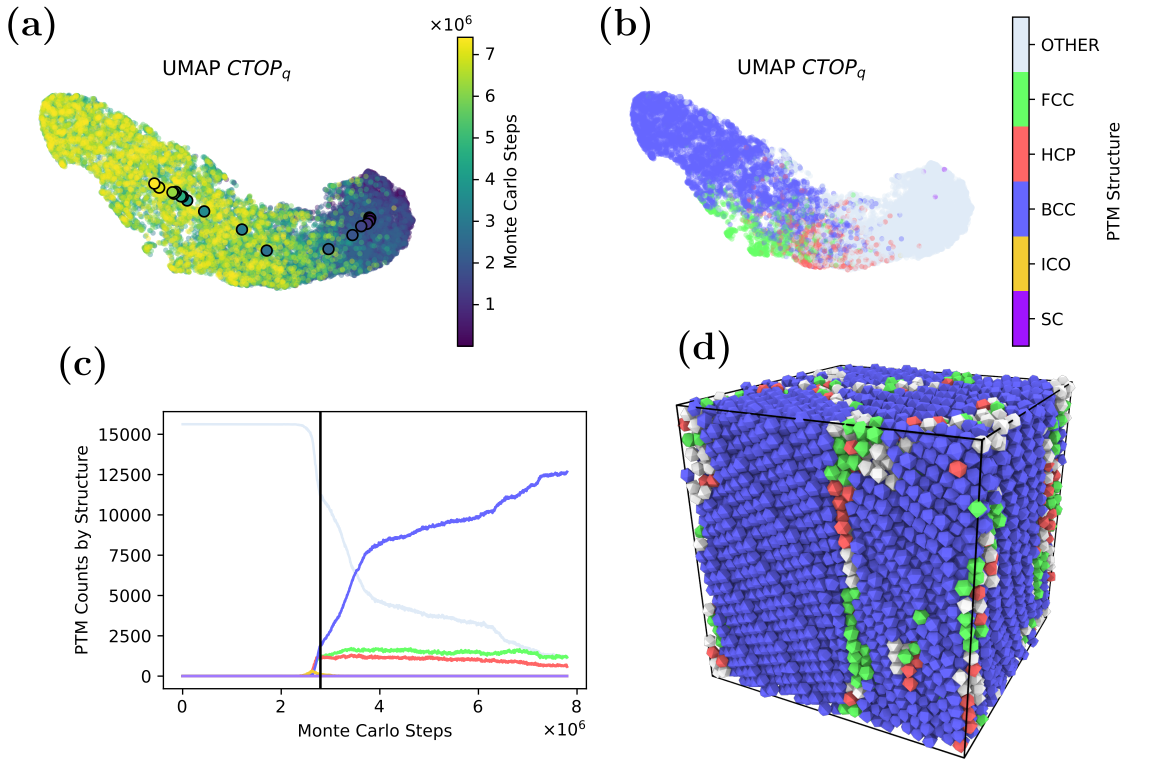


Figure 3.3: **(a)** A 2D UMAP embedding of the 6-dimensional $CTOP_q$ self-assembly pathway of 15,625 hard cuboctahedra crystallizing from a fluid into the bcc structure $cI2$-W. Each point in the scatter plot corresponds to the $CTOP_q$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from fluid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. **(b)** UMAP embedding of $CTOP_q$, colored by a reference analysis method, Polyhedral Template Matching, computed by OVITO [8, 9]. Here, the particles identified as fcc and hcp environments persist into the final frame shown in (d). **(c)** The reference analysis method, Polyhedral Template Matching, shows the emergence of particles in an hcp local environment, followed by the formation of a predominantly fcc phase. The black vertical line indicates the step at which the target packing fraction of $\phi = 0.63$ was reached. See (b) for the structure corresponding to each color. **(d)** Snapshot of the final frame of the simulation. See (b) for the structure corresponding to each color. The simulation protocol is described in Section 3.1.1. The system is slowly compressed to a target packing fraction over the course of the simulation.

contiguous regions of the embedding, just like in the previous example with fcc self-assembly. Like the fcc example we investigated, there are multiple grains separated by particles with local structures different from the bulk phase. In Figure 3.4, we show how clusters in the $CTOP_q$ space can be used to identify these kinds of different local environments.

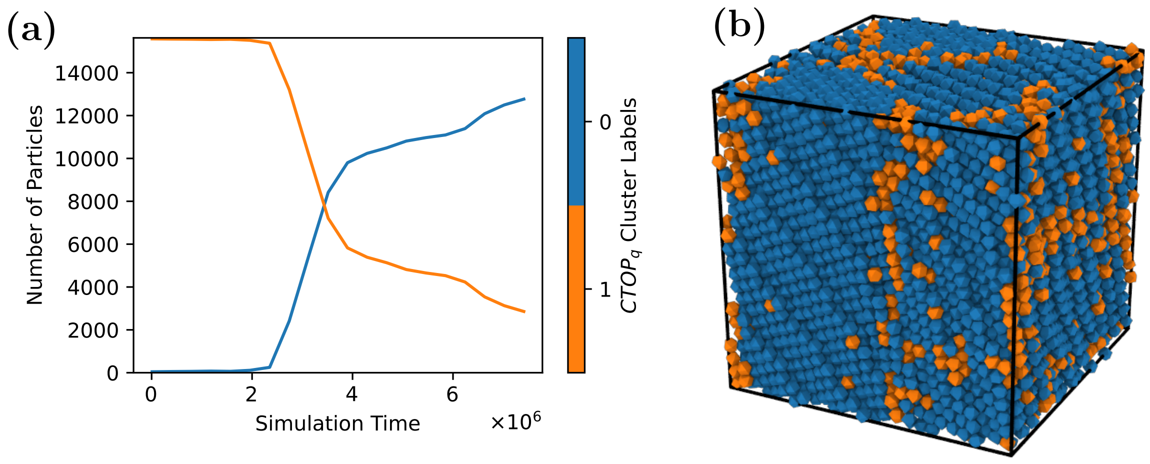


Figure 3.4: **(a)** Number of particles in each cluster of the $CTOP_q$ feature space. The trend for environment 0 (blue) closely follows the number of bcc-like particles identified by Polyhedral Template Matching in Figure 3.3(c). **(b)** A rendering of the final simulation frame, colored by clusters in the $CTOP_q$ feature space. The particles in environment 1 (orange) correspond to the same regions as the fcc, hcp, and "other" particles seen in Figure 3.3(d).

3.1.4 Self-Assembly of Hard Polyhedra into sc

In Figure 3.5, we show the self-assembly of cubes into the simple cubic structure, with a target packing fraction of 0.61. In this instance, the $CTOP_{\overline{qw}}$ order parameter in Figure 3.5(b) exhibits two distinct clusters of fluid and simple cubic particles. However, no explicit instructions were given to analyze simple cubic order – this result comes directly from the CTOP manifold structure.

For the analysis in Figure 3.5(c), the cuML library was used to compute clusters using the k-means algorithm [72, 5, 68]. From the two clusters in the $CTOP_{\overline{qw}}$ embedding space, we calculated the number of solid particles in each frame (without relying on the reference data from Polyhedral Template Matching). Additionally, the freud library was used to find the largest contiguous cluster (a cluster of particles in real space, not feature space) of solid particles using a cutoff distance of $1.3a$, where a is the side length of the cube.

Sharma et al. have previously demonstrated that the disorder-to-order phase

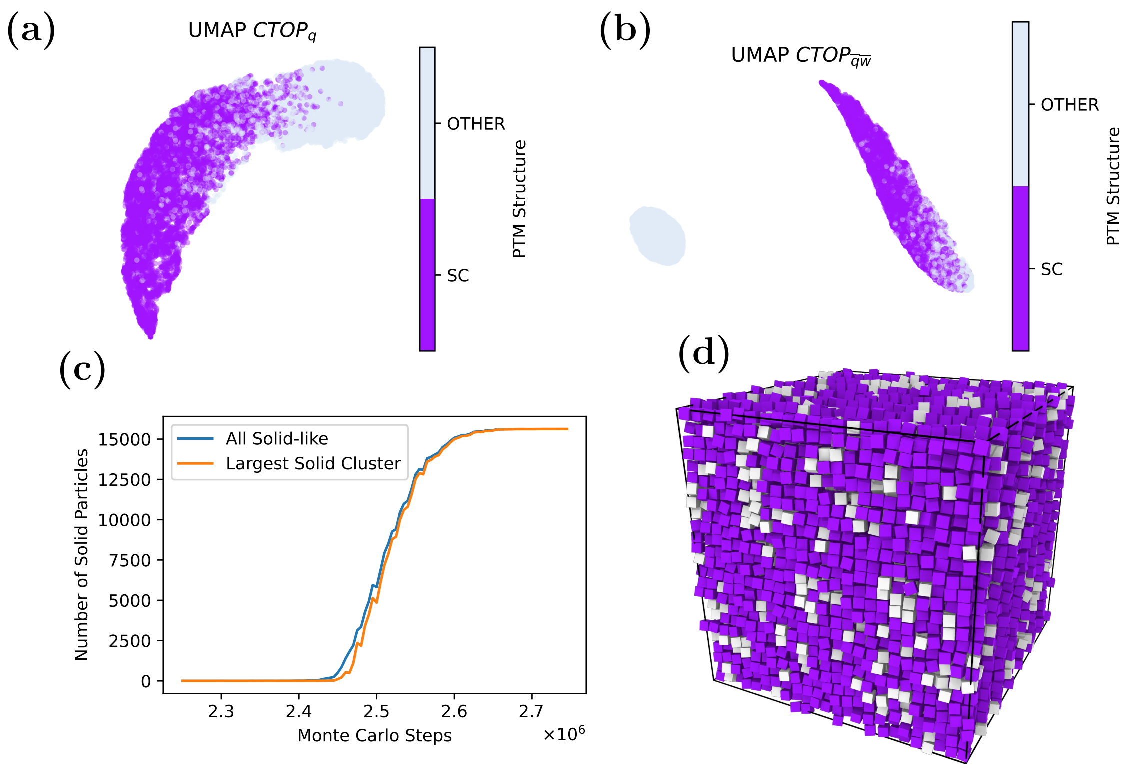


Figure 3.5: Self-assembly of cubes into a simple cubic structure at a packing fraction of 0.61. **(a)** UMAP embedding of $CTOP_q$, colored by Polyhedral Template Matching. **(b)** UMAP embedding of $CTOP_{\overline{qw}}$, colored by Polyhedral Template Matching. **(c)** Counts of all particles in the solid cluster of (b), and counts of the largest contiguous cluster of solid-like particles. See text for details. **(d)** Snapshot of the final frame of the simulation.

transition in colloidal cubes exhibits multiple ordered domains that appear at the same time [73]. We note that this simulation is compressed to a higher packing fraction and has a high degree of supersaturation. Thus, the collective ordering transition does not last very long and the ordered domains (though briefly visible) merge quickly. Nonetheless, the process of characterizing crystallization progress autonomously with the CTOP order parameter is applicable to many systems.

3.2 Self-Assembly of Complex Structures

In the previous section, we demonstrated how the CTOP order parameter can be used in conjunction with a reference method, Polyhedral Template Matching. This served as a comparison for visualization purposes, with classifications that closely follow the CTOP manifold. However, most of the “standard” order parameters (those not used in conjunction with machine learning) cannot be used for analyzing complex crystal structures. This is partly true for historical reasons – the interest in quantifying fcc, bcc, sc, and other simple structures is large because of the large number of systems that form those structures, especially in atomistic materials modeling. Methods designed for identifying complex unit cells and unusual symmetries are rare, because those do not occur as frequently in commonly studied materials.

As mentioned in Section 2.3.1, recent literature has used supervised machine learning methods, such as neural networks, to identify local structure in complex crystals [55, 54, 29]. However, this approach requires prior knowledge of the target structure, initial structure, and any common structures that might arise as defects or competing motifs. In most supervised methods, researchers must configure and run separate simulations for equilibrium states of each target structure (sometimes for a range of temperatures or other thermodynamic variables that affect positional disorder), load all the data, and finally train the classifier. By contrast, because CTOP is an unsupervised method, it can be computed from a single trajectory and identify local environments with no prior training.

Previous applications of unsupervised machine learning methods in the literature have not explicitly considered how the choice of features influences the topology and continuity of feature space, which in turn affects the interpretability of the manifold structure via dimensionality reduction. Here, we show that selecting features that form a continuous feature space and adhere to physical invariants can dramatically simplify the analysis of many complex crystal structures. In this section, we will apply the CTOP order parameter to a selection of complex crystals and quasicrystals previously studied by the Glotzer group, demonstrating how it can provide information about local structure and solidification progress.

3.2.1 A15 Structure ($cP8$-Cr$_3$Si)

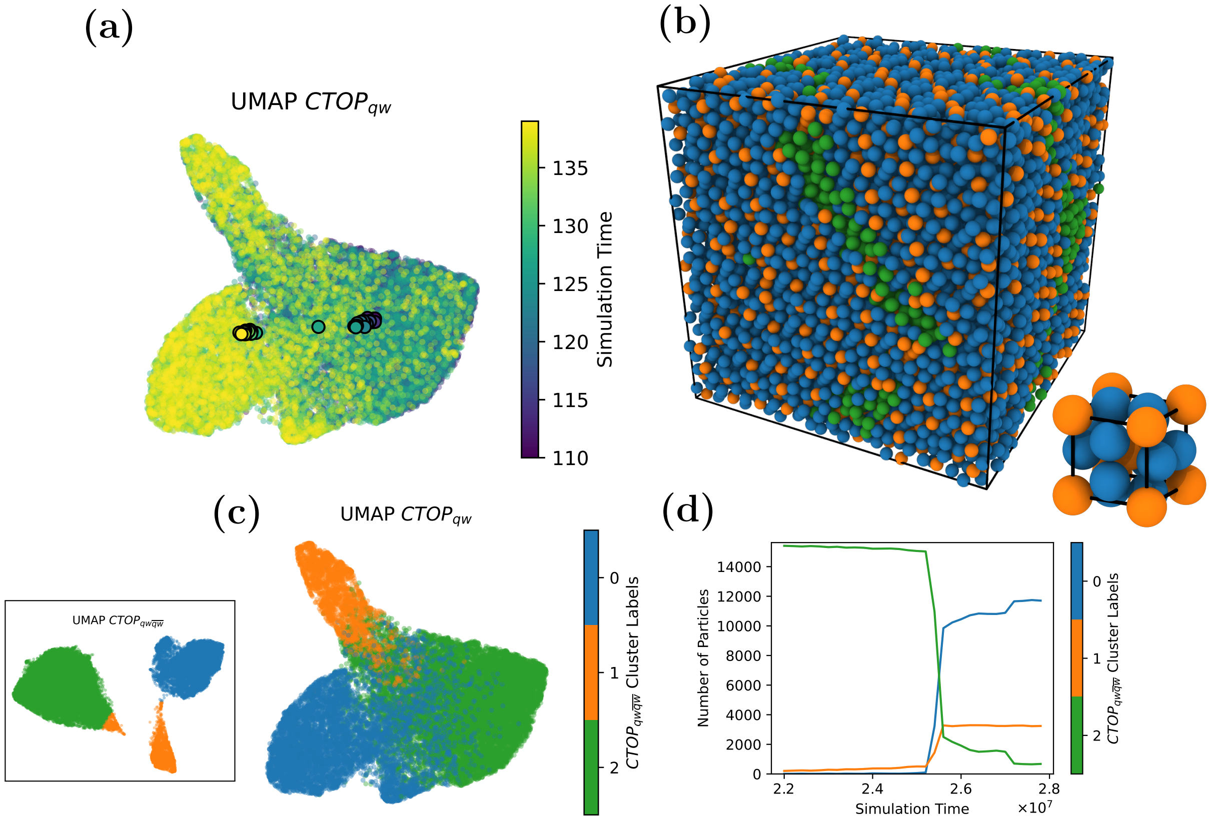


Figure 3.6: **(a)** UMAP embedding of the 9-dimensional $CTOP_{qw}$ self-assembly pathway of 15,625 particles crystallizing from a fluid into the A15 structure $cP8$-Cr$_3$Si. Each point in the scatter plot corresponds to the $CTOP_{qw}$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from liquid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. Each type of local environment present in the final structure (Cr-like and Si-like) appears as a cluster in the embedding. **(b)** Snapshot of the final frame of the simulation. The inset shows the A15 unit cell [10]. Particles are colored by their environment type cluster, shown in (c) (Cr-like in blue, Si-like in orange). The defect and liquid-like particle environments (colored green) form a grain boundary. **(c)** Clustering by $CTOP_{qw\overline{qw}}$ readily shows the dominant clusters' local environments. The inset shows the $CTOP_{qw\overline{qw}}$ feature space. **(d)** This shows the counts of each cluster in $CTOP_{qw\overline{qw}}$ as a function of time. The Cr and Si environments appear at the same time, and are in roughly a 3:1 proportion as is expected from the chemical formula of the prototype structure.

Figure 3.6 shows the assembly pathway of the A15 structure $cP8$-Cr$_3$Si via $CTOP_{qw}$ and clustered by $CTOP_{qw\overline{qw}}$. This simulation data was provided by Carl Simon Adorf with a pair potential optimized using an inverse design called Fourier-Filtered Relative Entropy Minimization (FF-REM) to produce the A15 structure [15]. This data was previously analyzed by Adorf et al. using unsupervised machine learning methods [30]. See that previous paper for more information about the simulation protocol. In this section, we demonstrate how the CTOP analysis provides an improved understanding of the particles' local environments, with stronger ability to distinguish between fluid-like environments and environments with icosahedral order.

Figure 3.6(a) shows the time evolution of particles' local environments embedded in the $CTOP_{qw}$ feature space.[2] The Uniform Manifold Approximation and Projection (UMAP) dimensionality reduction technique produces a two-dimensional embedding of the 9-dimensional $CTOP_{qw}$ feature vector while preserving its topological structure. With this method, the transition pathways (e.g., between a fluid and a crystal) form a continuous path in the embedding space. In Figure 3.6(b), we see the final image of the crystal structure, colored by clusters found in the $CTOP_{qw\overline{qw}}$ descriptor space, shown in (c). The most prominent environments in the final self-assembled structure are Cr-like (blue) and Si-like (orange), corresponding to the dominant Wyckoff positions in the crystal. We use the second-shell descriptors for clustering because the topology of that feature space clearly demonstrates two features Figure 3.6(d) shows the time evolution of particles' local environments. The system starts as a fluid (green), and both Cr-like and Si-like environments appear at the same time. Later, around 27 million time steps, there is an increase in the Cr-like environment and a simultaneous decrease in the fluid environment, suggesting that local rearrangement led to a shift in the detected environments. The clusters colored in this figure were found by k-means clustering from the cuML library [68]. The number of clusters was chosen manually. A future direction for this research

[2]Nearly equivalent results can be found for embeddings in the $CTOP_q$ feature space for this structure.

would be to choose the number of clusters automatically, which was skipped in the current work because of the inherent difficulties of identifying cluster boundaries in a continuously sampled space.

Compared to the previous analysis by Adorf et al. that used bispectrum descriptors, the CTOP approach has a few key differences. First, the CTOP approach does not require any tuning for the identification of neighbor shells because it uses the parameter-free Voronoi neighbors. Adorf et al. selected 12, 13, 24, and 26 nearest neighbors to match the first and second neighbor shells. However, the diversity of complex binary structures may not always fit well into an analysis paradigm that requires a defined number of neighbors for the descriptors. The use of Minkowski Structure Metrics in the CTOP descriptors and Voronoi neighbors (which are parameter free) reduces the number of parameters that must be chosen by hand for each structural analysis. Second, the environments found are not the same (refer to Figure 6 of Ref. [30]). Both the CTOP analysis and the bispectrum analysis locate a fluid-like environment in the early stages of crystallization, but this environment does not persist in the bispectrum analysis and *does* persist (at the grain boundary) in the CTOP analysis. While the CTOP approach appears to find two crystalline environments in the 3-to-1 stoichiometric ratio expected for the $cP8$-Cr_3Si structure, the bispectrum analysis appears to conflate particles in a grain boundary with the Si-like environment. The "intermittently dominant icosahedral environment" of the bispectrum analysis is never dominant in the CTOP analysis. Adorf et al. links this environment to Wyckoff site a (the Si-like environment), which the CTOP analysis observes as its own cluster appearing simultaneously with Wyckoff site c (the Cr-like environment) in an approximately 1:3 stoichiometric ratio. No temporarily dominant environment appears during the crystallization process for any of the feature spaces ($CTOP_q$, $CTOP_{qw}$, $CTOP_{\overline{qw}}$, and $CTOP_{qw\overline{qw}}$) or numbers of clusters (2 to 6) that were analyzed. We conclude that the CTOP approach differentiates itself from past unsupervised methods by three factors: continuity in feature space, fewer structure-specific parameters to tune, and identified clusters that align with the expected stoichiometric proportions.

3.2.2 Diamond ($cF8$-C)

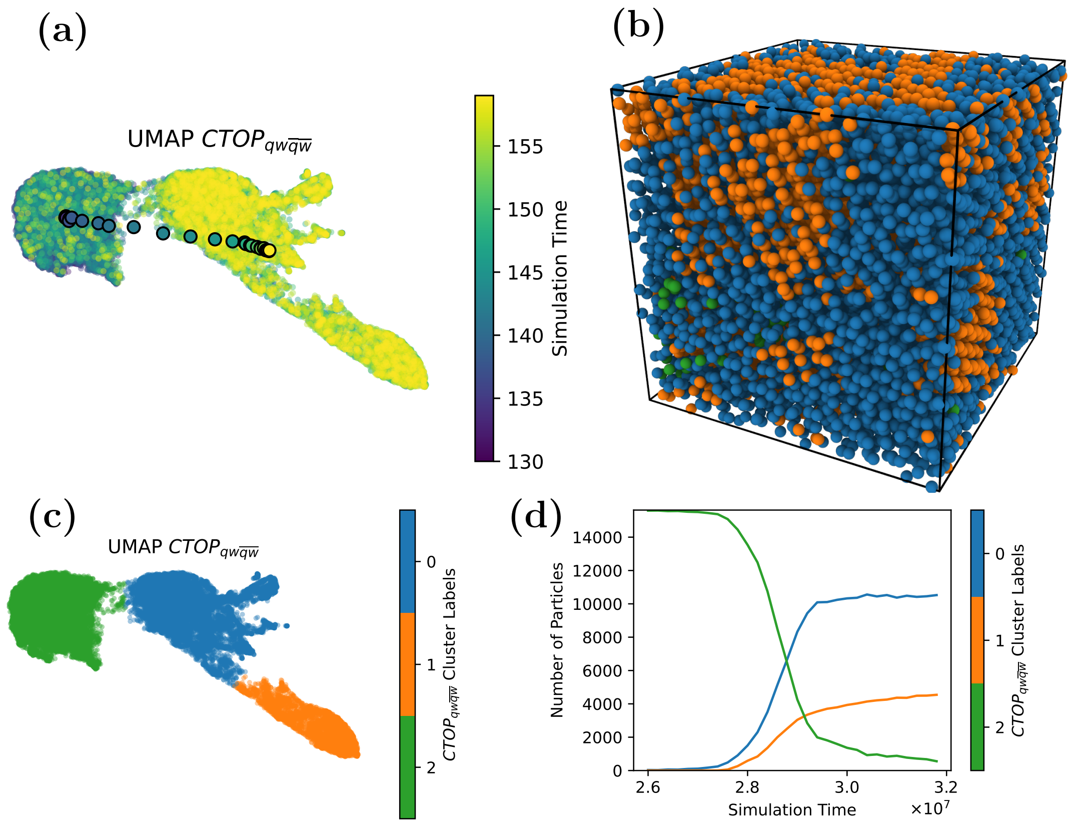


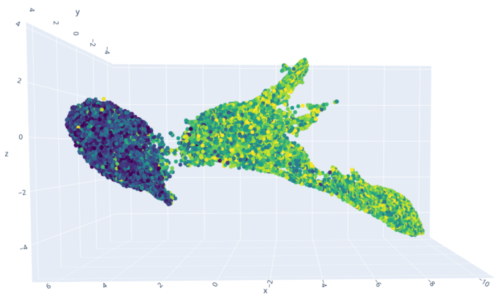
Figure 3.7: **(a)** UMAP embedding of the 18-dimensional $CTOP_{qw\overline{qw}}$ self-assembly pathway of 15,625 particles crystallizing from a fluid into the diamond structure $cF8$-C. Each point in the scatter plot corresponds to the $CTOP_{qw\overline{qw}}$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from liquid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. **(b)** Snapshot of the final frame of the simulation. Particles are colored by their environment type cluster, shown in (c). The dominant environment 0 particles have not crystallized to the $cF8$-C local structure, described further in the text. The diamond-like environment 1 is colored orange. The fluid-like particle environment 2 is colored green. **(c)** Clustering by $CTOP_{qw\overline{qw}}$ readily shows the dominant clusters' local environments. The inset shows the $CTOP_{qw\overline{qw}}$ feature space. **(d)** This shows the counts of each cluster in $CTOP_{qw\overline{qw}}$ as a function of time. The Cr and Si environments appear at the same time, and are in roughly a 3:1 proportion as is expected from the chemical formula of the prototype structure.

In Figure 3.7, we show the self-assembly of another FF-REM pair potential, this time optimized to form the $cF8$-C diamond structure. This simulation data, provided by Carl Simon Adorf, offers an interesting demonstration of the CTOP method when the system does not fully transition from fluid to solid. The simulation does not fully self-assemble, but instead freezes into a state with some particles in a cubic diamond environment shown in orange (cluster 1, verified by Polyhedral Template Matching). The remainder of the particles shown in blue (cluster 0) have different, lower symmetries that are not diamond-like but are distinguishable from the fluid in green (cluster 2) only by their increase in second-shell order. The onset of freezing leaves the system in a state that is out of equilibrium but changes slowly. Figure 3.7(d) shows a slowly increasing number of diamond-like particles in cluster 1. While the simulation never fully self-assembles the diamond structure, the environments detected by CTOP reflect the expected target structure.

The smaller scale structure present in the blue cluster 0 presents us with a question: what symmetries differentiate the regions contained in that cluster? Looking at a 3D embedding in Figure 3.8, multiple branches emerge from the middle region previously identified as cluster 0. Investigating the raw Minkowski Structure Metrics' values indicates that this middle region has q_l' and w_l' values that are very similar to the fluid-like cluster 2 (green), but separates itself from the fluid in this embedding because of strong distinctions in $\overline{q}_l'$ and $\overline{w}_l'$ features that indicate ordering in the second shell neighborhoods. As the system is cooled, some particles are able to locate the target free energy minimum corresponding to the $cF8$-C (diamond) structure for which the potential was designed. However, some particles instead fall into local minima that have less order. While looking only at first-shell descriptors $CTOP_{qw}$, we cannot distinguish the blue and green clusters we see in the $CTOP_{qw\overline{q}\overline{w}}$

descriptor space, we hypothesize that the emergence of second shell order is actually a manifestation of many particles and their neighbors finding local minima in the free energy landscape with some level of order that is less than that of $cF8$-C. Particles in a fluid do explore local minima like these in cluster 0, but only display second-shell order like this when the particle *and* its neighbors lack the thermal energy needed to escape the minima via fluctuations.

(a)



(b)

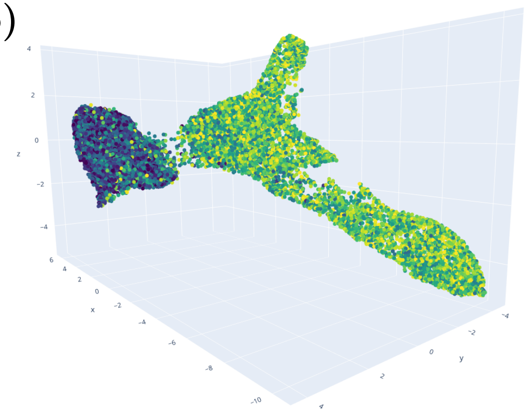


Figure 3.8: **(a)** 3D UMAP embedding of the 18-dimensional $CTOP_{qw\overline{qw}}$ self-assembly pathway of 15,625 particles crystallizing from a fluid into the diamond structure $cF8$-C. Each point in the scatter plot corresponds to the $CTOP_{qw\overline{qw}}$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from liquid (purple and dark blue-green) to solid (light green and yellow). Visualization generated using Plotly [7]. **(b)** Another view of the embedding in (a).

We also note that the diamond environment (cluster 1, orange) appears to be nearly two-dimensional when embedded in 3D. Using Principal Components Analysis on that cluster, we identify that the primary defining features of that submanifold are $\overline{q}'_{10}$, $-q'_5$, $\overline{q}'_4$, and $\overline{q}'_6$ along the first principal component and $-q'_5$, $-q'_8$, $-\overline{q}'_4$, and $-\overline{q}'_{10}$ along the second principal component. While this analysis neglects that UMAP's clusters may have nonlinear structure, these principal components give some sense of the geometry within the cluster. Coloring the UMAP by each of the Minkowski Structure Metric features (not shown) indicates positive correlation along the direction of increasing $cF8$-C order for the $\overline{q}'_{10}$, $\overline{q}'_4$, and $\overline{q}'_6$ averaged quantities in the first principal component and negative correlation with q'_5 as indicated in the PCA's first component. Similarly, the direction orthogonal to increasing $cF8$-C order shows negative correlation with q'_5, q'_8, $\overline{q}'_4$, and $\overline{q}'_{10}$ as expected from the PCA analysis. These observations from PCA give evidence that some qualitative conclusions can be made about the local geometry without concern for the global nonlinearity of the UMAP embedding, as long as the submanifold being analyzed is well sampled, has a low-dimensional tangent space, and has low curvature, like this analysis of the diamond environment.

3.2.3 σ Phase (*tP*30-CrFe)

In Figure 3.9, we show the self-assembly of a third FF-REM pair potential from Carl Simon Adorf, optimized to form the $tP30$-CrFe σ phase. This simulation tests the CTOP feature space's ability to recognize complex structures because the alloy σ-CrFe contains five different Wyckoff positions. These Wyckoff positions are labeled A, B, C, D, E by Yakel [74]. Yakel describes that the B ($4f$) and C ($8(i)(1)$) positions are 15- and 14-coordinated, forming a "major skeleton." The E ($8j$) positions form secondary layers. Finally, the A ($2a$) and D ($8(i)(2)$) positions have 12-fold icosahedral coordination.

Initially, we hypothesized that each position would have unique local symmetries and thus result in five clusters in the UMAP embedding of the CTOP feature space. However, we discovered through the CTOP analysis that this is not the case. An important piece of background knowledge for this analysis is that in real $\mathrm{Cr}_x\mathrm{Fe}_{1-x}$ alloys that form this crystal, the occupancy of each site may be either Cr or Fe. That is, iron or chromium atoms are distributed randomly over the Wyckoff positions [10, 74]. This suggests that there is significant similarity in behavior between the two species, which makes it less surprising that a single-component system with the

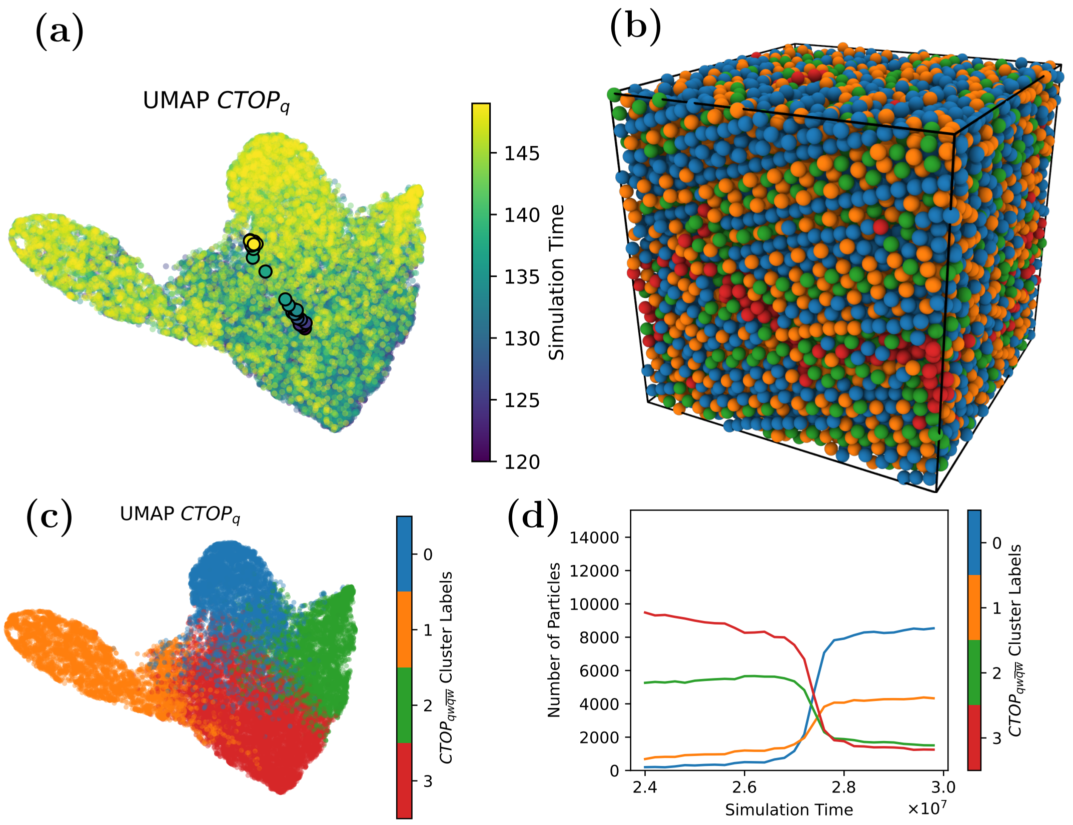


Figure 3.9: **(a)** UMAP embedding of the 9-dimensional $CTOP_q$ self-assembly pathway of 15,625 particles crystallizing from a fluid into the structure $tP30$-CrFe. Each point in the scatter plot corresponds to the $CTOP_q$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from liquid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. **(b)** Snapshot of the final frame of the simulation. Particles are colored by their environment type cluster, shown in (c) and described in the text. **(c)** Clustering by $CTOP_{qw\overline{qw}}$ readily shows the dominant clusters' local environments. The inset shows the $CTOP_{qw\overline{qw}}$ feature space. **(d)** This shows the counts of each cluster in $CTOP_{qw\overline{qw}}$ as a function of time.

FF-REM optimized potential can form this complex structure.[3] While there are 5 crystallographically distinct Wyckoff positions, the local symmetries are similar between some of these environments, resulting in only three clusters in the CTOP feature space shown in Figure 3.9.

Visually inspecting the unit cell of the crystal and comparing with slices of the self-assembled structure allows us to confirm which Wyckoff positions appear in each cluster. Environment 0 (blue) corresponds to the C ($8(i)(1)$) and E ($8(j)$) sites. En-

[3]We note there is also a single-component atomic crystal, β-U, that is claimed to form the same structure [10, 74].

vironment 1 (orange) can be clearly identified with the A ($2a$) and D ($8(i)(2)$) sites, separately confirmed with Polyhedral Template Matching, which shows icosahedral order in this cluster (no other sites had an identifiable structure in the Polyhedral Template Matching analysis). Environment 2 (green) is the B (4(f)) site. Finally, environment 3 (red) appears in boundary regions where the crystal is not fully formed, and is the dominant environment in the fluid before crystallization occurs, along with some of environment 2. This appears to be a conflation of fluid and solid by the clustering algorithm – we believe the true environment 2 should contain far fewer "fluid" particles but it is difficult to form sharp cluster boundaries when the data are distributed continuously and the ideal data density is not known *a priori*. However, we will show that the fluid-solid distinction is made clear when looking only at the $CTOP_{\overline{qw}}$ second-shell average feature space.

We observe in Figure 3.10 that the $CTOP_{\overline{qw}}$ descriptor space, which contains only second-shell averaged Minkowski Structure Metrics, can easily distinguish fluid from crystal environments, but cannot distinguish *between* crystalline environments because averaging over the second neighbor shell destroys the important information from the local environment about which site is represented by a feature vector. The construction of the second-shell average order parameters incorporates some similar information to the solid-liquid order parameter used in previous simulation studies of nucleation, which helps to explain why the second-shell descriptors are able to clearly distinguish fluid and solid particles [75, 76].

3.2.4 Decagonal Quasicrystal

Figure 3.11 shows the self-assembly of a decagonal quasicrystal from a single seed. This simulation data was provided by Kelly Wang as part of a study of self-healing behavior in quasicrystal grains [77]. The $CTOP_{qw\overline{qw}}$ features for the particles are able

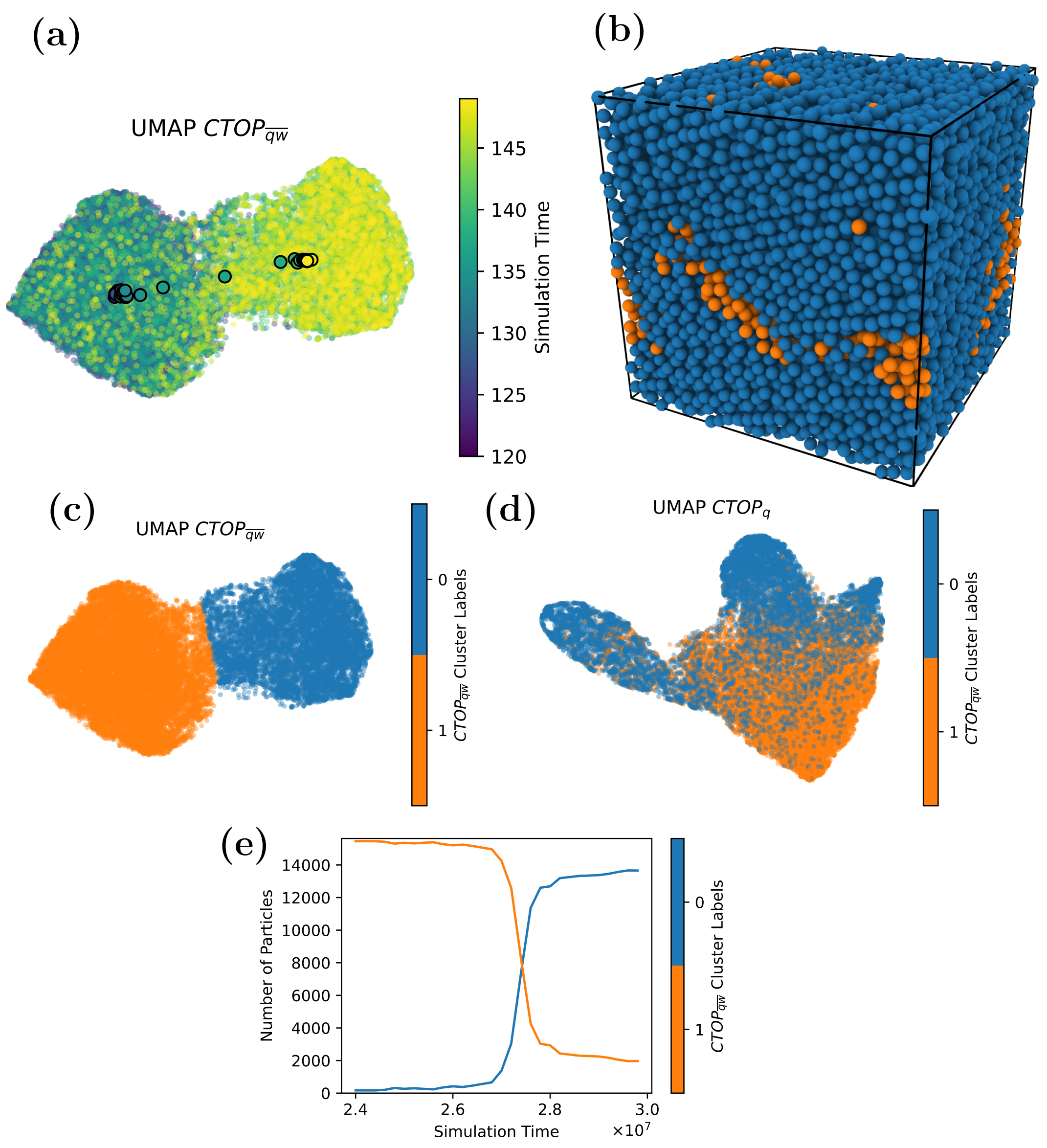


Figure 3.10: **(a)** UMAP embedding of the 12-dimensional $CTOP_{\overline{qw}}$ self-assembly pathway of 15,625 particles crystallizing from a fluid into the structure $tP30$-CrFe. Each point in the scatter plot corresponds to the $CTOP_q$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from liquid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. **(b)** Snapshot of the final frame of the simulation. Particles are colored by their environment cluster. Solid-like particles in environment 0 are blue. Fluid-like and defect particles in environment 1 are orange. **(c)** Clustering by $CTOP_{\overline{qw}}$ distinguishes fluid from solid, but cannot identify the Wyckoff positions or their local symmetries. **(d)** The $CTOP_q$ embedding previously shown in Figure 3.9, colored by clusters in the $CTOP_{\overline{qw}}$ space. **(e)** The counts of each cluster in $CTOP_{\overline{qw}}$ as a function of time, showing the fluid to crystal transition with only a small number of fluid-like particles remaining as defects in the crystal structure shown in (b).

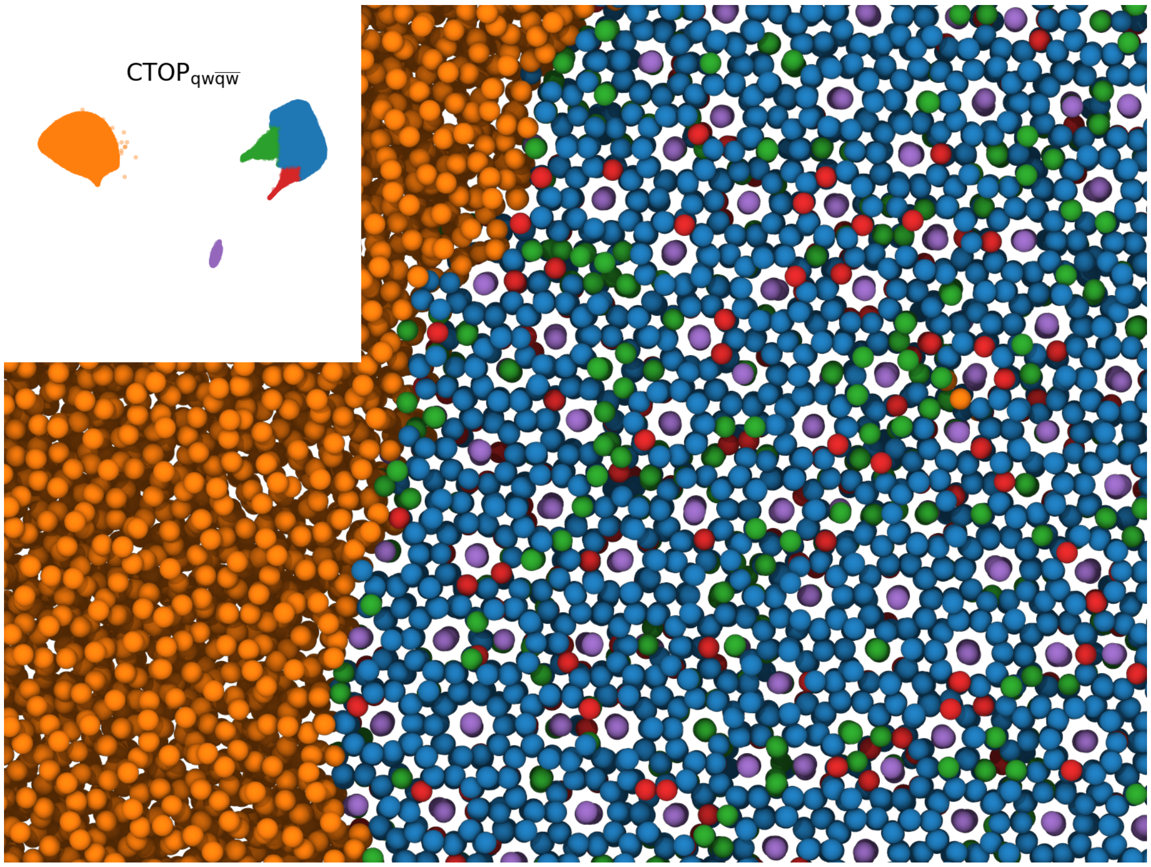


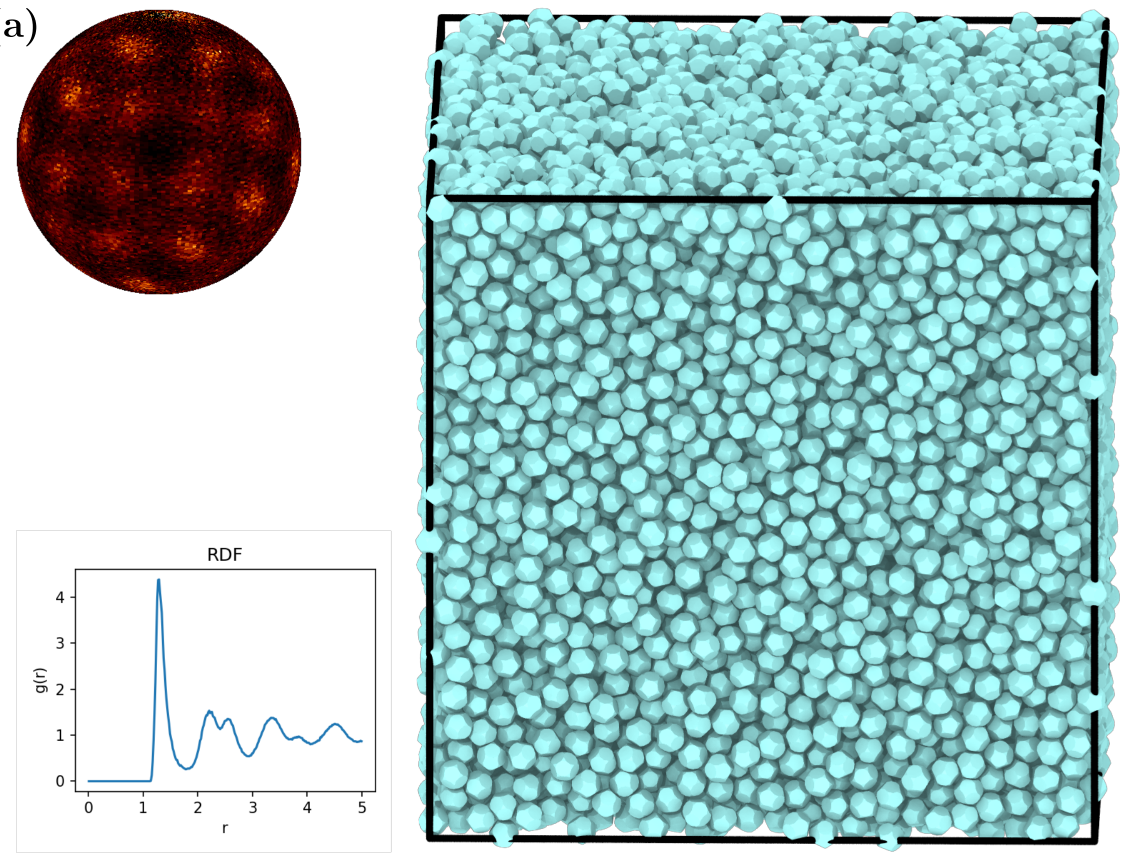
Figure 3.11: UMAP embedding of a decagonal quasicrystal self-assembling. Simulation data provided by Kelly Wang.

to distinguish between fluid-like and solid-like particles, as well as identify a number of unique environments present in the structure. These local environments appear to correspond to decagonal centers (purple), decagonal edges (blue), pinched motifs (green), edge defects (red), and fluid environments (yellow-orange). This simulation is the largest single trajectory we analyze, with 500,074 particles and a file size of 3.32 GB. These clusters are selected by hand, rather than with k-means clustering, due to difficulties with software compatibility and large data size.

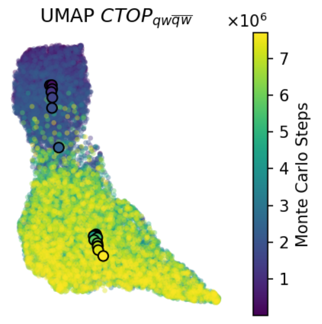
3.2.5 β-Mn Structure

In Figure 3.12, we show the self-assembly of hard truncated dodecahedra into the β-Mn crystal structure. This simulation, run alongside the other hard polyhedra

(a)



(b)



(c)

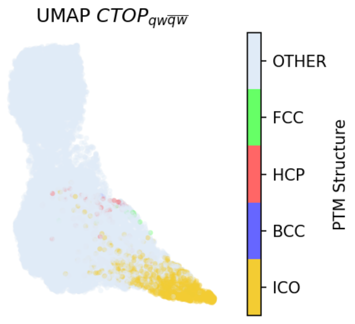


Figure 3.12: **(a)** Snapshot of the self-assembled β-Mn structure formed by 15,625 truncated dodecahedra. Particles are colored arbitrarily for high contrast. The bond order diagram in the upper left and radial distribution function in the lower left confirm that the obtained structure is β-Mn. **(b)** UMAP embedding of the 18-dimensional $CTOP_{qw\overline{qw}}$ self-assembly pathway of 15,625 particles crystallizing from a fluid into the β-Mn structure. Each point in the scatter plot corresponds to the $CTOP_{qw\overline{qw}}$ embedding for one of the particles at a given point in time. Color indicates simulation time, as the system progresses from liquid (purple and dark blue-green) to solid (light green and yellow). Solid circles show the mean of the data over time. **(c)** The $CTOP_{qw\overline{qw}}$ UMAP embedding, colored by Polyhedral Template Matching.

simulations discussed in Section 3.1, is an example of a complex crystal formed solely by entropy [23]. In this simulation, the $CTOP_{qw\overline{qw}}$ order parameter identifies changes in the particles' local environments that correspond to the onset of crystalline order that cannot be detected through the Polyhedral Template Matching shown in Figure 3.12(c). Many particles in the solid are labeled as "other" by the Polyhedral Template Matching algorithm, or are misclassified as hcp if the RMSD tolerance of the Polyhedral Template Matching algorithm is increased. However, the $CTOP_{qw\overline{qw}}$ demonstrates a clear separation between fluid and solid, and additionally shows a wide spread among the solid particles' local environments. This wide spread is expected from past results, because it corresponds to two Wyckoff positions discussed below.

A similar shape and density has previously been studied in Chapter V of Ref. [78]. This previous study indicated that the two Wyckoff positions of the structure have different local volumes, and in particular that the Mn1 position appears to have a higher local density (smaller Voronoi volume) and is closer to an ideal icosahedral environment. Similarly to the past study, we observe some particles in icosahedral environments, but there is no clear spatial ordering among the icosahedral particles identified by Polyhedral Template Matching. The wide spread of solid-like local environments represents both of these Wyckoff positions, because they have similar local symmetries and thus their distributions in the embedding overlap with one another.

3.3 Comparisons of Pathways

In this section, we demonstrate how the CTOP and UMAP methods can be combined to compare the pathways of different simulations. We call the embedding of multiple pathways in a single UMAP a "co-embedding." The previous UMAP embeddings were generated by computing the CTOP order parameter for particles in each frame of the simulation, then concatenating the data and computing the UMAP embedding of those CTOP vectors over all time. Similarly, we can concatenate the CTOP data from *multiple trajectories* before computing the UMAP to produce a co-embedding.

3.3.1 Monte Carlo and Molecular Dynamics Reveal Similar Paths

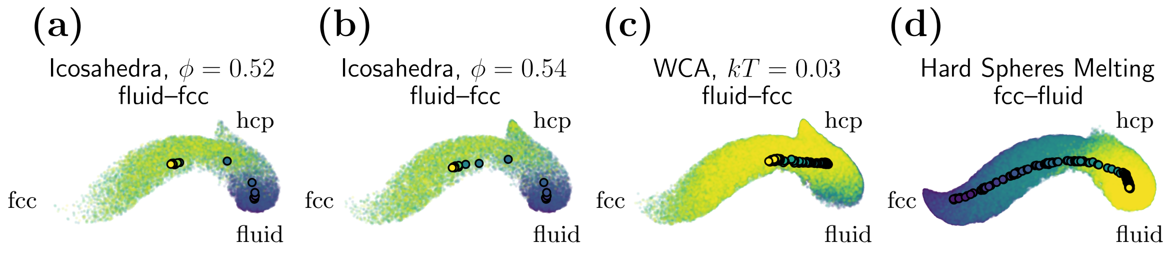


Figure 3.13: A co-embedding of four different simulations transitioning between fluid and fcc structures. The simulation times are normalized such that the dark purple color is the initial state and light yellow is the final state. **(a)** UMAP of the hard icosahedra compressed to $\phi = 0.52$, previously shown in Figure 3.1. **(b)** UMAP of the hard icosahedra compressed to $\phi = 0.54$, previously shown in Figure 3.2. **(c)** UMAP of WCA spheres at $kT = 0.03$. WCA self-assembly simulation data provided by Allen LaCour. **(d)** A simulation of hard spheres initialized in a close-packed fcc structure and slowly expanded, allowing the solid to melt into a fluid. Note that the color scheme is reversed for this simulation because it begins in an fcc solid (dark purple) and melts to the fluid (yellow).

In Figure 3.13, we show a co-embedding of four systems. These four systems all start or end in an fcc phase, but the simulations are performed in very different ways. The embeddings in parts (a) and (b) are from hard icosahedra. Part (c) shows a molecular dynamics trajectory with particles interacting via the Weeks-Chandler-Anderson potential to form the fcc structure. Part (d) shows a hard sphere simulation that is *melting into a fluid*, rather than undergoing self-assembly into a solid. This shows a combination of molecular dynamics and Monte Carlo simulation methods, as well as both self-assembly (freezing) and melting trajectories.

Overall, we see that the paths in these simulations are similar. However, we can make several more detailed observations about the CTOP approach from this co-embedding. First, we note that the melting simulation in (d) starts in a “perfect” close-packed fcc phase of hard spheres. The shape of the co-embedding places those particles at the farthest point, because the symmetries are the most ideal and least fluid-like. As the system gets less dense (but remains stable in the fcc phase), the noise in particles’ environments makes the local environments’ symmetries less perfect and thus the particles move towards the fluid phase at the bottom, long before melting. This shows that the CTOP feature space can be used to study local distortion or deformation in the Voronoi polyhedra due to noise (which may be thermal noise or simply hard particles exploring the nearby free volume of the system).

Second, we see that the structures in part (b) and (c) show a dominant hcp peak, while this is not seen as clearly for the embedding in part (a), at least in the final time in yellow, and not at all for (d). This is because of the present of hcp layer defects in both (b) and (c), while hcp environments only appear for a short time in (a) before annealing out. The melting simulation in (d) does not have any significant number of hcp environments appear during or after melting from its original fcc phase.

3.3.2 Comparing fcc, bcc, and sc Structures

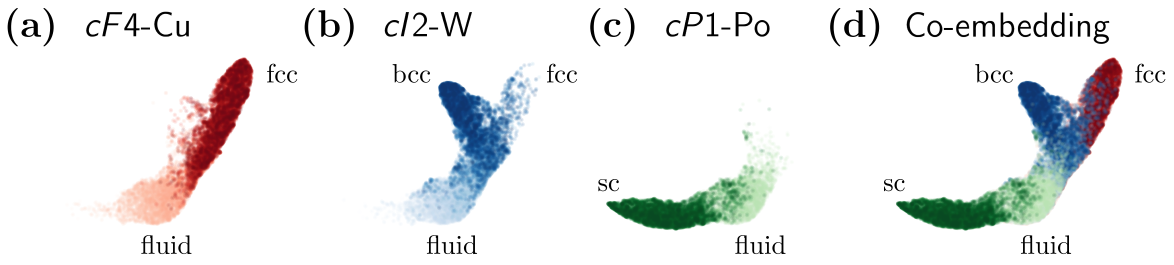


Figure 3.14: A co-embedding of three different simulations in the $CTOP_q$ feature space. The simulation times are normalized such that the lighter color is the initial state (fluid) and the darker color is the final state (solid). Colors are chosen arbitrarily to help identify the data in the co-embedding. **(a)** UMAP of the hard icosahedra compressed to $\phi = 0.52$, previously shown in Figure 3.1. **(b)** UMAP of the hard cuboctahedra compressed to $\phi = 0.63$, previously shown in Figure 3.3. **(c)** UMAP of the hard cubes compressed to $\phi = 0.61$, previously shown in Figure 3.5. **(d)** The co-embedding with all simulations shown on the same axes.

In Figure 3.14, we show the comparison of three different crystal structures self-assembling from hard polyhedra. The fluid region is shared among all the simulations, because the isotropic fluid state shows no local symmetries. Each crystal structure occupies a unique region of the embedding according to the local environments' symmetries. The small number of fcc-like environments in the simulation shown in (b) are also visible in Figure 3.3.

Importantly, this figure establishes that the $CTOP_q$ feature space is in fact a manifold with a topological structure that captures the progression of local environments during self-assembly in a way that can be compared across different types of structures. We are able to co-embed multiple simulations that self-assemble different structures and observe the formation of many unique types of local order with no training data or prior knowledge of the local environments. This type of generality in the ability to characterize microscopic details of changing local environments across many structures is a novel feature of the approach presented here.

3.3.3 Solid-Solid Transitions

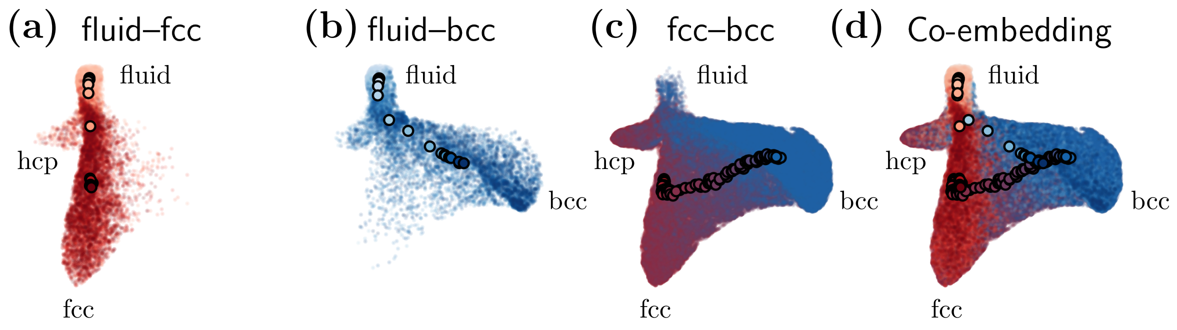


Figure 3.15: A co-embedding of three different simulations in the $CTOP_{qw}$ feature space. Color schemes are chosen arbitrarily to help identify the data in the co-embedding. The average UMAP value for each frame is shown in a series of circles colored by the frame time. **(a)** UMAP of the hard icosahedra compressed to $\phi = 0.52$, previously shown in Figure 3.1. **(b)** UMAP of the hard cuboctahedra compressed to $\phi = 0.63$, previously shown in Figure 3.3. **(c)** UMAP of a solid-solid phase transition (fcc to bcc). Data provided by Chrisy Xiyu Du [11]. **(d)** The co-embedding with all simulations shown on the same axes. The UMAP averages of each trajectory form a triangle for the fluid, fcc, and bcc states.

Solid-solid transitions, such as the martensitic transition [79], permit diffusionless changes between crystal structures. Here, we show that the manifold structure of CTOP has nontrivial cycles formed by the connections between solid phases and the disordered fluid phase. Figure 3.15 shows fcc and bcc self-assembly simulations (results gathered in the Hard Particle Monte Carlo simulations of polyhedra described in Section 3.1) compared to an fcc-bcc phase transition from a study by Du et al. of particles whose shapes are optimized to undergo solid-solid phase transformations as pressure changes [11]. The particles' local environments during this transition break some symmetries while gaining others. Looking at the co-embedding of this fcc-bcc transition with two self-assembly simulations forming the fcc and bcc phases from a disordered fluid, we see that the local environments of the solid-solid transition do not resemble those of the disordered fluid. These three distinct pathways exist in the "local structure space" implied by the manifold embedding, and form a triangular shape shown in part (d). While this set of pathways can be embedded into 2D, adding more transitions between solid phases would likely require additional projection dimensions to adequately represent the manifold. Graphs have planar embeddings if they do not contain the subgraphs K_5 or $K_{3,3}$, but we have observed that real data in a UMAP may have more complicated geometries that are not easy to embed into the plane.

3.3.4 Comparing Many Pathways

In Figure 3.16, we show a collection of many pathways from the compiled data sets, which showcase a broader picture of the $CTOP_q$ manifold than what was previously shown in Figure 3.14. In this embedding, all of the self-assembly pathways emerge from a common fluid origin. We observe that even before self-assembly begins, particles' local environments begin to weakly display some symmetries of their

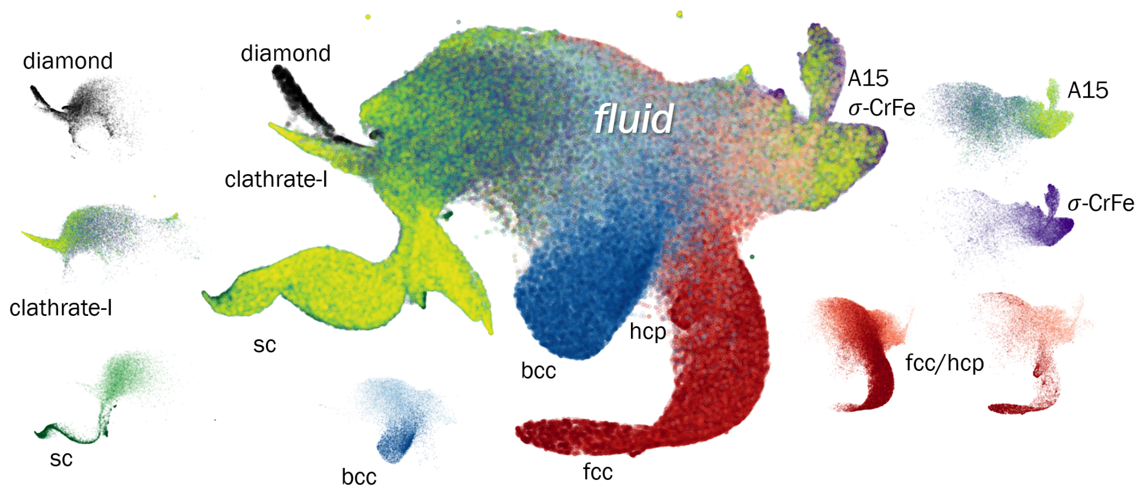


Figure 3.16: A co-embedding of 10 different simulations in the $CTOP_q$ feature space. The subfigures show the pathway embeddings of 8 of the simulations (selected as representative trajectories of the whole). Clockwise from top, the embedding shows local environments from the A15 ($cP8$-Cr_3Si), σ-CrFe ($tP30$-CrFe), fcc ($cF4$-Cu) and hcp ($hP2$-Mg), bcc ($cI2$-W), sc ($cP1$-Po), clathrate-I ($cP54$-K_4Si_{23}), and diamond ($cF8$-C) crystal structures. Color schemes are chosen arbitrarily to help identify the data in the co-embedding. Some trajectories shown were provided by Carl Simon Adorf, Sangmin Lee, and Allen LaCour.

final structure, causing the distribution of fluid-like particles to shift slightly in the direction of the final structure before the solid forms with long-range crystalline order. Nearby points in the embedding have similar symmetries. Some structures, such as A15 and σ-CrFe, appear to have some local environments with similar local symmetries.

Additionally, once a co-embedding of many simulations like this one is generated, it can be used to embed new trajectories and observe their pathways. If the trajectory's particles have local symmetries that have been seen before, their corresponding CTOP vectors will be close to points that already exist in the UMAP embedding. For example,

Figure 3.16 reveals the broader structure of the $CTOP_q$ embedding space, when considering some of the diverse types of crystal structures where local environment analysis can be applied. However, many questions remain. For example, Du et al.

successfully designed particles that exhibit pressure-driven solid-solid phase transformations from fcc to bcc and bcc to simple cubic – both of which appear to be "close" in the UMAP embedding shown here. However, it is unclear whether the embedding reflects meaningful distances in structure space: is it possible to design structures that transition from fcc to diamond, which appear to be further apart in this embedding?

While some of these questions can be solved through further investigation of the $CTOP_q$ manifold, the manifold is not yet a perfect tool for identifying all crystal structures. For example, the β-Sn structure ($tI4$-Sn) is difficult to distinguish from a fluid state because its local symmetries are not well captured by the $CTOP_{qw}$ set of descriptors. This is because β-Sn is a distorted form of the diamond structure, and appears somewhere between the fluid and diamond structure in the UMAP embedding. However, its "squashed" form of diamond breaks most of the local symmetries that the $CTOP_q$ method relies on to distinguish the diamond structure, and offers no strong signatures of other local symmetries observed in the $CTOP_q$ feature space. Detecting this structure reliably may require other continuous order parameters (possibly extensions of the Minkowski Structure Metrics) that can identify stretched or sheared symmetries. Similarly, the local order formed by hard tetrahedra appears to be close to the fluid state in the $CTOP_{qw}$ descriptor space, which was also observed by van Damme et al. with a different set of Minkowski Structure Metrics visualized with PCA dimensionality reduction [46]. Most of these challenging structures can be easily identified as solid-like by taking a second-shell average and looking at the $CTOP_{\overline{qw}}$ feature space. However, improvements in the descriptors are necessary to achieve the goal of detecting microscopic *local* (first neighbor shell) order for these structures. Next, we will discuss potential improvements to the descriptor space, machine learning models, and other aspects of the problem that could extend on the novel aspects of this method while solving some remaining issues that limit the method's generality.

3.4 Outlook and Future Directions

I plan to incorporate the CTOP method into the open-source Pythia library[4], which contains a number of methods for generating features from particle systems that are useful for machine learning applications. The CTOP method, like other methods in Pythia, is implemented using features from **freud** and can be extended and customized for users' needs.

The CTOP analysis method successfully identifies ordered local structures in a wide range of self-assembling systems. In this section, I describe a number of extended approaches and alternative choices that I hypothesize could be used to improve the method. Extensions of the CTOP method could come in a variety of forms, such as increasing the classes of systems whose local environments can be distinguished from one another. The essential finding that all of these future directions builds on is that continuity in feature space significantly influences what methods are best suited to explore the microscopic origins of self-assembly.

3.4.1 Experimental Applications of the CTOP Method

Real-time imaging approaches like in situ X-ray tomography (XRT) are enabling experimental acquisition of particle-level information at atomic length scales [80]. Particle-level information is already being used at the colloidal scale, with some experiments computing Minkowski Structure Metrics from the acquired 3D particle tracks to characterize local environments in fcc and bcc crystal structures [60]. Be-

[4]`https://github.com/glotzerlab/pythia`

cause the CTOP method presented here only requires the computation of Minkowski Structure Metrics from particle positions, it is possible to compute the CTOP and its UMAP embedding from this information. Analyzing the progress of phase transformations and characeterizing local environments in experimental data can be accelerated through this method, because of its applicability to a wide range of crystalline systems that are of experimental interest. Finally, comparisons of particles' local environments between experimental data and simulation data may be useful to understand aspects of real systems that are not captured effectively by simulations, or to explore pathways engineering with high-throughput experiments.

3.4.2 Local Symmetry Analysis with Point Groups

A recently developed method by Engel et al. [81] may present another way to understand the local environments of particles undergoing self-assembly. The CTOP method presented herein, based on Minkowski Structure Metrics, provides a "continuous fingerprint" that can be used to identify many types of local environments. However, as noted above, there are some structures that are difficult to detect using the Minkowski Structure Metrics. Instead, another way of looking at local environments would be to measure the level of point group symmetry around each particle for a set of desired point groups. It may be possible to continuously classify the types of local symmetries for each particle in a way that builds on the hierarchical structure of point groups. Achieving a feature space that is continuous with respect to particles' positions, like the CTOP feature space, may be possible through the use of Voronoi neighbors as is done for Minkowski Structure Metrics.

3.4.3 Applications to 2D Systems

In this work, we use 3D Minkowski Structure Metrics. However, a two-dimensional form of the Minkowski Structure Metric exists as well, using circular harmonics (instead of spherical harmonics). This approach could be used to characterize local structure in 2D simulations, such as hard polygons which have shown diverse phase behavior including continuous KTHNY hexatic transitions [82], plastic crystals [83], and complex host-guest structures [84].

Just as 3D Minkowski Structure Metrics closely resemble the Steinhardt order parameters, 2D Minkowski Structure Metrics resemble the hexatic order parameter (or k-atic order parameter). The k-atic order parameter is implemented in the **freud** library and can be defined as:

$$\psi_k(i) = \frac{1}{N_b} \sum_{j=1}^{N_b} e^{ik\phi_{ij}} \tag{3.1}$$

In 2D, we can obtain the Voronoi polygon of each point and compute its edge lengths, which are used as weights just like the facet areas in the 3D case. As with the 3D Minkowski Structure Metrics, we modify the formula to incorporate Voronoi edge weights w_{ij} as follows:

$$\psi'_k(i) = \frac{1}{\sum_{j=1}^{N_b} w_{ij}} \sum_{j=1}^{N_b} w_{ij} e^{ik\phi_{ij}} \tag{3.2}$$

3.4.4 Handling Particle Shape with Set Voronoi Constructions

The Voronoi construction used in this work treats all particles as points. This is reasonable for the molecular dynamics simulations shown in this work, where particles have isotropic interactions defined by their potential $V(r)$. However, treating particles as isotropic objects may not be a good model for highly anisotropic particles such as the cubes shown in Figure 3.5. The Set Voronoi diagram [85] has been proposed as a way to account for anisotropy in the context of Voronoi diagrams. The resulting Set Voronoi diagrams reduce to traditional Voronoi diagrams when used with points or monodisperse spheres, but are able to more accurately capture the local volumes occupied by each particle in a packing, whether disordered or ordered. The software package Pomelo [86] has been developed to compute Set Voronoi diagrams, and could be used to analyze anisotropic shapes. We note that the Minkowski Structure Metrics, which are currently computed by the **freud** library, could no longer be computed in the same way (according to the facet areas of neighbor bonds), due to the added complexity of having arbitrary surfaces whose facets are not strictly related to the "bonds" of the system (the dual of the usual Voronoi diagram, the Delaunay triangulation, can be trivially mapped onto **freud**'s concepts of neighbor bonds because both structures can be handled as weighted graphs). The Minkowski Structure Metrics on these arbitrary surfaces would have to be computed according to the more general formulas used for Minkowski Tensors, described below.

3.4.5 Minkowski Tensors and Continuous Morphometric Measures

Mickel et al. notes that while the construction of the Minkowski Structure Metrics (q'_l) used in this work are closely related to Steinhardt order parameters (q_l), they also correspond to Minkowski tensors [6]. These and related concepts are sometimes called "morphometric" approaches [6]. For example, the Minkowski Structure Metric q'_2 is related to a quantity $\beta_1^{0,2}$ defined as the ratio of the smallest and largest eigenvalues of the second-rank Minkowski tensor $W_1^{0,2}$. The derivation of the Minkowski Structure Metrics used in this work from Minkowski tensors is thoroughly described in the doctoral theses of Kapfer (Chapter 9) [87] and Mickel (Chapter 7) [88]. We re-derive the essential results in this section to provide a firm mathematical grounding for the extensive use of Minkowski Structure Metrics in this work. Another relevant paper by Schröder-Turk et al. [12] discusses the algorithms used to compute Minkowski tensors and their applications in spatial data extending beyond the realm of particulate matter into microphases seen in polymers and fibrous materials.

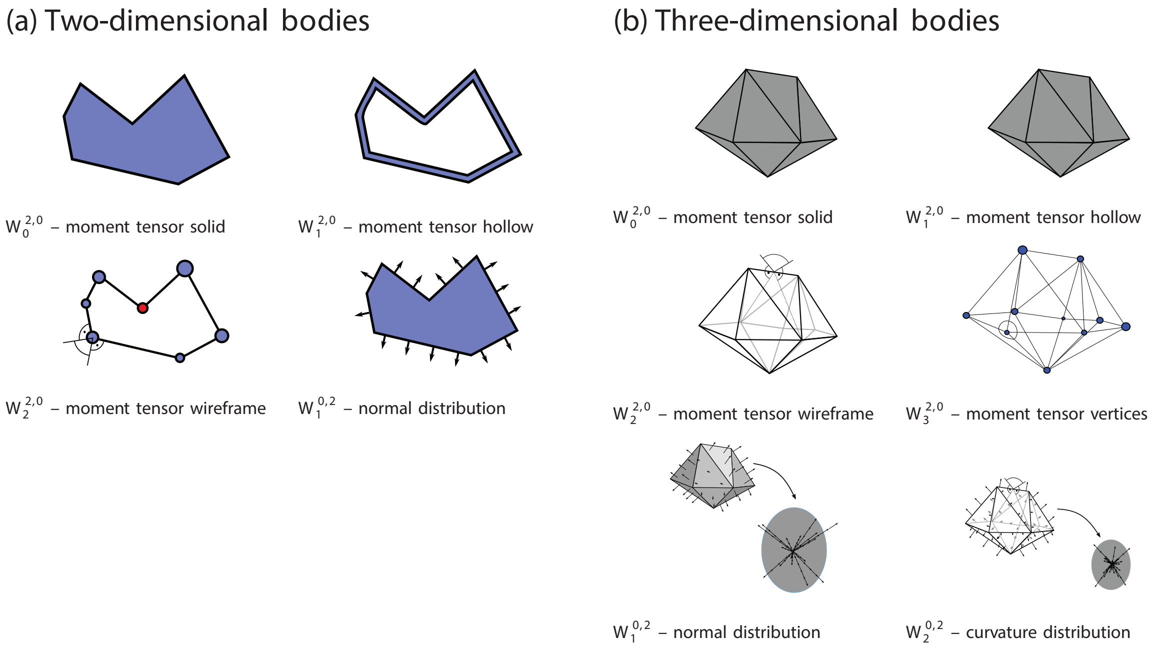

(a) Two-dimensional bodies
(b) Three-dimensional bodies
$W_0^{2,0}$ – moment tensor solid
$W_1^{2,0}$ – moment tensor hollow
$W_2^{2,0}$ – moment tensor wireframe
$W_1^{0,2}$ – normal distribution
$W_0^{2,0}$ – moment tensor solid
$W_1^{2,0}$ – moment tensor hollow
$W_2^{2,0}$ – moment tensor wireframe
$W_3^{2,0}$ – moment tensor vertices
$W_1^{0,2}$ – normal distribution
$W_2^{0,2}$ – curvature distribution

Figure 3.17: Illustration of the linearly independent Minkowski tensors. **(a)** In two dimensions, there are four linearly independent Minkowski tensors. **(b)** In three dimensions there are six. Taken from Ref. [12] with permission.

The Minkowski tensors are denoted $W_\nu^{a,b}$ and shown in Figure 3.17. The rank 0 and 1 Minkowski tensors for solid bodies correspond to properties such as volume, surface area, center of mass, mean curvature, and Gaussian curvature. For a solid body K (a compact set with non-empty interior) in $\mathbb{R}^3$, the rank 2 Minkowski tensors are defined as:

$$W_0^{2,0}(K) = \int_K \mathbf{r}^2 dV \tag{3.3}$$

$$W_\nu^{a,b}(K) = \frac{1}{3}\int_{\partial K} G_\nu \mathbf{r}^a \mathbf{n}^b dA \tag{3.4}$$

where vectors taken to powers use the tensor product (e.g., $\mathbf{r}^2 = \mathbf{r} \otimes \mathbf{r}$) and the functions G_ν are defined using the principal curvatures k_1, k_2 on ∂K:

$$G_1 = 1 \tag{3.5}$$

$$G_2 = \frac{k_1 + k_2}{2} \tag{3.6}$$

$$G_3 = k_1 k_2 \tag{3.7}$$

For this work, we specifically use quantities related to $W_1^{0,l}$ (we relabel the index l to foreshadow its relationship to the Minkowski Structure Metric q_l' obtained below). This tensor captures the distribution of normal vectors on the surface of the body K, shown in Figure 3.17(b) as $W_1^{0,2}$. We take K to be the Voronoi polyhedron of a particle. We take the Cartesian tensor $W_1^{0,l}$ and consider its decomposition into irreducible spherical tensors $W_1^{0,l}|_m^l$. For simplicity, we use Mickel's naming convention and drop some indices, calling these tensors $W_1|_m^l$. Following Mickel's equation (7.29) [88], we define the following with a normalization factor and then transform the integral into a summation over the facets of the convex Voronoi polyhedron K:

$$W_1|_m^l(K) = \int_{\partial K} dA \sqrt{\frac{4\pi}{2l+1}} Y_m^l(\mathbf{n}) \tag{3.8}$$

$$= \sqrt{\frac{4\pi}{2l+1}} \int_{\partial K} Y_m^l(\mathbf{n}) \tag{3.9}$$

$$= \sqrt{\frac{4\pi}{2l+1}} \sum_{j=1}^{N_b} A_j Y_m^l(\mathbf{n}) \tag{3.10}$$

where N_b is the number of facets of the polyhedron K. Note that this is very nearly the same construction as Eqn. 2.9. Applying the facet weight normalization from Eqn. 2.8, $w_{ij} = \frac{A_j}{\sum_j A_j}$ and rewriting this as a second-order rotational invariant (a bilinear form), we get precisely the Minkowski Structure Metrics q'_l:

$$W_1|^l_m(K) = \sqrt{\frac{4\pi}{2l+1}} \sum_{j=1}^{N_b} A_j Y^l_m(\mathbf{n}) \tag{3.11}$$

$$q'_l(i) = \frac{1}{\sum_j A_j} \sqrt{\sum_{m=-l}^{l} |(W_1|^l_m)|^2} \tag{3.12}$$

The third-order rotational invariant (3-form) w'_l can be constructed in a similar way [88]. However, there is nothing that limits us to consider only $W_1^{0,l}$ in our analysis. Other ways to characterize Minkowski tensors include the anisotropy measures $\beta_\nu^{a,b}$, defined as a ratio of the smallest and largest eigenvalues of the corresponding Minkowski tensor [89, 90]:

$$\beta_\nu^{a,b}(K) = \frac{\lambda_{\min}(W_\nu^{a,b}(K))}{\lambda_{\max}(W_\nu^{a,b}(K))} \tag{3.13}$$

Another approach involves reducing the rank-4 tensor $W_1^{0,4}$ to a 6×6 symmetric matrix (in continuum mechanics, a similar reduction is often performed on the rank-4 stiffness tensor). The six eigenvalues of that matrix (denoted $\varsigma_1 \ldots \varsigma_6$) have also been used as rotationally invariant quantities to identify and characterize local environments [91, 43].

3.4.6 Alternate Unsupervised Machine Learning Approaches

I have shown many ways to project the Minkowski tensors into scalars such as the Minkowski Structure Metrics q'_l, anisotropy measures $\beta_\nu^{a,b}$, or eigenvalues $\varsigma_1 \ldots \varsigma_6$. One possible machine learning approach for incorporating the tensors *directly* as structural descriptors would be to train a model such as an autoencoder (as in Ref. [31]) that uses an $\mathbb{E}^3$ equivariant neural network [92, 93] with the Minkowski tensors as features. The tensor networks in the e3nn package are able to learn weights for different ranks of tensors to mix with one another through rotationally-equivariant interactions. An autoencoder model architecture could include several interaction layers allowing for mixing between representations of different ranks, with an encoding space containing only low-rank quantities (e.g., 2-3 scalars). Such a model might be able to learn the relevant symmetries of the Minkowski tensors while exhibiting $\mathbb{E}^3$ equivariance.

3.4.7 $\mathbb{E}^3$ Equivariant Neural Networks

The recent development and application of equivariant neural networks to particle-like data embedded in 3D Euclidean space is one example where a combination of carefully chosen data representations and novel neural network architectures have led to new possibilities such as fast approximations of quantum mechanical force fields [94] and the prediction of phononic bands for a range of atomic crystals [95]. It may be possible to use supervised machine learning with $\mathbb{E}^3$ equivariant neural networks to achieve some of the same characteristics of this study, namely learned order parameters that respect local symmetries and continuity (perhaps through the use of continuous activation functions in the neural network).

CHAPTER IV

Predicting Properties of Photonic Crystals with Machine Learning

4.1 Photonic Materials and Machine Learning

In this chapter, I will discuss the design of materials with photonic properties. Photonic crystals exhibit a photonic band gap, a range of frequencies of light that are fully reflected by the materials. Photonic band gaps (PBGs) can be directional (only certain incident directions are reflected) or omnidirectional. For this chapter, I consider omnidirectional photonic band gaps in 3D materials. PBGs result in many fascinating phenomena in nature, such as the iridescent scales of the beetle *Lamprocyphus augustus* [96]. Furthermore, PBGs allow for the design of materials with applications in optics, such as waveguides [97].

Recently published work by R. Cersonsky, J. Antonaglia, B. Dice, and S. Glotzer describes our findings of three-dimensional photonic band gaps in a wide range of crystal structures [13]. High throughput analyses powered by the signac framework [15] enabled the computation of the first twenty photonic bands in over 130,000 crystal structures. First, I will describe the physical origins of photonic band gaps. Then, I will motivate the use of machine learning models for predicting photonic behavior as a route towards optimization of material properties. Finally, I will discuss a series of machine learning model designs applied to this problem. While no model has shown convergence to the desired results, I present these designs and the data used to train them in hopes of advancing future efforts on photonic materials design.

4.1.1 Electromagnetic Waves in Heterogeneous Macroscopic Media

The material models we consider in this chapter are crystal structures with spheres of radius r and dielectric constant ε placed on each lattice position, with void space filled with a dielectric constant of ε_0. We make a simplifying assumption that the dielectric constant is independent of frequency. This can be used to represent systems such as nanoparticles or colloids that self-assemble a crystal structure. We also consider so-called *inverse* structures, where the spheres have dielectric constant ε_0 and the void space is filled with a dielectric constant of ε. Inverse structures can be formed by methods such as colloidal crystal templating, where structures composed of polymer spheres are filled with a fluid that solidifies, followed by calcination or solvent extraction to remove the original polymer spheres [98].

The dispersion relations of a photonic system can be computed using Maxwell's Equations. We start from Jackson Eqn. 6.6 [99]:

$$\nabla \cdot \mathbf{D}(\mathbf{r}, t) = \rho \tag{4.1}$$

$$\nabla \cdot \mathbf{B}(\mathbf{r}, t) = 0 \tag{4.2}$$

$$\nabla \times \mathbf{H}(\mathbf{r}, t) = \mathbf{J}(\mathbf{r}, t) + \frac{\partial \mathbf{D}(\mathbf{r}, t)}{\partial t} \tag{4.3}$$

$$\nabla \times \mathbf{E}(\mathbf{r}, t) + \frac{\partial \mathbf{B}(\mathbf{r}, t)}{\partial t} = 0 \tag{4.4}$$

Following the derivation in Joannopoulos Chapter 2 [100], we consider only linear and lossless media. Through harmonic mode analysis, we obtain the master equation:

$$\nabla \times \left(\frac{1}{\varepsilon(\mathbf{r})} \nabla \times \mathbf{H}(\mathbf{r}) \right) = \left(\frac{\omega}{c} \right)^2 \mathbf{H}(\mathbf{r}) \tag{4.5}$$

4.1.2 Eigensolver Methods for Computing Photonic Bands

The master equation (Eqn. 4.5) can be treated as an eigenvalue / eigenvector problem, which allows the application of a wide range of computational tools [100]. In this problem, the eigenvalue is the frequency $\left(\frac{\omega}{c}\right)^2$ and the eigenvector is the field $\mathbf{H}(\mathbf{r})$. The MIT Photonic Bands software [101] implements an eigensolver acting in the frequency domain, which was used for calculations in Ref. [13] and this chapter.

The inputs to a photonic band calculation are the periodic box vectors, dielectric geometry contained within the periodic box, material properties corresponding to each geometry object, a set of $\mathbf{k}$ vectors, and the number of bands to compute. The periodic dielectric structure $\varepsilon(\mathbf{r})$ is discretized into a number of voxels. For each $\mathbf{k}$ vector, the MPB code computes eigenvalues (frequencies) of electromagnetic plane waves interacting with the material, and the corresponding fields $\mathbf{H}(\mathbf{r})$. However, this computation can take considerable time, with evaluation of 2744 (14^3) $\mathbf{k}$ vectors (for a density of states calculation) taking around 10 minutes when split across 128 cores, or nearly equivalent to a full day if computed on a single core.

4.2 Past Approaches to Photonic Structure Design

Past work in applying machine learning to photonic crystal design has focused on 2D systems and simple 3D structures with a small number of degrees of freedom in a constrained geometry [102, 103]. These models saw a speedup of between 10^4 and 10^6 when comparing the evaluation time of the neural network to the evaluation time of the MIT Photonic Bands computation. Another relevant work is that of Men et al. [104], which performs 3D optimizations of a dielectric structure represented as voxels to increase the size of a photonic band gap. While this work presents a powerful method for structural optimization, this approach is considerably more computationally expensive than a single evaluation. The ability to perform faster photonic band computations through approximation with a neural network would be useful in the context of materials design, where photonic band gaps could be evaluated in real time for guiding simulations towards structures with desired photonic properties.

4.3 Designing Convolutional Neural Networks for Predicting Photonic Properties

This section's work on CNN design and training was performed in collaboration with Thomas Waltmann. The first approach taken for this study was to build a convolutional neural network (CNN) acting on three-dimensional images of the periodic dielectric structure. The model accepted a 3D image $\varepsilon(\mathbf{r})$ and a reciprocal vector $\mathbf{k}$ as input. We used the ASE Python package [105] to compute the irreducible Brillouin zone (IBZ) and band path for each structure. The $\mathbf{k}$ vectors used for training/validation/testing were sampled from this IBZ band path. The target output was the frequency of the first 20 bands $(\omega_1, \ldots, \omega_{20})$ for that crystal structure, dielectric sphere radius, and $\mathbf{k}$ vector.

4.3.1 Data Preprocessing and Augmentation

Because of the large number of Bravais lattices in the input data set, we could not assume that the 3D input image would be cubic. Instead, we adopted a strategy seen commonly in CNN training: data augmentation, through multiplying the training set by adding many copies of an input image that are rotated or translated. One of the purposes of data augmentation is to train the model to generalize over (or be invariant to) various transformations. For instance, an algorithm being trained to identify objects in an image might include data augmentation that accounts for changes in the lighting conditions, by adding images that are artificially made brighter, darker, or more saturated. For our data, we wished for the resulting model to be invariant to rotation and translation of the crystal structure and thus augmented the data with examples of such transformations. To build the rotational invariance of the band structure into our model, we augmented the dataset with 10 random rotations for each combination of structure and particle radius in the dataset.

The high spatial resolution required for distinguishing geometric features in the 3D images (100x100x100 voxels) meant that the full data set of all crystal structures (130,000) and augmentations (10 per structure) was too large to save to disk, and instead had to be generated on the fly. To train the neural network, we used the Summit supercomputer with the TensorFlow library [106] and IBM Watson Compute Environment software. We used TensorFlow features for distributed memory parallelism and GPU acceleration, specifically optimized for use with Summit's compute nodes containing six NVIDIA V100 GPUs each.

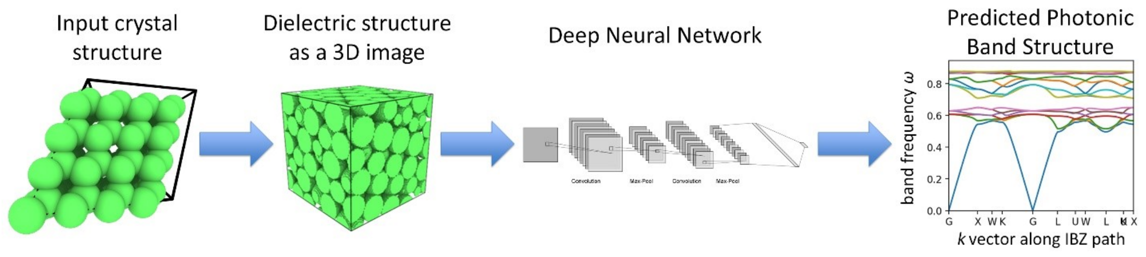


Figure 4.1: The flow of data from an input crystal structure to a predicted photonic band structure. The dielectric image $\varepsilon(\mathbf{r})$ is generated from the input crystal structure and processed by a deep neural network (the figure shows a simplified model) to predict the band frequencies $\omega_n(\mathbf{k}, \varepsilon(\mathbf{r}))$ for each band number n in the first 20 bands and each wavevector $\mathbf{k}$ in the band path over the Irreducible Brillouin Zone (IBZ).

The preprocessing steps and model architecture are shown in Figure 4.1. I built a new feature for the **freud** library called `SphereVoxelization` for efficiently computing 3D voxelized representations of crystal structures with spheres on each lattice site, shown in the second image of Figure 4.1. These voxelizations could be performed on the fly, with input data processed on the CPU while GPUs were used to train the model. This image generation code had to be heavily optimized to ensure that it could be completed quickly and would not become a bottleneck for the GPUs training the model.

4.3.2 Building a Scalable Model

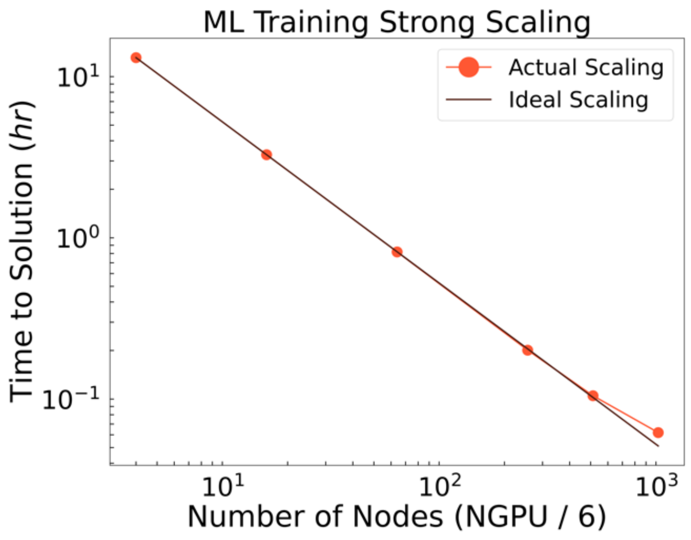


Figure 4.2: Benchmark on Summit (Oak Ridge National Laboratory) for scaling the training of a deep CNN machine learning model. Shown is time to train our model for 1 epoch vs. number of nodes. The strong scaling is nearly ideal, up to the tested limit of 1024 nodes (6144 NVIDIA V100 GPUs in total).

To take advantage of the Summit hardware and distribute our training across multiple GPUs and GPU nodes, we integrated our TensorFlow training code with Horovod [107]. Horovod utilizes an MPI-like interface exposing rank, size, local rank, and local size to the programmer. While training, we first randomly split all the data into three parts: training, validation, and testing. Each rank trains on a randomly selected subset of the training data. We selected the batch size and epoch size to optimize for performance and distributed training efficiency. Each batch of data is pre-processed on the CPU, and the optimal batch size was determined empirically to optimize performance (considering the trade-off of CPU compute, GPU compute, and memory bandwidth). Typically, one “epoch” is defined as one training pass over all data, but here one epoch is defined as a training pass over 10% of all the training data. The epoch size was chosen to be a fraction of the total training data size to boost distributed training efficiency, since the ranks broadcast their learned weights’ gradients at the end of each epoch. This ensures that the gradients will converge to the results expected from a single-node training, while enabling massive parallelism. TensorFlow takes advantage of GPU hardware through the cuDNN deep neural network package which provides libraries for specific deep neural network routines like convolution, pooling, and activation layers. We scale TensorFlow to multiple GPUs and GPU nodes using Horovod with MPI support provided by OpenMPI.

After developing this model with TensorFlow and Horovod, we benchmarked its performance and achieved good strong scaling behavior shown in Figure 4.2. This model was capable of predicting 555 band structures per hour on a single V100 GPU, which we computed would provide around a 30x speedup compared to the MPB calculation running on a CPU. This would have been fast enough to couple to simulations that aim to optimize materials for photonic properties. However, experiments with each of the model architectures described below showed poor convergence on both training and validation data sets.

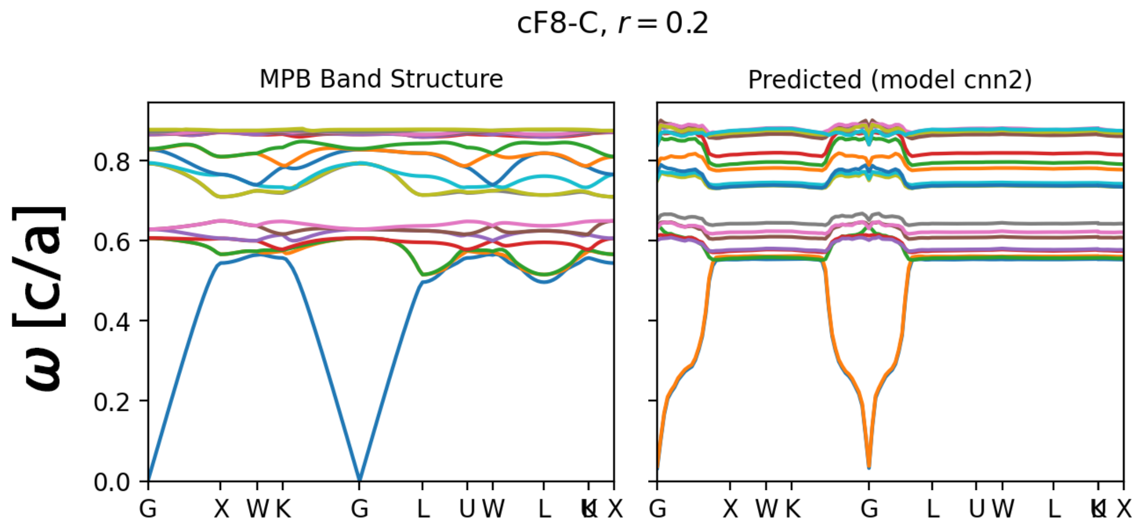


Figure 4.3: The MPB calculated photonic band structure (left) compared with the band structure prediction of our trial ML model, labeled cnn2 (right). The band structure above was predicted after training the model on the XSEDE Bridges GPU-AI partition for 50 epochs on a dataset consisting of only 127 sample structures using 10 random rotations. The dielectric structure is diamond (cF8-C) with particles at the lattice sites having a radius of 0.2 times the length of a unit cell. The trial model learns curvature near the gamma point, but otherwise converges to the average band frequency value over all $\mathbf{k}$ vectors.

4.3.3 Iterations on CNN Model Architecture

Several iterations on the model were attempted. The convolutional portion of the network ranged from 3 to 6 convolutional layers with 8 to 16 filters and `relu` activation functions. Next, the convolution outputs were fed to a series of 4 dense layers that reduced in size from 160 to 20 neurons. The size of the last dense layer is 20, because it is the number of band frequencies that the model predicts. One of the challenges faced was how to combine the 3D information of the crystal structure with the information contained in the $\mathbf{k}$ vector. In all models, the $\mathbf{k}$ vector was rotated to match the random rotation of the dielectric structure. The first model iteration used a four-channel dielectric image, where each channel of the 100x100x100 image contained the scalar values $(\varepsilon(\mathbf{r}), \mathbf{k}_x, \mathbf{k}_y, \mathbf{k}_z)$. That is, every voxel contained the same values for the components of $\mathbf{k}$ and a spatially varying value of $\varepsilon(\mathbf{r})$. The predicted band structure from this model showed some variation near the Γ point ($\mathbf{k} = 0$) but not away from it; instead, the model simply predicts the average frequency value when away from the Γ point. A second model iteration used a single-channel 3D image containing only $\varepsilon(\mathbf{r})$ and the components of the $\mathbf{k}$ vector were used as an additional input to each dense layer. A third model used a separate 3D image with the function $\Re\left[e^{i\mathbf{k}\cdot\mathbf{r}}\right]$, in an attempt to carry the $\mathbf{k}$ vector's information into the same 3D space as the crystal structure. A final model architecture inspired by Ref. [108] used a diffraction pattern computed for a plane orthogonal to the incident radiation. Initially we hoped to find some model architecture and set of hyperparameters that converged, which we could then fine-tune in order to improve the model's accuracy. However, none of the models appeared to converge. To predict the full band structure and predict the location of any photonic band gaps, we evaluated the network at each wavevector in the band path along the IBZ. Sample predictions for the second model are shown in Figure 4.3.

We tried to reduce the size of the training data set, and narrow its scope to only include a small number of closely related crystal structures from the same space group. We hypothesized that reducing the complexity of the full dataset and allowing the model to see examples of the same inputs many times while training could help with convergence (training for a full epoch of the entire data set took a long time, and thus each example was not seen very many times for a given number of compute hours spent training). However, this did not help with convergence towards the desired outputs.

There were two primary challenges with this approach that we sought to resolve. First, the primary goal of computing the photonic bands as a function of $\mathbf{k}$ was to identify photonic band gaps (frequencies where $\omega(\mathbf{k})$ has no real solutions). The computation of $\omega(\mathbf{k})$ isn't strictly necessary if the density of states is known as a function of ω, because photonic band gaps simply correspond to places where $\text{DOS}(\omega) = 0$. While the density of states had not been calculated for the training set of crystal structures, that quantity could be computed. Second, the CNN approach of using images led to a number of undesirable parameters. We wanted the voxelized images to contain multiple repetitions of the unit cell, so that the periodic structure could be learned and used to help predict the band frequencies. However, unit cells with large aspect ratios or angles different from $90°$ (which account for a large portion of the data) were hard to fit into this paradigm. We would need a variant of CNN designed specifically for analyzing data in a triclinic periodic box. Also, choosing an appropriate spatial resolution (here, 100x100x100) that could be applied to structures with different dielectric sphere radii was difficult. Using too high of a resolution means that the model is extremely difficult to train, while too low of a value prevents the model from accurately capturing the 3D geometry.

4.4 Computing the Photonic Density of States

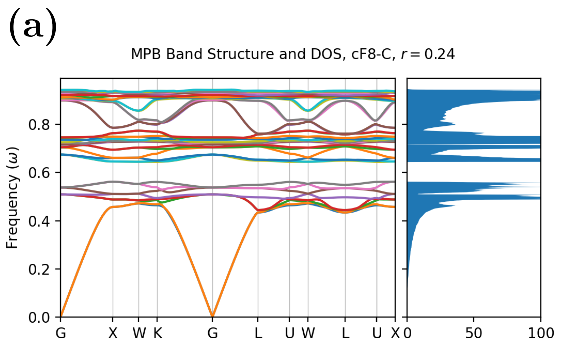
With these challenges in mind, I sought to try a different approach that could solve the problems faced with the CNN model. Recent work by Chen et al. [95] on the problem of *phononic* materials design has shown successful high-throughput predictions of the phononic densities of states for a wide range of atomic crystal structures. While some important aspects of the physics are different between phononic and photonic bands, there are also key similarities. For example, both involve the computation of eigenmodes as a function of a reciprocal wavevector. Both band functions are constrained by crystallographic symmetries and suffer from similar issues with data scarcity (both types of band structures are computationally difficult to generate). Although photonic bands obey the macroscopic laws of electromagnetism and phononic bands relate to thermal and vibrational motion, we saw an opportunity to try again.

In addition to the differences in physics, our photonics data set differs in a few ways. For example, we consider dielectric spheres with a provided radius and dielectric constant, whereas Chen et al. consider atomic elements. Additionally, there is no reference data set of photonic densities of states: our recently published work with Cersonsky et al. in Ref. [13] is, to our knowledge, the largest computational study (and largest data set) of three-dimensional photonic crystals to date. The key advantage to using the density of states as a target, rather than the band structure as in the previous CNN model, is that the dependence on the wavevector $\mathbf{k}$ is integrated out. Additionally, phenomena like band crossings do not need to be explicitly considered if all the bands are integrated, rather than indexed individually. Thus, the physics being learned is vastly simpler in principle, with no need to learn the complex relationships between 3D geometry and a reciprocal space vector.

We undertook calculations on the Bridges-2 cluster that expand on the results of Cersonsky et al. by computing a larger number of bands and integrating over the first Brillouin Zone to get the photonic density of states for each structure as a scalar function of frequency. To accomplish this, we adapted the MATLAB and Scheme code of Boyuan Liu et al., published in Ref. [109]. We ported this MPB code and its corresponding Gilat-Raubenheimer integration method [110] to Python, which is available as an MIT-licensed code on GitHub.[1] This code can be used to compute the photonic density of states for a provided 3D structure. Importantly, this code can be used with MPB's Python interface, meaning that it can be called

[1] `https://github.com/boyuanliuoptics/DOS-calculation`

from any Python application. Moreover, this code is easier to parallelize for large computations, because the MPB computations can be split so that each core takes a subset of the **k** vectors to be evaluated, then all results can be combined at the end. We observed that this was a very efficient approach to parallelization and was simple to implement with mpi4py [111], without needing to use MPB's own MPI interface (which we found somewhat difficult to build from source).



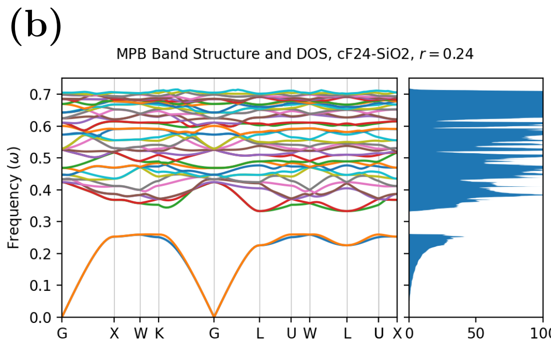


Figure 4.4: Densities of states computed for two crystal structures exhibiting band gaps. **(a)** The band structure and density of states for the diamond structure ($cF8$-C) with dielectric sphere radius $r = 0.24$. **(b)** The band structure and density of states for the β-cristobalite structure ($cF24$-SiO_2) with dielectric sphere radius $r = 0.24$.

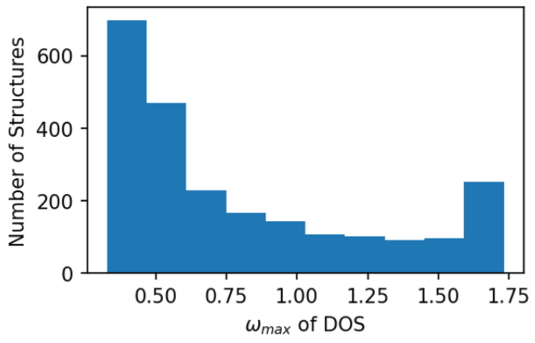


Figure 4.5: Histogram of $\omega_{\max}$ for the 2,400 crystal structures in space group 227 whose densities of states were computed.

We computed photonic densities of states only for structures in space group 227, the space group of the diamond structure ($cF8$-C). Example densities of states are shown in Figure 4.4. The density of states is truncated at the lowest frequency across **k** space for the highest band number (40 bands were calculated), which we label $\omega_{\max}$.

The distribution of these truncation frequencies is shown in Figure 4.5.

4.4.1 $\mathbb{E}^3$ Equivariant Models with Graph Convolutional Networks

We developed a new model architecture that could be trained on the results of the density of states calculations. We use the package e3nn [93] to implement an $\mathbb{E}^3$ equivariant model. This approach is driven by several motivations. First, this package was used in the work of Chen et al. for phononic density of states calculations, which is the closest machine learning model we are aware of for handling arbitrary 3D geometry and predicting band structures. Second, we know that the results of our model (the density of states) must be $\mathbb{E}^3$ invariant. That is, the photonic density of states for a given crystal structure should not vary if the system is rotated, translated, or reflected – the outputs should be invariant with respect to the Euclidean group $\mathbb{E}^3$. Thus, it makes sense to use $\mathbb{E}^3$ equivariant operations in the model's hidden layers. Point groups and space groups are subgroups of $\mathbb{E}^3$, which motivates this choice from the perspective of crystallographic symmetry constraints on the learnable functions. Last, the e3nn approach is data efficient and capable of learning from less training data than some other model architectures [94].

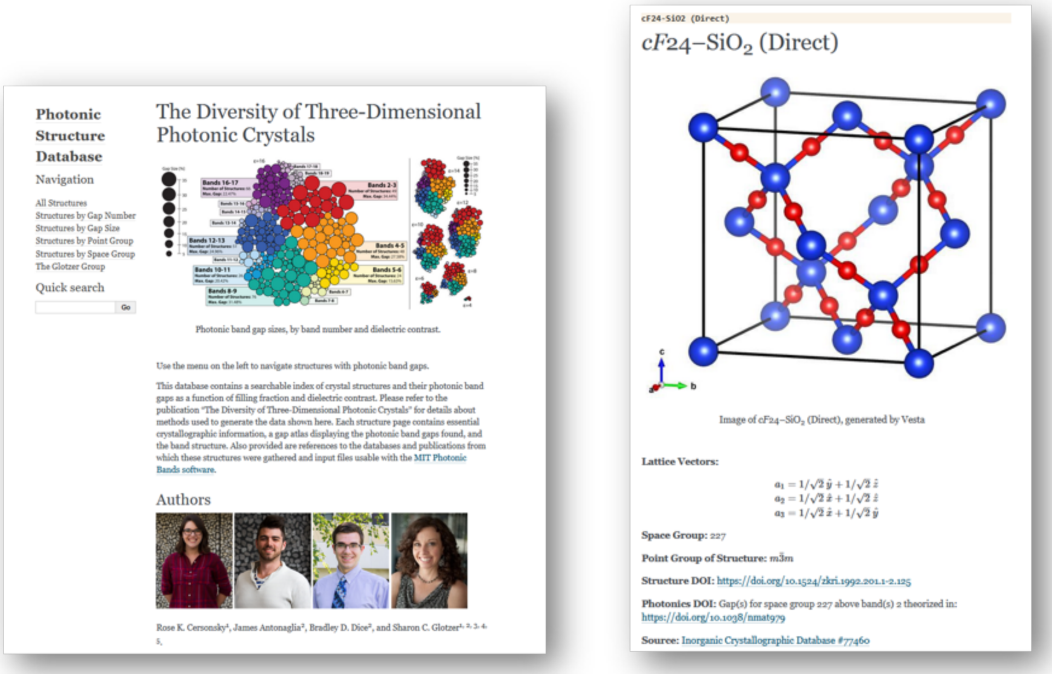
The e3nn is based on graph convolutional neural networks, which perform convolutions that combine information from nodes and edges of a graph. Features belonging to nodes or edges can be scalars, vectors, pseudovectors, or other types of tensorial quantities with a known irreducible representation (irrep) that includes its order $l >= 0$ and parity $p = \pm 1$. On each convolution, nodes' features are scattered to each edge, then an $\mathbb{E}^3$ equivariant tensor product is performed among the desired edge features. This tensor product can include arbitrary irreps, potentially with weighted paths that can be learned by the network. The resulting edge features are scattered back to the nodes, and summed. Then the convolution may be repeated. Other layer types, such as linear layers or gated activations, can also be applied. The resulting predictions may be quantities of any irrep. Following Chen et al., we predict 51 scalar values (with parity $p = 1$ to allow inversion symmetry) corresponding to frequencies in the range $[0, \omega_{\max}]$.

Here, we describe the preprocessing for this model. Following the approach in Chen et al. [95], we aim to predict the values of the density of states for the frequency range $\omega \in [0, \omega_{\max}]$, with 51 values of ω evenly spaced in that range. We choose a truncation frequency $\omega_{\max}$ from the distribution of frequencies shown in Figure 4.5 and drop all structures with densities of states that do not reach that frequency. Here, we chose a value of $\omega_{\max} = 0.5$. However, many structures in the data set have interesting behavior above this frequency. We must truncate at a lower frequency because we cannot exceed the frequency range at which the density of states integration over $\mathbf{k}$ space is valid. Gathering band data for higher frequencies involves the calculation of more bands with the MPB eigensolver, which becomes very computationally expensive as the desired frequency range grows. Since the density of states goes like ω^2, the photonic bands are more dense at higher frequencies, making it quadratically more difficult (on average) to expand the frequency range computed for a given structure.

A set of features is computed for each particle, which include the dielectric sphere volume, radius, and dielectric constant. Edge features include an edge length embedding, which is used by a radial layer (this is a common approach, related to the work of Behler and Parrinello [112]) and edge spherical harmonics $Y_l^m(\theta_{ij}, \phi_{ij})$ for a range of irreps l.

Early experiments with this model architecture did not converge to the desired results. The network appears to learn the general quadratic form of the density of states as an average over all the input data, but does not incorporate band gaps (where the density of states is zero) and does not account for the changes in dielectric filling fractions that appear to change the quadratic coefficient of the DOS curve. In future work, we may wish to explore improvements to the model architecture, input data quality, and additional training time.

4.5 Database of Photonic Crystal Structures



Photonic Structure Database

Navigation

All Structures
Structures by Gap Number
Structures by Gap Size
Structures by Point Group
Structures by Space Group
The Glotzer Group

Quick search

Go

The Diversity of Three-Dimensional Photonic Crystals

Photonic band gap sizes, by band number and dielectric contrast.

Use the menu on the left to navigate structures with photonic band gaps.

This database contains a searchable index of crystal structures and their photonic band gaps as a function of filling fraction and dielectric contrast. Please refer to the publication "The Diversity of Three-Dimensional Photonic Crystals" for details about methods used to generate the data shown here. Each structure page contains essential crystallographic information, a gap atlas displaying the photonic band gaps found, and the band structure. Also provided are references to the databases and publications from which these structures were gathered and input files usable with the MIT Photonic Bands software.

Authors

Rose K. Cersonsky[1], James Antonaglia[2], Bradley D. Dice[2], and Sharon C. Glotzer[1, 2, 3, 4, 5]

cF24-SiO2 (Direct)

$cF24$–SiO_2 (Direct)

Image of $cF24$–SiO_2 (Direct), generated by Vesta

Lattice Vectors:

$$a_1 = 1/\sqrt{2}\,\hat{y} + 1/\sqrt{2}\,\hat{z}$$
$$a_2 = 1/\sqrt{2}\,\hat{x} + 1/\sqrt{2}\,\hat{z}$$
$$a_3 = 1/\sqrt{2}\,\hat{x} + 1/\sqrt{2}\,\hat{y}$$

Space Group: 227

Point Group of Structure: $m\bar{3}m$

Structure DOI: https://doi.org/10.1524/zkri.1992.201.1-2.125

Photonics DOI: Gap(s) for space group 227 above band(s) 2 theorized in: https://doi.org/10.1038/nmat979

Source: Inorganic Crystallographic Database #77460

Figure 4.6: Screenshots of the photonics database online, produced to accompany Ref. [13].

As a part of the work done for Ref. [13], I designed a web-accessible database of the crystal structures obtained in that study. Using the **signac** data management framework and Sphinx documentation generator for Python, I created a searchable static website that is hosted on our group's webserver.[2] The website, shown in Figure 4.6, contains a list of all structures found to have photonic band gaps. The data can be sorted by gap size, point group, space group, or the band numbers

[2]`https://glotzerlab.engin.umich.edu/photonics/`

where the gap exists. The website contains information about each structure that was found to have a band gap, including the lattice vectors, space group, point group, data source, and Digital Object Identifiers (DOIs) for related publications where the structure or its photonic properties have been previously studied.

4.6 Outlook and Future Directions

Despite the challenges of training machine learning models for this problem, I can make some conclusions about what I have learned in the process. For example, in both this project and the CTOP analysis method demonstrated in Chapters II and III, I see that machine learning models are most capable when model and feature invariants/equivariants align with the physics being learned. Additionally, I find that data augmentation is not an efficient or robust replacement for model architectures that can capture the physics innately – and it's not easy to know when "brute force" (in the form of extensive data augmentation and longer training times) can make up for it. I believe that new model architectures like e3nn hold promise for future work on this and related problems in physics-based machine learning.

Additionally, I made improvements to open-source tools during the course of this work that will benefit future research in the field. These include an open-source (MIT licensed) Python implementation of the photonic density of states solver by Boyuan Liu[3], the `SphereVoxelization` feature of **freud**, and improved software packages on conda-forge for MPB and its companion tool Meep (a time-domain electromagnetic solver package that also supplies the Python interface for MPB).[4] Using these tools to simplify the computation of photonic densities of states may enable other applications. For example, *electronic* density of states have been used

[3]`https://github.com/boyuanliuoptics/DOS-calculation`
[4]`https://github.com/conda-forge/pymeep-feedstock/pull/56`

to train a machine learning model called DOSnet that accurately predicts adsorption energy for a range of surfaces and adsorbates [113]. In that work, the density of states was an input to the model, rather than an output being used to characterize band gaps as described in this chapter. The robust and computationally efficient pipeline developed for calculating photonic densities of states for this study would be valuable in a model like DOSnet that aims to predict other material properties from photonic densities of states.

CHAPTER V

The freud Library for Particle Analysis

5.1 Introduction

The open-source **freud** Python library[1] provides a simple, flexible, powerful set of tools for analyzing trajectories obtained from molecular dynamics or Monte Carlo simulations. High performance, parallelized C++ is used to compute standard tools such as radial distribution functions, correlation functions, order parameters, and clusters, as well as original analysis methods including potentials of mean force and torque (PMFTs) and local environment matching. The freud library supports many input formats and outputs NumPy arrays [114], enabling integration with the scientific Python ecosystem for many typical materials science workflows.

During my PhD, I have been a core developer and maintainer of the library, collaborating closely with Vyas Ramasubramani on extensive improvements to the code base driven by research needs for fast computations of order parameters, data visualization, and support for the many data formats used in molecular sciences. This joint work with Ramasubramani resulted in two co-authored publications: Ramasubramani et al. [27] about the library design and feature set, and Dice et al. [115] about the use of the **freud** library for machine learning and data visualization in the molecular sciences. As release manager for the software, I produced 23 releases since

[1] `https://github.com/glotzerlab/freud/`

2017, from version 0.6.4 to 2.6.2, including two major versions (the second of which is discussed further in this chapter). Crucially, my work on **freud**'s algorithms, APIs, and software features enabled the methodological development and results discussed in Chapters II and III. Among the core features I have contributed are improved neighbor finding in periodic boxes, Voronoi diagrams using the voro++ library [48], improved Steinhardt order parameters including the computation of Minkowski Structure Metrics discussed in Chapter II, and a new algorithm for computing particle clusters that greatly improved computation time for the Solid-Liquid order parameter [116]. It has enabled many research projects within our group and beyond, already resulting in 33 citations of the software (Ref. [27]) since the paper's publication in September 2020 (less than a year at the time of writing this). The library has 41 code contributors according to GitHub.[2] The **freud** library is available from the Python Package Index[3] and conda-forge[4], as well as in Docker and Singularity prebuilt images that users can execute on Mac, Linux, and HPC clusters.[5]

In this chapter, I will describe my contributions to the **freud** library and discuss machine learning pipelines, data visualization, and recently published research using features from **freud** for wide-ranging analyses of particle systems. Some contents of this chapter are adapted from Dice et al. [115], under the Creative Commons Attribution License. The full contents of that paper are reproduced in Appendix A.

5.2 Library Design and Features

The **freud** Python package offers a unique feature set that targets the analysis of colloidal systems. The library avoids trajectory management and the analysis of

[2] `https://github.com/glotzerlab/freud/`
[3] `https://pypi.org/project/freud-analysis/`
[4] `https://anaconda.org/conda-forge/freud`
[5] `https://github.com/glotzerlab/software`

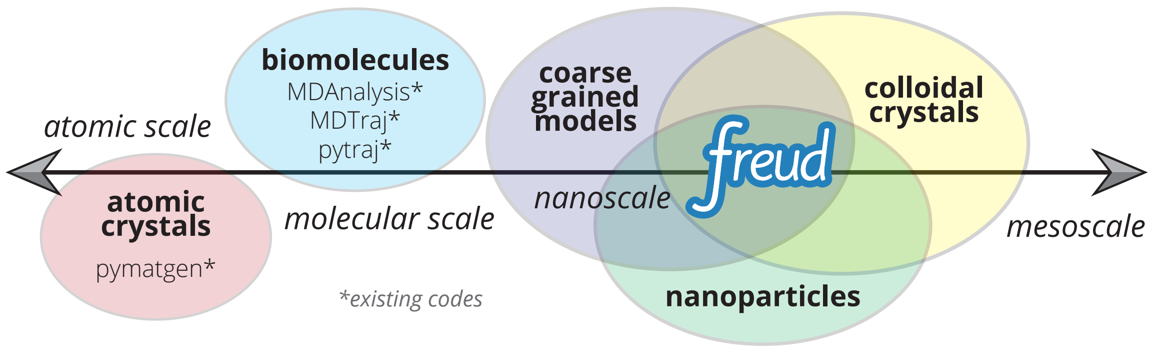


Figure 5.1: Common Python tools for simulation analysis at varying length scales. The **freud** library is designed for nanoscale systems, such as colloidal crystals and nanoparticle assemblies. In such systems, interactions are described by coarse-grained models where particles' atomic constituents are often irrelevant and particle anisotropy (non-spherical shape) is common, thus requiring a generalized concept of particle "types" and orientation-sensitive analyses. These features contrast the assumptions of most analysis tools designed for biomolecular simulations and materials science.

chemically bonded structures, which are the province of most other analysis platforms like MDAnalysis [117] and MDTraj [118] (see Figure 5.1). In particular, **freud** excels at performing analyses based on characterizing local particle environments, which makes it a powerful tool for tasks such as calculating order parameters to track crystallization or finding prenucleation clusters. Among the unique methods present in freud are the potential of mean force and torque, which allows users to understand the effects of particle anisotropy on entropic self-assembly [24, 119, 120, 121, 82], and various tools for identifying and clustering particles by their local crystal environments [44]. All such tasks are accelerated by **freud**'s extremely fast neighbor finding routines and are automatically parallelized, making it an ideal tool for researchers performing peta- or exascale simulations of particle systems. The **freud** library's scalability is exemplified by its use in computing correlation functions on systems of over a million particles, calculations that were used to illuminate the elusive hexatic phase transition in two-dimensional systems of hard polygons [82].

The freud 2.0 design changed the user interface (APIs) for the library, to reflect a much more general approach to analysis. For example, virtually all analysis methods can be performed between two disjoint sets of coordinates, called *points* and *query points*. Neighbor bonds always go from a *query point* to a *point*. This allows for the construction of asymmetric neighbor configurations, such as k-nearest neighbors, as well as the computation of quantities like partial radial distribution functions between two types A and B as $g_{AB}(r)$. This conceptual model is distinct from the approaches commonly used in packages for biomolecular simulations, where macromolecular topology (e.g., bonds between carbon and oxygen atoms) and easily distinguishable atomic identities provides a shared basis for neighbor finding. In nanoparticle self-assembly, however, simulations often involve monodisperse building blocks of the same type. The 2.0 design of **freud** allows for multiple paradigms of analysis, defined by NumPy indexing instead of fixed atomic type identifiers, which can be useful for nanoparticles, polymers, and coarse-grained models of many kinds. These ideas and implementations are discussed further in Ramasubramani et al. [27].

Other major changes in the 2.0 release included 4x faster performance through on-the-fly neighbor finding, integration with Jupyter notebooks [122] for inline data visualization and plotting, support for 25+ file formats through integrations with MDAnalysis [117], OVITO [9], HOOMD-blue [28], and file readers such as GSD[6] and garnett[7], and support for the Windows operating system in addition to Linux and macOS.

[6]`https://github.com/glotzerlab/gsd/`
[7]`https://github.com/glotzerlab/garnett/`

5.3 Machine Learning Pipelines

As previously shown in Figure 2.3, many approaches for machine learning analysis of particle systems involve the computation of features that form a high-dimensional feature space, followed by the application of a machine learning algorithm to that high-dimensional feature space. The **freud** library can be used as part of these data generation pipelines for machine learning (ML) algorithms used to analyze particle simulations. Among Python packages used in the computational molecular sciences, **freud** offers a unique set of analysis methods designed for nanoscale simulations.

5.4 Performance and Real-Time Visualization

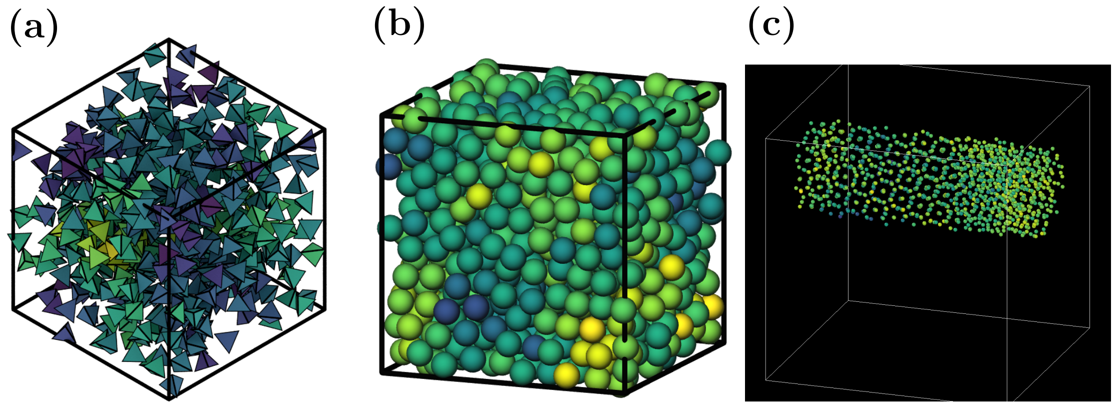


Figure 5.2: Overview of data visualization tools that can be coupled with the **freud** library. **(a)** Interactive visualization of a Lennard-Jones particle system, rendered in a Jupyter notebook using plato [3] with the pythreejs backend [4]. **(b)** Hard tetrahedra colored by local density, path traced with fresnel [14]. **(c)** A crystalline grain identified using **freud**'s `LocalDensity` module and cut out for display using OVITO [9]. The image shows a $tP30$-CrFe structure formed from an isotropic pair potential optimized to generate this structure [15].

Computational researchers are often scientists first and programmers second. As a consequence, it is imperative that scientific software is designed with its users in mind. One design requirement identified for the freud analysis library was that it must be possible to integrate with real-time visualization applications. In Figure 5.2, we show examples of such visualization applications. Over time, the focus of this design requirement shifted from in-house software using a Qt GUI (platoviz) to the popular OVITO application [9]. The OVITO software uses pipelines wherein data passes from one stage to another, starting with a file input and resulting in the data shown on screen. For example, pipeline stages may perform analysis, select particles, and add or delete properties corresponding to particles, bonds, types, or other attributes. The technical requirement that this imposes on freud developers is that simulation trajectory playback should be "real time," capable of rendering on the order of 30 frames per second. Following from that, any pipeline stages using freud must compute their target quantities in roughly 10-30 milliseconds. The ability to use freud as a "real time" library, rather than an expensive postprocessing step for each simulation, offers a high degree of flexibility to users.

5.5 Wide-Ranging Applications of Data Analysis in Particle Systems

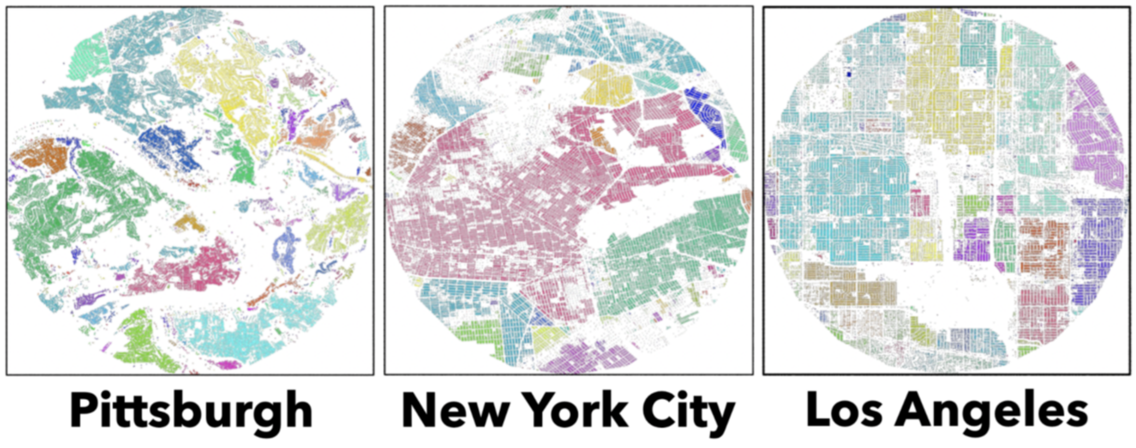


Figure 5.3: Cluster analysis of buildings in three American cities, revealing different levels of city "crystallinity." Figure reproduced from Ref. [16] under the Creative Commons Attribution International 4.0 License.

Among the citations of the **freud** library are some highly interesting applications of statistical physics beyond the nanoparticle regime. These kinds of use cases are enabled by the high level of generality that **freud** offers, with few assumptions about the types of input data. In this section, we highlight two such recently published papers using **freud**. The first, by Ryan Rusali and Gerald Wang, applies methods from molecular simulation to analyze “urban textures,” clusters of buildings in U.S. cities including Pittsburgh, Los Angeles, and New York City. The study uses **freud** to calculate quantities such as the radial distribution function, $g(r)$, the 2-fold order, ψ_2, and clusters of buildings using data from Geographic Information Systems (GIS). This analysis is shown in Figure 5.3. Another such analysis is that of Teich et al. [123], which uses **freud**’s environment matching module (originally contributed by Teich) to analyze structurally homogeneous regions of white matter in the human brain. These interdisciplinary applications of methods from materials physics and statistical mechanics in domains such as urban planning and neuroscience indicate the value of designing algorithms in a way that is neutral to their applications, encoding as few assumptions as possible, and encouraging interaction with the broader ecosystem of scientific Python tools.

5.6 Outlook and Future Directions

In this chapter, we have described library design principles, machine learning applications, and integrations with data visualization tools. We see future goals for the library in a few key areas, such as diffraction and scattering calculations, where there are few modern tools capable of handling the diversity of input formats and generalized assumptions (e.g., generic particles, not strictly atoms) that **freud** aims to handle. Additionally, we hope that newly developed methods for symmetry detection and other techniques applicable to nanoparticle self-assembly can be incorporated into the feature set of the library. The architecture of the library has been solidified, with high standards for code quality and performance. We anticipate that the standardization efforts on **freud** 2.0 will aid future developers significantly, as the code now provides a wide range of types of calculations that can serve as examples for future feature development.

CHAPTER VI

The signac Framework for Data Management and Workflow Automation

6.1 Introduction

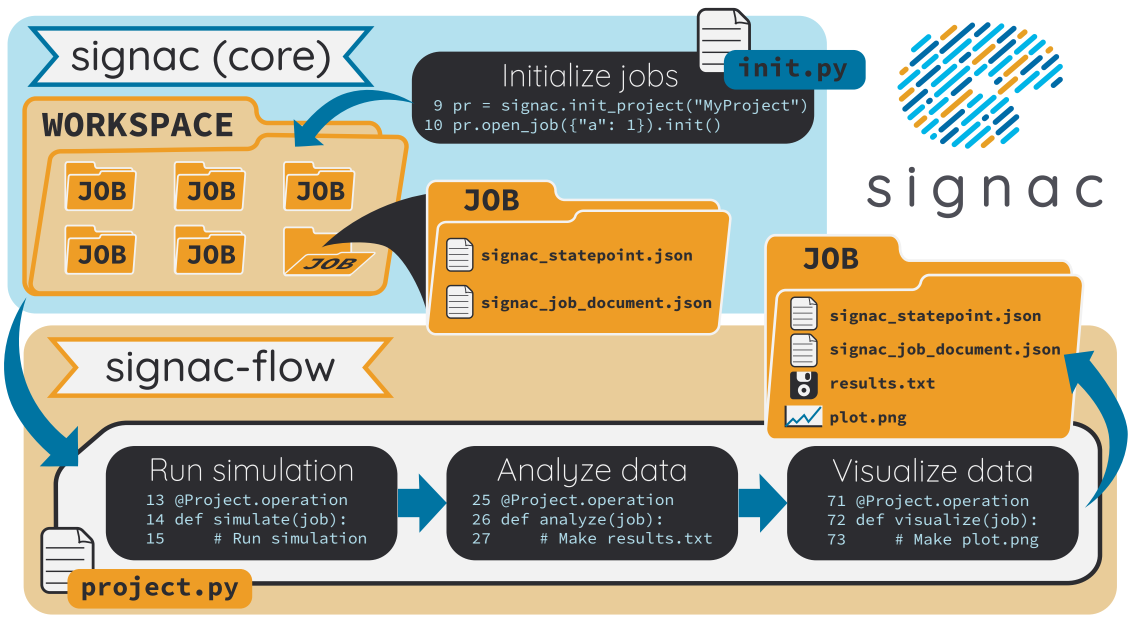
The data collection and computational workflow of the projects described in Chapters II, III and IV were managed with the **signac** framework. Since 2017, I have been a core developer and maintainer of the **signac** framework, along with a team of 7 other committers and maintainers. In this chapter, I will discuss my contributions to the **signac** data management framework and the novel scientific applications enabled by its use. Some of the contents of this chapter are reproduced from Dice et al. [124] under the Creative Commons Attribution License. The full contents of that paper are reproduced in Appendix B.

The **signac** data management framework helps researchers execute reproducible computational studies, scaling from laptops to supercomputers and emphasizing portability and fast prototyping. With **signac**, users can track data and metadata for file-based workflows (such as large molecular simulations) with features for searchability, collaboration, and archival. The companion package **signac-flow** automates workflow submission on high performance computing clusters. The architecture of **signac** is specifically aimed at research, where questions change rapidly, data models are always in flux, and computing infrastructure varies widely from project to project. The **signac** framework is a NumFOCUS affiliated project, is available for Python 3.6+ via pip or conda, and is licensed BSD-3. To date, over 45 people have contributed code to the **signac** framework.

The **signac** framework has been cited 59 times, according to Google Scholar, and has been used in a range of scientific fields with various types of computational workflows. Some of these studies include quantum calculations of small molecules [125], 4,480 simulations of epoxy curing (each containing millions of particles) [126], inverse design of pair potentials [15], identifying photonic band gaps in 151,593 crystal structures [13], benchmarking atom-density representations for use in machine learning [127], simulating fluid flow in polymer solutions [128], design of optical metamaterials [129], and economic analysis of drought risk in agriculture [130]. These diverse applications of the software show the generality of its approach to data management and computational workflows. Much of the software design and development has been carried out in conjunction with the Molecular Simulation Design Framework (MoSDeF) [131], a National Science Foundation Cyberinfrastructure for Sustained Scientific Innovation (CSSI) effort.

6.2 Data Model and Integration with Scientific Applications

Here, we present an overview of the **signac** data model and **signac-flow** workflow paradigm, shown in Figure 6.1. The core **signac** package provides a database (called a *project*) that is managed on the file system, containing *jobs*. Jobs are identified by their state points, key-value stores encoded as JSON. In a **signac** project's workspace directory, each job has a directory named according to a hash of its state point. Users of scientific research codes typically have file-based workflows, executing specialized software (often designed for HPC systems) that generates output files in a particular



```
pr = signac.init_project("MyProject")
pr.open_job({"a": 1}).init()
```

```
@Project.operation
def simulate(job):
    # Run simulation
```

```
@Project.operation
def analyze(job):
    # Make results.txt
```

```
@Project.operation
def visualize(job):
    # Make plot.png
```

Figure 6.1: Overview of the **signac** framework. Users first create a project, which initializes a workspace directory on disk. Users define state points which are dictionaries that uniquely identify a job. The workspace holds a directory for each job, containing JSON files that store the state point and job document. The job directory name is a hash of the state point's contents. Here, the `init.py` file initializes an empty project and adds one job with state point {`"a": 1`}. Next, users define a workflow using a subclass of **signac-flow**'s `FlowProject`. The workflow shown has three operations (simulate, analyze, visualize) that, when executed, produce two new files `results.txt` and `plot.png` in the job directory. Special thanks to Kelly Wang for contributing the design and concept of this figure.

format. In the **signac** paradigm, these files are then stored on disk in a job workspace directory corresponding to the state point parameters that were used to generate the data.

The companion package **signac-flow** provides tools for users to write reproducible workflows that can be executed on local machines (such as a laptop) and scale all the way up to large HPC clusters such as Oak Ridge's Summit. The design of **signac-flow** involves a `FlowProject` that builds on top of **signac**'s `Project` class to provide the tools needed for a conditional workflow. These include *operations* that act on jobs, *labels* that provide status information to the user, and *precondi-*

tions/postconditions that indicate when operations should run. With these tools, users can check the status of a workflow, run operations locally, and submit operations to a cluster running a SLURM, PBS, or LSF scheduler. On top of these basic workflow tools, **signac-flow** also presents advanced workflow capabilities described below in Figure 6.2. The operations in a workflow may be Python functions or shell commands, providing users the ability to call external programs such as simulation tools or other programming languages besides Python.

6.3 HDF5 Data Stores

Many modern scientific applications can be used as *libraries*, such as the **freud** library discussed in Chapter V and the new design of HOOMD-blue version 3. In these application designs, typically driven by Python or another scripting language, the software cedes control to the user to perform most actions involving file input/output. As a result, scientific applications that act as libraries often handle data from existing objects in memory, eliminating the need for reading data from or writing data to disk. Objects such as NumPy arrays [114] provide a transparent and nearly-universal format for numerical arrays in memory. Having intermediate data in a common memory layout that can be accessed efficiently is crucial for analyzing large data sets, and is thus essential to modern computational scientific work.

The HDF5 format is designed to store numerical arrays like those described above. For my research, I developed an integration layer between **signac** and HDF5 data stores, enabling users to quickly save and load numerical arrays and minimizing the complexities of serializing binary data to disk. The h5py software [132] provides APIs for reading and writing HDF5 files in Python, which we use to handle the interactions between **signac** and the underlying HDF5 files. For example, in the research shown in Chapter III, I used HDF5 files extensively to cache results computed by the **freud** library. This allowed for rapid iteration on other parts of the data pipeline such as tuning the UMAP dimensionality reduction and plotting code, without the need to recompute data from **freud** for every change in the data pipeline. The **freud** library has no direct support for saving its results to disk, and instead relies on the scientific Python ecosystem for saving its NumPy arrays. Thus, the integration of h5py and HDF5 with **signac** provides a convenient mechanism for storing output data of computational libraries that have no native methods for data storage.

The choice of the HDF5 format has both advantages and disadvantages. One advantage of the format is its hierarchical nature, allowing users to store nested keys and values in the same way that **signac** handles the job document (which is a JSON file). This is well suited for the types of data and metadata that **signac** users often wish to store. Additionally, the core HDF5 library is commonly found in HPC contexts. The h5py library greatly simplifies the usage of HDF5, making it user-friendly and mostly transparent to users who are accustomed to the behavior of NumPy arrays. Furthermore, the use of HDF5 files allows for binary data like floating point numbers to be directly stored, rather than performing expensive and lossy serialization of those values as text. A significant disadvantage of the format is its inability to recover space in the file when deleting datasets (the file size can only grow unless the user runs a repacking program). However, in practice many HDF5 stores are written once and read many times, which mitigates this to some extent. When designing this feature, I found that the performance, flexibility, and long history of stability of the HDF5 format made it a strong choice for both convenience in the short term and archival over the long term.

The **signac** library offers a few ways to utilize HDF5 Data Stores. Users create a context manager that contains all of the input/output actions to be performed, which ensures the file is opened and closed safely. The `Job` object has two attributes, `job.data` and `job.stores`, that respectively refer to a specific data store named `signac_data.h5` and a dictionary of stores whose keys correspond to file names, e.g., `job.stores["my_data"]` corresponds to the file `my_data.h5` in the job workspace. Likewise, the `Project` class has attributes `project.data` and `project.stores` for holding project-wide data.

6.4 Improving Performance and Scalability

In this section, I discuss improvements to the scalability of the **signac** and **signac-flow** packages. These changes enabled the study undertaken in Chapter IV, which contained more than 100,000 jobs representing different crystal structures and sphere radii. The serverless design of **signac**'s data model imposes limitations relating to disk I/O and filesystem latency that cannot be easily concealed from the user (e.g., via asynchronous transactions). In the past study of Cersonsky et al., which also handled this large number of photonics computations, a hierarchical model of **signac** subprojects nested within a parent **signac** project was used. However, this approach made workflows and HPC submission cumbersome, compared to what can be done with a typical single-level project with **signac-flow**. By focusing on algorithmic efficiency, minimizing system calls, and improving data access patterns throughout the **signac** framework, I was able to extend the framework's scalability, solving a challenge for my own research as well as enabling larger computational studies for future users of **signac**.

The **signac-flow** package supports a number of complex patterns affecting the execution and submission of jobs and operations, summarized in Figure 6.2. In

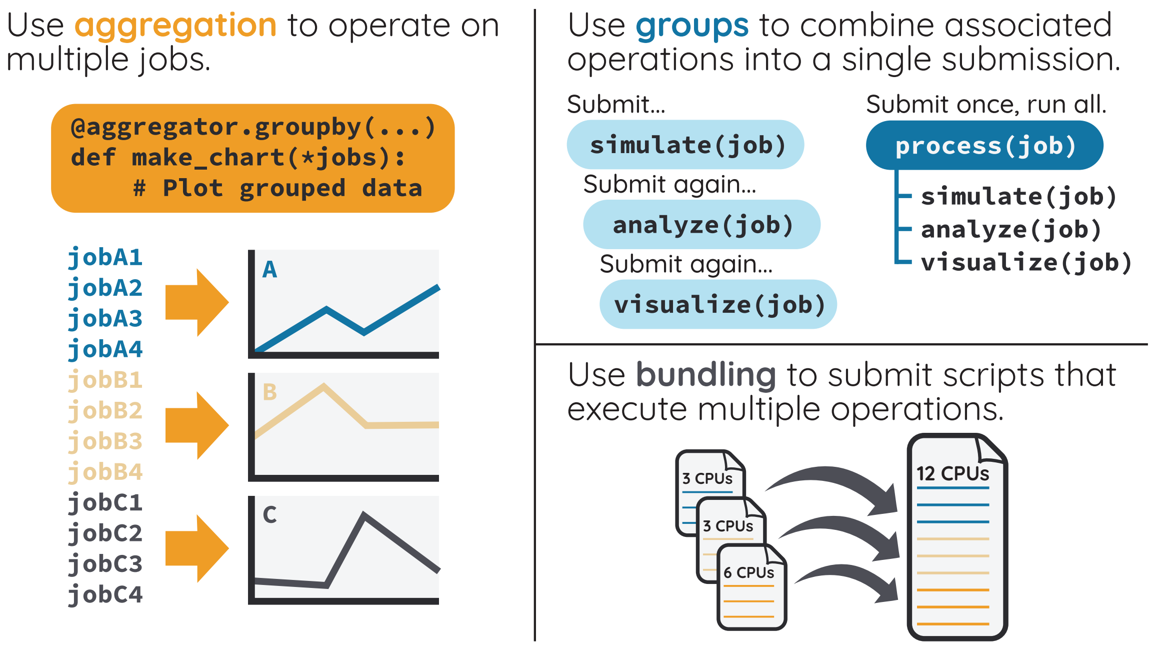


Figure 6.2: Aggregation, groups, and bundling allow users to build complex workflows. The features are orthogonal, and can be used in any combination. Aggregation enables one operation or group to act on multiple jobs. Groups allow users to combine multiple operations into one, with dependencies among operations resolved at run time. Bundling helps users efficiently leverage HPC schedulers by submitting multiple commands in the same script, to be executed in serial or parallel.

summer 2020, Google Summer of Code student Hardik Ojha worked with me and other mentors from the **signac** team to implement a feature called "aggregation," allowing users to submit and execute operations acting on more than one job at a time. This new feature solves a common request from users, who want to be able to plot data from multiple jobs (perhaps grouped by a common state point parameter) or execute complex parallel operations on large supercomputers (particularly Oak Ridge National Laboratory's Summit system) with MPI communicators split among multiple jobs.

As a part of the redesign of **signac-flow** to support aggregation, I refactored a large portion of the internal workings of the package in January 2021. The core functions of **signac-flow** (`status`, `run`, and `submit`) shared common algorithms but had disparate implementations, some of which were aware of groups or aggregation, but often handled nontrivial groups (groups with more than one operation) or nontrivial aggregates (aggregates with more than one job) in a separate and inefficient manner. The result was a significant improvement, with more shared logic among the core features of **signac-flow** and fewer unique code paths to test and maintain. Now, most internal functions call a single generator (coroutine) that yields pairs of aggregates and groups along with their execution eligibility and/or scheduler status information. While undertaking this redesign, I profiled the code extensively using Python's `cProfile` module, identifying and refactoring internal methods that scaled poorly with the number of jobs or were making unnecessary function calls that could be eliminated through restructuring the flow of data. The improved evaluation of operation eligibility and data preparation steps common to all core functions led to a speedup of approximately 4x for a sample workflow with 1000 jobs, 3 operations, and 2 label functions (comparing version 0.12.0 with version 0.11.0).

At the same time, the process of profiling **signac-flow** also pointed to inefficiencies within the core **signac** package. For instance, creating an instance of the `Job` class (which is created $O(N_{jobs})$ times for most **signac-flow** commands) always loaded the state point from disk, even if the user had opened the job by its id (unique hash) and not yet accessed the `statepoint` attribute. I refactored the core `Job` and `Project` classes to load data lazily (on request), cache internal properties, and initialize data on access rather than pre-emptively. These modifications were somewhat difficult to do while maintaining the database invariants required (or assumed) by **signac**'s data model, especially under circumstances that could involve race conditions in parallel workflows on HPC clusters. Once completed, these refactorings of the data access patterns to minimize disk I/O resulted in much faster access for large data spaces.

On a solid-state disk, these changes cumulatively created a 4x to 7x speedup for large projects (tested with 100,000 jobs, comparing version 1.6.0 with version 1.5.0). The simultaneous development of new code for "synced collection" objects by Vyas Ramasubramani and Google Summer of Code student Vishav Sharma served as a helpful validation of these optimizations, which we confirmed could be implemented safely without compromising the validity of the **signac** data model.

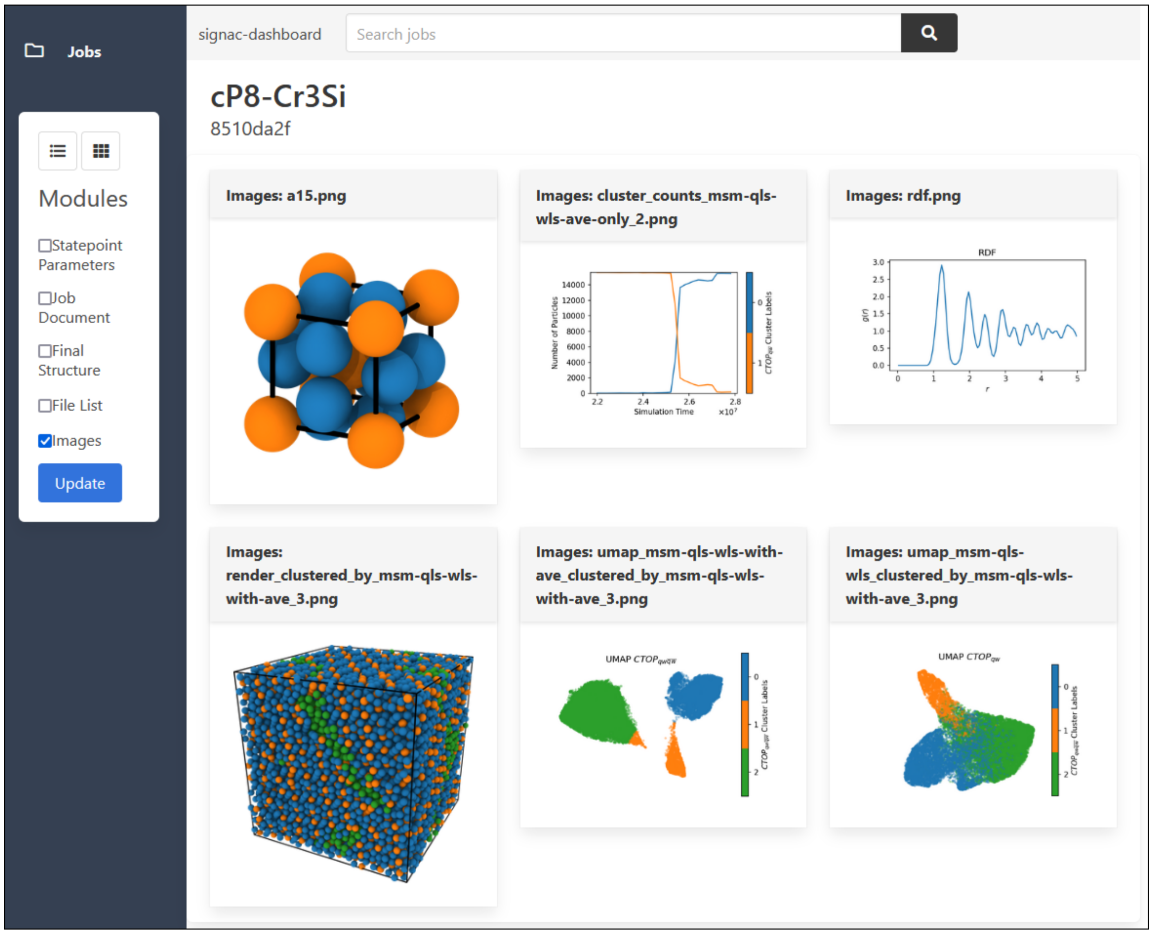
Prior to this work, the **signac** framework was quite difficult to use with large numbers of jobs, because basic actions like checking a workflow status or submitting jobs to an HPC cluster could take a long time. The measured "user time" for these operations is very important, because long wait times to check job status negatively impact research productivity and users' ability to understand the current state of their work.

6.5 signac-dashboard: Visualizing Research Data

While performing an early stage research project, I found that I needed a high-throughput way to review data generated in a **signac** project for hundreds of jobs. To address this need, I designed the **signac-dashboard** application for web-based data visualization of **signac** projects. This application can be used to view job state points, job documents, images, text, video, file lists, and other types of content held in a **signac** data space. An example of the project used with the data presented in Chapter III is shown in Figure 6.3.

6.5.1 Dashboard Features

The dashboard can be broken into a few components. *Modules* generate one or more *cards* for each job in the **signac** project. Modules can be enabled or disabled by the user at run time, to condense or expand the amount of information shown.


Jobs
signac-dashboard
Search jobs
cP8-Cr3Si
8510da2f
Modules
Update
Images: a15.png
Images: cluster_counts_msm-qls-wls-ave-only_2.png
Images: rdf.png
Images: render_clustered_by_msm-qls-wls-with-ave_3.png
Images: umap_msm-qls-wls-with-ave_clustered_by_msm-qls-wls-with-ave_3.png
Images: umap_msm-qls-wls_clustered_by_msm-qls-wls-with-ave_3.png
RDF
Number of Particles
Simulation Time
Cluster Labels

☐ Statepoint Parameters
☐ Job Document
☐ Final Structure
☐ File List
☑ Images

Figure 6.3: An example of **signac-dashboard** showing the image viewer modules with data collected for Chapter III.

For example, the *ImageViewer* module generates a card for each PNG, JPG, or GIF image present in the job workspace. These cards are titled according to the file name being displayed, and display a thumbnail image that can be clicked to expand the image size. The modules currently included with **signac-dashboard** are shown in Table 6.1. Users can write custom modules with Python. Custom modules can also include JavaScript that can be served alongside the dashboard for interactive content. For example, the `Notes` and `DocumentEditor` modules include JavaScript code that allows data to be submitted to the dashboard server dynamically (without reloading the full page), when users click “Save” next to a text box in the web interface.

`DocumentEditor`	Provides an interface to edit the job document.
`DocumentList`	Displays the job document.
`FileList`	Lists files in the job workspace with download links.
`FlowStatus`	Show job labels from a flow.FlowProject.
`ImageViewer`	Displays images in the job workspace that match a glob.
`Notes`	Displays a text box that is synced to the job document.
`StatepointList`	Displays the job state point. (See Figure 6.4(b).)
`TextDisplay`	Render custom text or Markdown in a card.
`VideoViewer`	Displays videos in the job workspace that match a glob.

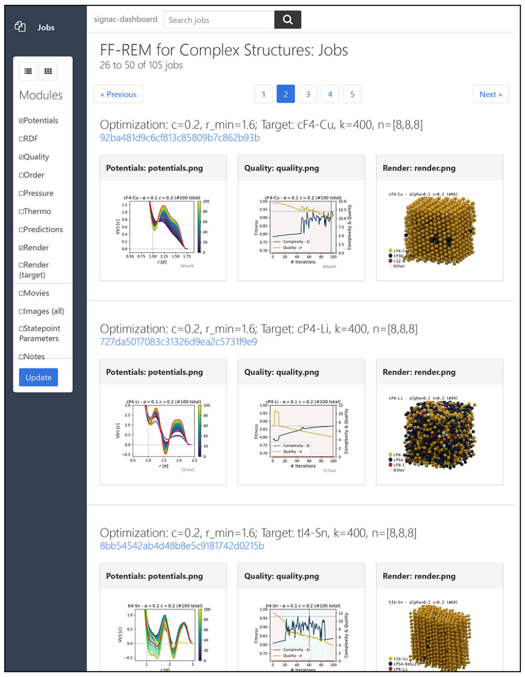
Table 6.1: List of **signac-dashboard** modules.

The class `signac_dashboard.Dashboard` provides the basic infrastructure for configuring a dashboard, connecting it to a **signac** project, and choosing modules. The configuration includes options for the host and port where the web server will be run, as well as options for debugging, profiling the application, and managing web site security. Most users only need to use a few of the methods of this class, such as defining a function `job_title(job)` that returns a string of a human-readable name for the given job (e.g., a name based on its state point keys and values) – see Figure 6.4(a) for an example.

The dashboard has two main viewing modes: list view and grid view. In list view, users are simply shown a list of job titles. This view does not show any of the cards generated for each job until the user clicks on the job title to see that job's page. The grid view shows cards generated for all jobs on the page (by default, the dashboard shows 25 jobs per page).

The dashboard's search feature works the same way as the **signac** command line interface's `signac find` command, allowing users to type queries in "simple syntax" like `pressure 10` to match jobs that have a key "pressure" with value 10 in their state point. The full JSON syntax for finding jobs is also supported, like {`"pressure": 10`}.

(a)



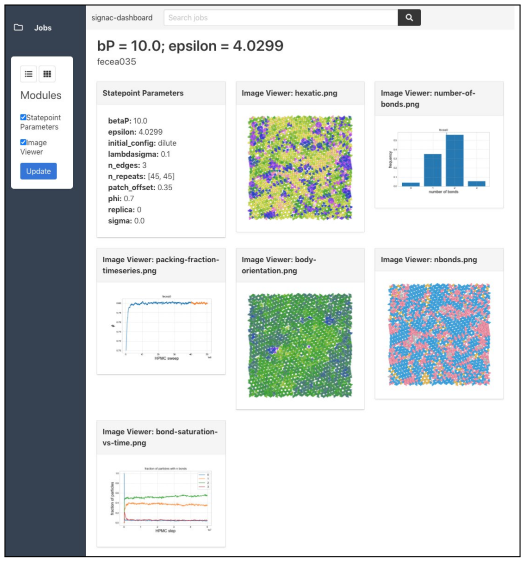
FF-REM for Complex Structures: Jobs
26 to 50 of 105 jobs

Optimization: c=0.2, r_min=1.6; Target: cF4-Cu, k=400, n=[8,8,8]
92ba481d9c6cf813c85809b7c862b93b

Optimization: c=0.2, r_min=1.6; Target: cP4-Li, k=400, n=[8,8,8]
727da5017083c31326d9ea2c5731f9e9

Optimization: c=0.2, r_min=1.6; Target: tI4-Sn, k=400, n=[8,8,8]
8bb54542ab4d48b8e5c9181742d0215b

(b)



Jobs
signac-dashboard
Search jobs
Modules
☑Statepoint Parameters
☑Image Viewer
Update
Statepoint Parameters
Image Viewer: hexatic.png
Image Viewer: number-of-bonds.png
Image Viewer: packing-fraction-timeseries.png
Image Viewer: body-orientation.png
Image Viewer: nbonds.png
Image Viewer: bond-saturation-vs-time.png

bP = 10.0; epsilon = 4.0299
fecea035

betaP: 10.0
epsilon: 4.0299
initial_config: dilute
lambdasigma: 0.1
n_edges: 3
n_repeats: [45, 45]
patch_offset: 0.35
phi: 0.7
replica: 0
sigma: 0.0

Figure 6.4: Additional examples of **signac-dashboard**. **(a)** Custom job titles are shown for each job, generated by a overridden `job_title` function. **(b)** The `StatepointList` module shows the keys and values for the job state point in the upper left.

6.6 Outlook and Future Directions

The **signac** data management framework offers a comprehensive set of tools for computational researchers to manage and scale file-based workflows with virtually any combination of scientific applications, machine learning pipelines, or data processing tools. The features I developed for the framework have improved its ability to handle and store binary data, scale up to larger data spaces, and enable rapid visualization of computed results.

As the framework continues to grow in popularity among computational researchers, I also hope to see the framework adopted by experimental users. Essentially all experimental apparatuses include complex sensors, cameras, detectors, and/or robots interfaced with computers and programmed for data collection. Enabling these types of devices to store their file-based data in a **signac** project through hardware interfaces controlled by Python would allow for reproducible analyses to be run directly from the device outputs.

CHAPTER VII

Conclusions

In this dissertation, I have discussed applications of physics-informed machine learning to two important problems in materials design, as well as developed two major software packages that will accelerate future discoveries through powerful data analysis and reproducible workflows.

First, I developed and applied the Continuous Topological Order Parameter, a novel method using unsupervised machine learning for the characterization of crystallization in self-assembling colloidal systems. I hope that this analysis method and the motifs it reveals can be used for engineering applications, such as identifying the connections between local motifs and the quality or yield of the crystal being formed. Though engineering crystallization pathways is a major challenge, having a better understanding of the microscopic processes underlying crystallization is an important step.

I began this dissertation with a few questions about the microscopic behaviors of particles that lead to their crystallization pathways. While these questions are likely to remain a challenge for the near future, the progress shared in this dissertation may be a helpful step towards answering them. In particular, the CTOP and UMAP approach enables us to autonomously detect local motifs in many types of ordered structures, including simple crystals, Frank-Kasper phases, quasicrystals, and solid-solid phase transitions, whether simulated with molecular dynamics or Monte Carlo methods, or performed with isotropic pair potentials or anisotropic hard particles. The pathways traversed by particles in these simulations can be directly analyzed in the CTOP manifold, and pathways may be compared against one another with no need for constructing separate training data. I hope that this method can be coupled with other tools such as transition path sampling, to enable efficient and autonomous explorations of new phase diagrams, the measurement of energy barriers and nucleation rates in crystal formation, and engineering of material properties through the direct manipulation of kinetic pathways.

Next, I discussed the application of machine learning for predicting photonic properties of crystal structures. I designed a convolutional neural network with TensorFlow and deployed it on Oak Ridge National Laboratory's Summit supercomputer. While the convolutional neural network's predictive capability was limited, I have begun exploration of using densities of states and equivariant neural networks that may be able to improve on these results. From both the crystallization pathways and photonics projects, I conclude that the complex challenges of physics-based machine learning can be addressed through the selection of features and model architectures that are well-motivated for the physics of the problem. The challenge lies in finding those representations suitable for a given question, and to extend those representations for use in related problems.

I summarized my contributions to the development of a suite of powerful, efficient, and flexible analysis tools for particle systems in the **freud** library. The **freud** library is already heavily used by the soft matter community and its capabilities have shown to be essential for the kinds of large-scale, data-driven simulations that I hope become commonplace in the field. Moreover, its general approach to the computations found in materials physics have shown to be useful in interdisciplinary applications.

Lastly, I have enabled scalable workflows through the **signac** framework for data management and workflow automation. The **signac** framework has found a wide user base from several fields of science and engineering, and continues to find new applications driven by the complex needs of computational researchers.

I will conclude with a discussion of scientific software and its role in advancing computational research.

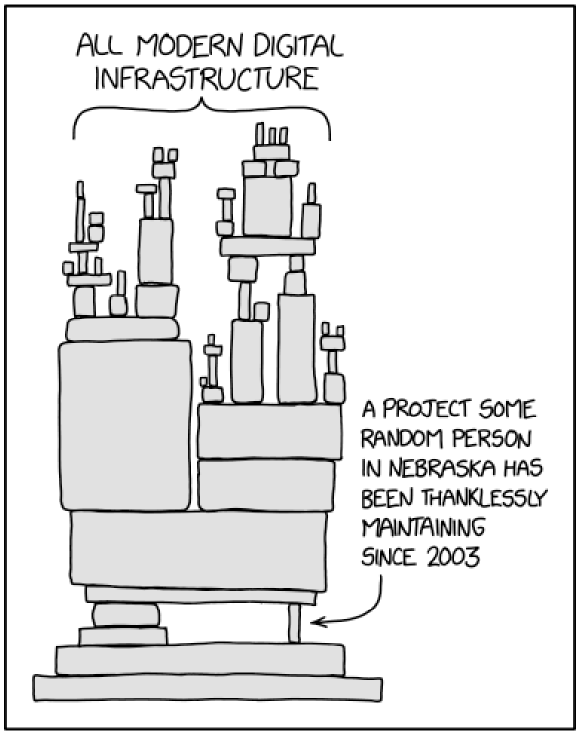

ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

Figure 7.1: "Dependency" by Randall Munroe. Alternate text: "Someday ImageMagick will finally break for good and we'll have a long period of scrambling as we try to reassemble civilization from the rubble." Reproduced under a Creative Commons Attribution-NonCommercial 2.5 License. Source: https://xkcd.com/2347/

Scientific software is a key infrastructural component of computational research. Packages such as **freud** and **signac** take a combination of technical skills, project vision, and teamwork to develop and maintain. As the field of materials science strives to answer questions through computation, we rely more and more on a tree of complex software packages built upon one another (see Figure 7.1). Combining machine learning and data science tools with molecular simulations quickly ascends this tower of complexity – for instance, **freud** has 31 software dependencies, while cuML (which I used for its GPU-accelerated UMAP algorithm in Chapter III) has 88. However, the benefits of easily usable, freely available, and open-source software dramatically exceed the costs. With well-developed software that includes documentation, examples, and tests, computational research can be heavily accelerated.

Since the **signac** framework was created in 2015, our group has seen many PhD students pass down projects to newer students, where the organizational strategy of the framework has made it easier to understand the inner workings of the project and its core scientific aims. This has been seen in other labs using **signac** as well. Matthew Thompson, a **signac** user from Vanderbilt University, said, "**signac** has revolutionized the way we use molecular simulations for scientific research. It allows us to focus our efforts on the scientific objectives and not worry about where our data is, how our data is structured, how to submit thousands of jobs to a cluster, or if our future selves can remember what a simulation was meant to focus on." Mike Henry, a **signac** user who has become a maintainer of the software, said, "I needed to pull some data from a project that another grad student did. Because they used **signac**, it was easy to use the commands `$ signac schema` and `$ signac find` to find exactly what I needed." He added, "No one will be able to find/figure out the stuff I did before I used the **signac** framework." Tools like the **freud** library and **signac** framework form a foundation upon which scalable workflows can be built, enabling researchers to generate reproducible and ready-to-share data. The continually expanding number of users (and citations) of these software packages indicates their utility and underscores the wide range of problems they can help to solve. I look forward to seeing these and other related tools continue to grow in their capabilities and usage, which reflects the scientific progress and discoveries that they enable.

APPENDICES

APPENDIX A

Analyzing Particle Systems for Machine Learning and Data Visualization with freud

This appendix is reproduced from Dice, B., Ramasubramani, V., Harper, E., Spellings, M., Anderson, J., and Glotzer, S. (2019). Analyzing Particle Systems for Machine Learning and Data Visualization with freud. *Proceedings of the 18th Python in Science Conference*, 27–33. DOI: 10.25080/Majora-7ddc1dd1-004. This is an open-access article distributed under the terms of the Creative Commons Attribution License.

A.1 Abstract

The **freud** Python library analyzes particle data output from molecular dynamics simulations. The library's design and its variety of high-performance methods make it a powerful tool for many modern applications. In particular, **freud** can be used as part of the data generation pipeline for machine learning (ML) algorithms for analyzing particle simulations, and it can be easily integrated with various simulation visualization tools for simultaneous visualization and real-time analysis. Here, we present numerous examples both of using **freud** to analyze nano-scale particle systems by coupling traditional simulational analyses to machine learning libraries and of visualizing per-particle quantities calculated by **freud** analysis methods. We

include code and examples of this visualization, showing that in general the introduction of **freud** into existing ML and visualization workflows is smooth and unintrusive. We demonstrate that among Python packages used in the computational molecular sciences, **freud** offers a unique set of analysis methods with efficient computations and seamless coupling into powerful data analysis pipelines.

A.2 Introduction

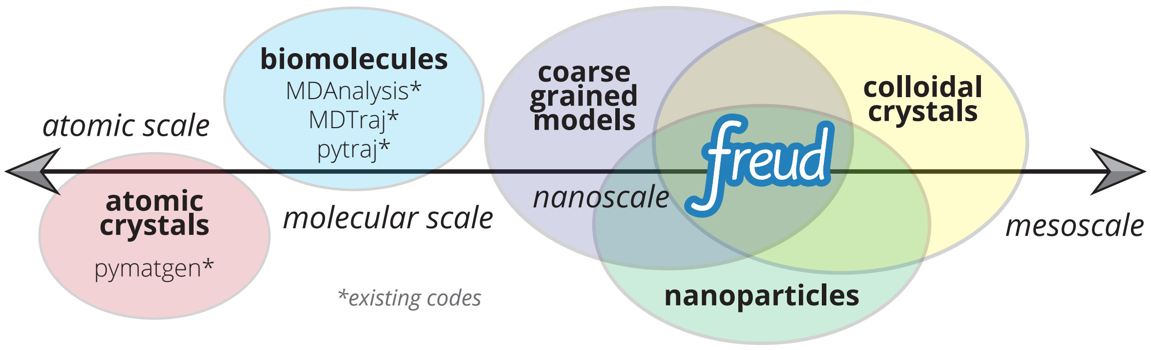


Figure A.1: Common Python tools for simulation analysis at varying length scales. The **freud** library is designed for nanoscale systems, such as colloidal crystals and nanoparticle assemblies. In such systems, interactions are described by coarse-grained models where particles' atomic constituents are often irrelevant and particle anisotropy (non-spherical shape) is common, thus requiring a generalized concept of particle "types" and orientation-sensitive analyses. These features contrast the assumptions of most analysis tools designed for biomolecular simulations and materials science.

The availability of "off-the-shelf" molecular dynamics engines (e.g., HOOMD-blue [133, 134], LAMMPS [135], GROMACS [136]) has made simulating complex systems possible across many scientific fields. Simulations of systems ranging from large biomolecules to colloids are now common, allowing researchers to ask new questions about reconfigurable materials [137] and develop coarse-graining approaches to access increasing timescales [138]. Various tools have arisen to facilitate the analysis of these simulations, many of which are immediately interoperable with the most popular simulation tools. The **freud** library is one such analysis package that differentiates itself from others through its focus on colloidal and nano-scale systems.

Due to their diversity and adaptability, colloidal materials are a powerful model system for exploring soft matter physics [139]. Such materials are also a viable platform for harnessing photonic [137], plasmonic [140], and other useful structurally-derived properties. In colloidal systems, features like particle anisotropy play an important role in creating complex crystal structures, some of which have no atomic analogues [23]. Design spaces encompassing wide ranges of particle morphology [23] and interparticle interactions [15] have been shown to yield phase diagrams filled with complex behavior.

The **freud** Python package offers a unique feature set that targets the analysis of colloidal systems. The library avoids trajectory management and the analysis of chemically bonded structures, which are the province of most other analysis platforms like MDAnalysis and MDTraj (see also Figure A.1) [117, 118]. In particular, **freud** excels at performing analyses based on characterizing local particle environments, which makes it a powerful tool for tasks such as calculating order parameters to track crystallization or finding prenucleation clusters. Among the unique methods present in **freud** are the potential of mean force and torque, which allows users to understand the effects of particle anisotropy on entropic self-assembly [24, 119, 120, 121, 82], and various tools for identifying and clustering particles by their local crystal environments [44]. All such tasks are accelerated by **freud**'s extremely fast neighbor finding routines and are automatically parallelized, making it an ideal tool for researchers performing peta- or exascale simulations of particle systems. The **freud** library's scalability is exemplified by its use in computing correlation functions on systems of over a million particles, calculations that were used to illuminate the elusive hexatic phase transition in two-dimensional systems of hard polygons [82]. More details on the use of **freud** can be found in Ref. [27]. In this paper, we will demonstrate that **freud** is uniquely well-suited to usage in the context of data pipelines for visualization and machine learning applications.

A.2.1 Data Pipelines

The **freud** package is especially useful because it can be organically integrated into a data pipeline. Many research tasks in computational molecular sciences can be expressed in terms of data pipelines; in molecular simulations, such a pipeline typically involves:

1. **Generating** an input file that defines a simulation.
2. **Simulating** the system of interest, saving its trajectory to a file.
3. **Analyzing** the resulting data by computing and storing various quantities.
4. **Visualizing** the trajectory, using colors or styles determined from previous analyses.

However, in modern workflows the lines between these stages is typically blurred, particularly with respect to analysis. While direct visualization of simulation trajectories can provide insights into the behavior of a system, integrating higher-order analyses is often necessary to provide real-time interpretable visualizations in that allow researchers to identify meaningful features like defects and ordered domains of self-assembled structures. Studies of complex systems are also often aided or accelerated by a real-time coupling of simulations with on-the-fly analysis. This simultaneous usage of simulation and analysis is especially relevant because modern machine learning techniques frequently involve wrapping this pipeline entirely within a higher-level optimization problem, since analysis methods can be used to construct objective functions targeting a specific materials design problem, for instance.

Following, we provide demonstrations of how **freud** can be integrated with popular tools in the scientific Python ecosystem like TensorFlow, Scikit-learn, SciPy, or Matplotlib. In the context of machine learning algorithms, we will discuss how the analyses in **freud** can reduce the $6N$-dimensional space of particle positions and orientations into a tractable set of features that can be fed into machine learning algorithms. We will further show that **freud** can be used for visualizations even outside of scripting contexts, enabling a wide range of forward-thinking applications including Jupyter notebook integrations, versatile 3D renderings, and integration with various standard tools for visualizing simulation trajectories. These topics are aimed at computational molecular scientists and data scientists alike, with discussions of real-world usage as well as theoretical motivation and conceptual exploration. The full source code of all examples in this paper can be found online.[1]

A.3 Performance and Integrability

Using **freud** to compute features for machine learning algorithms and visualization is straightforward because it adheres to a UNIX-like philosophy of providing modular, composable features. This design is evidenced by the library's reliance on NumPy arrays [141] for all inputs and outputs, a format that is naturally integrated with most other tools in the scientific Python ecosystem. In general, the analyses in **freud** are designed around analyses of raw particle trajectories, meaning that the inputs are typically $(N, 3)$ arrays of particle positions and $(N, 4)$ arrays of particle orientations, and analyses that involve many frames over time use `accumulate` methods that are called once for each frame. This general approach enables **freud** to be used for a range of input data, including molecular dynamics and Monte Carlo simulations as well as experimental data (e.g., positions extracted via particle tracking)

[1] `https://github.com/glotzerlab/freud-examples`

in both 3D and 2D. The direct usage of numerical arrays indicates a different usage pattern than that of tools, such as MDAnalysis [117] and MDTraj [118], for which trajectory parsing is a core feature. Due to the existence of many such tools which are capable of reading simulation engines' output files, as well as certain formats like `gsd`[2] that provide their own parsers, **freud** eschews any form of trajectory management and instead relies on other tools to provide input arrays. If input data is to be read from a file, binary data formats such as `gsd` or NumPy's `npy` or `npz` are strongly preferred for efficient I/O. Though it is possible to use a library like Pandas to load data stored in a comma-separated value (CSV) or other text-based data format, such files are often much slower when reading and writing large numerical arrays. Decoupling **freud** from file parsing and specific trajectory representations allows it to be efficiently integrated into simulations, machine learning applications, and visualization toolkits with no I/O overhead and limited additional code complexity, while the universal usage of NumPy arrays makes such integrations very natural.

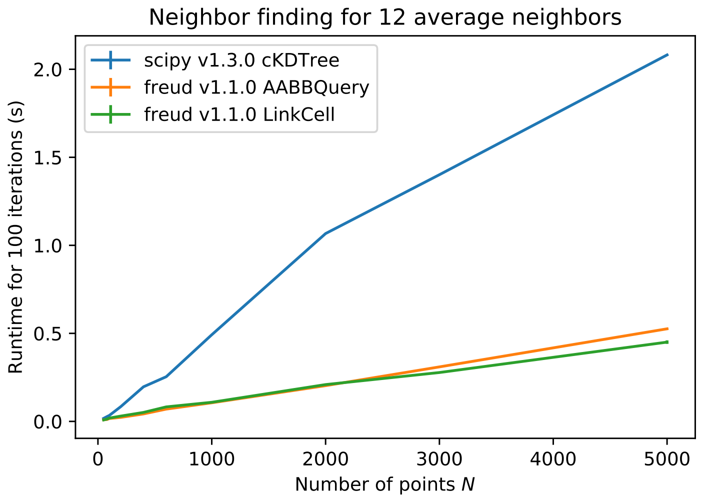


Figure A.2: Comparison of runtime for neighbor finding algorithms in **freud** and SciPy for varied system sizes. See text for details.

[2]`https://github.com/glotzerlab/gsd`

In keeping with this focus on composable features, **freud** also abstracts and directly exposes the task of finding particle neighbors, the task most central to all other analyses in **freud**. Since neighbor finding is a common need, the neighbor finding routines in **freud** are highly optimized and natively support periodic systems, a crucial feature for any analysis of particle simulations (which often employ periodic boundary conditions). In Figure A.2, a comparison is shown between the neighbor finding algorithms in **freud** and SciPy [142]. For each system size, N particles are uniformly distributed in a 3D periodic cube such that each particle has an average of 12 neighbors within a distance of $r_{cut} = 1.0$. Neighbors are found for each particle by searching within the cutoff distance r_{cut}. The methods compared are `scipy.spatial.cKDTree`'s `query_ball_tree`, `freud.locality.AABBQuery`'s `queryBall`, and `freud.locality.LinkCell`'s `compute`. The benchmarks were performed with 5 replicates on a 3.6 GHz Intel Core i3-8100B processor with 16 GB 2667 MHz DDR4 RAM.

Evidently, **freud** performs very well on this core task and scales well to larger systems. The parallel C++ backend implemented with Cython and Intel Threading Building Blocks makes **freud** perform quickly even for large systems [143, 144]. Furthermore, **freud** supports periodicity in arbitrary triclinic volumes, a common feature found in many simulations. This support distinguishes it from other tools like `scipy.spatial.cKDTree`, which only supports cubic boxes. The fast neighbor finding in **freud** and the ease of integrating its outputs into other analyses not only make it easy to add fast new analysis methods into **freud**, they are also central to why **freud** can be easily integrated into workflows for machine learning and visualization.

A.4 Machine Learning

A wide range of problems in soft matter and nano-scale simulations have been addressed using machine learning techniques, such as crystal structure identification [29]. In machine learning workflows, **freud** is used to generate features, which are then used in classification or regression models, clusterings, or dimensionality reduction methods. For example, Harper et al. used **freud** to compute the cubatic order parameter and generate high-dimensional descriptors of structural motifs, which were visualized with t-SNE dimensionality reduction [145, 62]. The library has also been used in the optimization and inverse design of pair potentials [15], to compute fitness functions based on the radial distribution function. The open-source `pythia`[3] library offers a number of descriptor sets useful for crystal structure identification, leveraging **freud** for fast computations. Included among the descriptors in `pythia` are quantities based on bond angles and distances, spherical harmonics, and Voronoi diagrams.

Computing a set of descriptors tuned for a particular system of interest (e.g., using values of Q_l, the higher-order Steinhardt W_l parameters, or other order parameters provided by **freud**) is possible with just a few lines of code. Descriptors like these (exemplified in the `pythia` library) have been used with TensorFlow for supervised and unsupervised learning of crystal structures in complex phase diagrams [29, 106].

Another useful module for machine learning with **freud** is `freud.cluster`, which uses a distance-based cutoff to locate clusters of particles while accounting for 2D or 3D periodicity. Locating clusters in this way can identify crystalline grains, helpful for building a training set for machine learning models.

To demonstrate a concrete example, we focus on a common challenge in molecular

[3]`https://github.com/glotzerlab/pythia`

sciences: identifying crystal structures. Recently, several approaches have been developed that use machine learning for detecting ordered phases [146, 29, 147, 33, 45]. The Steinhardt order parameters are often used as a structural fingerprint, and are derived from rotationally invariant combinations of spherical harmonics. In the example below, we create face-centered cubic (fcc), body-centered cubic (bcc), and simple cubic (sc) crystals with added Gaussian noise, and use Steinhardt order parameters with a support vector machine to train a simple crystal structure identifier. Steinhardt order parameters characterize the spherical arrangement of neighbors around a central particle, and combining values of Q_l for a range of l often gives a unique signature for simple crystal structures. This example demonstrates a simple case of how **freud** can be used to help solve the problem of structural identification, which often requires a sophisticated approach for complex crystals.

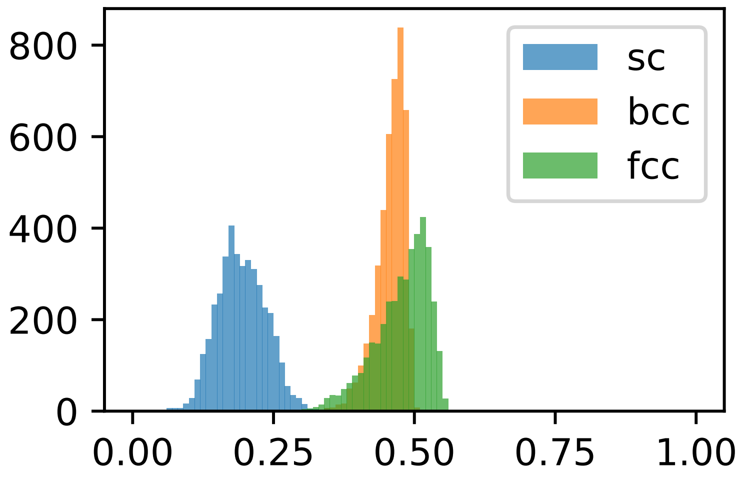


Figure A.3: Histogram of the Steinhardt Q_6 order parameter for 4000 particles in simple cubic, body-centered cubic, and face-centered cubic structures with added Gaussian noise.

In Figure A.3, we show the distribution of Q_6 values for sample structures with 4000 particles. Here, we demonstrate how to compute the Steinhardt Q_6, using neighbors found via a periodic Voronoi diagram. Neighbors with small facets in the Voronoi polytope are filtered out to reduce noise.

```python
import freud
import numpy as np
from util import make_fcc

def get_features(box, positions, structure):
    # Create a Voronoi compute object
    voro = freud.voronoi.Voronoi(
        box, buff=max(box.L)/2)
    voro.computeNeighbors(positions)

    # Filter the Voronoi NeighborList
    nlist = voro.nlist
    nlist.filter(nlist.weights > 0.1)

    # Compute Steinhardt order parameters
    features = {}
    for l in [4, 6, 8, 10, 12]:
        ql = freud.order.LocalQl(
            box, rmax=max(box.L)/2, l=l)
        ql.compute(positions, nlist)
        features['q{}'.format(l)] = ql.Ql.copy()

    return features

# Create a freud box object and an array of
# 3D positions for a face-centered cubic
# structure with 4000 particles
fcc_box, fcc_positions = make_fcc(
    nx=10, ny=10, nz=10, noise=0.1)

structures = {}
structures['fcc'] = get_features(
    fcc_box, fcc_positions, 'fcc')
# ... repeat for all structures
```

Then, using Pandas and Scikit-learn, we can train a support vector machine to identify these structures:

```python
# Build dictionary of DataFrames,
# labeled by structure
structure_dfs = {}
for i, struct in enumerate(structures):
    df = pd.DataFrame.from_dict(structures[struct])
    df['class'] = i
    structure_dfs[struct] = df

# Combine DataFrames for input to SVM
df = pd.concat(structure_dfs.values())
df = df.reset_index(drop=True)

from sklearn.preprocessing import normalize
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

# We use the normalized Steinhardt order parameters
# to predict the crystal structure
X = df.drop('class', axis=1).values
X = normalize(X)
y = df['class'].values
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)

svm = SVC()
svm.fit(X_train, y_train)
print('Score:', svm.score(X_test, y_test))
# The model is ~98% accurate.
```

To interpret crystal identification models like this, it can be helpful to use a dimensionality reduction tool such as Uniform Manifold Approximation and Projection (UMAP) [22], as shown in Figure A.4. The low-dimensional UMAP projection shown is generated directly from the Pandas `DataFrame`:

```
from umap import UMAP
umap = UMAP()

# Project the high-dimensional descriptors
# to a two dimensional manifold
data = umap.fit_transform(df)
plt.plot(data[:, 0], data[:, 1])
```

A.5 Visualization

Many analyses performed by the **freud** library provide a `plot(ax=None)` method (new in v1.2.0) that allows their computed quantities to be visualized with Matplotlib. Additionally, these plottable analyses offer IPython representations, allowing Jupyter notebooks to render a graph such as a radial distribution function $g(r)$ just by returning the compute object at the end of a cell. Analyses like the radial distribution function or correlation functions return data that is binned as a one-dimensional histogram – these are visualized with a line graph via `matplotlib.pyplot.plot`, with the bin locations and bin counts given by properties of the compute object. Other classes provide multi-dimensional histograms, like the Gaussian density or Potential

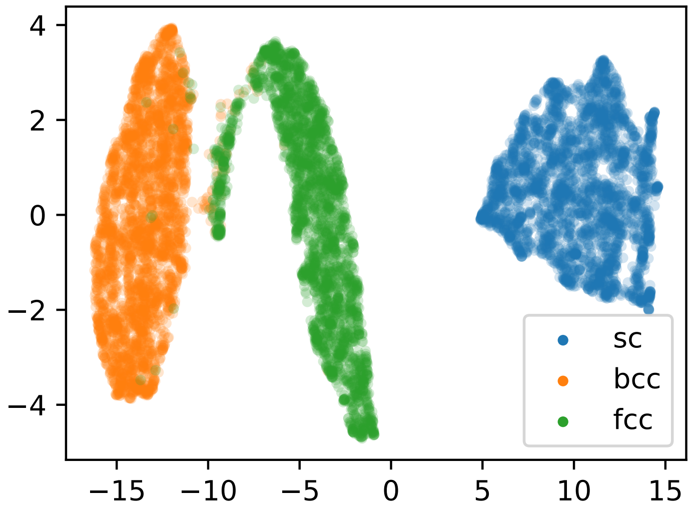


Figure A.4: UMAP of particle descriptors computed for simple cubic, body-centered cubic, and face-centered cubic structures of 4000 particles with added Gaussian noise. The particle descriptors include Q_l for $l \in \{4, 6, 8, 10, 12\}$. Some noisy configurations of bcc can be confused as fcc and vice versa, which accounts for the small number of errors in the support vector machine's test classification.

of Mean Force and Torque, which are plotted with `matplotlib.pyplot.imshow`.

The most complex case for visualization is that of per-particle properties, which also comprises some of the most useful features in **freud**. Quantities that are computed on a per-particle level can be continuous (e.g., Steinhardt order parameters) or discrete (e.g., clustering, where the integer value corresponds to a unique cluster ID). Continuous quantities can be plotted as a histogram over particles, but typically the most helpful visualizations use these quantities with a color map assigned to particles in a two- or three-dimensional view of the system itself. For such particle visualizations, several open-source tools exist that interoperate well with **freud**. Below are examples of how one can integrate **freud** with `plato`[4], `fresnel`[5], and OVITO[6] [9].

[4]`https://github.com/glotzerlab/plato`
[5]`https://github.com/glotzerlab/fresnel`
[6]`https://ovito.org`

A.5.1 plato

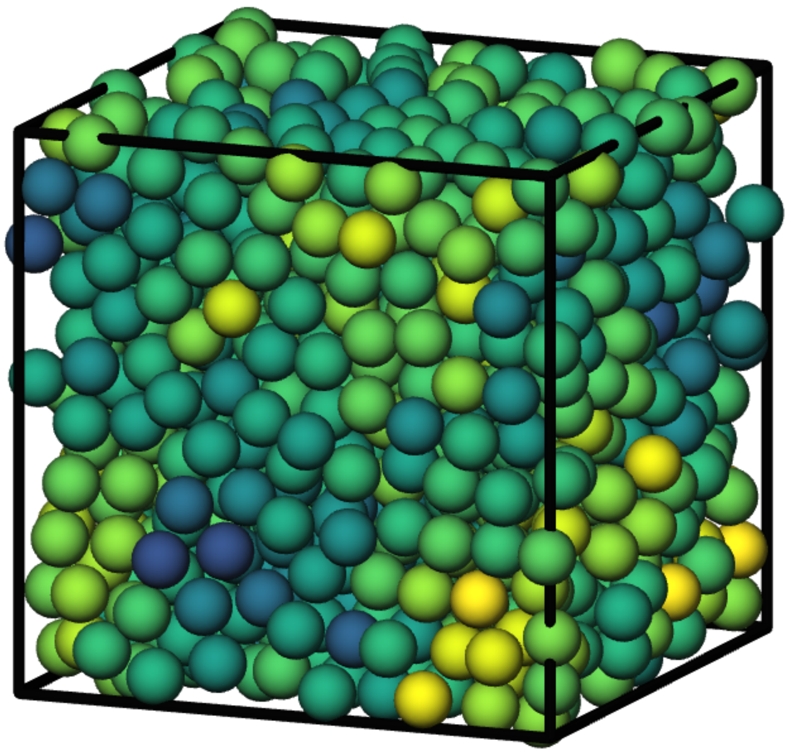


Figure A.5: Interactive visualization of a Lennard-Jones particle system, rendered in a Jupyter notebook using `plato` with the `pythreejs` backend.

`plato` is an open-source graphics package that expresses a common interface for defining two- or three-dimensional scenes which can be rendered as an interactive Jupyter widget or saved to a high-resolution image using one of several backends (PyThreejs, Matplotlib, `fresnel`, POVray[7], and Blender[8], among others). Below is an example of how to render particles from a HOOMD-blue snapshot, colored by the density of their local environment [133, 134]. The result is shown in Figure A.5.

```
import plato
import plato.draw.pythreejs as draw
import numpy as np
import matplotlib.cm
import freud
from sklearn.preprocessing import minmax_scale
```

[7]`https://www.povray.org`
[8]`https://www.blender.org`

```python
# snap comes from a previous HOOMD-blue simulation
box = freud.box.Box.from_box(snap.box)
positions = snap.particles.position

# Compute the local density of each particle
ld = freud.density.LocalDensity(
    r_cut=3.0, volume=1.0, diameter=1.0)
ld.compute(box, positions)

# Create a scene for visualization,
# colored by local density
radii = 0.5 * np.ones(len(positions))
colors = matplotlib.cm.viridis(
    minmax_scale(ld.density))
spheres_primitive = draw.Spheres(
    positions=positions,
    radii=radii,
    colors=colors)
scene = draw.Scene(spheres_primitive, zoom=2)
scene.show()  # Interactive view in Jupyter
```

A.5.2 fresnel

`fresnel`[9] is a GPU-accelerated ray tracer designed for particle simulations, with customizable material types and scene lighting, as well as support for a set of common anisotropic shapes. Its feature set is especially well suited for publication-quality graphics. Its use of ray tracing also means that an image's rendering time scales most strongly with the image size, instead of the number of particles – a desirable feature for extremely large simulations. An example of how to integrate `fresnel` is shown below and rendered in Figure A.6.

```python
# Generate a snapshot of tetrahedra using HOOMD-blue
import hoomd
import hoomd.hpmc
hoomd.context.initialize('')

# Create an 8x8x8 simple cubic lattice
system = hoomd.init.create_lattice(
    unitcell=hoomd.lattice.sc(a=1.5), n=8)

# Create tetrahedra, configure HPMC integrator
mc = hoomd.hpmc.integrate.convex_polyhedron(seed=123)
mc.set_params(d=0.2, a=0.1)
vertices = [( 0.5, 0.5, 0.5),
            (-0.5,-0.5, 0.5),
            (-0.5, 0.5,-0.5),
            ( 0.5,-0.5,-0.5)]
```

[9]https://github.com/glotzerlab/fresnel

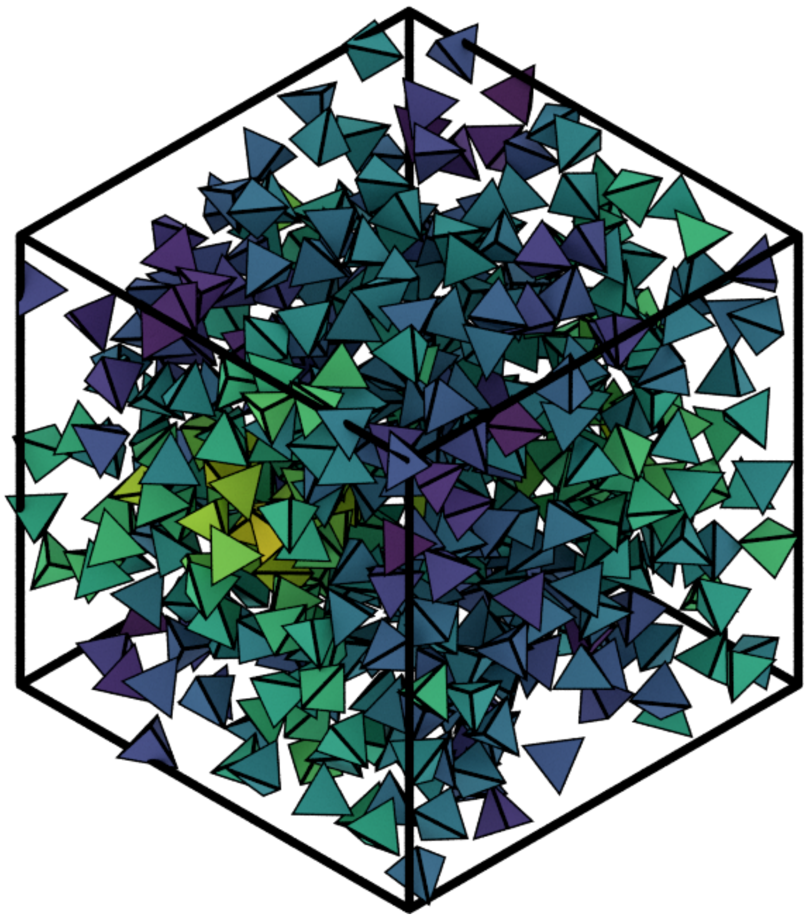


Figure A.6: Hard tetrahedra colored by local density, path traced with `fresnel`.

```
mc.shape_param.set('A', vertices=vertices)

# Run for 5,000 steps
hoomd.run(5e3)
snap = system.take_snapshot()

# Import analysis & visualization libraries
import fresnel
import freud
import matplotlib.cm
from matplotlib.colors import Normalize
import numpy as np
device = fresnel.Device()

# Compute local density and prepare geometry
poly_info = \
    fresnel.util.convex_polyhedron_from_vertices(
        vertices)
positions = snap.particles.position
orientations = snap.particles.orientation
box = freud.box.Box.from_box(snap.box)
ld = freud.density.LocalDensity(3.0, 1.0, 1.0)
```

```
ld.compute(box, positions)
colors = matplotlib.cm.viridis(
    Normalize()(ld.density))
box_points = np.asarray([
    box.makeCoordinates(
        [[0, 0, 0], [0, 0, 0], [0, 0, 0],
         [1, 1, 0], [1, 1, 0], [1, 1, 0],
         [0, 1, 1], [0, 1, 1], [0, 1, 1],
         [1, 0, 1], [1, 0, 1], [1, 0, 1]]),
    box.makeCoordinates(
        [[1, 0, 0], [0, 1, 0], [0, 0, 1],
         [1, 0, 0], [0, 1, 0], [1, 1, 1],
         [1, 1, 1], [0, 1, 0], [0, 0, 1],
         [0, 0, 1], [1, 1, 1], [1, 0, 0]])])

# Create scene
scene = fresnel.Scene(device)
geometry = fresnel.geometry.ConvexPolyhedron(
    scene, poly_info,
    position=positions,
    orientation=orientations,
    color=fresnel.color.linear(colors))
geometry.material = fresnel.material.Material(
    color=fresnel.color.linear([0.25, 0.5, 0.9]),
    roughness=0.8, primitive_color_mix=1.0)
geometry.outline_width = 0.05
box_geometry = fresnel.geometry.Cylinder(
    scene, points=box_points.swapaxes(0, 1))
box_geometry.radius[:] = 0.1
box_geometry.color[:] = np.tile(
    [0, 0, 0], (12, 2, 1))
box_geometry.material.primitive_color_mix = 1.0
scene.camera = fresnel.camera.fit(
    scene, view='isometric', margin=0.1)
scene.lights = fresnel.light.lightbox()

# Path trace the scene
fresnel.pathtrace(scene, light_samples=64,
                  w=800, h=800)
```

A.5.3 OVITO

OVITO is a GUI application with features for particle selection, making movies, and support for many trajectory formats [9]. OVITO has several built-in analysis functions (e.g., Polyhedral Template Matching), which complement the methods in **freud**. The Python scripting functionality built into OVITO enables the use of **freud** modules, demonstrated in the code below and shown in Figure A.7.

```
import freud

def modify(frame, input, output):
```

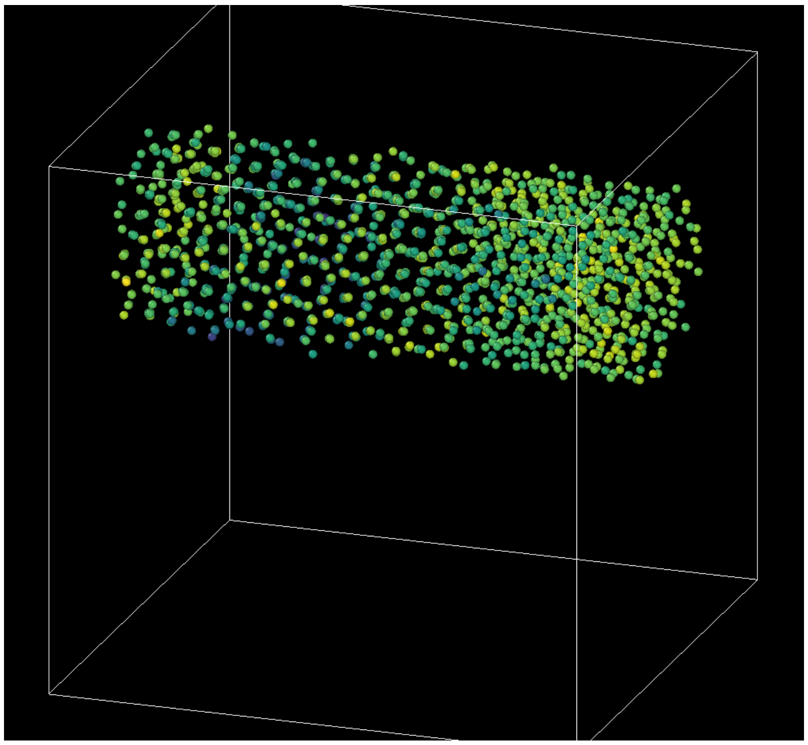


Figure A.7: A crystalline grain identified using **freud**'s `LocalDensity` module and cut out for display using OVITO. The image shows a $tP30$-CrFe structure formed from an isotropic pair potential optimized to generate this structure [15].

```
if input.particles != None:
    box = freud.box.Box.from_matrix(
        input.cell.matrix)
    ld = freud.density.LocalDensity(
        r_cut=3, volume=1, diameter=0.05)
    ld.compute(box, input.particles.position)
    output.create_user_particle_property(
        name='LocalDensity',
        data_type=float,
        data=ld.density.copy())
```

A.6 Conclusions

The **freud** library offers a unique set of high-performance algorithms designed to accelerate the study of nanoscale and colloidal systems. These algorithms are enabled by a fast, easy-to-use set of tools for identifying particle neighbors, a common first step in nearly all such analyses. The efficiency of both the core neighbor finding algorithms and the higher-level analyses makes them suitable for incorporation into real-time visualization environments, and, in conjunction with the transparent NumPy-based interface, allows integration into machine learning workflows using iterative optimization routines that require frequent recomputation of these analyses. The use of **freud** for real-time visualization has the potential to simplify and accelerate existing simulation visualization pipelines, which typically involve slower and less easily integrable solutions to performing real-time analysis during visualization. The application of **freud** to machine learning, on the other hand, opens up entirely new avenues of research based on treating well-known analyses of particle simulations as descriptors or optimization targets. In these ways, **freud** can facilitate research in the field of computational molecular science, and we hope these examples will spark new ideas for scientific exploration in this field.

A.7 Getting freud

The **freud** library is tested for Python 2.7 and 3.5+ and is compatible with Linux, macOS, and Windows. To install **freud**, execute

```
conda install -c conda-forge freud
```

or

```
pip install freud-analysis
```

Its source code is available on GitHub[10] and its documentation is available via ReadTheDocs.[11]

A.8 Acknowledgments

Thanks to Jin Soo Ihm for benchmarking the neighbor finding features of **freud** against SciPy. The **freud** library's code development and public code releases are supported by the National Science Foundation, Division of Materials Research under a

[10]`https://github.com/glotzerlab/freud`

[11]`https://freud.readthedocs.io`

Computational and Data-Enabled Science & Engineering Award # DMR 1409620 (2014-2018) and the Office of Advanced Cyberinfrastructure Award # OAC 1835612 (2018-2021). B.D. is supported by a National Science Foundation Graduate Research Fellowship Grant DGE 1256260. M.P.S acknowledges funding from the Toyota Research Institute; this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity. Data for Figure A.7 generated on the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575; XSEDE award DMR 140129.

APPENDIX B

signac: Data Management and Workflows for Computational Researchers

This appendix is reproduced from Dice, B. D., Butler, B. L., Ramasubramani, V., Travitz, A., Henry, M. M., Ojha, H., Wang, K. L., Adorf, C. S., Jankowski, E., and Glotzer, S. C. (2021). signac: Data Management and Workflows for Computational Researchers. *Proceedings of the 20th Python in Science Conference*, 23-32. DOI: 10.25080/majora-1b6fd038-003. This is an open-access article distributed under the terms of the Creative Commons Attribution License.

B.1 Abstract

The **signac** data management framework (https://signac.io) helps researchers execute reproducible computational studies, scales workflows from laptops to supercomputers, and emphasizes portability and fast prototyping. With **signac**, users can track, search, and archive data and metadata for file-based workflows and automate workflow submission on high performance computing (HPC) clusters. We will discuss recent improvements to the software's feature set, scalability, scientific applications, usability, and community. Newly implemented synced data structures, features for generalized workflow execution, and performance optimizations will be covered, as well as recent research using the framework and changes to the project's

outreach and governance as a response to its growth.

B.2 Introduction

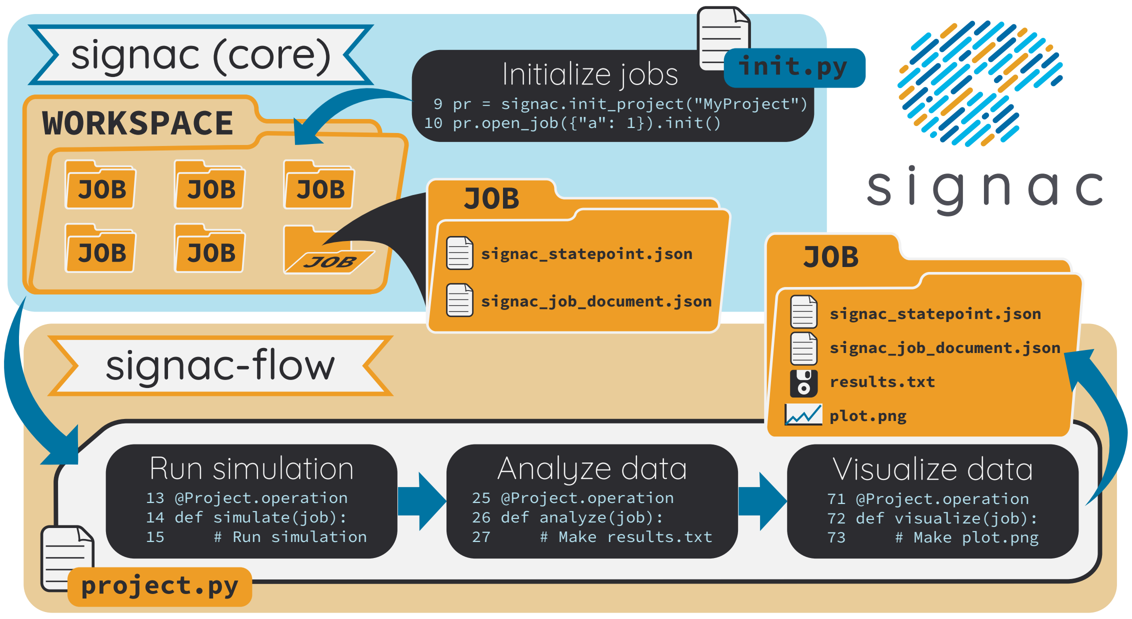


Figure B.1: Overview of the **signac** framework. Users first create a project, which initializes a workspace directory on disk. Users define state points which are dictionaries that uniquely identify a job. The workspace holds a directory for each job, containing JSON files that store the state point and job document. The job directory name is a hash of the state point's contents. Here, the `init.py` file initializes an empty project and adds one job with state point {`"a": 1`}. Next, users define a workflow using a subclass of **signac-flow**'s `FlowProject`. The workflow shown has three operations (simulate, analyze, visualize) that, when executed, produce two new files `results.txt` and `plot.png` in the job directory. Special thanks to Kelly Wang for contributing the design and concept of this figure.

Scientific research addresses problems where questions often change rapidly, data models are always in flux, and compute infrastructure varies widely from project to project. The **signac** data management framework [25] is a tool designed by researchers, for researchers, to simplify the process of prototyping and then performing reproducible scientific computations. It forgoes encoding complex data files into a database in favor of working directly on file systems, providing fast indexing utilities for a set of directories. Using **signac**, a data space on the file system can be initialized, searched, and modified using either a Python or command-line interface. By its general-purpose design, **signac** is agnostic to data content and format. The companion package **signac-flow** interacts with the data space to generate and analyze data through reproducible workflows that scale from laptops to supercomputers. Arbitrary shell commands can be run by **signac-flow** as part of a workflow, making it as flexible as a script in any language of choice.

This paper will focus on developments to the **signac** framework over the last 3 years, during which features, flexibility, usability, and performance have been greatly improved. The core data structures in **signac** have been overhauled to provide a powerful and generic implementation of *synced collections*, that we will leverage in future versions of **signac** to enable more performant data indexing and flexible data layouts. In **signac-flow**, we have added support for submitting *groups* of operations with conditional dependencies, allowing for more efficient utilization of large HPC resources. Further developments allow for operations to act on arbitrary subsets of the data space via *aggregation*, rather than single jobs alone. Moving beyond code development, this paper will also discuss the scientific research these features have enabled and organizational developments supported through key partnerships. We will share our project's experience in continuously revising project governance to encourage sustained contributions, adding more entry points for learning about the project (Slack support, weekly public office hours), and participating in Google Summer of Code in 2020 as a NumFOCUS Affiliated Project. Much of the work has been carried out in conjunction with the Molecular Simulation Design Framework (MoSDeF) [131], a National Science Foundation Cyberinfrastructure for Sustained Scientific Innovation (CSSI) effort.

B.3 Structure and Implementation

With **signac**, file-based data and metadata are organized in folders and JSON files, respectively (see Figure B.1). A **signac** data space, or *workspace*, contains jobs, which are individual directories associated with a single primary key known as a *state point* stored in a file `signac_statepoint.json` in that directory. The JSON files allow **signac** to index the data space, providing a database-like interface to a collection of directories. Arbitrary user data may be stored in user-created files in these jobs, although **signac** also provides convenient facilities for storing simple lightweight data or array-like data via JSON (the "job document") and HDF5 (the "job data") utilities. Readers seeking more details about **signac** are referred to past **signac** papers [25, 26] as well as the **signac** website[1] and documentation.[2]

This filesystem-based approach has both advantages and disadvantages. Its key advantages lie in flexibility and portability. The serverless design removes the need for any external running server process, making it easy to operate on any filesystem. The design is also intrinsically distributed, making it well suited for highly parallel workflows where multiple processes concurrently read or write file-based data stored in job directories. Conversely, this distributed approach precludes the performance advantages of centralized data stores with persistent indexes in memory. Typically, the **signac** approach works very well for projects up to 100,000 jobs, while significantly larger projects may have wait times that constrain interactive usage. These limits are inherent to **signac**'s use of small files for each job's state point, but the framework has been aggressively optimized and uses extensive caching/buffering to maximize the achievable throughput within this model.

[1]`https://signac.io`
[2]`https://docs.signac.io`

The framework is a strong choice for applications meeting one or more of the following criteria:

- input/output data is primarily file-based
- prototype research code where data schemas may change or evolve
- computations will use an HPC cluster
- the amount of computation per job is large
- parameter sweeps over a range of values (with values on a grid or dynamically determined by e.g., active learning)
- heterogeneous data (not all jobs have the same keys present in their state points)

For example, M. W. Thompson et al. in Ref. [148] used 396 jobs/state points to execute computer simulations of room-temperature ionic liquids with GROMACS [149, 150, 151, 152] simulations. The study investigated 18 compositions (by mass fraction) and 22 unique solvents from five chemical families (nitriles, alcohols, halocarbons, carbonyls, and glymes), with a state point for each pairing of mass fraction and solvent type.

Users working with large tabular data (e.g., flat files on disk or data from a SQL database) may prefer to use libraries like pandas [153, 154], Dask [155, 156], or RAPIDS [68] that are specifically designed for those use cases. However, it is possible to create a **signac** project with state points corresponding to each row, which may be a good use of **signac** if there is file-based data affiliated with each row's parameters.

Code examples of features presented in this paper can be found online.[3]

[3]`https://github.com/glotzerlab/signac-examples`

B.4 Applications of signac

The **signac** framework has been cited 54 times, according to Google Scholar, and has been used in a range of scientific fields with various types of computational workflows. Some of these studies include quantum calculations of small molecules [125], 4,480 simulations of epoxy curing (each containing millions of particles) [126], inverse design of pair potentials [15], identifying photonic band gaps in 151,593 crystal structures [13], benchmarking atom-density representations for use in machine learning [127], simulating fluid flow in polymer solutions [128], design of optical metamaterials [129], and economic analysis of drought risk in agriculture [130]. To date, **signac** users have built workflows utilizing a wide range of software packages including simulation tools such as Cassandra and MoSDeF-Cassandra [157, 158], foyer [159], GROMACS [149, 150, 151, 152], HOOMD-blue [28, 134, 160], mBuild [161], MIT Photonic Bands [101], Quantum-ESPRESSO [162], Rigorous Coupled Wave Analysis (RCWA) [163], and VASP [164], machine learning libraries including Keras [165], scikit-learn [5], and TensorFlow [106], and analysis libraries for postprocessing data such as freud [27], librascal [127], MDAnalysis [117], MDTraj [118], and OVITO [9]. Much of the published research using **signac** comes from chemical engineering, materials science, or physics, the fields of many of **signac**'s core developers and thus fields where the project has had greatest exposure. Computational materials research commonly requires large HPC resources with shared file systems, a use case where **signac** excels. However, there are many other fields with similar hardware needs where **signac** can be applied. These include simulation-heavy HPC workloads such as fluid dynamics, atomic/nuclear physics, or genomics, data-intensive fields such as economics or machine learning, and applications needing fast, flexible prototypes for optimization and data analysis.

While there is no “typical” **signac** project, factors such as computational complexity and data sizes offer some rough guidelines for when **signac**’s database-on-the-filesystem is appropriate. For instance, the time to check the status of a workflow depends on the number of jobs, number of operations, and number of conditions to evaluate for those jobs. Typical **signac** projects have 100 to 10,000 jobs, with each job workspace containing arbitrarily large data sizes (the total file size of the job workspace has little effect on the speed of the **signac** framework). To give a rough idea of the limits of scalability, **signac** projects can contain up to around 100,000 jobs while keeping common tasks like checking workflow status in an “interactive” time scale of 1-2 minutes. Some users that primarily wish to leverage **signac-flow**’s workflows for execution and submission may have a very small number of jobs (< 10). One example of this would be executing a small number of expensive biomolecular simulations using different random seeds in each job’s state point. Importantly, projects with a small number of jobs can be expanded at a later time, and make use of the same workflow defined for the initial set of jobs. The abilities to grow a project and change its schema on-the-fly catalyze the kind of exploration that is crucial to answering research questions.

The workflow submission features of **signac-flow** interoperates with popular HPC schedulers including SLURM, PBS/TORQUE, and LSF automating the generation and submission of scheduler batch scripts. Directives are set through Python decorators and define resource and execution requests for operations. Examples of directives include number of CPUs or GPUs, the walltime, and memory. The use of directives allows **signac-flow** workflows to be portable across HPC systems by generating resource requests that are specific to each machine’s scheduler.

B.5 Overview of New Features

The last three years of development of the **signac** framework have expanded its usability, feature set, user and developer documentation, and potential applications. Some of the largest architectural changes in the framework will be discussed in their own sections, namely extensions of the workflow model (support for executing groups of operations and aggregators that allow operations to act on multiple jobs) and a much more performant and flexible re-implementation of the core "data structure" classes that synchronize **signac**'s Python representation of state points and job documents with JSON-encoded dictionaries on disk.

B.5.1 Data Archival

The primary purpose of the core **signac** package is to simplify and accelerate data management. The **signac** command line interface is a common entry point for users, and provides subcommands for searching, reading, and modifying the data space. New commands for import and export simplify the process of archiving **signac** projects into a structure that is both human-readable and machine-readable for future access (with or without **signac**). Archival is an integral part of research data operations that is frequently overlooked. By using highly compatible and long-lived formats such as JSON for core data storage with simple name schemes, **signac** aims to preserve projects and make it easier for studies to be independently reproduced. This is aligned with the principles of TRUE (Transparent, Reproducible, Usable by others, and Extensible) simulations put forth by the MoSDeF collaboration [166].

B.5.2 Improved Data Storage, Retrieval, and Integrations

Data access via the shell: The `signac shell` command allows the user to quickly enter a Python interpreter that is pre-populated with variables for the current project or job (when in a project or job directory). This means that manipulating a job document or reading data can be done through a hybrid of bash/shell commands and Python commands that are fast to type. For example:

```
 ~/project $ ls
signac.rc workspace
~/project $ cd workspace/42b7b4f2921788ea14dac5566e6f06d0/
~/project/workspace/42b7b4f2921788ea14dac5566e6f06d0 $ signac shell
Python 3.8.3
signac 1.6.0

Project:        test
Job:            42b7b4f2921788ea14dac5566e6f06d0
Root:           ~/project
Workspace:      ~/project/workspace
Size:           1

Interact with the project interface using the
"project" or "pr" variable. Type "help(project)"
or "help(signac)" for more information.

>>> job.sp
{'a': 1}
```

HDF5 support for storing numerical data: Many applications used in research generate or consume large numerical arrays. For applications in Python, NumPy arrays are a de facto standard for in-memory representation and manipulation. However, saving these arrays to disk and handling data structures that mix dictionaries and numerical arrays can be cumbersome. The **signac** H5Store feature offers users a convenient wrapper around the h5py library [132] for loading and saving both hierarchical/key-value data and numerical array data in the widely-used HDF5 format [167]. The `job.data` attribute is an instance of the `H5Store` class, and is a key-value store saved on disk as `signac_data.h5` in the job workspace. Users who prefer to split data across multiple files can use the `job.stores` API to save in multiple HDF5 files. Corresponding `project.data` and `project.stores` attributes exist, which save data files in the project root directory. Using an instance of `H5Store` as a context manager allows users to keep the HDF5 file open while reading large chunks of the data:

```python
with job.data:
    # Copy array data from the file to memory
    # (which will persist after the HDF5 file is
    # closed) by indexing with an empty tuple:
    my_array = job.data["my_array"][()]
```

Advanced searching and filtering of the workspace: The `signac diff` command, available on both the command line and Python interfaces, returns the difference between two or more state points and allows for easily assessing subsets of the data space. By unifying state point and document queries, filtering, and searching workspaces can be more fine-grained and intuitive.

B.5.3 Data Visualization and Integrations

Integrating with the PyData ecosystem: Users can now summarize data from a **signac** project into a pandas DataFrame for analysis. The `project.to_dataframe()` feature exports state point and job document information to a pandas DataFrame in a consistent way that allows for quick analysis of all jobs' data. Support for Jupyter notebooks [122] has also been added, enabling rich HTML representations of **signac** objects.

Dashboards: The companion package **signac-dashboard** allows users to quickly visualize data stored in a **signac** data space. The dashboard runs in a browser and allows users to display job state points, edit job documents, render images and videos, download any file from a job workspace, and search or browse through state points in their project. Dashboards can be hosted on remote servers and accessed via port forwarding, which makes it possible to review data generated on a remote HPC system without needing to copy it back to a local system for inspection. Users can quickly save notes into the job document and then search those notes, which is useful for high throughput studies that require some manual investigation (e.g., reviewing plots).

B.5.4 Performance Enhancements

In early 2021, a significant portion of the codebase was profiled and refactored to improve performance and these improvements were released in **signac** 1.6.0 and **signac-flow** 0.12.0. As a result of these changes, large **signac** projects saw 4-7x speedups for operations such as iterating over the jobs in a project compared to the 1.5.0 release of **signac**. Similarly, performance of a sample workflow that checks status, runs, and submits a FlowProject with 1,000 jobs, 3 operations, and 2 label functions improved roughly 4x compared to **signac-flow** 0.11.0. These improvements allow **signac** to scale to 100,000 jobs.

In **signac**, the core of the `Project` and `Job` classes were refactored to support lazy attribute access and delayed initialization, which greatly reduces the total amount of disk I/O by waiting until data is actually requested by the user. Other improvements include early exits in functions, reducing the number of required system calls with smarter usage of the `os` library, and switching to algorithms that operate in constant time, $O(1)$, instead of linear time, $O(N_{jobs})$. Optimizations were identified by profiling the performance of common operations on small and large real-world projects with cProfile and visualized with snakeviz [168].

Similarly, performance enhancements were also made in the **signac-flow** package. Some of the optimizations identified include lazy evaluation of run commands and directives, and caching of job status information. In addition, the improvements in **signac** such as faster iteration over large **signac** projects used in **signac-flow** made **signac-flow**'s primary functions — checking project status, executing operations, and submitting operations to a cluster — significantly faster.

B.5.5 Improved User Output

Workflow graph detection: The preconditions and postconditions of operations in a **signac-flow** `FlowProject` implicitly define a graph. For example, if the operation "analyze" depends on the operation "simulate" via the precondition `@FlowProject.pre.after(simulate)`, then there is a directed edge from "simulate" to "analyze." This graph can now be detected from the workflow conditions and returned in a NetworkX [169] compatible format for display or inspection.

Templated status output: Querying the status of a **signac-flow** project now has many options controlling the information displayed and has been templated to allow for plain text, Markdown, or HTML output. In doing so, the output has also become cleaner and compatible with external tools.

B.5.6 Enhanced Workflows

Directives: Execution directives (or *directives* for short) provide a way to specify required resources on HPC schedulers such as number of CPUs/GPUs, MPI ranks, OpenMP threads, walltime, memory, and others. Directives can be a function of the job as well as the operation, allowing for great flexibility. In addition, directives work seamlessly with operation groups, job aggregation, and submission bundling (all of which are described in the following section).

Dynamic workspaces: The **signac-flow** package can now handle workspaces where jobs are created as the result of operations on other jobs. This is crucial for optimization workflows and iteratively sampling parameter spaces, and allows projects to become more automated with some data points only run if a prior condition on another data point is reached.

B.6 Executing Complex Workflows via Groups and Aggregation

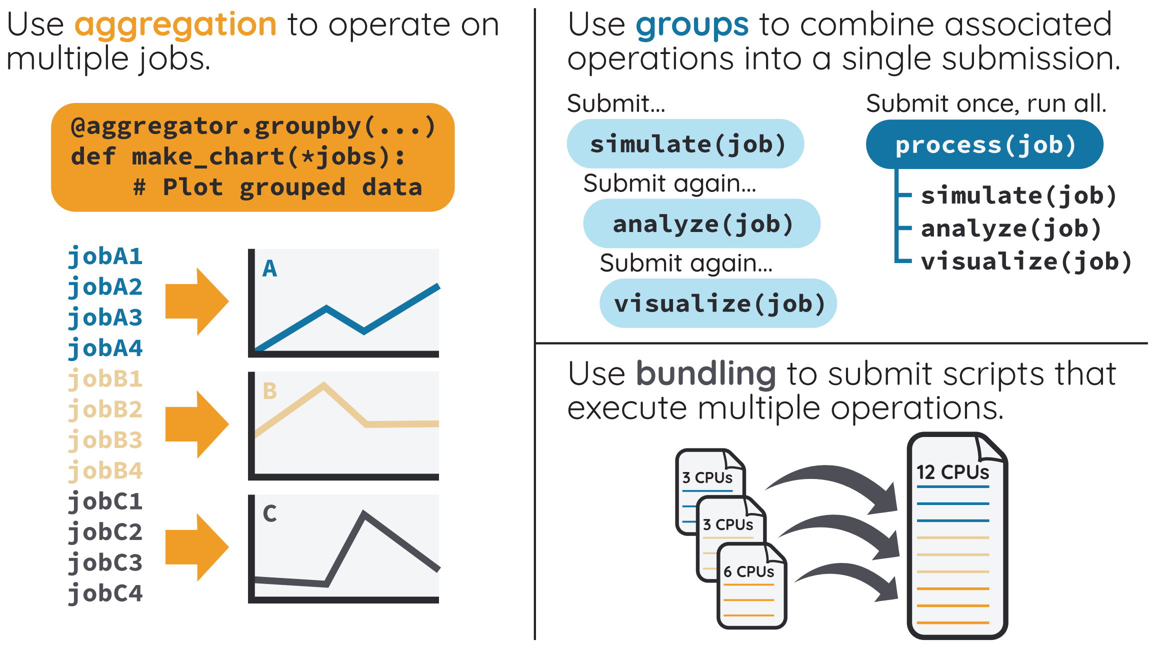


Figure B.2: Aggregation, groups, and bundling allow users to build complex workflows. The features are orthogonal, and can be used in any combination. Aggregation enables one operation or group to act on multiple jobs. Groups allow users to combine multiple operations into one, with dependencies among operations resolved at run time. Bundling helps users efficiently leverage HPC schedulers by submitting multiple commands in the same script, to be executed in serial or parallel.

Two new concepts in **signac-flow** provide users with significantly more power to implement complex workflows: *groups* and *aggregation*. A related third concept – *bundling* – which is not new, also provides flexibility to users in their workflows, but exclusively affects scheduler submission, not workflow definition. Figure B.2 shows a graphical illustration of the three concepts.

As the names of both groups and aggregation imply, the features enable the "grouping" or "aggregating" of existing concepts: operations in the case of groups and jobs in the case of aggregates. The conceptual model of **signac-flow** builds on **signac**'s notions of the `Project` and `Job` (the unit of the data space) through a `FlowProject` class that adds the ability to define and execute operations (the unit of a workflow) that act on jobs. Operations are Python functions or shell commands that act on a job within the data space, and are defined using Python decorator syntax. For example:

```
# project.py
from flow import FlowProject

@FlowProject.operation
@Flowproject.post.true("initialized")
def initialize(job):
    # perform necessary initialize steps
    # for simulation
    job.doc.initialized == True

if __name__ == "__main__":
    FlowProject().main()
```

When this project is run using **signac-flow**'s command line API (`python project.py run`), the current state point is prepared for simulation. Operations can have preconditions and postconditions that define their eligibility. All preconditions must be met in order for a operation to be eligible for a given job. If all postconditions are met, that indicates an operation is complete (and thus ineligible). Examples of such conditions include the existence of an input file in a job's workspace or a key in the job document (as shown in the above snippet). However, this type of conditional workflow can be inefficient when sequential workflows are coupled with an HPC scheduler interface, because the user must log on to the HPC and submit the next operation after the previous operation is complete. The desire to submit large and long-running jobs to HPC schedulers encourages users to write large operation functions which are not modular and do not accurately represent the individual units of the workflow, thereby limiting **signac-flow**'s utility and reducing the readability of the workflow.

B.6.1 Groups

Groups, implemented by the `FlowGroup` class and `FlowProject.make_group` method, allows users to combine multiple operations into a single entity that can be run or submitted. Submitting a group allows **signac-flow** to dynamically resolve preconditions and postconditions of operations as each operation is executed, making it possible to combine separate operations (e.g., for simulation and analysis and plotting) into a single submission script that will execute eligible operations in sequence. This allows users to write smaller, modular functions, which may require a specific order of execution, without sacrificing the ability to submit large, long-running jobs on HPCs. Furthermore, groups are aware of directives and can properly combine the directives of their constituent operations to specify resources and quantities like walltime whether executing in parallel or serial. For example:

```python
from flow import FlowProject

example_group = FlowProject.make_group(
    name="example_group")

@example_group.with_directives(
    {"ngpu": 2,
     "walltime": lambda job: job.doc.hours_to_run})
@FlowProject.post.true("simulated")
@FlowProject.operation
def simulate(job):
    # run simulation
    job.doc.simulated = True

@example_group
@FlowProject.pre.after(simulate)
@FlowProject.post.true("analyzed")
@FlowProject.operation
def analyze(job):
    # analyze simulation results
    job.doc.analyzed = True
```

Groups also allow for specifying multiple machine specific resources (CPU or GPU) with the same operation. An operation can have unique directives for each distinct group to which it belongs. By associating an operation's directives with respect to a specific group, groups can represent distinct compute environments, such as a local workstation or a remote supercomputing cluster. The below snippet shows an `expensive_simulate` operation which can be executed with three different directives depending on how it is written. If executed through `cpu_group` the operation will request 48 cores, if `gpu_group` 4 GPUs, if neither then it will request 4 cores. This represents the real use case where a user may want to run an operation locally (in this case without a group), or on a CPU or GPU focused HPC/workstation. For example:

```
from flow import FlowProject

cpu_group = FlowProject.make_group(name="cpu")
gpu_group = FlowProject.make_group(name="gpu")

@cpu_group.with_directives({"np": 48})
@gpu_group.with_directives({"ngpu": 4})
@FlowProject.operation.with_directives({"np": 4})
def expensive_simulate(job):
    # expensive simulation run on CPUs or GPUs
    pass
```

B.6.2 Aggregation

Users also frequently work with multiple jobs when performing tasks such as plotting data from all jobs in the same figure. Though the **signac** package has methods like `Project.groupby`, which can generate subsets of the project that are grouped by a state point key, there has been no way to use these "aggregation" features in **signac-flow** for defining workflows. The concept of aggregation provides a straightforward way for users to write and submit operations that act on arbitrary subsets of jobs in a **signac** data space through functions analogous to `Project.groupby`. Just as the groups feature acts as an abstraction over operations, aggregation can be viewed as an abstraction over jobs. When decorated with an aggregator, operations can accept multiple job instances as positional arguments through Python's argument unpacking. Decorators are used to define aggregates, encompassed in the `@aggregator` decorator for single operations and in the argument `aggregator_function` to `FlowProject.make_group` for groups of operations. For example:

```python
from flow import FlowProject

@aggregator
@FlowProject.operation
def plot_enzyme_activity(*jobs):
    import matplotlib.pyplot as plt
    import numpy as np

    x = [job.sp.temperature for job in jobs]
    y = [job.doc.activity for job in jobs]
    fig, ax = plt.subplots()
    ax.scatter(x, y)
    ax.set_title(
        "Enzymatic Activity Across Temperature")
    fig.savefig("enzyme-activity.png")
```

Like groups, there are many reasons why a user might wish to use aggregation. For example, a **signac** data space that describes weather data for multiple cities in multiple years might want to plot or analyze data that uses `@aggregator.groupby("city")` to show changes over time for each city in the data space. Similarly, aggregating over replicas (e.g., the same simulation with different random seeds) facilitates computing averaged quantities and error bars. Another example is submitting aggregates with a fixed number of jobs in each aggregate to enable massive parallelization by breaking a large MPI communicator into a smaller communicator for each independent job, which is necessary for efficient utilization of leadership-class supercomputers like OLCF Summit.

B.6.3 Bundling

Finally, bundling is another way to use workflows in conjunction with an HPC scheduling system. Whereas aggregates are concerned with jobs and groups operations, bundling is concerned with combining executable units into a single submission script. This distinction means that bundling is not part of the workflow definition, but is a means of tailoring batch scripts for different HPC systems. Bundles allow users to leverage scheduler resources effectively and minimize queue time, and can be run in serial (the default) or parallel. Users enable bundling by passing the command line argument `--bundle`, optionally with another argument `--parallel` to run each command in the bundle in parallel (the Python API has corresponding options as well). The simplest case of a bundle is a submission script with the same operation being executed for multiple jobs. Bundling is what allows the submission script to contain multiple jobs executing the same operation. By storing information about the generated bundles during submission, **signac-flow** prevents accidental resubmission just as in the unbundled case. While the example mentioned above does not use either groups or aggregation, bundles works seamlessly with both.

B.6.4 Cluster Templates

The **signac-flow** software includes automatic detection and script support for SLURM, PBS/TORQUE, and LSF schedulers. However, effective HPC utilization frequently relies on specific information such as numbers of cores per compute node or designated partitions for GPU or large memory applications. To this end, **signac-flow** includes templates for a number of HPC clusters including OLCF Summit and Andes, XSEDE [170] clusters such as PSC Bridges-2, SDSC Comet, and TACC Stampede2, and university clusters such as the University of Michigan's Great Lakes and University of Minnesota's Mangi. These cluster templates change frequently as HPC systems are brought online and later decommissioned. Users can create their own templates to contribute to the package or use locally.

B.7 Synced Collections: Backend-Agnostic, Persistent, Mutable Data Structures

B.7.1 Motivation

At its core, **signac** is a tool for organizing and working with data on the filesystem, presenting a Pythonic interface for tasks like creating directories and modifying files. In particular, **signac** makes modifying the JSON files used to store a job's state points and documents as easy as working with Python dictionaries. Despite heavy optimization, when seeking to scale **signac** to ever-larger data spaces, we quickly realized that the most significant performance barrier was the overhead of parsing and modifying large numbers of text files. Unfortunately, the usage of JSON files in this manner was deeply embedded in our data model, which made switching to a more performant backend without breaking APIs or severely complicating our data model a daunting task.

While attempting to separate the **signac** data model from its original backend implementation (manipulating JSON files on disk), we identified a common pattern: providing a dictionary-like interface for an underlying resource. Several well-known Python packages such as h5py [132] and zarr [171] also use dictionary-like interfaces to make working with complex resources feel natural to Python users. Most such packages implement this layer directly for their particular use case, but the nature of the problem suggested to us the possibility of developing a more generic representation of this interface. Indeed, the purpose of the Python standard library's `collections.abc` module to make it easy to define objects that "look like" standard Python objects while having completely customizable behavior under the hood. As such, we saw an opportunity to specialize this pattern for a specific use case: the transparent synchronization of a Python object with an underlying resource.

The *synced collections* framework represents the culmination of our efforts in this direction, providing a generic framework in which interfaces of any abstract data type can be mapped to arbitrary underlying synchronization protocols. In **signac**, this framework allows us to hide the details of a particular file storage medium (like JSON) behind a dictionary-like interface, but it can just as easily be used for tasks such as creating a new, list-like interface that automatically saves all its data in a plain-text CSV format. This section will offer a high-level overview of the synced collections framework and our plans for its use within **signac**, with an eye to potential users in other domains as well.

B.7.2 Summary of Features

We designed synced collections to be flexible, easily extensible, and independent of **signac**'s data model. Most practical use cases for this framework involve an underlying resource that may be modified by any number of associated in-memory objects that behave like standard Python collections, such as dictionaries or lists. Therefore, all normal operations must be preceded by loading from this resource and updating the in-memory store, and they must be succeeded by saving to that resource. The central idea behind synced collections is to decouple this process into two distinct groups of tasks: the saving and loading of data from a particular resource backend, and the synchronization of two in-memory objects of a given type. This delineation allows us to, for instance, encapsulate all logic for JSON files into a single `JSONCollection` class and then combine it with dictionary- or list-like `SyncedDict` / `SyncedList` classes via inheritance to create fully functional JSON-backed dictionaries or lists. Such synchronization significantly lowers performance, so the framework also exposes an API to implement buffering protocols to collect operations into a single transaction before submitting them to the underlying resource.

Previously, **signac** contained a single `JSONDict` class as part of its API, along with a separately implemented internal-facing `JSONList` that could only be used as a member of a `JSONDict`. With the new framework, users can create fully-functional, arbitrarily nested `JSONDict` and `JSONList` objects that share the same logic for reading from and writing to JSON files. Just as importantly, **signac** can now combine these data structures with a different backend, allowing us to swap in different storage mechanisms for improved performance and flexibility with no change in our APIs. Since different types of resources may have different approaches to batching transactions — for example, a SQLite backend may want to exploit true SQL transactions, while a Redis backend might simply collect all changes in memory and delay sending memory to the server — synced collections also support customizable buffering protocols, again via class inheritance.

B.7.3 Applications of Synced Collections

The new synced collections promise to substantially simplify both feature and performance enhancements to the **signac** framework. Performance improvements in the form of Redis-based storage are already possible with synced collections, and as expected they show substantial speedups over the current JSON-based approach. We have also exploited the new and more flexible buffering protocol to implement and test alternatives to the previous approach. In certain cases, our new buffering techniques improve performance of buffered operations by 1-2 orders of magnitude. Some of these performance improvements are drop-in replacements that require no changes to our existing data models, and we plan to enable these in upcoming versions of **signac**.

The generality of synced collections makes them broadly useful even outside the **signac** framework. Adding Pythonic APIs to collection-like objects can be challenging, particularly when those objects should support arbitrary nesting, but synced collections enable nesting as a core feature to dramatically simplify this process. Moreover, while the framework was originally conceived to support synchronization of an in-memory data structure with a resource on disk, it can also be used to synchronize with another in-memory resource. A powerful example of this would be wrapping a C or C++ extension type, for instance by creating a `SyncedList` that synchronizes with a C++ `std::vector`, such that changes to either object would be transparently reflected in the other. With synced collections, creating this class just requires defining a conversion between a `std::vector` and a raw Python list, a trivial task using standard tools for exposing extension types such as pybind or Cython.

At a higher level, synced collections represent an important step in improving both the scalability and flexibility of **signac**. By abstracting away details of persistent file storage from the rest of **signac**, they make it much easier for the rest of **signac** to focus on offering flexible data models. One of the most common use cases of **signac** is creating data spaces with homogeneous schemas that fit naturally into tabular data structures. In future iterations of **signac**, we plan to allow users to opt into homogeneous schemas, which would enable us to replace file-based indexes with SQL-backed databases that would offer orders of magnitude in performance improvements. Using this flexibility, we could also move away from our currently rigid workspace model to allow more general data layouts on disk for cases where users may benefit from more general folder structures. As such, synced collections are a stepping stone to creating a more general and powerful version of **signac**.

B.8 Project Evolution

The **signac** project has evolved from being an open-source project mostly developed and managed by the Glotzer Group at the University of Michigan, to being supported by over 30 contributors and 8 committers/maintainers on 3 continents and with over 55 citations from academic and government research labs and 12 talks at large scientific, Python, and data science conferences. The growth in involvement with **signac** results from our focus on developing features based on user needs, as well as our efforts to transition **signac** users into **signac** contributors, through many initiatives in the past few years. Through encouraging users to become contributors, we ensure that **signac** addresses real users' needs. Early on, we identified that the framework had the potential to be used by a wide community of researchers and that its philosophy was aligned with other projects in the scientific Python ecosystem. We have expanded **signac**'s contributor base beyond the University of Michigan through research collaborations such as the MoSDeF CSSI with other universities, sharing the framework at conferences, and through the Google Summer of Code (GSoC) program, which we applied to under the NumFOCUS organization. Working with and mentoring students through GSoC led to a new committer and significant work on the synced collections and aggregation projects presented above. We provide active support and open discussion for the contributor and user community through Slack. In addition, we have started hosting weekly "office hours" for in-person (virtual) introduction and guided contributions to the code base. By pairing new contributors with experienced **signac** developers, we significantly reduce the knowledge barrier to joining a new project. Close interactions between developers and users during office hours has led to more features and documentation born directly out of user need. Contributing to documentation has been a productive starting point for new users-turned-contributors, both for the users and the project, since it improves the users' familiarity with the API as well as addresses weak spots in the documentation that are more obvious to new users.

In our growth with increasing contributors and users, we recognized a need to change our governance structure to make contributing easier and provide a clear organizational structure to the community. We based our new model on the Meritocratic Governance Model and our manager roles on Numba [172] Czars. We decided on a four category system with maintainers, committers, contributors, and users. Code review and pull request merge responsibilities are granted to maintainers and committers, who are (self-) nominated and accepted by a vote of the project maintainers. Maintainers are additionally responsible for the strategic direction of the project and administrative duties. Contributors consist of all members of the community who have contributed in some way to the framework, which includes adding or refactoring code as well as filing issues and improving documentation. Finally, users refer to all those who use **signac** in any capacity.

In addition, to avoid overloading our committers and maintainers, we added three rotating manager roles to our governance model that ensure project management goes smoothly: triage, community, and release. These managers have specific rotation policies based on time (or release cycles). The triage manager role rotates weekly and looks at new issues or pull requests and handles cleanup of outdated issues. The community manager role rotates monthly and is in charge of meeting planning and outreach. Lastly, the release manager rotates with each release cycle and is the primary decision maker for the timeline and feature scope of package releases. This prevents burnout among our senior developers and provides a sense of ownership to a greater number of people, instead of relying on a “benevolent dictator/oligarchy for life” mode of project leadership.

B.9 Conclusions

From the birth of the **signac** framework in 2015 to now, **signac** has grown in usability, performance, and use. In the last three years, we have added exciting new features such as groups, aggregation, and synced collections, while learning how to manage outreach and establish sustainable project governance in a burgeoning scientific open-source project. We hope to continue expanding the framework through user-oriented development, reach users in research fields beyond materials science that routinely have projects suited for **signac**, and welcome new contributors with diverse backgrounds and skills to the project.

B.10 Installing signac

The **signac** framework is tested for Python 3.6+ and is compatible with Linux, macOS, and Windows. The software is available under the BSD-3 Clause license. To install, execute:

```
conda install -c conda-forge signac \
signac-flow signac-dashboard
```

or:

```
pip install signac signac-flow signac-dashboard
```

Source code is available on GitHub[4] and documentation is hosted online by ReadTheDocs.[5]

[4] https://github.com/glotzerlab/signac, https://github.com/glotzerlab/signac-flow

[5] https://docs.signac.io

B.11 Acknowledgments

We would also like to thank NumFOCUS for providing helpful advice on open-source governance, project sustainability, and community outreach, as well as funding for the design of the **signac** project logo.

This work was supported by the National Science Foundation, Office of Advanced Cyberinfrastructure Awards OAC 1835612 and OAC 1835593. B.D. and B.B. acknowledge fellowship support from the National Science Foundation under ACI 1547580, S212: Impl: The Molecular Sciences Software Institute [1, 2]. B.D. was also supported by a National Science Foundation Graduate Research Fellowship Grant DGE 1256260 (2016–2019). V.R. acknowledges the 2019-2020 J. Robert Beyster Computational Innovation Graduate Fellowship at the University of Michigan. A.T. is supported by the National Science Foundation under DMR 1707640. Software was deployed and validated and benchmarked on the Extreme Science and Engineering Discovery Environment (XSEDE) [170], which is supported by National Science Foundation Grant No. ACI-1053575 (XSEDE award DMR 140129) and on resources of the Oak Ridge Leadership Computing Facility which is a DOE Office of Science User Facility supported under Contract No. DE-AC05-00OR22725.

B.12 Author Contributions

Conceptualization, B.D.D., B.L.B., V.R., A.T., M.M.H., H.O., and C.S.A.; data curation, B.D.D., B.L.B., V.R., A.T., M.M.H., H.O., and C.S.A.; funding acquisition, E.J. and S.C.G.; methodology, B.D.D., B.L.B., V.R., A.T., M.M.H., H.O., and C.S.A.; project administration, B.D.D., B.L.B., V.R., A.T., M.M.H., H.O., and C.S.A.; software, B.D.D., B.L.B., V.R., A.T., M.M.H., H.O., and C.S.A.; supervision, S.C.G.; visualization, B.D.D., B.L.B., A.T., and K.W.; writing – original draft,

B.D.D., B.L.B., V.R., A.T., and H.O.; writing – review & editing, B.D.D., B.L.B., V.R., A.T., M.M.H., H.O., K.W., C.S.A., and S.C.G. All authors have read and agreed to the published version of the manuscript.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] N. Wilkins-Diehr and T. D. Crawford, "NSF's Inaugural Software Institutes: The Science Gateways Community Institute and the Molecular Sciences Software Institute," *Computing in Science Engineering*, vol. 20, no. 5, pp. 26–38, 2018.

[2] A. Krylov, T. L. Windus, T. Barnes, E. Marin-Rimoldi, J. A. Nash, B. Pritchard, D. G. A. Smith, D. Altarawy, P. Saxe, C. Clementi, T. D. Crawford, R. J. Harrison, S. Jha, V. S. Pande, and T. Head-Gordon, "Perspective: Computational Chemistry Software and Its Advancement As Illustrated Through Three Grand Challenge Cases for Molecular Science," *The Journal of Chemical Physics*, vol. 149, no. 18, p. 180901, 2018.

[3] M. Spellings and plato-draw contributors, "https://github.com/glotzerlab/plato/," May 2021.

[4] J. Grout and PyThreejs Development Team, "https://github.com/jupyter-widgets/pythreejs," October 2020.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] W. Mickel, S. C. Kapfer, G. E. Schröder-Turk, and K. Mecke, "Shortcomings of the Bond Orientational Order Parameters for the Analysis of Disordered Particulate Matter," *Journal of Chemical Physics*, vol. 138, no. 4, 2013.

[7] Plotly Technologies Inc., "Collaborative Data Science," 2015.

[8] P. M. Larsen, S. Schmidt, and J. Schiøtz, "Robust Structural Identification via Polyhedral Template Matching," *Modelling and Simulation in Materials Science and Engineering*, vol. 24, p. 055007, June 2016.

[9] A. Stukowski, "Visualization and Analysis of Atomistic Simulation Data with OVITO - The Open Visualization Tool," *Modelling and Simulation in Materials Science and Engineering*, vol. 18, p. 015012, January 2010.

[10] M. J. Mehl, D. Hicks, C. Toher, O. Levy, R. M. Hanson, G. Hart, and S. Curtarolo, "The AFLOW Library of Crystallographic Prototypes: Part 1," *Computational Materials Science*, vol. 136, pp. S1–S828, August 2017.

[11] C. X. Du, G. van Anders, R. S. Newman, and S. C. Glotzer, "Shape-Driven Solid-Solid Transitions in Colloids," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 114, pp. E3892–E3899, May 2017.

[12] G. E. Schröder-Turk, W. Mickel, S. C. Kapfer, F. M. Schaller, B. Breidenbach, D. Hug, and K. Mecke, "Minkowski Tensors of Anisotropic Spatial Structure," *New Journal of Physics*, vol. 15, p. 083028, August 2013.

[13] R. K. Cersonsky, J. Antonaglia, B. D. Dice, and S. C. Glotzer, "The Diversity of Three-Dimensional Photonic Crystals," *Nature Communications*, vol. 12, p. 2543, May 2021.

[14] J. Anderson and fresnel contributors, "https://github.com/glotzerlab/fresnel/," June 2021.

[15] C. S. Adorf, J. Antonaglia, J. Dshemuchadse, and S. C. Glotzer, "Inverse Design of Simple Pair Potentials for the Self-Assembly of Complex Structures," *The Journal of Chemical Physics*, vol. 149, p. 204102, November 2018.

[16] R. Rusali and G. J. Wang, "High-Throughput Analysis of Urban Textures Using Methods from Molecular Simulation," in *Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, (New York, NY, USA), pp. 298–301, Acm, November 2020.

[17] E. Fermi, J. Pasta, S. Ulam, and M. Tsingou, "Studies of Nonlinear Problems," tech. rep., Los Alamos Scientific Laboratory of the University of California, 1955.

[18] J. Jung, W. Nishima, M. Daniels, G. Bascom, C. Kobayashi, A. Adedoyin, M. Wall, A. Lappala, D. Phillips, W. Fischer, C.-S. Tung, T. Schlick, Y. Sugita, and K. Y. Sanbonmatsu, "Scaling Molecular Dynamics Beyond 100,000 Processor Cores for Large-Scale Biophysical Simulations," *Journal of Computational Chemistry*, vol. 40, pp. 1919–1930, August 2019.

[19] Z. M. Sherman, M. P. Howard, B. A. Lindquist, R. B. Jadrich, and T. M. Truskett, "Inverse Methods for Design of Soft Materials," *The Journal of Chemical Physics*, vol. 152, p. 140902, April 2020.

[20] Y. Geng, G. van Anders, P. M. Dodd, J. Dshemuchadse, and S. C. Glotzer, "Engineering Entropy for the Inverse Design of Colloidal Crystals From Hard Shapes," *Science Advances*, vol. 5, p. eaaw0514, July 2019.

[21] G. C. Sosso, J. Chen, S. J. Cox, M. Fitzner, P. Pedevilla, A. Zen, and A. Michaelides, "Crystal Nucleation in Liquids: Open Questions and Future Challenges in Molecular Dynamics Simulations," *Chemical Reviews*, vol. 116, pp. 7078–7116, June 2016.

[22] L. McInnes and J. Healy, "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction," February 2018.

[23] P. F. Damasceno, M. Engel, and S. C. Glotzer, "Predictive Self-Assembly of Polyhedra Into Complex Structures," *Science (New York, N.Y.)*, vol. 337, pp. 453–7, July 2012.

[24] G. van Anders, N. K. Ahmed, R. Smith, M. Engel, and S. C. Glotzer, "Entropically Patchy Particles: Engineering Valence Through Shape Entropy," *ACS Nano*, vol. 8, no. 1, pp. 931–940, 2014.

[25] C. S. Adorf, P. M. Dodd, V. Ramasubramani, and S. C. Glotzer, "Simple Data and Workflow Management with the signac Framework," *Computational Materials Science*, vol. 146, pp. 220–229, April 2018.

[26] V. Ramasubramani, C. S. Adorf, P. M. Dodd, B. D. Dice, and S. C. Glotzer, "signac: A Python Framework for Data And Workflow Management," in *Proceedings of the 17th Python in Science Conference* (F. Akici, D. Lippa, D. Niederhut, and M. Pacer, eds.), pp. 152–159, 2018.

[27] V. Ramasubramani, B. D. Dice, E. S. Harper, M. P. Spellings, J. A. Anderson, and S. C. Glotzer, "freud: A Software Suite for High Throughput Analysis of Particle Simulation Data," *Computer Physics Communications*, vol. 254, p. 107275, September 2020.

[28] J. A. Anderson, J. Glaser, and S. C. Glotzer, "HOOMD-blue: A Python Package for High-Performance Molecular Dynamics and Hard Particle Monte Carlo Simulations," *Computational Materials Science*, vol. 173, p. 109363, February 2020.

[29] M. Spellings and S. C. Glotzer, "Machine Learning for Crystal Identification and Discovery," *AIChE Journal*, vol. 64, pp. 2198–2206, June 2018.

[30] C. S. Adorf, T. C. Moore, Y. J. U. Melle, and S. C. Glotzer, "Analysis of Self-Assembly Pathways with Unsupervised Machine Learning Algorithms," *The Journal of Physical Chemistry B*, vol. 124, pp. 69–78, January 2020.

[31] E. Boattini, M. Dijkstra, and L. Filion, "Unsupervised Learning for Local Structure Detection in Colloidal Systems," *The Journal of Chemical Physics*, vol. 151, p. 154901, October 2019.

[32] W. F. Reinhart, A. W. Long, M. P. Howard, A. L. Ferguson, and A. Z. Panagiotopoulos, "Machine Learning for Autonomous Crystal Structure Identification," *Soft Matter*, vol. 13, no. 27, pp. 4733–4745, 2017.

[33] P. J. Steinhardt, D. R. Nelson, and M. Ronchetti, "Bond-Orientation Order in Liquids and Glasses," *Physical Review B*, vol. 28, no. 2, pp. 784–805, 1983.

[34] A. S. Keys, C. R. Iacovella, and S. C. Glotzer, "Characterizing Structure Through Shape Matching and Applications to Self-Assembly," *Annual Review of Condensed Matter Physics*, vol. 2, pp. 263–285, March 2011.

[35] J. D. Honeycutt and H. C. Andersen, "Molecular Dynamics Study of Melting and Freezing of Small Lennard-Jones Clusters," *The Journal of Physical Chemistry*, vol. 91, pp. 4950–4963, September 1987.

[36] F. A. Lindemann, "Über die berechnung molekularer eigenfrequenzen," *Phys. Z*, vol. 11, pp. 609–612, 1910.

[37] B. Peters, "Reaction Coordinates and Mechanisms," in *Reaction Rate Theory and Rare Events Simulations*, pp. 539–571, Elsevier, January 2017.

[38] C. L. Kelchner, S. J. Plimpton, and J. C. Hamilton, "Dislocation Nucleation and Defect Structure During Surface Indentation," *Physical Review B*, vol. 58, pp. 11085–11088, November 1998.

[39] G. J. Ackland and A. P. Jones, "Applications of Local Crystal Structure Measures in Experiment and Simulation," *Physical Review B*, vol. 73, p. 054104, February 2006.

[40] A. Stukowski, "Structure Identification Methods for Atomistic Simulations of Crystalline Materials," *Modelling and Simulation in Materials Science and Engineering*, vol. 20, p. 045021, June 2012.

[41] A. Malins, S. R. Williams, J. Eggers, and C. P. Royall, "Identification of Structure in Condensed Matter with the Topological Cluster Classification," *The Journal of Chemical Physics*, vol. 139, p. 234506, December 2013.

[42] C. Dietz and M. H. Thoma, "Investigation and Improvement of Three-Dimensional Plasma Crystal Analysis," *Physical Review E*, vol. 94, p. 033207, September 2016.

[43] C. Dietz, T. Kretz, and M. H. Thoma, "Machine-Learning Approach for Local Classification of Crystalline Structures in Multiphase Systems," vol. 011301, pp. 1–5, 2017.

[44] E. G. Teich, G. van Anders, and S. C. Glotzer, "Identity Crisis in Alchemical Space Drives the Entropic Colloidal Glass Transition," *Nature Communications*, vol. 10, p. 64, December 2019.

[45] W. Lechner and C. Dellago, "Accurate Determination of Crystal Structures Based on Averaged Local Bond Order Parameters," *Journal of Chemical Physics*, vol. 129, no. 11, 2008.

[46] R. van Damme, G. M. Coli, R. van Roij, and M. Dijkstra, "Classifying Crystals of Rounded Tetrahedra and Determining Their Order Parameters Using Dimensionality Reduction," *ACS Nano*, vol. 14, pp. 15144–15153, November 2020.

[47] M. Li, Y. Chen, H. Tanaka, and P. Tan, "Revealing Roles of Competing Local Structural Orderings in Crystallization of Polymorphic Systems," *Science Advances*, vol. 6, p. eaaw8938, July 2020.

[48] C. Rycroft, "Voro++: A Three-Dimensional Voronoi Cell Library in C++," tech. rep., Lawrence Berkeley National Laboratory (LBNL), Berkeley, CA, January 2009.

[49] A. Ng, "Machine Learning and AI via Brain Simulations," March 2013.

[50] R. S. DeFever, C. Targonski, S. W. Hall, M. C. Smith, and S. Sarupria, "A Generalized Deep Learning Approach for Local Structure Identification in Molecular Simulations," *Chemical Science*, vol. 10, pp. 7503–7515, August 2019.

[51] T. Xie and J. C. Grossman, "Crystal Graph Convolutional Neural Networks for Accurate and Interpretable Prediction of Material Properties," *Physical Review Letters*, vol. 120, no. 14, p. 145301, 2018.

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, vol. 60, pp. 84–90, June 2017.

[53] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation," in *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 1724–1734, Association for Computational Linguistics (ACL), 2014.

[54] G. M. Coli and M. Dijkstra, "An Artificial Neural Network Reveals the Nucleation Mechanism of a Binary Colloidal AB_{13} crystal," *ACS Nano*, vol. 15, pp. 4335–4346, March 2021.

[55] E. Boattini, M. Ram, F. Smallenburg, and L. Filion, "Neural-Network-Based Order Parameters for Classification of Binary Hard-Sphere Crystal Structures," *Molecular Physics*, vol. 116, pp. 3066–3075, November 2018.

[56] J. A. van Meel, L. Filion, C. Valeriani, and D. Frenkel, "A Parameter-Free, Solid-Angle Based, Nearest-Neighbor Algorithm," *Journal of Chemical Physics*, vol. 136, p. 114707, June 2012.

[57] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," December 2013.

[58] H. Tanaka, "Bond Orientational Order in Liquids: Towards a Unified Description of Water-Like Anomalies, Liquid-Liquid Transition, Glass Transition, and Crystallization," *The European Physical Journal E*, vol. 35, p. 113, October 2012.

[59] J. Russo and H. Tanaka, "The Microscopic Pathway to Crystallization in Supercooled Liquids," *Scientific Reports*, vol. 2, p. 505, December 2012.

[60] S. Arai and H. Tanaka, "Surface-Assisted Single-Crystal Formation of Charged Colloids," *Nature Physics*, vol. 13, pp. 503–509, May 2017.

[61] K. Pearson, "On Lines and Planes of Closest Fit to Systems of Points in Space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, pp. 559–572, November 1901.

[62] L. van der Maaten and G. Hinton, "Visualizing Data Using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.

[63] D. W. H. Swenson, J.-H. Prinz, F. Noe, J. D. Chodera, and P. G. Bolhuis, "OpenPathSampling: A Python Framework for Path Sampling Simulations. 1. Basics," *Journal of Chemical Theory and Computation*, vol. 15, pp. 813–836, February 2019.

[64] D. W. H. Swenson, J.-H. Prinz, F. Noe, J. D. Chodera, and P. G. Bolhuis, "OpenPathSampling: A Python Framework for Path Sampling Simulations. 2. Building and Customizing Path Ensembles and Sample Schemes," *Journal of Chemical Theory and Computation*, vol. 15, pp. 837–856, February 2019.

[65] H. Sidky, Y. J. Colón, J. Helfferich, B. J. Sikora, C. Bezik, W. Chu, F. Giberti, A. Z. Guo, X. Jiang, J. Lequieu, J. Li, J. Moller, M. J. Quevillon, M. Rahimi, H. Ramezani-Dakhel, V. S. Rathee, D. R. Reid, E. Sevgen, V. Thapar, M. A. Webb, J. K. Whitmer, and J. J. de Pablo, "SSAGES: Software Suite for Advanced General Ensemble Simulations," *The Journal of Chemical Physics*, vol. 148, p. 044104, January 2018.

[66] E. Becht, L. McInnes, J. Healy, C.-A. Dutertre, I. W. H. Kwok, L. G. Ng, F. Ginhoux, and E. W. Newell, "Dimensionality Reduction for Visualizing Single-Cell Data Using UMAP," *Nature Biotechnology*, vol. 37, pp. 38–44, January 2019.

[67] C. J. Nolet, V. Lafargue, E. Raff, T. Nanditale, T. Oates, J. Zedlewski, and J. Patterson, "Bringing UMAP Closer to the Speed of Light with GPU Acceleration," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 418–426, May 2021.

[68] RAPIDS Development Team, *RAPIDS: Collection of Libraries for End to End GPU Data Science*, 2018.

[69] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[70] J. A. Anderson, M. Eric Irrgang, and S. C. Glotzer, "Scalable Metropolis Monte Carlo for Simulation of Hard Shapes," *Computer Physics Communications*, vol. 204, pp. 21–30, July 2016.

[71] V. Ramasubramani, B. D. Dice, T. T. Dwyer, and S. C. Glotzer, "coxeter: A Python Package for Working with Shapes," *Journal of Open Source Software*, vol. 6, no. 63, p. 3098, 2021.

[72] J. MacQueen *et al.*, "Some Methods for Classification and Analysis of Multivariate Observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.

[73] A. K. Sharma and F. A. Escobedo, "Disorder Foreshadows Order in Colloidal Cubes," *The Journal of Physical Chemistry B*, vol. 122, pp. 9264–9273, October 2018.

[74] H. L. Yakel, "Atom Distributions in Sigma Phases. I. Fe and Cr Atom Distributions in a Binary Sigma Phase Equilibrated at 1063, 1013 and 923 K," *Acta Crystallographica Section B Structural Science*, vol. 39, pp. 20–28, February 1983.

[75] P. R. ten Wolde, M. J. Ruiz-Montero, and D. Frenkel, "Numerical Evidence for bcc Ordering at the Surface of a Critical fcc Nucleus," *Physical Review Letters*, vol. 75, pp. 2714–2717, October 1995.

[76] 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*, vol. 133, p. 244115, December 2010.

[77] I. Han, K. L. Wang, A. T. Cadotte, Z. Xi, H. Parsamehr, X. Xiao, S. C. Glotzer, and A. J. Shahani, “Self-Healing Behavior of Quasicrystals Upon Hard Collision,” June 2021.

[78] A. Karas, *Understanding and Controlling Directional Entropic Forces in Hard Particle Self-Assembly.* PhD thesis, University of Michigan, 2018.

[79] D. A. Porter and K. E. Easterling, *Phase Transformations in Metals and Alloys (Revised Reprint).* CRC Press, 2009.

[80] P. Chao, X. Xiao, and A. J. Shahani, “Flexible Unsupervised Binary Change Detection Algorithm Identifies Phase Transitions in Continuous Image Streams,” *Integrating Materials and Manufacturing Innovation*, vol. 10, pp. 72–81, March 2021.

[81] M. Engel, “Point Group Analysis in Particle Simulation Data,” June 2021.

[82] J. A. Anderson, J. Antonaglia, J. A. Millan, M. Engel, and S. C. Glotzer, “Shape and Symmetry Determine Two-Dimensional Melting Transitions of Hard Regular Polygons,” *Physical Review X*, vol. 7, p. 021001, April 2017.

[83] W. Shen, J. Antonaglia, J. A. Anderson, M. Engel, G. van Anders, and S. C. Glotzer, “Symmetries in Hard Polygon Systems Determine Plastic Colloidal Crystal Mesophases in Two Dimensions,” *Soft Matter*, vol. 15, pp. 2571–2579, March 2019.

[84] T. C. Moore, J. A. Anderson, and S. C. Glotzer, “Shape-Driven Entropic Self-Assembly of an Open, Reconfigurable, Binary Host-Guest Colloidal Crystal,” *Soft Matter*, vol. 17, pp. 2840–2848, March 2021.

[85] F. M. Schaller, S. C. Kapfer, M. E. Evans, M. J. Hoffmann, T. Aste, M. Saadatfar, K. Mecke, G. W. Delaney, and G. E. Schröder-Turk, “Set Voronoi Diagrams of 3D Assemblies of Aspherical Particles,” *Philosophical Magazine*, vol. 93, pp. 3993–4017, November 2013.

[86] S. Weis, P. W. A. Schönhöfer, F. M. Schaller, M. Schröter, and G. E. Schröder-Turk, “Pomelo, a Tool for Computing Generic Set Voronoi Diagrams of Aspherical Particles of Arbitrary Shape,” *EPJ Web of Conferences*, vol. 140, p. 06007, June 2017.

[87] S. Kapfer, *Morphometry and Physics of Particulate and Porous Media.* PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2012.

[88] W. Mickel, *Geometry Controlled Phase Behavior in Nanowetting and Jamming.* PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2011.

[89] S. C. Kapfer, W. Mickel, F. M. Schaller, M. Spanner, C. Goll, T. Nogawa, N. Ito, K. Mecke, and G. E. Schröder-Turk, “Local Anisotropy of Fluids Using Minkowski Tensors,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2010, p. P11010, November 2010.

[90] G. E. Schröder-Turk, W. Mickel, M. Schröter, G. W. Delaney, M. Saadatfar, T. J. Senden, K. Mecke, and T. Aste, “Disordered Spherical Bead Packs Are Anisotropic,” *EPL (Europhysics Letters)*, vol. 90, p. 34001, May 2010.

[91] S. C. Kapfer, W. Mickel, K. Mecke, and G. E. Schröder-Turk, “Jammed Spheres: Minkowski Tensors Reveal Onset of Local Crystallinity,” *Physical Review E*, vol. 85, p. 030301, March 2012.

[92] T. E. Smidt, “Euclidean Symmetry and Equivariance in Machine Learning,” *Trends in Chemistry*, vol. 3, pp. 82–85, February 2021.

[93] M. Geiger, T. Smidt, A. M., B. K. Miller, W. Boomsma, B. Dice, K. Lapchevskyi, M. Weiler, M. Tyszkiewicz, S. Batzner, J. Frellsen, N. Jung, S. Sanborn, J. Rackers, and M. Bailey, “e3nn/e3nn: 2021-06-21,” June 2021.

[94] S. Batzner, T. E. Smidt, L. Sun, J. P. Mailoa, M. Kornbluth, N. Molinari, and B. Kozinsky, "SE(3)-Equivariant Graph Neural Networks for Data-Efficient and Accurate Interatomic Potentials," January 2021.

[95] Z. Chen, N. Andrejevic, T. Smidt, Z. Ding, Q. Xu, Y. T. Chi, Q. T. Nguyen, A. Alatas, J. Kong, and M. Li, "Direct Prediction of Phonon Density of States with Euclidean Neural Networks," *Advanced Science*, vol. 2021, p. 2004214, 2021.

[96] J. W. Galusha, L. R. Richey, J. S. Gardner, J. N. Cha, and M. H. Bartl, "Discovery of a Diamond-Based Photonic Crystal Structure in Beetle Scales," *Physical Review E*, vol. 77, p. 050904, May 2008.

[97] S. G. Johnson, P. R. Villeneuve, S. Fan, and J. D. Joannopoulos, "Linear Waveguides in Photonic-Crystal Slabs," *Physical Review B*, vol. 62, pp. 8212–8222, September 2000.

[98] R. C. Schroden, M. Al-Daous, C. F. Blanford, and A. Stein, "Optical Properties of Inverse Opal Photonic Crystals," *Chemistry of Materials*, vol. 14, no. 8, pp. 3305–3315, 2002.

[99] J. D. Jackson, *Classical Electrodynamics.* American Association of Physics Teachers, 1999.

[100] J. D. Joannopoulos, S. G. Johnson, J. N. Winn, and R. D. Meade, *Photonic Crystals.* Princeton University Press, 2011.

[101] S. G. Johnson and J. D. Joannopoulos, "Block-Iterative Frequency-Domain Methods for Maxwell's Equations in a Planewave Basis," *Opt. Express*, vol. 8, pp. 173–190, Jan 2001.

[102] A. da Silva Ferreira, G. N. M. Silveira, and H. E. H. Figueroa, "Predicting Complete Band-Gaps of 2d Photonic Crystals by Using Artificial Neural Networks," in *2017 SBMO/IEEE MTT-S International Microwave and Optoelectronics Conference (IMOC)*, pp. 1–5, IEEE, August 2017.

[103] A. da Silva Ferreira, G. N. Malheiros-Silveira, and H. E. Hernández Figueroa, "Designing Artificial Neural Networks for Band Structures Computations in Photonic Crystals," in *Physics and Simulation of Optoelectronic Devices XXVII* (M. Osiński, Y. Arakawa, and B. Witzigmann, eds.), vol. 10912, p. 60, Spie, February 2019.

[104] H. Men, K. Y. K. Lee, R. M. Freund, J. Peraire, and S. G. Johnson, "Robust Topology Optimization of Three-Dimensional Photonic-Crystal Band-Gap Structures," *Optics Express*, vol. 22, p. 22632, September 2014.

[105] A. H. Larsen, J. J. Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. B. Jensen, J. Kermode, J. R. Kitchin, E. L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, and K. W. Jacobsen, "The Atomic Simulation Environment–A Python Library for Working with Atoms," *Journal of Physics: Condensed Matter*, vol. 29, no. 27, p. 273002, 2017.

[106] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. Software available from tensorflow.org.

[107] A. Sergeev and M. Del Balso, "Horovod: Fast and Easy Distributed Deep Learning in TensorFlow," February 2018.

[108] A. Souza, L. B. Oliveira, S. Hollatz, M. Feldman, K. Olukotun, J. M. Holton, A. E. Cohen, and L. Nardi, "DeepFreak: Learning Crystallography Diffraction Patterns with Automated Machine Learning," April 2019.

[109] B. Liu, S. G. Johnson, J. D. Joannopoulos, and L. Lu, "Generalized Gilat-Raubenheimer Method for Density-of-States Calculation in Photonic Crystals," *Journal of Optics (United Kingdom)*, vol. 20, p. 044005, April 2018.

[110] G. Gilat and L. J. Raubenheimer, "Accurate Numerical Method for Calculating Frequency-Distribution Functions in Solids," *Physical Review*, vol. 144, pp. 390–395, April 1966.

[111] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, "Parallel Distributed Computing Using Python," *Advances in Water Resources*, vol. 34, pp. 1124–1139, September 2011.

[112] J. Behler and M. Parrinello, "Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces," *Physical Review Letters*, vol. 98, p. 146401, April 2007.

[113] V. Fung, G. Hu, P. Ganesh, and B. G. Sumpter, "Machine Learned Features from Density of States for Accurate Adsorption Energy Prediction," *Nature Communications*, vol. 12, p. 88, December 2021.

[114] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array Programming with NumPy," *Nature*, vol. 585, pp. 357–362, September 2020.

[115] B. Dice, V. Ramasubramani, E. Harper, M. Spellings, J. Anderson, and S. Glotzer, "Analyzing Particle Systems for Machine Learning and Data Visualization with freud," in *Proceedings of the 18th Python in Science Conference*, pp. 27–33, 2019.

[116] P. R. ten Wolde, M. J. Ruiz-Montero, and D. Frenkel, "Numerical Calculation of the Rate of Crystal Nucleation in a Lennard-Jones System At Moderate Undercooling," *The Journal of Chemical Physics*, vol. 104, pp. 9932–9947, June 1996.

[117] N. Michaud-Agrawal, E. J. Denning, T. B. Woolf, and O. Beckstein, "MDAnalysis: A Toolkit for the Analysis of Molecular Dynamics Simulations," *Journal of Computational Chemistry*, vol. 32, pp. 2319–2327, July 2011.

[118] R. T. McGibbon, K. A. Beauchamp, M. P. Harrigan, C. Klein, J. M. Swails, C. X. Hernández, C. R. Schwantes, L.-P. Wang, T. J. Lane, and V. S. Pande, "MDTraj: A Modern Open Library for the Analysis of Molecular Dynamics Trajectories," *Biophysical Journal*, vol. 109, pp. 1528–1532, October 2015.

[119] G. van Anders, D. Klotsa, N. K. Ahmed, M. Engel, and S. C. Glotzer, "Understanding Shape Entropy Through Local Dense Packing," *Proceedings of the National Academy of Sciences*, vol. 111, no. 45, pp. E4812–e4821, 2014.

[120] A. S. Karas, J. Glaser, and S. C. Glotzer, "Using Depletion to Control Colloidal Crystal Assemblies of Hard Cuboctahedra," *Soft Matter*, vol. 12, pp. 5199–5204, June 2016.

[121] E. S. Harper, R. L. Marson, J. A. Anderson, G. van Anders, and S. C. Glotzer, "Shape Allophiles Improve Entropic Assembly," *Soft Matter*, vol. 11, pp. 7250–7256, September 2015.

[122] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and Jupyter Development Team, "Jupyter Notebooks - A Publishing Format for Reproducible Computational Workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (F. Loizides and B. Scmidt, eds.), (Netherlands), pp. 87–90, IOS Press, 2016.

[123] E. G. Teich, M. Cieslak, B. Giesbrecht, J. M. Vettel, S. Grafton, T. D. Satterthwaite, and D. S. Bassett, "Crystallinity Characterization of White Matter in the Human Brain," *New Journal of Physics*, July 2021.

[124] B. D. Dice, B. L. Butler, V. Ramasubramani, A. Travitz, M. M. Henry, H. Ojha, K. L. Wang, C. S. Adorf, E. Jankowski, and S. C. Glotzer, "signac: Data Management and Workflows for Computational Researchers," in *Proceedings of the 20th Python in Science Conference*, pp. 23–32, 2021.

[125] M. Govoni and G. Galli, "GW100: Comparison of Methods and Accuracy of Results Obtained with the WEST Code," *Journal of Chemical Theory and Computation*, vol. 14, no. 4, pp. 1895–1909, 2018.

[126] S. Thomas, M. Alberts, M. M. Henry, C. E. Estridge, and E. Jankowski, "Routine Million-Particle Simulations of Epoxy Curing with Dissipative Particle Dynamics," *Journal of Theoretical and Computational Chemistry*, vol. 17, p. 1840005, April 2018.

[127] F. Musil, M. Veit, A. Goscinski, G. Fraux, M. J. Willatt, M. Stricker, T. Junge, and M. Ceriotti, "Efficient Implementation of Atom-Density Representations," *The Journal of Chemical Physics*, vol. 154, p. 114109, March 2021.

[128] M. P. Howard, T. M. Truskett, and A. Nikoubashman, "Cross-Stream Migration of a Brownian Droplet in a Polymer Solution Under Poiseuille Flow," *Soft Matter*, vol. 15, no. 15, pp. 3168–3178, 2019.

[129] E. S. Harper, E. J. Coyle, J. P. Vernon, and M. S. Mills, "Inverse Design of Broadband Highly Reflective Metasurfaces Using Neural Networks," *Physical Review B*, vol. 101, p. 195104, May 2020.

[130] D. Rodziewicz and J. Dice, "Drought Risk to the Agriculture Sector," *The Federal Reserve Bank of Kansas City Economic Review*, December 2020.

[131] P. T. Cummings, C. McCabe, C. R. Iacovella, A. Ledeczi, E. Jankowski, A. Jayaraman, J. C. Palmer, E. J. Maginn, S. C. Glotzer, J. A. Anderson, J. I. Siepmann, J. Potoff, R. A. Matsumoto, J. B. Gilmer, R. S. DeFever, R. Singh, and B. Crawford, "Open-Source Molecular Modeling Software in Chemical Engineering Focusing on the Molecular Simulation Design Framework," *AIChE Journal*, vol. 67, no. 3, p. e17206, 2021.

[132] A. Collette, *Python and HDF5.* O'Reilly, 2013.

[133] J. A. Anderson, C. D. Lorenz, and A. Travesset, "General Purpose Molecular Dynamics Simulations Fully Implemented on Graphics Processing Units," *Journal of Computational Physics*, vol. 227, no. 10, pp. 5342–5359, 2008.

[134] J. Glaser, T. D. Nguyen, J. A. Anderson, P. Lui, F. Spiga, J. A. Millan, D. C. Morse, and S. C. Glotzer, "Strong Scaling of General-Purpose Molecular Dynamics Simulations on GPUs," *Computer Physics Communications*, vol. 192, pp. 97–107.

[135] S. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," *Journal of Computational Physics*, vol. 117, pp. 1–19, Mar 1995.

[136] H. Berendsen, D. van der Spoel, and R. van Drunen, "GROMACS: A Message-Passing Parallel Molecular Dynamics Implementation," *Computer Physics Communications*, vol. 91, pp. 43–56, September 1995.

[137] R. K. Cersonsky, J. Dshemuchadse, J. A. Antonaglia, G. van Anders, and S. C. Glotzer, "Pressure-Tunable Photonic Band Gaps in an Entropic Colloidal Crystal," *Physical Review Materials*, vol. 2, p. 125201, 2018.

[138] A. J. Simon, Y. Zhou, V. Ramasubramani, J. Glaser, A. Pothukuchy, J. Gollihar, J. C. Gerberich, J. Leggere, B. R. Morrow, C. Jung, S. C. Glotzer, D. W. Taylor, and A. D. Ellington, "Supercharging Enables Organized Assembly of Synthetic Biomolecules," *Nature Chemistry*, vol. 11, pp. 204–212, 2019.

[139] S. C. Glotzer and M. J. Solomon, "Anisotropy of Building Blocks and Their Assembly Into Complex Structures," *Nature Materials*, vol. 6, pp. 557–562, Aug 2007.

[140] S. J. Tan, M. J. Campolongo, D. Luo, and W. Cheng, "Building Plasmonic Nanostructures with DNA," *Nature Nanotechnology*, vol. 6, pp. 268–276, May 2011.

[141] T. E. Oliphant, *A Guide to NumPy.* Trelgol Publishing, 2006.

[142] E. Jones, T. Oliphant, P. Peterson, and others, "SciPy: Open Source Scientific Tools for Python," 2001.

[143] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The Best of Both Worlds," *Computing in Science & Engineering*, vol. 13, pp. 31–39, March 2011.

[144] Intel, "Intel Threading Building Blocks," 2018.

[145] E. S. Harper, B. Waters, and S. C. Glotzer, "Hierarchical Self-Assembly of Hard Cube Derivatives," *Soft Matter*, vol. 15, pp. 3733–3739, 2019.

[146] S. S. Schoenholz, E. D. Cubuk, E. Kaxiras, and A. J. Liu, "A Structural Approach to Relaxation in Glassy Liquids," *Nature Physics*, no. February, pp. 1–11, 2015.

[147] M. Fulford, M. Salvalaglio, and C. Molteni, "DeepIce: A Deep Neural Network Approach to Identify Ice and Water Molecules," *Journal of Chemical Information and Modeling*, p. acs.jcim.9b00005, March 2019.

[148] M. W. Thompson, R. Matsumoto, R. L. Sacci, N. C. Sanders, and P. T. Cummings, "Scalable Screening of Soft Matter: A Case Study of Mixture of Ionic Liquids and Organic Solvents," *J. Phys. Chem. B*, vol. 123, no. 6, pp. 1340–1347.

[149] S. Pronk, S. Páll, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M. R. Shirts, J. C. Smith, P. M. Kasson, D. van der Spoel, B. Hess, and E. Lindahl, "GROMACS 4.5: A High-Throughput and Highly Parallel Open Source Molecular Simulation Toolkit," *Bioinformatics*, vol. 29, no. 7, pp. 845–854.

[150] E. Lindahl, B. Hess, and D. van der Spoel, "GROMACS 3.0: A Package for Molecular Simulation and Trajectory Analysis," *J Mol Model*, vol. 7, no. 8, pp. 306–317.

[151] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, "GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation," *J. Chem. Theory Comput.*, vol. 4, no. 3, pp. 435–447.

[152] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, "GROMACS: High Performance Molecular Simulations Through Multi-Level Parallelism from Laptops to Supercomputers," *SoftwareX*, vol. 1-2, pp. 19–25.

[153] The pandas Development Team, "pandas-dev/pandas: Pandas," February 2020.

[154] W. McKinney, "Data Structures for Statistical Computing in Python," *Proceedings of the 9th Python in Science Conference*, pp. 56–61.

[155] D. D. Team, *Dask: Library for Dynamic Task Scheduling*, 2016.

[156] M. Rocklin, "Dask: Parallel Computation with Blocked Algorithms and Task Scheduling," in *Proceedings of the 14th Python in Science Conference* (K. Huff and J. Bergstra, eds.), pp. 130–136, 2015.

[157] J. K. Shah, E. Marin-Rimoldi, R. G. Mullen, B. P. Keene, S. Khan, A. S. Paluch, N. Rai, L. L. Romanielo, T. W. Rosch, B. Yoo, *et al.*, "Cassandra: An Open Source Monte Carlo Package for Molecular Simulation," 2017.

[158] R. S. DeFever, R. A. Matsumoto, A. W. Dowling, P. T. Cummings, and E. J. Maginn, "MoS-DeF Cassandra: A Complete Python Interface for the Cassandra Monte Carlo Software," *Journal of Computational Chemistry*, 2021.

[159] C. Klein, A. Z. Summers, M. W. Thompson, J. B. Gilmer, C. McCabe, P. T. Cummings, J. Sallai, and C. R. Iacovella, "Formalizing Atom-Typing and the Dissemination of Force Fields with Foyer," *Computational Materials Science*, vol. 167, pp. 215–227.

[160] 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 *Proceedings of the 19th Python in Science Conference* (Meghann Agarwal, Chris Calloway, Dillon Niederhut, and David Shupe, eds.), pp. 24–31, 2020.

[161] C. Klein, J. Sallai, T. J. Jones, C. R. Iacovella, C. McCabe, and P. T. Cummings, "A Hierarchical, Component Based Approach to Screening Properties of Soft Matter," in *Foundations of Molecular Modeling and Simulation: Select Papers from FOMMS 2015* (R. Q. Snurr, C. S. Adjiman, and D. A. Kofke, eds.), Molecular Modeling and Simulation, pp. 79–92, Springer.

[162] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. D. Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, and R. M. Wentzcovitch, "QUANTUM ESPRESSO: A Modular and Open-Source Software Project for Quantum Simulations of Materials," *Journal of Physics: Condensed Matter*, vol. 21, p. 395502, September 2009.

[163] V. Liu and S. Fan, "S4: A Free Electromagnetic Solver for Layered Periodic Structures," *Computer Physics Communications*, vol. 183, no. 10, pp. 2233–2244, 2012.

[164] G. Kresse and J. Furthmüller, "Efficient Iterative Schemes for Ab Initio Total-Energy Calculations Using a Plane-Wave Basis Set," *Phys. Rev. B*, vol. 54, pp. 11169–11186, Oct 1996.

[165] F. Chollet *et al.*, "Keras," 2015.

[166] M. W. Thompson, J. B. Gilmer, R. A. Matsumoto, C. D. Quach, P. Shamaprasad, A. H. Yang, C. R. Iacovella, C. McCabe, and P. T. Cummings, "Towards Molecular Simulations That Are Transparent, Reproducible, Usable by Others, and Extensible (TRUE)," *Molecular Physics*, vol. 118, p. e1742938, June 2020.

[167] T. H. Group, "Hierarchical Data Format, Version 5," 1997-2021.

[168] M. Davis, "snakeviz."

[169] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function Using NetworkX," in *Proceedings of the 7th Python in Science Conference* (G. Varoquaux, T. Vaught, and J. Millman, eds.), (Pasadena, CA USA), pp. 11–15, 2008.

[170] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr, "XSEDE: Accelerating Scientific Discovery," *Computing in Science Engineering*, vol. 16, no. 5, pp. 62–74, 2014.

[171] A. Miles, jakirkham, M. Durant, M. Bussonnier, J. Bourbeau, T. Onalan, J. Hamman, Z. Patel, M. Rocklin, shikharsg, R. Abernathey, J. Moore, V. Schut, raphael dussin, E. S. de Andrade, C. Noyes, A. Jelenak, A. Banihirwe, C. Barnes, G. Sakkis, J. Funke, J. Kelleher, J. Jevnik, J. Swaney, P. S. Rahul, S. Saalfeld, john, T. Tran, pyup.io bot, and sbalmer, "zarr-developers/zarr-python: v2.5.0," October 2020.

[172] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A LLVM-Based Python JIT Compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, pp. 1–6, Association for Computing Machinery.