Inverse Design and Analysis of Crystallization Pathways of Colloidal Systems

by

Carl Simon Adorf

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Chemical Engineering)
in The University of Michigan
2019

Carl Simon Adorf

csadorf@umich.edu

ORCID iD: 0000-0003-4962-2495

Gewidmet meiner kleinen und meiner großen Familie.

ACKNOWLEDGEMENTS

First of, I would like to thank my advisor and mentor Sharon C. Glotzer for supporting me in all aspects of my research and for being an incredible champion of my work. You inspire me as a researcher and a mentor. Thank you for the opportunity to be a part of this amazing group of people and for always pushing me to be better.

I would also like to thank the many mentors, friends, and colleagues I have met along the way, including the late Ahmed E. Ismail, Peter Cummings, and Christopher Iacovella, who encouraged and supported me in pursuing a doctoral degree in the first place. Thank you to Joshua A. Anderson, Jens Glaser, and Karen Coulter for your support and guidance. Thank you to Pablo Damasceno, Ryan Marson, James Antonaglia, Erin Teich, Paul Dodd, Chrisy Xiyu Du, Julia Dshemuchdase, and Tim Moore for your friendship and mentorship. Thank you to Vyas Ramasubramani and Bradley D. Dice, it has been incredible fun to work with you and I hope that we will be able to continue to work on our shared projects in the future.

I would like to thank my mum, my dad, and my sister for their love and continued support over the past five years. Although we only had very few chances to see each other in person since I moved abroad, I have never felt closer to you. Thank you so much for being there to listen and encourage me.

Finally, I would like to thank my wife and best friend Rosy, who was always there to help me push through the lows and celebrate the highs. You are a wonderful person and an amazing scientist and I am extremely grateful that you are part of my life.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The ability to produce particles on the nano-scale that self-assemble into soft materials with specifically targeted properties is a promising avenue for a completely new class of materials with novel applications. New synthesis routes and increased computational resources have led to the proposition, modeling, simulation, and experimental realization of a whole plethora of new particle types, including but not limited to faceted polyhedra, Janus particles, lock-and-key particles, and self-propelled particles. The *inverse design* problem of engineering particles for targeted self-assembly is challenging because of the large available design space. Unlike atoms which are more easily classified by their composition and energy state, nanoparticles can be almost arbitrarily shaped and interactions between particles can be further modulated to various degrees, for example with complete or partial surface coatings.

One of the main determinants of material properties is the breaking of symmetries manifested in the formation of crystal structures. In Chapter II of this dissertation, I present a computational method for the optimization of isotropic pair potentials (IPPs) as a general model for the effective interaction between colloidal particles. Using this method, which is based on the minimization of the relative entropy, I optimize IPPs for various simple and complex crystal structures, including simple cubic ($cP1$), body-centered cubic ($cI2$), face-centered cubic ($cF4$), β-tin $tI4$-Sn ($tI4$), diamond ($cF8$), A15-type $cP8$-Cr_3Si ($cP8$), σ-phase $tP30$-CrFe ($tP30$), and clathrate-I $cP54$-K_4Si_{23} ($cP54$). Reducing the design space to IPPs allows us to explore it more effectively, however solutions found in this way still need to be mapped to a physical model for experimental synthesis.

Chapter III is a study of crystallization pathways, including pathways of some of the

models presented in Chapter II. I developed and applied an unsupervised machine learning (ML) workflow for the analysis of self-assembly pathways, which allows us to make observations about the general mechanism of, as well as identify local particle environments that play a role in, the nucleation and growth of the crystal structure. I observe two-step nucleation for all tested crystal structures at moderate supercooling and can demonstrate that random fluctuations of local order play a crucial role in mediating nucleation and growth.

In Chapter IV, I present `signac`, a software framework that assists researchers in managing their computational data and implementing workflows that operate on that data. The `signac` framework was originally designed and implemented by me and has since its early inception matured into an open-source project with a team of core maintainers and many internal and external contributors and users. Recent work has been focused on enabling the implementation of more complex workflows and increasing the user base, which is facilitated by an affiliation of the project with the NumFOCUS organization.

Chapter V provides guidelines on how to efficiently develop robust and reusable software within an academic research environment. These guidelines are presented in the form of a heuristic named *lazy refactoring*, which in essence priorities the incremental development of working solutions over general complete solutions. Determining the correct scope and Application Program Interface (API) for general solutions from scratch is typically much more difficult compared to refactoring existing partial solutions.

I conclude this dissertation with a general summary of the presented work and provide a brief outlook on potential future research directions.

CHAPTER I

Introduction

1.1 Historical Background

The use of computational simulations for the study of materials and physical processes has come a long way since Metropolis et al. published on their proposed method of using "fast computing machines" for the calculation of an equation of state of a monatomic fluid in 1953 [1]. In its early stage often regarded as a secondary, complementary technique to merely verify or support experimental results, the field of computational research has matured to an essential pillar of modern science and engineering and is now more often found in the role of driver of innovation for both the science itself and the technology it is supported by [2, 3].

The evaluation of molecular models with statistical mechanics is especially attractive and amenable to computational research since the fundamental principles [4] are both simple to implement and due to the small time and length scales elusive to experimental observation at the same time. The simplicity of implementation of course only holds for the most naive implementations; highly optimized implementations for modern architectures can be arbitrarily complex [5–8] and are undergoing continuous innovation to this day [9, 10].

However, employing computational "experiments" as opposed to physical experiments has other advantages despite simplicity of implementation and the ability to collect statistics on all observables at full resolution. Possibly the biggest strength is their versatility. A computational model of, *e.g.*, a colloidal particle on the nano- to microscale [11] is not constrained by available routes to fabrication, costs of fabrication posed by materials and labor, and even the laws of physics in general. For instance, while *hard spheres* can be posed theoretically, it is impossible to physically fabricate a sphere that has absolutely zero repulsion or attraction at finite distances, and a zero probability of overlap below a certain diameter. A hard sphere model represents an approximation of a physical object with very similar properties (highly spherical, extremely high bulk modulus) in a vacuum and microgravity, but is nonetheless highly informative.

1.2 Colloidal Self-Assembly

Alder and Wrainwright used computer simulations in 1957 [12] to demonstrate that hard spheres – a particle model with maximal symmetry – will self-assemble into a symmetry-breaking face-centered cubic (FCC) crystal structure absent any external forces or energetic contributions, *i.e.*, through pure entropy maximization alone. This at the time highly surprising result inspired a heated debate about the nature of the formation of crystals, and inspired researchers to explore the self-assembly behavior of other "simple" hard particle models such as rods [13] and ellipses [14, 15], and more recently polyhedra [16, 17].

The study of hard objects with statistical mechanics is most easily described within the canonical ensemble, which represents the collection of all possible states of a system with constant number of particles N and constant volume V, and which is in equilibrium with an infinitely large heat bath at temperature T. The free energy of this ensemble is equivalent to the Helmholtz free energy A and is defined as

$$A = U - TS, \tag{1.1}$$

where U denotes the internal energy as a function of all particle coordinates and momenta, and S is the entropy of the system.

The probability of a particular microstate i with internal energy U_i to occur within the canonical ensemble is equal to

$$P_i = \mathrm{e}^{\beta(A-U_i)}, \tag{1.2}$$

where β is the inverse reduced temperature $(k_B T)^{-1}$. For a system that is comprised of purely hard shapes with no other interactions, the probability of a particular microstate to occur is either zero (at least one shape overlaps with another shape) or constant across all possible microstates (no overlaps). That in turn means that the energetic term U of the free energy A must be zero for *all* valid configurations, which means that minimizing the free energy A is equivalent to maximizing the entropic term TS, *i.e.*, maximizing the entropy S since T is constant. The crystallization or increase in order through free energy minimization of a system comprised of hard particles is therefore also referred to as “entropic ordering“ [18] or “entropic crystallization” [19] since the only remaining and therefore relevant term is the entropic term TS.

To this date, the concept of “entropic crystallization” is still not widely known within the scientific community, albeit having been shown to be real beyond doubt both in theory and experiment. There has also been some confusion as to how the packing of hard shapes is related to crystallization of hard shapes at finite pressures, often mistaken as the underlying driving mechanism. While local dense packing plays a significant role in entropic crystallization [20], Cersonsky et al. showed that the relationship between global dense packing and self-assembly is purely correlative, with additional factors determining the self-assembly structure of colloidal crystals [21].

1.3 Forward vs. Inverse Design

One of the main advantages in using computational experiments compared to physical experiments is the ability to rapidly test different models and physical conditions, vastly expanding the design space available for the invention of novel materials with highly unique properties. However, an increased number of degrees of freedom also comes with an increase in complexity. Glotzer and Solomon devised a framework for the design of complex materials with anisotropic building blocks [22] which serves as a guideline on how to describe and parameterize the design space of anisotropic shapes.

Damasceno et al. [17] and Klotsa et al. [23] explored this space for the shape class of convex polyhedra and showed that simple shapes are indeed a possible avenue for the self-assembly of highly complex crystal structures. And while both studies sought after and revealed certain correlations that can be translated into design rules, they are also an example of "forward design", essentially a brute-force approach to exploring the design space with the goal of uncovering interesting phenomena.

In contrast, "inverse design" is a more targeted approach, where the design targets are set *a priori*, and the design space is explored in a more directed manner [24]. Materials inverse design can be expressed as an optimization problem, in which case it is made accessible to solutions via established optimization algorithm such as Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [25–27], or as an extended thermodynamical ensemble, such as the "Digital Alchemy Framework" devised by van Anders et al. [28]. The former approach is more general and may be advantageous when relevant degrees of freedom can be made accessible to the algorithm, whereas the second approach can be embedded and evaluated via existing algorithms for the integration of statistical mechanical ensembles.

1.4 Relative Entropy

Regardless of the methodology for inverse design, mathematically it is most elegantly expressed within the framework of relative entropy [29]. The relative entropy or Kullback-Leibler divergence [30]

$$D_{\mathrm{KL}}(P|Q) = -\sum_{x\in\chi} P(x)\log\left(\frac{Q(x)}{P(x)}\right) \tag{1.3}$$

is a concept out of information theory and is a measure for the similarity of two probability distributions P and Q evaluated for a continuous random variable x out of distribution χ. Assuming that we have two models[1] a and b which are both sampled within the same statistical ensemble, and that we collect statistics on a set of observables $\mathbf{O}$, then the relative entropy of the distribution of said observables will be minimized if and only if the two models have a similar probability distribution of microstates that is correlated with said observables. For example, the probability distribution of nearest neighbor distances as expressed in the radial distribution function $g(r)$ will be similar for both models at the same state point. Scott Shell was able to contextualize many independently developed algorithms that are aimed at minimizing the difference between two models, such as algorithms developed for the purpose of coarse-graining atomistic models, within the framework of relative entropy minimization [29].

A minimal relative entropy between two distributions of a specific observable is a necessary, but not a sufficient condition for two models to behave identically in all aspects. It is possible that two models produce the exact same distribution for a particular set of observables at a specific state point, and are in that respect equivalent for all intents and purposes, but then behave completely orthogonal with respect to a different set of observables and/or at a different state point. For example, Moore et al. [31] showed that it is possible to reduce the state-dependence of isotropic pair potentials (IPPs), which are optimized via the Iterative Boltzmann Inversion (IBI) algorithm, to match the radial distribution function (RDF) of a given target state by optimizing the model for multiple state points simultaneously. This means that the relative entropy for this particular observable is globally reduced by refining the model, even though it might have appeared minimized locally.

[1]A model in this context is equivalent to a Hamiltonian.

1.5 Outline

In Chapter II, I present an approach on how to use relative entropy minimization to optimize tabulated IPP models for the self-assembly of specifically targeted crystal structures. I combine the approach presented by Edlund et al. [32, 33], who showed that designing IPPs directly in Fourier space is a highly effective way to guarantee that a targeted crystal structure presents a ground state for a given energy function, with the method published by Lindquist et al. [34], based on the minimization of relative entropy. In this way we are able to optimize multiple IPPs for the self-assembly of various simple to complex crystal structures while at the same time controlling against overfitting of the function.

How and if a model optimized in this way can be mapped to a physical system of particles depends strongly on the complexity of the interaction function, *i.e.*, the number of features and the interaction range. In either case, the function's complexity serves as an upper bound on how complex a physical particle that assembles the same structure would need to be.

But even without an immediate experimental realization, we can still study the model to learn more about potential mechanisms for the self-assembly of a particular colloidal crystal. In Chapter III, I present a machine learning (ML)-based approach for the semi-automated identification and analysis of self-assembly pathways. Among other systems, I show results for the analysis of crystallization pathways for some of the models obtained in Chapter II.

The methods presented as part of Chapters II and III allow us to overcome interesting scientific problems, however their implementation and large-scale execution also poses tough technical challenges. Optimizing many different potential functions for a multitude of different target crystal structures with a variety of optimization parameters for the work presented in Chapter II requires a lot of compute power and a well-managed data space with diligent tracking of all relevant parameters. Similarly, the unsupervised ML workflow presented in Chapter III involves the execution of hundreds of interdependent data space operations starting with the computation of the descriptor space, its dimensionality reduction, model training, and numerous plotting and rendering operations.

The main challenges to overcome are the establishment of a well defined data space, where data and metadata are robustly linked. Metadata in this context are all attributes necessary to place the data into context and thus make the data space accessible to oneself and others. The second challenge is the definition and execution of workflows in a way that they are reproducible, but also scalable.

Much of the work presented here is driven by massively parallelized and partially GPU-accelerated algorithms implemented in HOOMD-blue [6, 8] and freud [35]. That includes for example Molecular Dynamics (MD) and Hard-particle Monte Carlo (HPMC) simulations of particle systems and the parallelized computation of neighbor-based order parameters, which can be executed efficiently on hundreds or thousands of processing units in parallel and require massive amounts of random-access memory (RAM) and on-disk storage. However, many other operations are much less computationally intensive, such as the collection of statistics and graph plotting. The challenge is to have an integrated workflow that supports both the very small and the very large data space operations for both early exploration and large-scale production.

In Chapter IV, I present **signac**, a software framework that I designed and implemented to overcome exactly these challenges. The first components of the framework were released open-source in 2016, which spawned a significant expansion of its developer and user base. As of the day of writing, the software has approximately twenty internal and external contributors and on the order of hundreds of users.[2]

However, developing reusable and robust software within an academic environment poses its own set of challenges. Chapter V provides a guideline on how to efficiently implement reusable scientific software in an academic setting. My colleague Vyas Ramasubramani and I together with Joshua Anderson and Sharon C. Glotzer developed the *lazy refactoring* approach, which is targeted at researchers in academic groups who find themselves in the position of needing to develop software as part of their general research efforts.

[2]The size of the user base is an estimate based on the number of known users as well as website access and download statistics.

While software developers both in industry and academia generally face very similar challenges, there are a few crucial differences that make it difficult and in some cases ill-advisable to simply adopt best practices and standards developed for industry within the academic research group. For example, on the one hand many software developers in academia are non-experts and software development represents only a small fraction of their actual work and may not be rewarded as much by principle investigators and program managers. On the other hand, researchers who do aspire to developing high-quality research software and are generally supported in this effort may find it challenging to determine the right scope and consequently spend either too little or too much effort on certain components of their software.

The *lazy refactoring* approach asks developers to prioritize the development of working solutions over general solutions while consequently generalizing these the moment that they need to be reused more than once. This is motivated by the experience that composing a general solution from one or more specialized ones is usually orders of magnitude easier than trying to understand and design software for a general problem space without even having solved parts of it.

I conclude with a summary of the presented work and an outlook for potential future research directions in Chapter VI.

Bibliography

[1] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. doi: 10.1063/1.1699114.

[2] N. Jouppi, C. Young, N. Patil, and D. Patterson. Motivation for and Evaluation of the First Tensor Processing Unit. *IEEE Micro*, 38(3):10–19, 2018. ISSN 0272-1732. doi: 10.1109/MM.2018.032271057.

[3] D. Luebke. CUDA: Scalable parallel programming for high-performance scientific computing. In *2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 836–838, May 2008. doi: 10.1109/ISBI.2008.4541126.

[4] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation: From Algorithms to Applications*. Elsevier, October 2002. ISBN 978-0-12-267351-1. doi: 10.1016/B978-0-12-267351-1.X5000-7.

[5] Steve Plimpton. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics*, 117(1):1–19, 1995. doi: 10.1006/jcph.1995.1039.

[6] Joshua A. Anderson, Chris D. Lorenz, and A. Travesset. General purpose molecular dynamics simulations fully implemented on graphics processing units. *Journal of Computational Physics*, 227(10):5342–5359, 2008. doi: 10.1016/j.jcp.2008.01.047.

[7] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilrd Pll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1-2:19–25, 2015. doi: 10.1016/j.softx.2015.06.001.

[8] Joshua A. Anderson, M. Eric Irrgang, Sharon C. Glotzer, M. Eric Irrgang, and Sharon C. Glotzer. Scalable Metropolis Monte Carlo for simulation of hard shapes. *Computer Physics Communications*, 204:21–30, 2016. doi: 10.1016/j.cpc.2016.02.024.

[9] Jens Glaser, Trung Dac Nguyen, Joshua A Anderson, Pak Lui, Filippo Spiga, Jaime A Millan, David C Morse, and Sharon C. Glotzer. Strong scaling of general-purpose molecular dynamics simulations on GPUs. *Computer Physics Communications*, 192:97–107, 2015. doi: 10.1016/j.cpc.2015.02.028.

[10] Xiaohui Duan, Ping Gao, Tingjian Zhang, Meng Zhang, Weiguo Liu, Wusheng Zhang, Wei Xue, Haohuan Fu, Lin Gan, Dexun Chen, Xiangxu Meng, and Guangwen Yang. Redesigning lammps for peta-scale and hundred-billion-atom simulation on sunway taihulight. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18, pages 12:1–12:12, Piscataway, NJ, USA, 2018. IEEE Press. doi: 10.1109/SC.2018.00015.

[11] Bo Li, Di Zhou, and Yilong Han. Assembly and phase transitions of colloidal crystals. *Nature Reviews Materials*, 1(2):15011, 2016. doi: 10.1038/natrevmats.2015.11.

[12] B. J. Alder and T. E. Wainwright. Phase Transition for a Hard Sphere System. *The Journal of Chemical Physics*, 27(5):1208, 1957. doi: 10.1063/1.1743957.

[13] Lars Onsager. The Effects of Shape on the Interaction of Colloidal Particles. *Annals of the New York Academy of Sciences*, 51(4):627–659, 1949. doi: 10.1111/j.1749-6632.1949.tb27296.x.

[14] Dominique Levesque, Daniel Schiff, and Jacques Vieillard-Baron. Structure Factor of a Two-Dimensional Fluid of Hard Ellipses. *J. Chem. Phys.*, 51(8):3625–3626, 1969. doi: 10.1063/1.1672567.

[15] Jacques Vieillard-Baron. Phase Transitions of the Classical Hard-Ellipse System. *J. Chem. Phys.*, 56(10):4729–4744, 1972. doi: 10.1063/1.1676946.

[16] Vinothan N. Manoharan. Dense Packing and Symmetry in Small Clusters of Microspheres. *Science*, 301(5632):483–487, 2003. ISSN 0036-8075. doi: 10.1126/science.1086189.

[17] P. F. Damasceno, M. Engel, and S. C. Glotzer. Predictive Self-Assembly of Polyhedra into Complex Structures. *Science*, 337(6093):453–457, 2012. doi: 10.1126/science.1220869.

[18] Daan Frenkel. Order through entropy. *Nature Materials*, 14(1):9–12, 2014. doi: 10.1038/nmat4178.

[19] Sangmin Lee, Erin G. Teich, Michael Engel, and Sharon C. Glotzer. Entropic colloidal crystallization pathways via fluid-fluid transitions and multidimensional prenucleation motifs. *PNAS*, 116(30):14843–14851, 2019. doi: 10.1073/pnas.1905929116.

[20] Greg van Anders, Daphne Klotsa, N Khalid Ahmed, Michael Engel, and Sharon C Glotzer. Understanding shape entropy through local dense packing. *Proceedings of the National Academy of Sciences*, 111(45):E4812–E4821, 2014. doi: 10.1073/pnas.1418159111.

[21] Rose K. Cersonsky, Greg van Anders, Paul M. Dodd, and Sharon C. Glotzer. Relevance of packing to colloidal self-assembly. *Proceedings of the National Academy of Sciences*, pages 1439–1444, 2018. doi: 10.1073/pnas.1720139115.

[22] Sharon C. Glotzer and Michael J Solomon. Anisotropy of building blocks and their assembly into complex structures. *Nature Materials*, 6(7):557–562, 2007. doi: 10.1038/nmat1949.

[23] Daphne Klotsa, Elizabeth R.Chen, Michael Engel, and Sharon C.Glotzer. Intermediate crystalline structures of colloids in shape space. *Soft Matter*, 14(43):8692–8697, 2018. doi: 10.1039/C8SM01573B.

[24] Avni Jain, Jonathan A. Bollinger, and Thomas M. Truskett. Inverse methods for material design. *AIChE Journal*, 60(8):2732–2740, 2014. doi: 10.1002/aic.14491.

[25] Christian Igel, Nikolaus Hansen, and Stefan Roth. Covariance Matrix Adaptation for Multi-objective Optimization. *Evolutionary Computation*, 15(1):1–28, 2007. doi: 10.1162/evco.2007.15.1.1.

[26] Nikolaus Hansen. *The CMA Evolution Strategy: A Comparing Review*, pages 75–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-32494-2. doi: 10.1007/3-540-32494-1_4.

[27] Rajneesh Kumar, Gabriele M. Coli, Marjolein Dijkstra, and Srikanth Sastry. Inverse design of charged colloidal particle interactions for self assembly into specified crystal structures. *arXiv:1905.11061 [cond-mat]*, May 2019. arXiv: 1905.11061.

[28] Greg van Anders, Daphne Klotsa, Andrew S Karas, Paul M Dodd, and Sharon C Glotzer. Digital Alchemy for Materials Design: Colloids and Beyond. *ACS Nano*, 9(10):9542–9553, 2015. doi: 10.1021/acsnano.5b04181.

[29] M. Scott Shell. The relative entropy is fundamental to multiscale and inverse thermodynamic problems. *Journal of Chemical Physics*, 129(14):144108, 2008. doi: 10.1063/1.2992060.

[30] S. Kullback and R. A. Leibler. On Information and Sufficiency. *Ann. Math. Statist.*, 22 (1):79–86, 1951. doi: 10.1214/aoms/1177729694.

[31] Timothy C Moore, Christopher R Iacovella, and Clare McCabe. Derivation of coarse-grained potentials via multistate iterative Boltzmann inversion. *The Journal of Chemical Physics*, 140(22):224104, 2014. doi: 10.1063/1.4880555.

[32] E. Edlund, O. Lindgren, and M. Nilsson Jacobi. Designing Isotropic Interactions for Self-Assembly of Complex Lattices. *Physical Review Letters*, 107(8):085503, 2011. doi: 10.1103/PhysRevLett.107.085503.

[33] E. Edlund, O. Lindgren, and M. Nilsson Jacobi. Using the uncertainty principle to design simple interactions for targeted self-assembly. *The Journal of Chemical Physics*, 139(2):024107, 2013. doi: 10.1063/1.4812727.

[34] Beth A. Lindquist, Ryan B. Jadrich, and Thomas M. Truskett. Communication: Inverse design for self-assembly via on-the-fly optimization. *The Journal of Chemical Physics*, 145(11):111101, 2016. doi: 10.1063/1.4962754.

[35] Vyas Ramasubramani, Bradley D. Dice, Eric S. Harper, Matthew P. Spellings, Joshua A. Anderson, and Sharon C. Glotzer. freud: A Software Suite for High Throughput Analysis of Particle Simulation Data. *arXiv:1906.06317 [cond-mat, physics:physics]*, June 2019. arXiv: 1906.06317.

CHAPTER II

Inverse Design of Isotropic Pair Potentials for the Self-Assembly of Complex Structures

This chapter is adapted from a publication in the *Journal of Chemical Physics* in 2018 authored by me, James Antonaglia, Julia Dshemuchadse, and Sharon C. Glotzer. The article was assigned DOI:10.1063/1.5063802.

2.1 Introduction

The ability to synthesize novel complex materials *via* the self-assembly of building blocks on the nanoscale presents an enormous opportunity for the design of materials with targeted behavior, including mechanical and optical properties [1, 2]. Following the definition of Whitesides *et al.* [3], a self-assembly process is characterized by the emergence of structure from disordered, distinct constituents and governed by their shapes and interactions. In order to design a material for synthesis *via* self-assembly, we need to answer the question "What constituents are required for the targeted self-assembly behavior?" This question represents the *inverse* problem in contrast to the *forward* problem of "What is the self-assembly behavior of certain predefined constituents?" [1, 4] The major challenge in solving the *inverse* problem is the vast search space constituted by the sheer limitless choice and possible combinations of feasible building blocks and interactions [5–7]. Of course, simply identifying the constituents

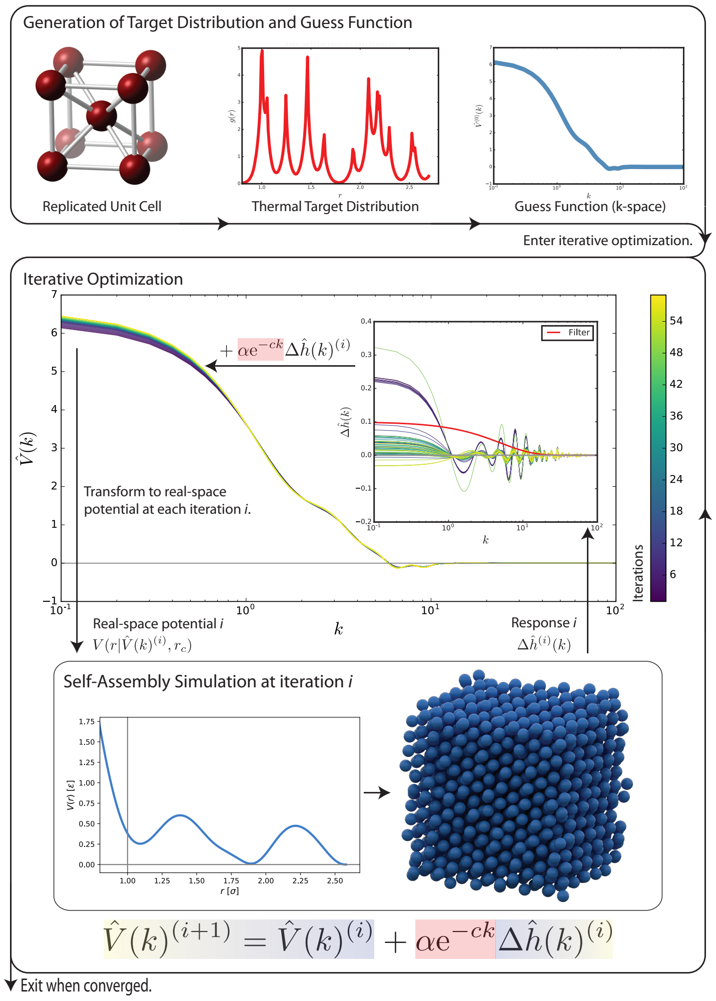

Generation of Target Distribution and Guess Function
Replicated Unit Cell
Thermal Target Distribution
Guess Function (k-space)
Enter iterative optimization.
Iterative Optimization
$\hat{V}(k)$
$+\alpha e^{-ck}\Delta\hat{h}(k)^{(i)}$
Filter
$\Delta\hat{h}(k)$
Transform to real-space potential at each iteration *i*.
Real-space potential *i*
$V(r|\hat{V}(k)^{(i)}, r_c)$
k
Response *i*
$\Delta\hat{h}^{(i)}(k)$
Iterations
Self-Assembly Simulation at iteration *i*
$V(r)$ [ϵ]
r [σ]
$\hat{V}(k)^{(i+1)} = \hat{V}(k)^{(i)} + \alpha e^{-ck}\Delta\hat{h}(k)^{(i)}$
Exit when converged.

Figure 2.1: To generate an isotropic pair potential (IPP) for the self-assembly of complex structures, the radial distribution function (RDF) is measured from a thermalized ideal crystal, from which we generate a smooth guess function in k-space. This guess function is then iteratively updated by transforming the potential into real space at each iteration, executing a self-assembly simulation, measuring the response, and then updating the potential accordingly in Fourier space. The initial guess, as well as all updates are smoothened *via* a low-pass filter (shown in red) in order to ensure that the optimization is biased toward smoother potentials that carry only those length scales that are crucial for the assembly of the target structure.

that produce a thermodynamic target structure does not guarantee the existence of a robust kinetic pathway to that structure.

Although directing self-assembly processes with highly specific interactions is technically possible [8–10], it is often more informative to know what is the *simplest* interaction needed to achieve a specific structure *via* facile and robust self-assembly [11], that is, on short time scales and without the need for seeding the target crystal. This so-called simplest interaction will not only provide insight into the underlying mechanisms of self-assembly, but may also be easier to realize experimentally and produce higher yields. Simple interactions with features whose length scales are on the order of the interparticle distances are experimentally realizable through, for example, DNA-mediated surface functionalization of nanoparticles [12–19].

In this work we optimize isotropic pair potentials (IPPs) as a model for the interaction between point particles that self-assemble into a specific target crystal structure from a fluid (disordered) state. That is, we seek pair potentials that not only have shapes containing minimal features, but also which drive assembly of the target structure rapidly, without need for a seed and without long waiting times for nucleation. It was previously shown that Fourier space filters provide an elegant way to *design* simple IPPs for the self-assembly of complex structures [20, 21]. Here we apply this knowledge to advance the relative entropy minimization (REM) approach outlined by Lindquist et al. [22] to be carried out directly in Fourier space and with the repeated application of a smooth low-pass filter at each iteration in order to effectively steer the optimization process towards simpler solutions. The proposed Fourier-filtered relative entropy minimization (FF-REM) method (Fig. 2.1) is designed to optimize for potentials with relatively few minima and maxima while suppressing noisy fluctuations on length scales smaller than those features.

Standard methods for the derivation of IPPs for *fluids* [23–26], many of which fall under the general umbrella of the relative entropy minimization framework [27, 28], cannot be readily used for solids, because the radial distribution functions (RDFs) of solids have many more characteristic length scales compared to their fluid counterparts. Applying standard

REM or Iterative Boltzmann Inversion (IBI) directly to solids results in potentials that tend to be overfitted. That means in this context that they contain too many features and length scales that are not actually critical and are possibly even detrimental for the robust self-assembly of the target structure. Since we know that complex structures may be assembled from much simpler potential functions [29], an efficient optimization algorithm needs to be biased towards those length scales that are essential for robust self-assembly.

One approach to steer the optimization of potentials towards simpler solutions is to apply constraints, *e.g.*, by limiting the solution space to a specific functional form [30]. Overfitting may also be prevented with early stopping for more broadly constrained search spaces, for example when the solutions are limited to a specific class of functions, such as repulsive, monotonically decreasing functions [22, 31, 32]. The FF-REM method does not rely on such constraints, but instead steers the optimization towards smoother and simpler solutions by the repeated application of a filter function in Fourier space (k-space). This approach is especially advantageous during early exploration, *e.g.*, to determine whether any solution exists at all, or when there is no specific desired functional form. Conversely, the presented method does not allow one to target a specific functional form, even if desired.

2.2 Fourier-filtered Relative Entropy Minimization

For the algorithm's derivation we recognize the potential energy E of a three-dimensional system of interacting point particles in a volume $V = N/\rho$, where ρ denotes the number density, may be expressed as a function of the RDF, $g(r)$, both in real space,

$$\frac{E}{N} = 2\pi\rho \int_0^\infty \mathrm{d}r\, r^2 g(r) V(r), \tag{2.1}$$

and equivalently in reciprocal space,

$$\frac{E}{N} = 2\pi\rho \int_0^\infty \mathrm{d}k\, k^2 \hat{g}(k)\hat{V}(k), \tag{2.2}$$

where $f(r) \mapsto \hat{f}(k)$ is the Fourier transform, defined by

$$\hat{f}(k) = \frac{1}{k}\sqrt{\frac{2}{\pi}} \int_0^\infty \mathrm{d}r\, r f(r) \sin(kr)\,. \tag{2.3}$$

The Fourier transformation is unique and invertible and thus preserves all the information of the real-space potential. However, in practice, in order to meet the complexity constraints introduced above, a real-space potential is strongly limited, especially in its range. This means that traditional optimization techniques—carried out exclusively in real-space—are inherently tying the information exploited for the optimization process to the range of the potential energy function. In other words, a potential optimized with, *e.g.*, IBI is inherently biased to match short-range distance distributions since any long-range information contained in the RDF beyond the real-space potential cut-off is completely discarded. By instead optimizing the pairwise interaction model directly in Fourier space, we introduce no inherent constraint on the potential range and the potential function is only transformed into real space for the sake of carrying out the integration of forces as part of simulating the assembly process using Molecular Dynamics (MD).

For the overall process (shown in Fig. 2.1), we first propose an *ansatz* function $\hat{V}^{(0)}(k)$, which in our case is just the smoothened Fourier transform of the potential of mean force. Then we enter an iterative update process, where at each iteration we map the potential to real space and carry out a MD simulation of point particles. Specifically, we thermalize the system at an elevated temperature of $k_BT = 3.0\varepsilon$ to ensure that it is in a disordered fluid state, and then cool and compress the system over the next 4 million time steps to a final temperature of $k_BT = 1.0\varepsilon$ (see Section 2.5.3 for details). Whether the system assembled the targeted structure or not, we then calculate the shifted RDF $h(r) = g(r) - 1$ and Fourier transform to obtain $\hat{h}(k)$. The update step is then derived from the minimization of relative entropy directly in Fourier space (see Sections 2.5.1.1 to 2.5.1.3 for a more detailed derivation) and is expressed as a function of the difference between $\hat{h}^{(i)}(k)$ at iteration i and $\hat{h}^*(k)$ measured from the target structure

$$\hat{V}^{(i+1)}(k) = \hat{V}^{(i)}(k) + \alpha \mathrm{e}^{-ck} k_BT[\hat{h}^{(i)}(k) - \hat{h}^*(k)], \tag{2.4}$$

where α denotes the effective learning rate, c scales the low-pass filter, and k_BT is the thermal energy of the system. The learning rate α is a unitless dampening factor to stabilize the optimization process; we found values on the order of 0.1 to be small enough to yield stable optimization. The low-pass length scale of the exponential filter is set by c such that features in the real-space potential with wavelengths much smaller than $2\pi c$ are damped while features with much longer wavelengths are preserved (see Section 2.5.1.4 for details).

The studied filter strengths $c = 0.1\sigma$ and $c = 0.2\sigma$ are chosen empirically, such that features on length scales on the order of particle interactions $\mathcal{O}(1)$ are largely preserved, while features on smaller lengths scales are sufficiently suppressed.

IPPs mapped from Fourier space onto real space need to be truncated since they are in principle infinite in range. For this we applied the following cut-off algorithm:

$$r_{\mathrm{cut}} = \min_r \left(r \geq r_{\min} \wedge V(r) \leq \epsilon \wedge V'(r) \leq \epsilon' \right), \tag{2.5}$$

where we chose $r_{\min} \in \{1.6, 2.4\}$, $\epsilon = 0.3$ and $\epsilon' = 5.0$. This means that the potential is cut off at the first extremum beyond $r_{\min}$ that is sufficiently close to zero (see also Section 2.5.1.5). We ensure smoothness at this cut-off by applying the Stoddard-Ford algorithm [33] up to the

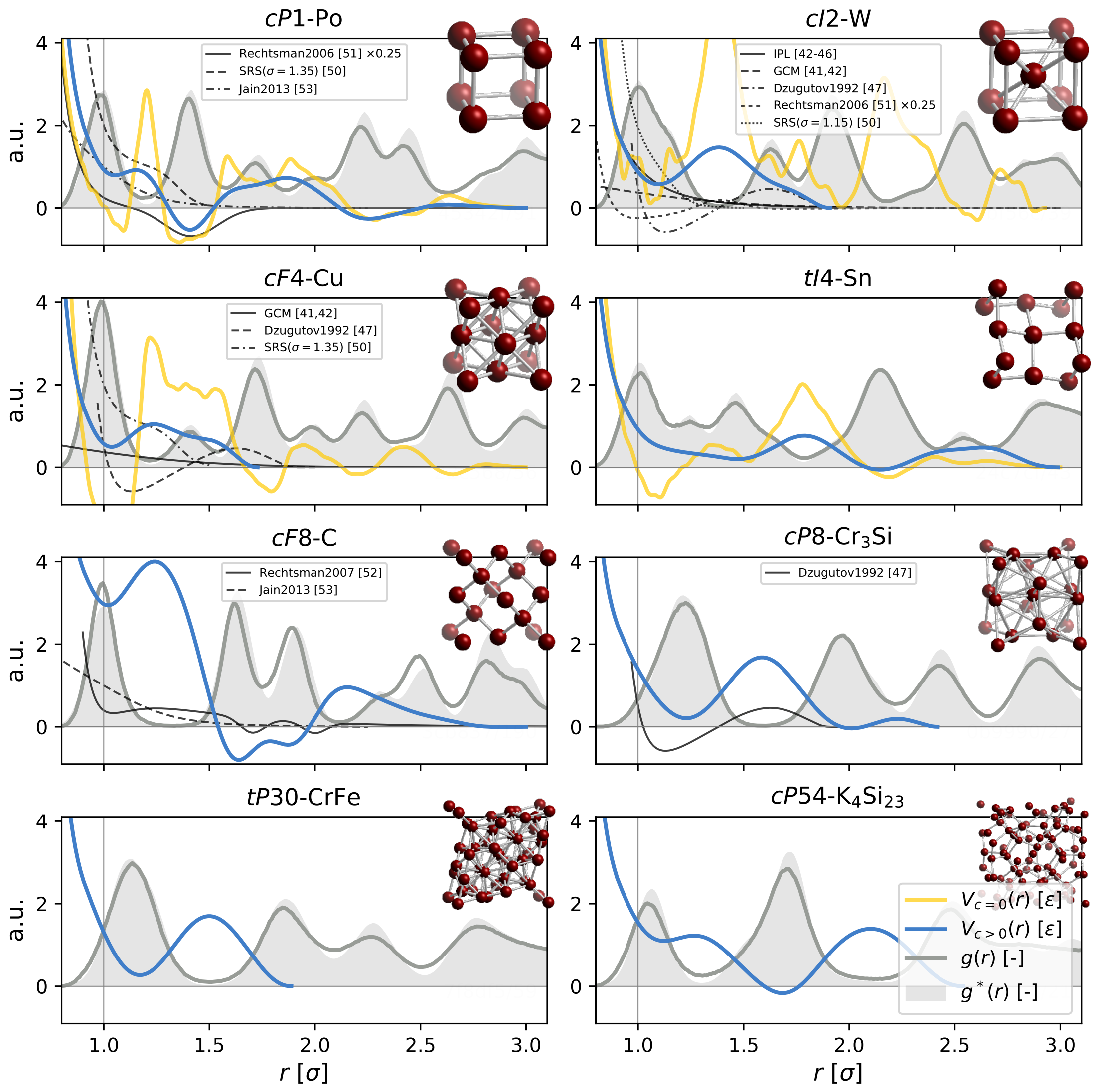


Figure 2.2: Here we show the objectively best IPPs (corresponding to $\varphi_{\max}$, see Eq. 2.7) optimized with FF-REM ($c > 0$) in blue and without filtering ($c = 0$) in yellow. The RDF measured from the assembled structures ($g(r)$, gray lines) are compared against those obtained from the target harmonic crystal ($g^*(r)$, shaded in gray). We found that a perfect fitting of all RDF features is not a critical requirement for the assembly of the target structure. The corresponding unit cells are depicted as ball-and-stick models (top right). While our method consistently produces simpler potentials compared to the control method without filtering, it is not guaranteed that our methodology results in the simplest possible interaction potential for a given target structure. This becomes obvious in comparison with select results from the literature, where some potentials are significantly simpler compared to our results, even though most share general characteristics. The potentials drawn from the literature and plotted here are not adjusted for differences in temperature and density of the assembly state point.

first derivative.

To apply this algorithm to the derivation of IPPs for the assembly of solid structures, we compute the RDF from position distributions of harmonic crystals, where particles are bound to their ideal crystal sites through harmonic bonds. The harmonic bond constant K was chosen such that the peaks within the measured RDF are sufficiently distinct to reliably characterize the structure, usually in a range of $K = [100, 800]$, but always low enough to avoid singularities.

All molecular dynamics simulations were carried out with HOOMD-blue [34, 35] on XSEDE resources [36] (including the Comet and Bridges clusters) and on the high-performance compute cluster of the University of Michigan. The computational workflow in general and data management in particular for this work was primarily supported by the signac data management framework [37].

Simulation trajectories were analyzed with the software package Freud [38] and visualizations were rendered with Fresnel [39]. Structures were analyzed and identified with the in house software Injavis. We trained a machine-learning model based on a deep neural network with spherical harmonic descriptors of particle environments to identify crystal structures from millions of simulation snapshots [40] (Section 2.5.2).

To benchmark the performance of FF-REM, we also attempted a control optimization using standard REM, which is equivalent to no filtering ($c = 0$). IPPs optimized using REM without any kind of filtering failed to self-assemble the target structure in about 70% of all cases.

2.3 Isotropic Pair Potentials for Complex Structures

Design and optimization of IPPs for simple and complex structures has yielded a plethora of different models ranging from repulsive to attractive, from short-ranged to long-ranged, from simple to complex. We have selected a few exemplary models to compare our results to, including the Gaussian core model (GCM) [41, 42], the inverse-power-law potential (IPL) [42–

46], the Dzugutov potential [47–49], the soft-repulsive-shoulder potential (SRS) [50], and potentials published by Rechtsman *et al.*[51, 52] and Jain *et al.*[53], that have been shown to assemble some of the structures we targeted as part of this study.

Using FF-REM we found IPPs for the assembly of simple cubic ($cP1$), body-centered cubic ($cI2$), face-centered cubic ($cF4$), β-tin $tI4$-Sn ($tI4$), A15-type $cP8$-Cr_3Si ($cP8$), diamond ($cF8$), clathrate-I $cP54$-K_4Si_{23} ($cP54$), and σ-phase $tP30$-CrFe ($tP30$) structures. The corresponding IPPs are plotted in Fig. 2.2. Without filtering, *i.e.*, $c = 0$, we were only able to optimize potentials for $cP1$, $cI2$, $cF4$, and $tI4$.

Attempts to find potentials for $cP4$-Li ($cP4$), β-manganese $cP20$-Mn ($cP20$), and γ-brass $cI52$-Cu_5Zn_8 ($cI52$) were not successful with parameters tested for this study, that means they did not assemble the target structure after a fixed number of time steps. This does not rule out the possibility of the obtained potentials to self-assemble the target structure using alternative protocols, *e.g.*, by starting from a seeded configuration or simply sampling longer to overcome potential nucleation barriers. This is evidenced by the fact that potentials that will self-assemble the targeted structures are known for all tested structures, including $cI52$[54] and $cP20$[55], and because the assembly yield is equal or greater for all potentials when the system is doped with a crystalline seed (Fig. 2.16). Within the realm of this study, we only report those potentials that assemble the target structure with the tested protocol, others were considered unsuccessful and consequently disregarded.

To quantify the effectiveness of our filtering, we introduced a measurement of complexity, Ω, defined as

$$\Omega \equiv \frac{1}{k_{\max}} \int_{k=0}^{k_{\max}} \mathrm{d}k \left[k\hat{V}(k) \right]^2 . \tag{2.6}$$

Ω is nonnegative and becomes large when the potential has small-scale real-space features (Section 2.5.5).

Table 2.1: The complexity Ω as defined by Eq. 2.6, measured for different structures and filter strengths c (Eq. 2.4).

Crystal Structure	$c = 0$	$c = 0.1\sigma$	$c = 0.2\sigma$	**Mean**
$cP1$-Po	1.04	0.22	0.18	0.48
$cI2$-W	0.46	0.08	0.06	0.20
$cF4$-Cu	0.99	0.06	0.04	0.37
$tI4$-Sn	0.71	0.07	0.04	0.27
$cF8$-C	-	-	0.70	0.70
$cP8$-Cr_3Si	-	-	0.09	0.09
$tP30$-CrFe	-	-	0.06	0.06
$cP54$-K_4Si_{23}	-	-	0.07	0.07

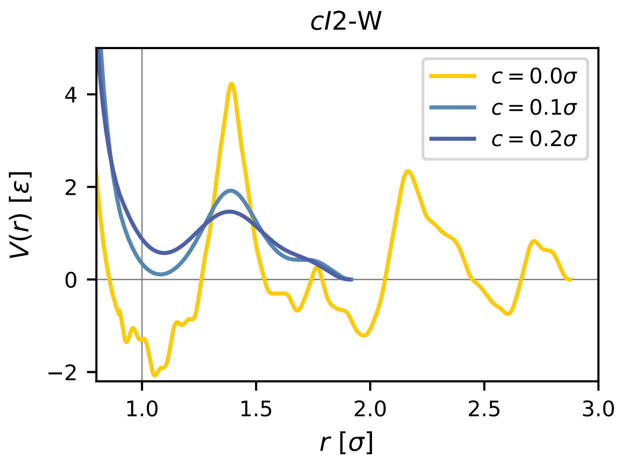


Figure 2.3: We can use the filter strength c to effectively control the complexity of our solution in k-space (Eq. 2.4), resulting in smoother potentials with fewer features on smaller length scales. Important features such as the location of extrema and their relative well-depth are preserved. The control optimization with $c = 0$ is obviously much more complex.

The effectiveness of the low-pass filter becomes obvious when comparing Ω between optimization procedures with different filter strengths c, see Tab. 2.1. Optimization runs with the higher c value of 0.2σ consistently yielded solutions with lower complexity.

This effect can be visualized when comparing solutions mapped onto real space for the identical target structure, but carried out with different filter strengths. Fig. 2.3 shows solutions for $cI2$, optimized with c ranging from 0 to 0.2σ. The solution for $c = 0$ is clearly much more complex compared to all other solutions, but the main characteristics of the potential functions with $c > 0$ are conserved. The solution with $c = 0.1\sigma$ clearly contains

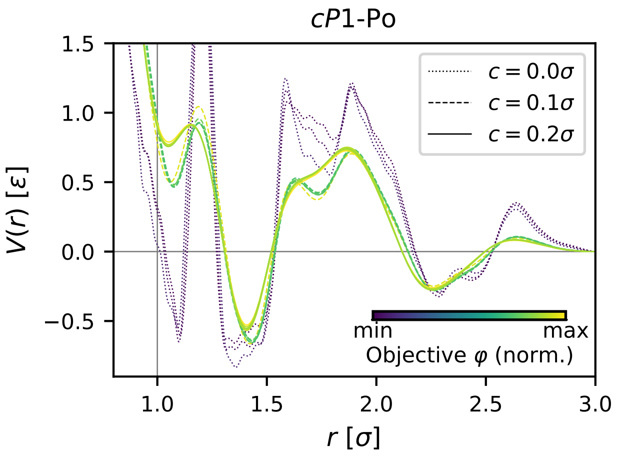


Figure 2.4: All optimized potentials were evaluated by the objective function φ (defined by Eq. 2.7 and visualized by color), which rewards a better fit with the target RDF and penalizes complexity (Eq. 2.6) and long-ranged potentials (Eq. 2.5). Plotted are potentials that were found as solutions for the self-assembly of the $cP1$ structure for a given set of optimization parameters and multiple replications. The potentials have about the same shape, but the complexity is sharply clustered with respect to the filter strength c.

additional non-critical features on smaller length scales compared to solutions obtained with $c = 0.2\sigma$.

2.4 Method Evaluation

To quantify the objective of minimizing the difference in the RDF measured from the thermalized ideal crystal and the distribution measured from the self-assembly result, with the additional constraint of minimizing complexity and potential range, we define the objective function,

$$\varphi^{(i)} = \frac{f^{(i)}}{\Omega^{(i)} \cdot r_{\text{cut}}^{(i)}}, \tag{2.7}$$

where

$$f^{(i)} = 1.0 - \frac{\sum_j |g^{(i)}(r_j) - g^*(r_j)|}{\sum_j [g^{(i)}(r_j) - g^*(r_j)]}. \tag{2.8}$$

The functions $g^{(i)}(r_j)$ and $g^*(r_j)$ denote the discrete RDF measured at iteration i and from the target structure, respectively, therefore f is a measure of how close the RDF matches the target distribution at iteration i. This means that the objective function naturally increases as the RDF matches better, but is reduced by increased complexity (Eq. 2.6) and the range of the potential evaluated in real space (Eq. 2.5). The objective function is used to rank different IPP solutions for the same structure as shown in Fig. 2.4 and Fig. 2.12.

For the structures that we were able to find a solution for, the Fourier space filtering not only results in a reduction of IPP complexity, but also generally yields an overall higher fitness (see Fig. 2.8). We found it both surprising and reassuring that the least complex solutions result in an overall better fitting of the target function.

In general, the fitness of the resulting RDFs is not a good indicator for successful assembly. In fact, when evaluated for the complete data set, the fitness is only weakly correlated with the yield. In other words, most optimization runs returned potentials that reliably reproduced the RDF, up to a specific precision, but from that alone we cannot discern that this potential assembles the target structure, or any ordered structure at all. However, for those potentials that did assemble the target structure, we can use the fitness as a quantitative measure of how well the solution matches the target distribution, which in turn allows us to rank multiple successful solutions.

While we were able to determine IPPs for many different structures, in many cases the optimization resulted in potentials that either failed to assemble any structure, *i.e.*, they formed some kind of fluid, or assembled highly defective structures that might or might not resemble the target structure. In only very few cases did the optimization result in a structure different from the desired structure. For example, none of the attempts to optimize an IPP for $cI52$ were successful, but some of them yielded $cI2$ instead. While the optimization did not succeed in this case, it at least yielded a closely related state: $cI52$ represents a $(3 \times 3 \times 3)$-fold superstructure of $cI2$. Similarly, some of the $cP20$ optimization runs yielded $cI2$ as well. A potential energy analysis (Fig. 2.17) shows that the FF-REM algorithm is able to determine a potential for which the targeted structure presents the ground state among the competitor pool in all cases except for $cI52$, where $cI2$ has a lower potential energy. We therefore presume that the failure to find a IPP for a targeted structure is related primarily to the assembly protocol in all but this case.

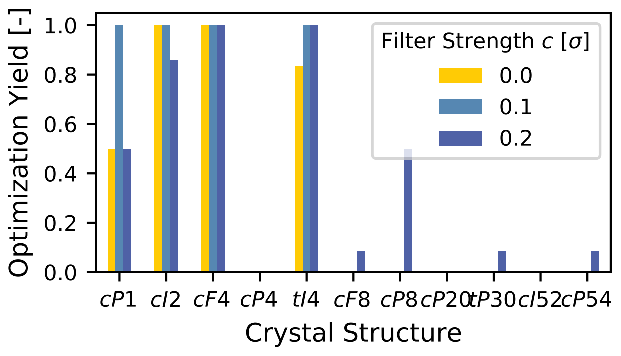
We analyzed the optimization performance with respect to the optimization parameters and with respect to the target structures. In particular we are interested in determining which parameters yield the best results and whether we can discern for which structures it is inherently harder to optimize potentials with the presented methods based on specific structural characteristics.

We evaluated the robustness of a specific parameter and structure combination by dividing the number of times they resulted in the successful optimization of a potential in at least one iteration with the total number of attempts for that combination. All combinations were replicated independently three times with a different random seed.

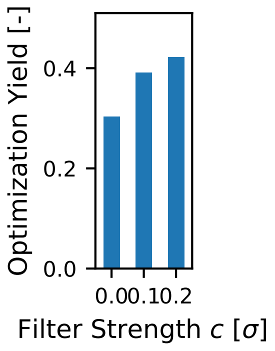
We found that the overall yield of FF-REM (41%) for finite filter strengths ($c > 0$) is higher compared to 30% with no filtering ($c = 0$). The yield among replication groups, that means all attempts with identical optimization parameters except for the random seed, where at least one attempt led to successful optimization, is much closer (87% (filtered) versus 95% (non-filtered)).

For the three tested filter strengths c, we find that FF-REM performs better with higher values of c, see Fig. 2.5b. This finding is surprising to us as we expected that increased smoothing would make it more difficult to assemble target structures.

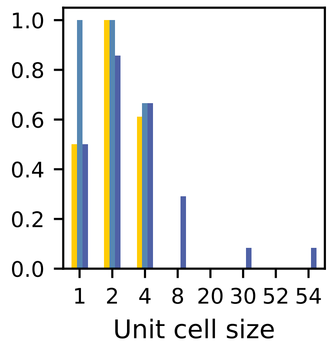
Secondly, we investigated how the optimization performs with respect to the target structure. We found that the yield for individual structures is usually higher with FF-REM



(a) Optimization yield by structure



(b) Optimization yield by filter strength C



(c) Optimization yield by unit cell size

Figure 2.5: The yield shown here is the number of successful attempts, *i.e.*, a structure and optimization parameter combination, where at least one iteration led to an IPP that would assemble the target structure, divided by the total number of attempts.

(Fig. 2.5a). That means we were able to find more structures more robustly with FF-REM in the vast majority of studied cases.

It appears that for the tested structures the yield is negatively correlated with the unit cell size (see Fig. 2.5c). However, we would need to test more structures to determine whether this is inherent to the optimization algorithm, or whether, *e.g.*, it is because structures with larger unit cells are generally more difficult to assemble with the used protocols and without seeds. Furthermore, the yield increases with the average coordination number of the first neighbor shell (see Fig. 2.13). With the exception of the extraordinarily robust optimization of potentials for the $cP1$ and $tI4$ structures, which both have an average coordination number

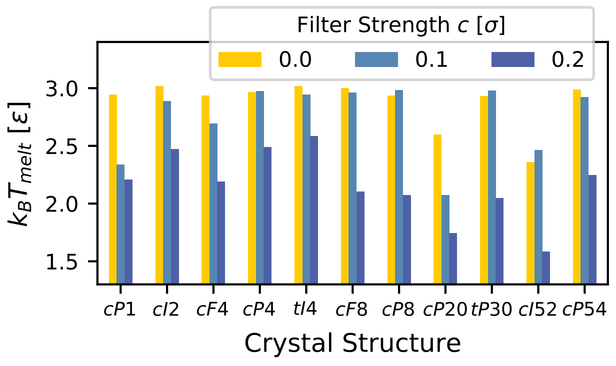


Figure 2.6: The melting temperature k_BT_{melt} is the temperature at which the target structure disintegrates, *i.e.*, is no longer stable with particles at the lattice coordinates interacting *via* the optimized IPP. The melting temperature was determined by initializing the system in the target structural configuration and then slowly raising the temperature from $k_BT = 1.0$ up to $k_BT = 3.0$ (Fig. 2.18).

of 6, there appears to be a a positive relationship between average coordination number and the yield. Specifically, it appears to be generally more difficult to optimize IPPs for lower-coordinated structures, such as $cP4$ or $cF8$ (see Fig. 2.5a).

To assess the optimization robustness for all runs, even for those parameter combinations where the target structure could not be assembled, we determined the melting temperature k_BT_{melt} for the best potential, *i.e.*, the potential with the objectively highest value ($\varphi_{\max}$) (see Fig. 2.6). The melting temperature is defined as the temperature at which the target structure comprised of particles interacting *via* the corresponding potential would start to disintegrate. We initialized simulation configurations with the target structure, and then slowly increased the temperature while evaluating the Lindemann criterion [56, 57]:

$$L = \left(\overline{(\boldsymbol{r}_i - \boldsymbol{r}_0)}\right)^{1/2}. \tag{2.9}$$

The melting temperature was then determined to be exactly the temperature at which L and $\frac{\delta L}{\delta t}$ were above a specific but universal threshold (see Section 2.5.9 for more information).

We found that the mean melting temperature is slightly higher with lower filter strengths

c. The mean melting temperature for individual structures was often comparable or even higher with no filter ($c = 0$), even when the latter did not result in a potential that would assemble the target structure.

In conclusion, we demonstrated that by taking advantage of the unique properties of Fourier space, we are able to implement a simple but effective optimization algorithm resulting in smooth IPPs for the self-assembly of complex structures. Not only is FF-REM more robust and demonstrated an overall higher yield compared to the control optimization with no filter, the resulting smoother potentials lead to a significantly higher yield when it comes to the self-assembly of complex crystal structures.

2.5 Supplemental Material

2.5.1 Background

2.5.1.1 The Potential Energy in Real and Fourier Space

We can describe the total potential energy E of a many-particle system as the sum of a pairwise isotropic potential function $V(r)$ over all particle coordinates $\boldsymbol{r}$

$$E = \sum_{i<j} V(|\boldsymbol{r}_i - \boldsymbol{r}_j|), \tag{2.10}$$

where $|\dots|$ denotes the Euclidean norm.

In 3 dimensions the total potential energy may be expressed as a function of the RDF

$$\frac{E}{N} = 2\pi\rho \int_0^\infty \mathrm{d}r\, r^2 g(r) V(r), \tag{2.11}$$

and equivalently in reciprocal space

$$\frac{E}{N} = 2\pi\rho \int_0^\infty \mathrm{d}k\, k^2 \hat{g}(k)\hat{V}(k), \tag{2.12}$$

where $f(r) \mapsto \hat{f}(k)$ is the Fourier transform for isotropic functions, defined by

$$\hat{f}(k) = \frac{1}{(2\pi)^{3/2}} \int \mathrm{d}^3\boldsymbol{r} f(\boldsymbol{r}) \mathrm{e}^{-i\boldsymbol{k}\cdot\boldsymbol{r}} = \frac{1}{k}\sqrt{\frac{2}{\pi}} \int_0^\infty \mathrm{d}r\, r f(r) \sin(kr)\,. \tag{2.13}$$

This means that we can use $V(r)$ and $\hat{V}(k)$ equivalently to evaluate the potential energy of the system.

2.5.1.2 Relative Entropy Minimization in Fourier Space

We largely follow the approach outlined by Lindquist *et al.* [22] for the REM, to derive the optimization algorithm for the FF-REM method. The probability p for a specific microstate $\boldsymbol{R}_i$ within the canonical ensemble (NVT) is defined as

$$\begin{aligned} p(\boldsymbol{R}_i) &= \frac{\mathrm{e}^{-\beta E(\boldsymbol{R}_i|\boldsymbol{\theta})}}{\sum_i \mathrm{e}^{-\beta E(\boldsymbol{R}_i|\boldsymbol{\theta})}} && (2.14) \\ &\equiv \frac{1}{Z(\boldsymbol{\theta})}\mathrm{e}^{-\beta E(\boldsymbol{R}_i|\boldsymbol{\theta})}, && (2.15) \end{aligned}$$

where $Z(\boldsymbol{\theta})$ denotes the partition function. We then define the joint probability

$$P(\boldsymbol{\Omega}|\boldsymbol{\theta}) = \prod_i^M \frac{1}{Z(\boldsymbol{\theta})}\mathrm{e}^{-\beta E(\boldsymbol{R}_i|\boldsymbol{\theta})} \tag{2.16}$$

where $\boldsymbol{\Omega}$ denotes the set of all M microstates sampled from the target distribution, which we want to maximize to find the optimal parameter set

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{\Omega}|\boldsymbol{\theta}). \tag{2.17}$$

Assuming that $P > 0$, this is equivalent to

$$\begin{aligned} \boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \ln P(\boldsymbol{\Omega}|\boldsymbol{\theta}) && (2.18) \\ &\equiv \arg\max_{\boldsymbol{\theta}} L(\boldsymbol{\Omega}|\boldsymbol{\theta}), && (2.19) \end{aligned}$$

where we denote the logarithm of the probability with L. We apply an iterative ascent algorithm to maximize L

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} + \bar{\alpha}\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\Omega}|\boldsymbol{\theta}), \tag{2.20}$$

where $\bar{\alpha}$ denotes the learning rate, an empirical constant that controls the size of each update step. To evaluate $\nabla_{\boldsymbol{\theta}} L$, we first expand L:

$$\begin{aligned} L &= \ln \prod_i^M \frac{1}{Z(\boldsymbol{\theta})} \mathrm{e}^{-\beta E(\boldsymbol{R}_i|\boldsymbol{\theta})} && (2.21) \\ &= \sum_i^M \left[-\beta E(\boldsymbol{R}_i|\boldsymbol{\theta}) - \ln Z(\boldsymbol{\theta})\right] && (2.22) \\ &= -\beta \sum_i^M E(\boldsymbol{R}_i|\boldsymbol{\theta}) - M \ln Z(\boldsymbol{\theta}). && (2.23) \end{aligned}$$

Then we evaluate the derivative for all terms individually:

$$\nabla_{\boldsymbol{\theta}}\left(\frac{-L}{M}\right)=\beta\frac{1}{M}\sum_{i}^{M}\nabla_{\boldsymbol{\theta}}(E(\boldsymbol{R}_i|\boldsymbol{\theta}))+\nabla_{\boldsymbol{\theta}}\ln Z(\boldsymbol{\theta}). \tag{2.24}$$

The derivative of the potential energy can be expressed in terms of $\hat{g}(k)$ by applying Eq. 2.12:

$$\nabla_{\boldsymbol{\theta}}E(\boldsymbol{R}_i|\boldsymbol{\theta})=2\pi\rho N\int_0^\infty \mathrm{d}k\, k^2\hat{g}_i(k)\nabla_{\boldsymbol{\theta}}\hat{V}(\boldsymbol{\theta}). \tag{2.25}$$

The first term of Eq. 2.24 is therefore:

$$\begin{aligned}\frac{1}{M}\sum_{i}^{M}\nabla_{\boldsymbol{\theta}}(E(\boldsymbol{R}_i|\boldsymbol{\theta}))&=\frac{1}{M}\sum_{i}^{M}2\pi\rho N\int_0^\infty \mathrm{d}k\, k^2\hat{g}_i(k)\nabla_{\boldsymbol{\theta}}\hat{V}(\boldsymbol{\theta}) &(2.26)\\ &=2\pi\rho N\int_0^\infty \mathrm{d}k\, k^2\overline{\hat{g}(k)}\nabla_{\boldsymbol{\theta}}\hat{V}(\boldsymbol{\theta}) &(2.27)\\ &\overset{*}{=}2\pi\rho N\int_0^\infty \mathrm{d}k\, k^2\left\langle\hat{g}(k)\right\rangle_*\nabla_{\boldsymbol{\theta}}\hat{V}(\boldsymbol{\theta}), &(2.28)\end{aligned}$$

where we assume for the last transformation $(*)$ that the target sample distribution $\boldsymbol{\Omega}$ is ergodic. Similarly, the second term can be evaluated to:

$$\nabla_{\boldsymbol{\theta}} \ln Z(\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} Z(\boldsymbol{\theta}) \tag{2.29}$$

$$= \frac{1}{Z(\boldsymbol{\theta})} \sum_{\boldsymbol{\Omega}} \nabla_{\boldsymbol{\theta}} \mathrm{e}^{-\beta E(\boldsymbol{R}_i|\boldsymbol{\theta})} \tag{2.30}$$

$$= -\beta \sum_{\boldsymbol{\Omega}} \frac{1}{Z(\boldsymbol{\theta})} \mathrm{e}^{-\beta E(\boldsymbol{R}_i|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} E(\boldsymbol{R}_i|\boldsymbol{\theta}) \tag{2.31}$$

$$= -\beta \sum_{\boldsymbol{\Omega}} \frac{1}{Z(\boldsymbol{\theta})} \mathrm{e}^{-\beta E(\boldsymbol{R}_i|\boldsymbol{\theta})} 2\pi\rho N \int_0^\infty \mathrm{d}k\, k^2 \hat{g}_i(k) \nabla_{\boldsymbol{\theta}} \hat{V}(k|\boldsymbol{\theta}) \tag{2.32}$$

$$= -2\pi\rho N\beta \int_0^\infty \mathrm{d}k k^2 \left[\sum_{\boldsymbol{\Omega}} \frac{1}{Z(\boldsymbol{\theta})} \mathrm{e}^{-\beta E(\boldsymbol{R}_i|\boldsymbol{\theta})} \hat{g}_i(k) \right] \nabla_{\boldsymbol{\theta}} \hat{V}(k|\boldsymbol{\theta}) \tag{2.33}$$

$$= -2\pi\rho N\beta \int_0^\infty \mathrm{d}k\, k^2 \left\langle \hat{g}(k) \right\rangle_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \hat{V}(k|\boldsymbol{\theta}). \tag{2.34}$$

Since both terms only vary with respect to the ensemble average, we can combine them to:

$$\nabla_{\boldsymbol{\theta}} L \propto \int_0^\infty \mathrm{d}k\, k^2 \left[\left\langle \hat{h}(k) \right\rangle_{\boldsymbol{\theta}} - \left\langle \hat{h}(k) \right\rangle_{*} \right] \nabla_{\boldsymbol{\theta}} \hat{V}(k|\boldsymbol{\theta}). \tag{2.35}$$

Here we use $h(r) \equiv g(r) - 1$ to avoid explicitly computing the Fourier transform $\hat{g}(k)$ which has a singularity at $k = 0$.

Assuming that $\hat{V}(k|\boldsymbol{\theta})$ is a tabulated potential, where $\boldsymbol{\theta} = [\epsilon_0, \epsilon_1, ..., \epsilon_P]^T$, then the m^{th} component of $\nabla_{\boldsymbol{\theta}} L$ can be expressed as

$$\frac{\mathrm{d}}{\mathrm{d}\epsilon_m} L \propto \sum_{n=0}^{P} \int_{k_{n,\min}}^{k_{n,\max}} \mathrm{d}k\, k^2 \left[\left\langle \hat{h}(k) \right\rangle_{\boldsymbol{\theta}} - \left\langle \hat{h}(k) \right\rangle_{*} \right] \frac{\mathrm{d}}{\mathrm{d}\epsilon_m} \epsilon_n, \tag{2.36}$$

where the integral—over all k values—has been split up into multiple integrals, each over all k values corresponding to a specific ϵ. Since all terms in the sum must be zero except where

$m = n$, our update formula for individual components reduces to:

$$\epsilon_m^{(i+1)} = \epsilon_m^{(i)} + \tilde{\alpha} \int\limits_{k_{m,\min}}^{k_{m,\max}} \mathrm{d}k\, k^2 \left[\left\langle \hat{h}(k) \right\rangle_{\boldsymbol{\theta}} - \left\langle \hat{h}(k) \right\rangle_{*} \right], \tag{2.37}$$

where we have combined $\bar{\alpha}$ and all other proportionality constants into $\tilde{\alpha}$. Choosing an appropriate discretization of $\hat{h}(k)$, this is approximately equivalent to

$$\epsilon_m^{(i+1)} \approx \epsilon_m^{(i)} + \tilde{\alpha} k_m^2 \Delta k \left[\left\langle \hat{h}(k_m) \right\rangle_{\boldsymbol{\theta}} - \left\langle \hat{h}(k_m) \right\rangle_{*} \right]^{(i)}. \tag{2.38}$$

To bias the optimization algorithm towards smoother functions, we apply a filter operator function $\hat{\xi}_c f(k) = k^{-2} \mathrm{e}^{-ck} f(k)$ to the update step:

$$\epsilon_m^{(i+1)} \approx \epsilon_m^{(i)} + \tilde{\alpha} \mathrm{e}^{-ck_m} \Delta k \left[\left\langle \hat{h}(k_m) \right\rangle_{\boldsymbol{\theta}} - \left\langle \hat{h}(k_m) \right\rangle_{*} \right]^{(i)}, \tag{2.39}$$

which is the basis for Eq. (2.4) shown earlier. This transformation biases the gradient ascent path on L in directions that are orthogonal to changes for large k, but preserves the positions of extrema on L.

2.5.1.3 The Ansatz Function

The iterative update scheme outlined in the previous section requires the proposition of an ansatz function $\hat{V}(k|\boldsymbol{\theta}_0)$ as a starting point. For this, we measure the potential of mean force,

$$V_0(r) = -\frac{1}{\beta}\ln g(r), \tag{2.40}$$

which we transform into Fourier space and then screen to amplify values within the low k range:

$$\hat{V}_0(k) = \mathrm{e}^{-ck}\hat{f}(V_0(r)). \tag{2.41}$$

2.5.1.4 The Filter Strength c (Fourier Space)

To determine an appropriate filter strength c to be used with the filter operator function $\hat{\xi}_c$ (Eq. 2.39), we evaluated the Fourier spectrum of the $g(r)$ of some representative crystal structures.

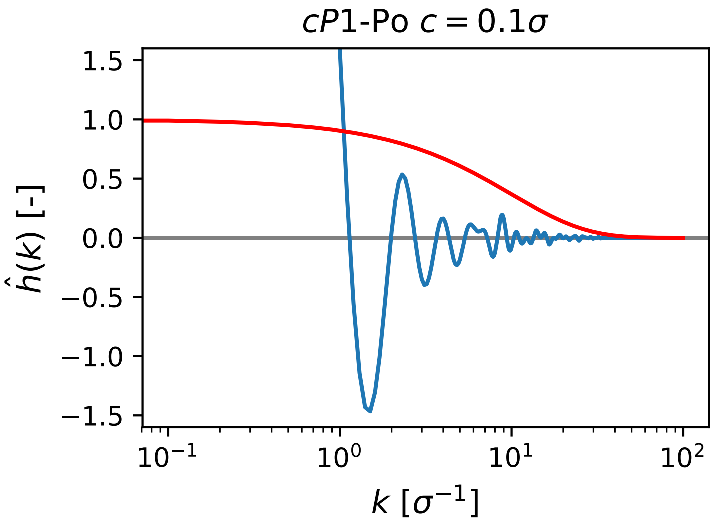


Figure 2.7: Here we have superimposed the function e^{-ck} (red) for $c = 0.1\sigma$ with $\hat{h}(k)$ for $cP1$ (blue). We see that the majority of features within the small-k range are preserved while most features within the high-k range are suppressed.

We found that values within the range $c/\sigma = [0.1, 0.2]$ scale the filter such that the most important features within the spectrum are preserved, but many features within the range of large k values are suppressed (see Fig. 2.7).

The low-pass filtering operation can also be expressed as a convolution in real space. If $V(r)$ is the real-space filtered potential and $V_0(r)$ is the real-space unfiltered potential, the operation in Equation 2.41 is equivalent to

$$V(r) = \frac{1}{\pi} \int_0^\infty \mathrm{d}r' \, V_0(r') \frac{4cr'^2}{(c^2 + (r - r')^2)(c^2 + (r + r')^2)}. \tag{2.42}$$

2.5.1.5 The Real Space Potential

To be able to simulate the assembly process at each iteration i, the Fourier potential $\hat{V}(k|\boldsymbol{\theta}^{(i)})$ was transformed into real space

$$V(r|\boldsymbol{\theta}) = \frac{1}{r}\sqrt{\frac{2}{\pi}} \int_0^\infty \mathrm{d}k \, k\hat{V}(k|\boldsymbol{\theta}) \sin(kr) \tag{2.43}$$

and then truncated according to Eq. (2.5)

After the range was determined, we applied the Stoddard-Ford algorithm [33] up to the first derivative to ensure that all potentials would smoothly go to zero at the truncation.

2.5.2 Crystal Structure Identification

We employed multiple techniques to identify crystal structures from simulation snapshots:

1. By directly inspecting the rendered structures (*i.e.*, their real-space coordinates).
2. By comparing the bond-orientational order diagram (BOD) against those of ideal and thermalized crystals.
3. By comparing the diffraction pattern against those of ideal and thermalized crystals.

4. By identifying the unit cell through symmetry detection and cluster reduction.

5. Through automated classification using a neural network trained on local harmonic descriptors as described by Spellings *et al.* [40].

Tasks 1-4 were performed using the in-house software Injavis. The machine learning model (task 5) was trained on a subset of the data and subsequently used to identify possible candidates for successful assembly.

2.5.3 The Optimization Self-Assembly Protocol

All simulations were executed with HOOMD-blue [34, 35] within the canonical ensemble (NVT) using the Langevin integrator and a time step of $dt = 0.001\tau$. All systems self-assembled for the potential optimization contained at least 2,000 particles and were first thermalized for 1 million steps at a slightly increased box size and an elevated temperature $k_BT_0 = 3.0\varepsilon$. The box was then compressed and the temperature reduced to $k_BT = 1.0\varepsilon$ over the next 1-2 million steps. The system was then thermalized for another 1-2 million steps and the RDF used to determine the potential update was measured from the last 800,000 steps.

The ultimately selected IPPs were verified to assemble the targeted structure in a larger self-assembly simulation with the self-assembly protocol as described above, but with system sizes of 15,625 particles in a cubic box, and 10-times more time steps.

2.5.4 Fitness

The fitness f as defined in Eq. (2.8) is a measure of how closely the discrete RDF $g(r_j)$ at iteration i matches the discrete RDF of the target. The fitness ranges from 0 (complete mismatch) to 1 (perfect match).

The FF-REM method generally generates IPP solutions that have a higher fitness compared to the non-filtered results (see Fig. 2.8 and Fig. 2.9).

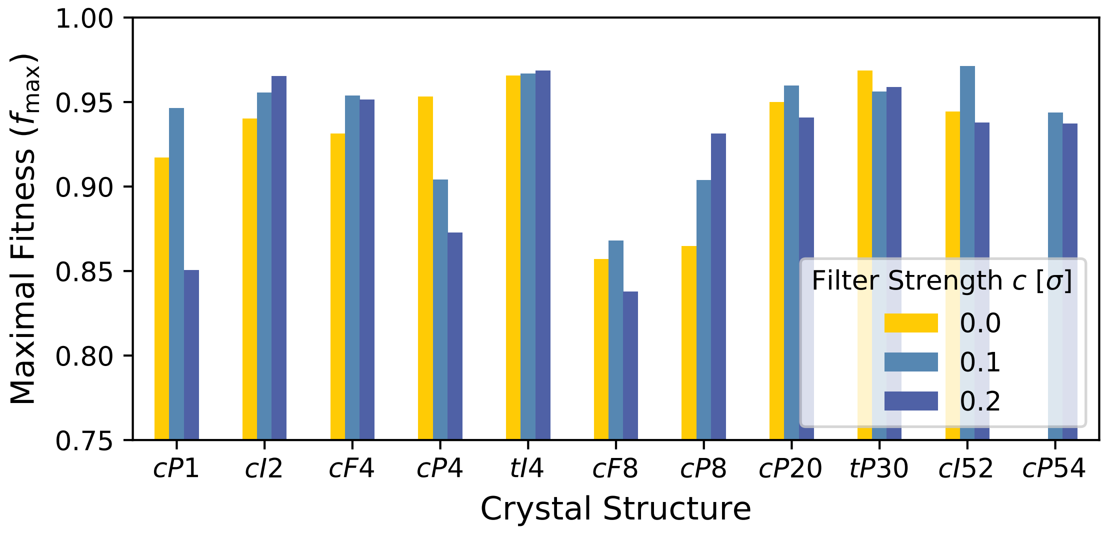


Figure 2.8: We observe that FF-REM outperforms non-filtered optimization with respect to the maximum fitness $f_{\max}$ (Eq. 2.8) averaged over the structure class in the majority of cases. Notable exceptions are $cP4$ and $tP30$, which lead to exceptionally high fitness without any smoothening. The fact that $cP4$ did not actually self-assemble is further evidence that a high fitness is not a good indicator for successful assembly.

2.5.5 Complexity

To objectively measure the complexity of a specific IPP solution, we introduced a complexity norm Ω defined in Eq. (2.6). This nonnegative number correlates with the presence of small-scale features in the real-space potential. We show complexity measures for potentials as a function of structure and filter strength in Fig. 2.10.

2.5.6 Objective Quality

To be able to evaluate and select the *objectively* best IPP solution among multiple possible solutions, we defined the objective function according to Eq. (2.7), which is proportional to the fitness f (Eq. (2.8)) and anti-proportional to the potential's complexity Ω (Eq. (2.6)) and range (Eq. (2.5)). We show a visual comparison of potentials with different quality in Fig. 2.12.

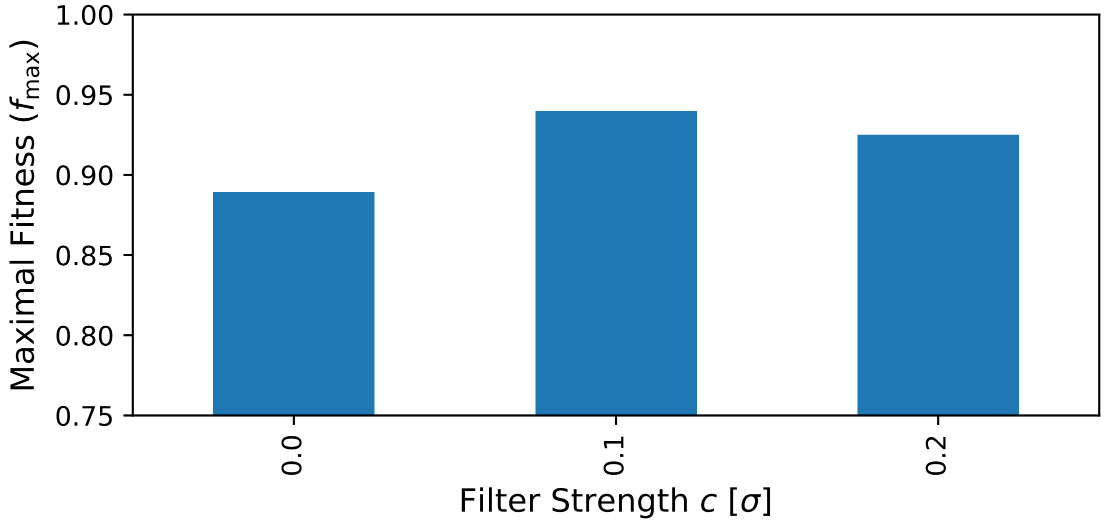


Figure 2.9: The average maximal fitness f_{max} is generally higher for filter strengths $c > 0$.

2.5.7 Yield

The optimization yield is defined as the number of successful attempts, *i.e.*, any optimization run that would yield *at least one* IPP that would assemble the targeted structure, divided by the number of all optimization runs. We evaluated the yield with respect to the optimization parameters and the targeted structure (Fig. 5), as well as selected structure properties (Fig. 2.13).

We performed additional self-assembly simulations with all potentials that successfully assembled the targeted structure during the optimization process in order to validate that they will assemble the targeted structure not only for small system sizes and unit cell commensurate box dimensions, but also for large systems in a cubic box. We made a total of three attempts per potential and only selected those potentials that assembled the targeted structure at least once. The assembly yield is defined as the number of successful attempts to assemble the targeted structure with a given potential in a large cubic system, divided by the total number of attempts.

Finally, we tested how much the nucleation barrier affects the assembly rate and ran additional tests with crystalline seeds in a large ($N > 10,000$ particles), but unit cell

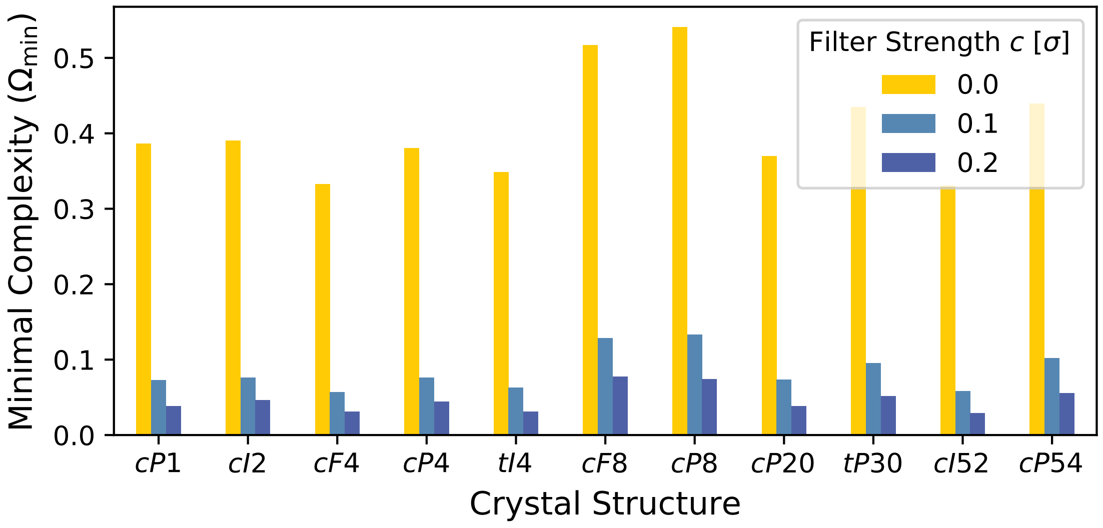


Figure 2.10: The average minimal complexity Ω_{min} (Eq. 2.6) is much higher for all potentials optimized without filter ($c = 0$) compared to those optimized with FF-REM.

commensurate box, where 8 to 20 particles were fixed at the center and compared them to the non-seeded assembly runs. The results show that a crystal seed greatly improves the assembly yield for all tested potentials (see Fig. 2.16).

2.5.8 Ground state energies

To further elucidate reasons for failure, we measured the potential energy of all computed potentials for all targeted ideal structures (Fig. 2.17). We then compared the minimum energy of a targeted crystal structure with the minimal energy from the competitor pool and found that in almost all cases the targeted crystal structure presents the ground state for a given potential within the competitor pool.

A notable exception is $cI52$, where $cI2$ presents the ground state which matches our observations that potentials optimized for $cI52$ sometimes yield $cI2$ instead. Furthermore, the energies for $tP30$ and $cP8$ are very close, which also matches our expectations, since these structures are structurally very similar.

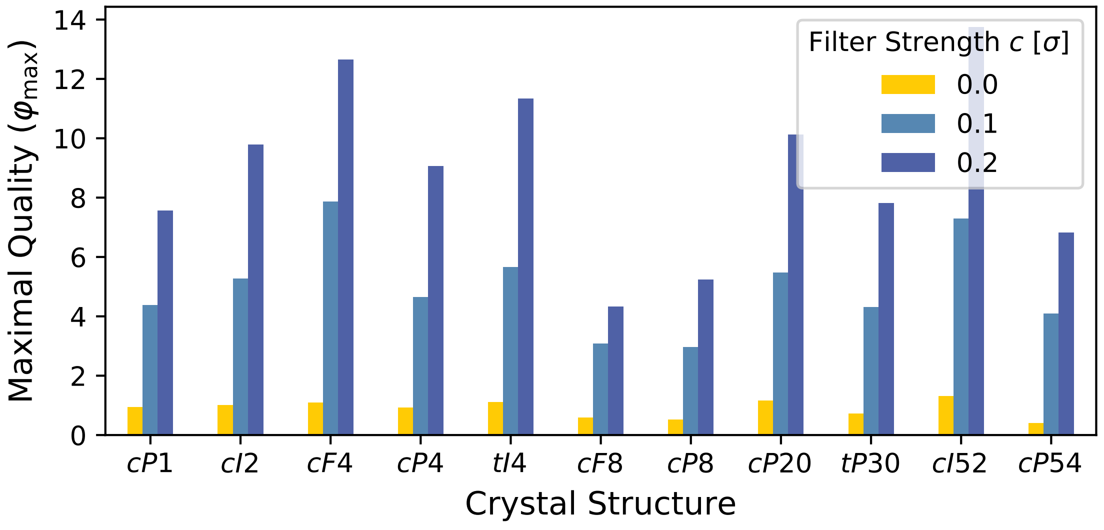


Figure 2.11: The quality φ of potentials is strongly governed by the chosen filter strength c, with a strong positive correlation between filter strength and potential quality.

2.5.9 Melting Temperature

The melting temperature $k_B T_{\text{melt}}$ was measured by initializing the system directly in the target structure and then slowly increasing the temperature from $k_B T = 1.0\varepsilon$ to $k_B T = 3.0\varepsilon$. We then measured the Lindemann order parameter [56, 57] defined as

$$L = \left(\overline{(\boldsymbol{r}_i - \boldsymbol{r}_0)}\right)^{1/2}. \tag{2.44}$$

The melting temperature was determined with

$$k_B T_{\text{melt}} = \arg\min_{kT}(L > L_{\max} \vee (L' > 4\sigma_{L'} \wedge L > L_{\min})), \tag{2.45}$$

where L' denotes the gradient of L, $\sigma_{L'}$ the standard deviation of L', and all parameters were tuned empirically with $L_{\min} = 3.0$ and $L_{\max} = 4.7$. The average measured melting temperatures as a function of filter strength c are plotted in Fig. 2.18.

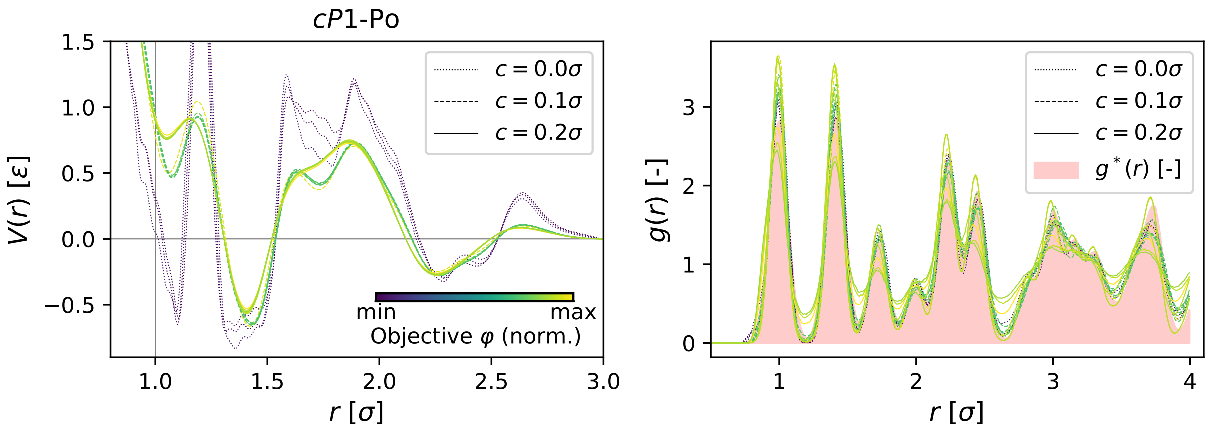


Figure 2.12: Comparison of different IPPs (a) and the resulting RDFs (b) for the $cP1$ structure with respect to the objective function φ (Eq. 2.7), optimized with different filter strengths c.

2.5.10 Crystal Structure Visualization

All structures are rendered with the Fresnel rendering software [39] (Fig. 2.19 to 2.26).

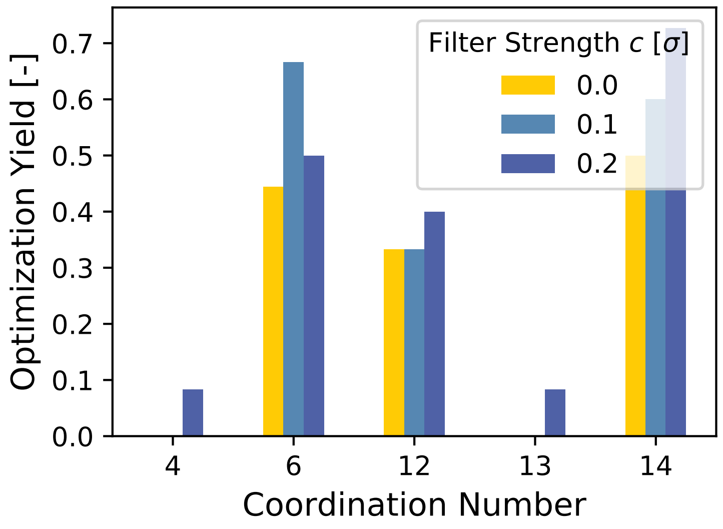


Figure 2.13: There appears to be a marginal trend for a higher yield for structures with a higher average coordination number. *Note: the data is plotted on a non-linear x-axis.*

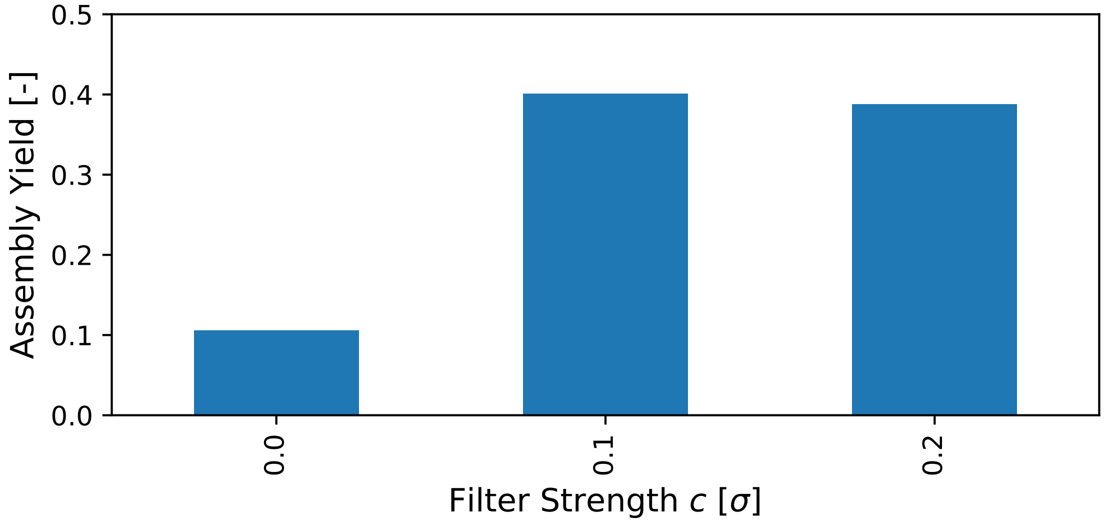


Figure 2.14: The assembly yield is on average much higher for values of $c > 0$.

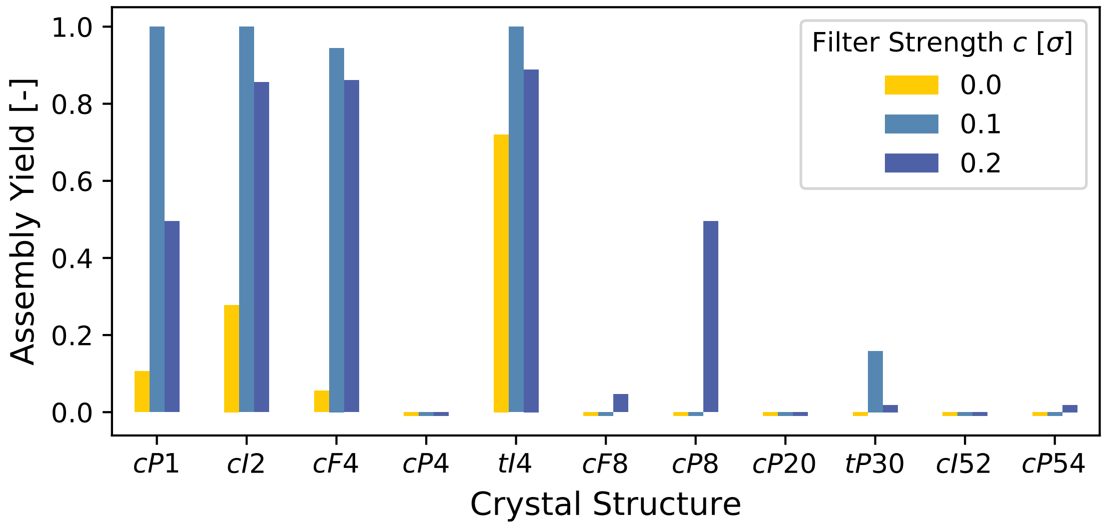


Figure 2.15: Assembly yield by structure.

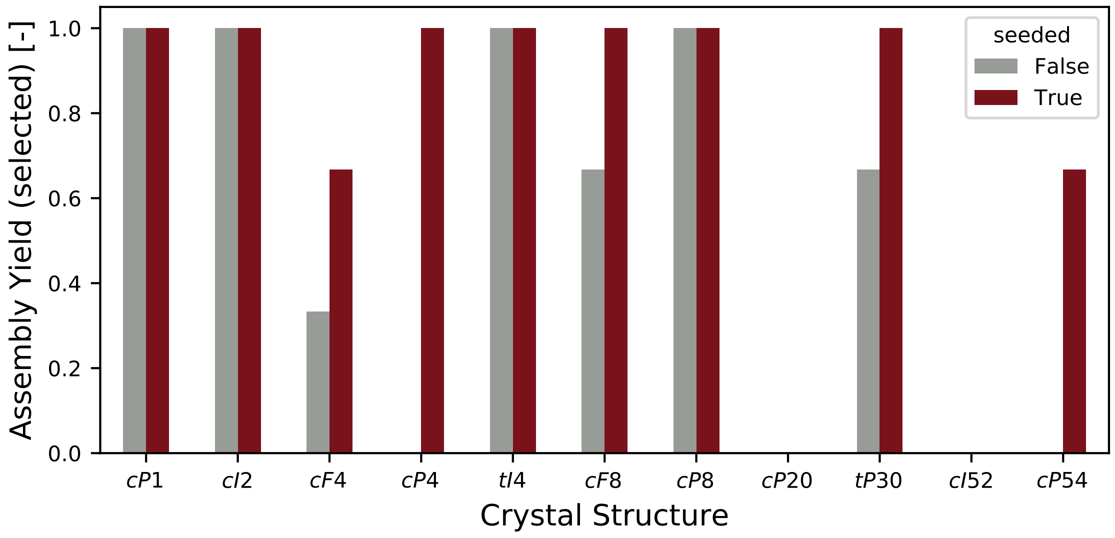


Figure 2.16: The assembly yield by structure for selected potentials ($\varphi_{\max}$, some shown in Fig. 2), is generally improved by providing a crystalline seed at the beginning of the simulation. Notably, the $cP4$ structure assembles *only with seed*, the $cP54$ structure did not assemble without seed in these additional tests.

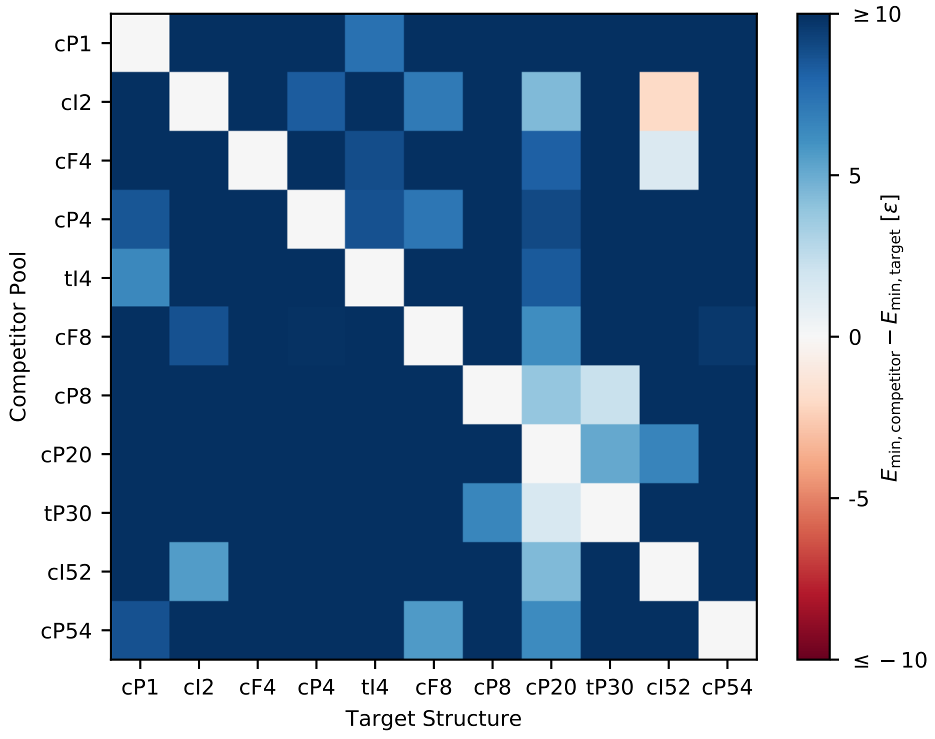


Figure 2.17: We calculated the per-particle potential energy of an ideal crystal for all computed potentials and compared the respective minimum energy for any structure with the respective minimum energy of all other structures. The results clearly show that FF-REM is able to determine at least one potential for each structure, where the targeted structure presents the ground state among the competitor pool with the exception of $cI52$.

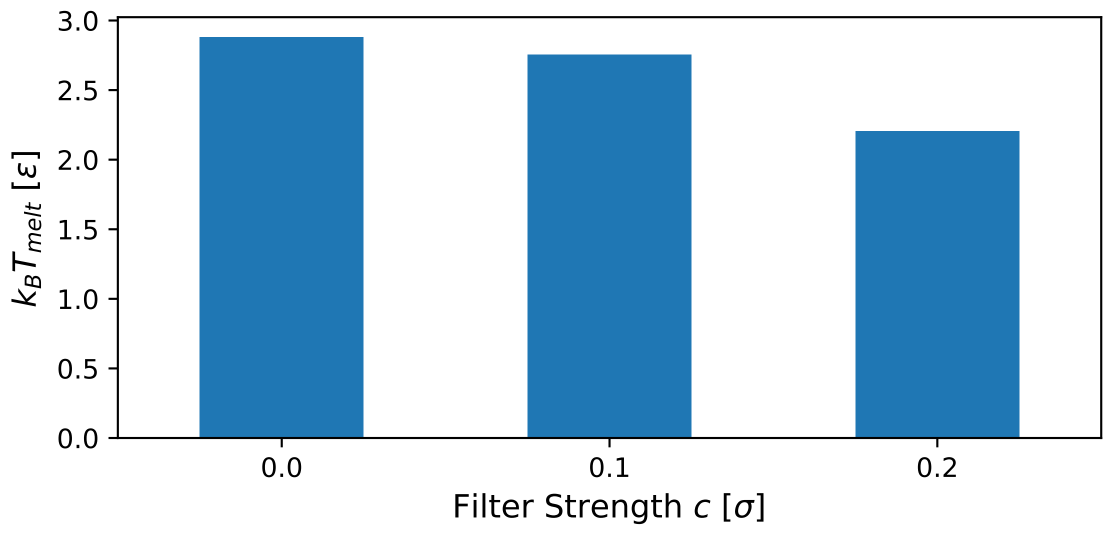


Figure 2.18: The melting temperature $k_B T_{\text{melt}}$ was determined for all structures and parameters, regardless of whether the optimization succeeded, according to Eq. 2.45. The melting temperature was generally higher for no and weak filtering ($c \leq 0.1\sigma$) compared to stronger filtering ($c = 0.2\sigma$).

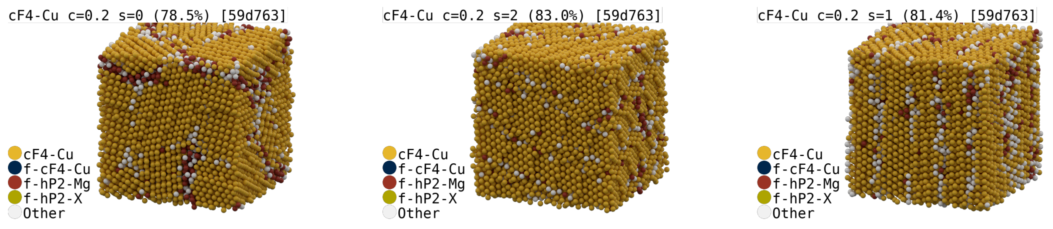

cF4-Cu c=0.2 s=0 (78.5%) [59d763]
cF4-Cu c=0.2 s=2 (83.0%) [59d763]
cF4-Cu c=0.2 s=1 (81.4%) [59d763]
● cF4-Cu
● f-cF4-Cu
● f-hP2-Mg
● f-hP2-X
● Other

Figure 2.19: $cF4$-Cu

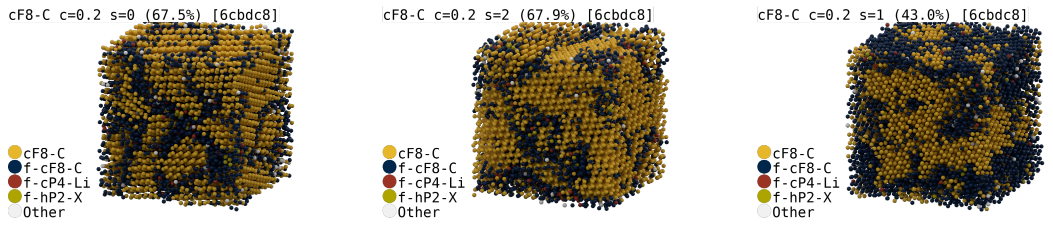

cF8-C c=0.2 s=0 (67.5%) [6cbdc8]
cF8-C c=0.2 s=2 (67.9%) [6cbdc8]
cF8-C c=0.2 s=1 (43.0%) [6cbdc8]
● cF8-C
● f-cF8-C
● f-cP4-Li
● f-hP2-X
● Other

Figure 2.20: $cF8$-C

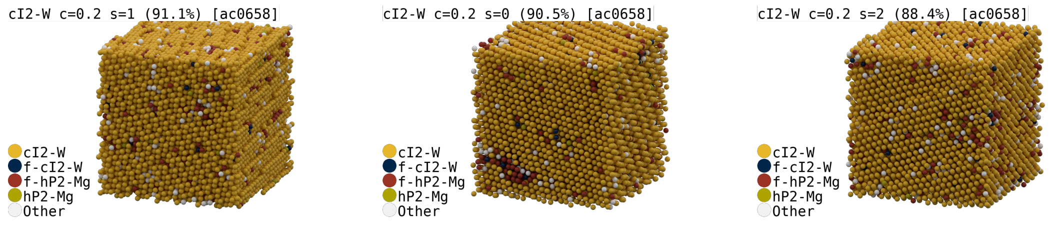

cI2-W c=0.2 s=1 (91.1%) [ac0658]
cI2-W c=0.2 s=0 (90.5%) [ac0658]
cI2-W c=0.2 s=2 (88.4%) [ac0658]
● cI2-W
● f-cI2-W
● f-hP2-Mg
● hP2-Mg
● Other

Figure 2.21: $cI2$-W

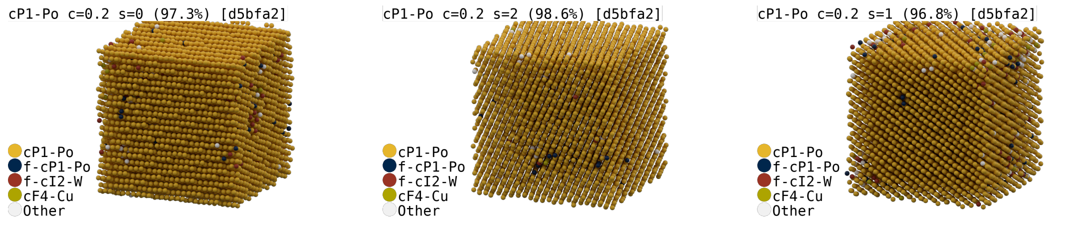

cP1-Po c=0.2 s=0 (97.3%) [d5bfa2]
cP1-Po c=0.2 s=2 (98.6%) [d5bfa2]
cP1-Po c=0.2 s=1 (96.8%) [d5bfa2]
cP1-Po
f-cP1-Po
f-cI2-W
cF4-Cu
Other

Figure 2.22: $cP1$-Po

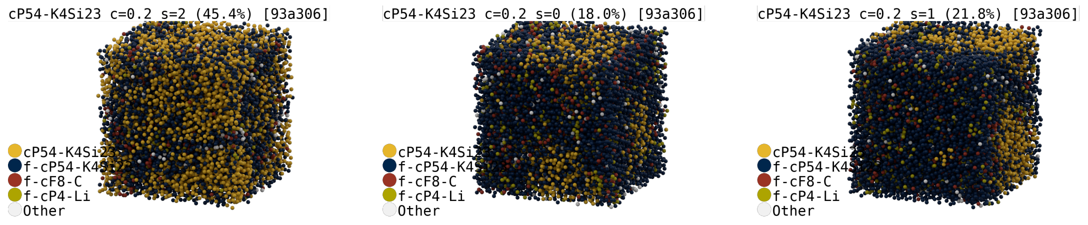

cP54-K4Si23 c=0.2 s=2 (45.4%) [93a306]
cP54-K4Si23 c=0.2 s=0 (18.0%) [93a306]
cP54-K4Si23 c=0.2 s=1 (21.8%) [93a306]
cP54-K4Si2
f-cP54-K4S
f-cF8-C
f-cP4-Li
Other

Figure 2.23: $cP54$-K_4Si_{23}

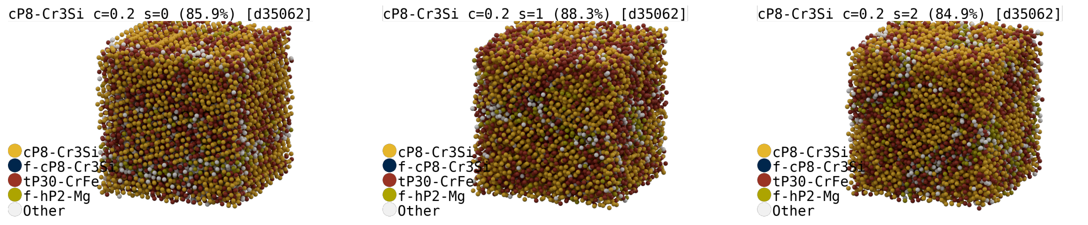

cP8-Cr3Si c=0.2 s=0 (85.9%) [d35062]
cP8-Cr3Si c=0.2 s=1 (88.3%) [d35062]
cP8-Cr3Si c=0.2 s=2 (84.9%) [d35062]
cP8-Cr3Si
f-cP8-Cr3S
tP30-CrFe
f-hP2-Mg
Other

Figure 2.24: $cP8$-Cr_3Si

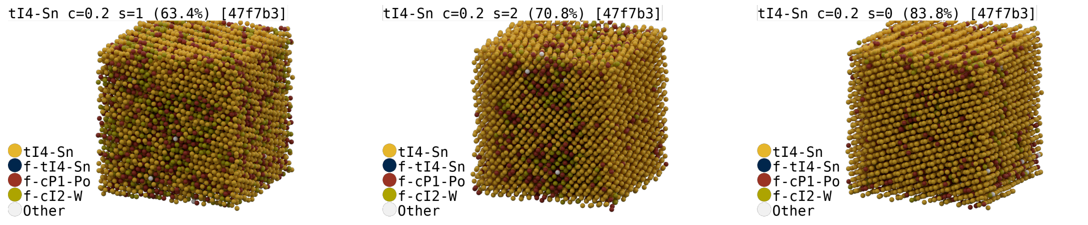

tI4-Sn c=0.2 s=1 (63.4%) [47f7b3]
tI4-Sn c=0.2 s=2 (70.8%) [47f7b3]
tI4-Sn c=0.2 s=0 (83.8%) [47f7b3]
tI4-Sn
f-tI4-Sn
f-cP1-Po
f-cI2-W
Other

Figure 2.25: $tI4$-Sn

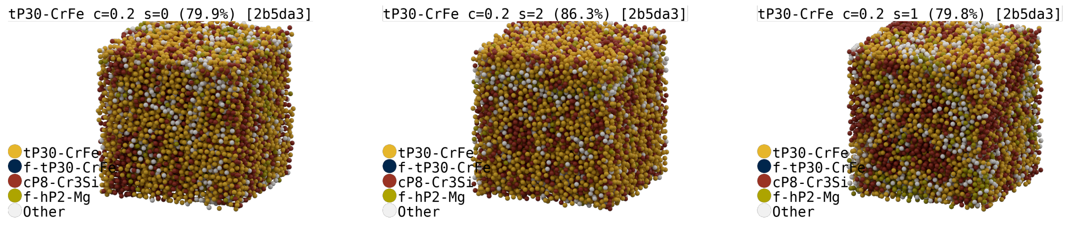

tP30-CrFe c=0.2 s=0 (79.9%) [2b5da3]
tP30-CrFe c=0.2 s=2 (86.3%) [2b5da3]
tP30-CrFe c=0.2 s=1 (79.8%) [2b5da3]
tP30-CrFe
f-tP30-CrFe
cP8-Cr3Si
f-hP2-Mg
Other

Figure 2.26: $tP30$-CrFe

2.6 Acknowledgments

We thank Paul M. Dodd, Pablo F. Damasceno, and Bryan J. VanSaders for helpful discussions and Rose K. Cersonsky for assistance with structure identification. We thank Matthew Spellings for support with the implementation of the machine-learning based crystal structure detection algorithm. This research was supported in part by the National Science Foundation, Division of Materials Research Award # DMR 1409620 and by a Simons Investigator Award from the Simons Foundation to Sharon Glotzer. J.A. is supported by a National Science Foundation Graduate Research Fellowship Grant No. DGE 1256260. J.D. acknowledges support through the Early Postdoc.Mobility Fellowship from the Swiss National Science Foundation, grant number P2EZP2_152128. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575; XSEDE award DMR 140129. Additional computational resources and services provided by Advanced Research Computing at the University of Michigan, Ann Arbor.

The source code for the execution of FF-REM optimizations with HOOMD-blue can be downloaded at `https://glotzerlab.engin.umich.edu/ff-rem`.

Bibliography

[1] Avni Jain, Jonathan A. Bollinger, and Thomas M. Truskett. Inverse methods for material design. *AIChE J.*, 60(8):2732–2740, 2014. doi: 10.1002/aic.14491.

[2] Michael A. Boles, Michael Engel, and Dmitri V. Talapin. Self-Assembly of Colloidal Nanocrystals: From Intricate Structures to Functional Materials. *Chem. Rev.*, 116(18): 11220–11289, 2016. doi: 10.1021/acs.chemrev.6b00196.

[3] George M Whitesides and Mila Boncheva. Beyond molecules: Self-assembly of mesoscopic and macroscopic components. *Proc. Natl. Acad. Sci.*, 99(8):4769–4774, 2002. doi: 10.1073/pnas.082065899.

[4] Sharon C Glotzer. Some Assembly Required. *Science*, 306(5695):419–420, 2004. doi: 10.1126/science.1099988.

[5] Sharon C. Glotzer and Michael J Solomon. Anisotropy of building blocks and their assembly into complex structures. *Nat. Mater.*, 6(7):557–562, 2007. doi: 10.1038/nmat1949.

[6] Salvatore Torquato. Inverse optimization techniques for targeted self-assembly. *Soft Matter*, 5(6):1157–1173, 2009. doi: 10.1039/b814211b.

[7] L. Cademartiri, K. J. M. Bishop, P. W. Snyder, and G. A. Ozin. Using shape for self-assembly. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, 370(1969):2824–2847, 2012. doi: 10.1098/rsta.2011.0254.

[8] William M. Jacobs, Aleks Reinhardt, and Daan Frenkel. Communication: Theoretical prediction of free-energy landscapes for complex self-assembly. *J. Chem. Phys.*, 142(2): 021101, 2015. doi: 10.1063/1.4905670.

[9] William M. Jacobs and Daan Frenkel. Self-Assembly of Structures with Addressable Complexity. *J. Am. Chem. Soc.*, 138(8):2457–2467, 2016. doi: 10.1021/jacs.5b11918.

[10] Miriam H Huntley, Arvind Murugan, and Michael P Brenner. Information capacity of specific interactions. *Proc. Natl. Acad. Sci.*, 113(21):5841–5846, 2016. doi: 10.1073/pnas.1520969113.

[11] Sanat K Kumar, Guruswamy Kumaraswamy, Bhagavatula L V Prasad, Rajdip Bandyopadhyaya, Steve Granick, Oleg Gang, Vinothan N Manoharan, Daan Frenkel, and Nicholas A Kotov. Nanoparticle assembly: a perspective and some unanswered questions. *Curr. Sci.*, 112(8):1635–1641, 2017. URL `http://www.currentscience.ac.in/Volumes/112/08/1635.pdf`.

[12] C. Knorowski, S. Burleigh, and A. Travesset. Dynamics and statics of DNA-programmable nanoparticle self-assembly and crystallization. *Phys. Rev. Lett.*, 106(21):3–6, 2011. doi: 10.1103/PhysRevLett.106.215501.

[13] Ting I.N.G. Li, Rastko Sknepnek, Robert J. Macfarlane, Chad A. Mirkin, and Monica Olvera de la Cruz. Modeling the Crystallization of Spherical Nucleic Acid Nanoparticle Conjugates with Molecular Dynamics Simulations. *Nano Lett.*, 12(5):2509–2514, 2012. doi: 10.1021/nl300679e.

[14] Robert J Macfarlane, M. R. Jones, B. Lee, Evelyn Auyeung, and Chad A Mirkin. Topotactic Interconversion of Nanoparticle Superlattices. *Science*, 341(6151):1222–1225, 2013. doi: 10.1126/science.1241402.

[15] Ting I. N. G. Li, Rastko Sknepnek, and Monica Olvera de la Cruz. Thermally Active Hybridization Drives the Crystallization of DNA-Functionalized Nanoparticles. *J. Am. Chem. Soc.*, 135(23):8535–8541, 2013. doi: 10.1021/ja312644h.

[16] Evelyn Auyeung, Ting I N G Li, Andrew J Senesi, Abrin L Schmucker, Bridget C Pals, Monica Olvera de la Cruz, and Chad A Mirkin. DNA-mediated nanoparticle crystallization into Wulff polyhedra. *Nature*, 505(7481):73–77, 2014. doi: 10.1038/nature12739.

[17] Matthew N. O'Brien, Martin Girard, Hai-Xin Lin, Jaime A Millan, Monica Olvera de la Cruz, Byeongdu Lee, and Chad A Mirkin. Exploring the zone of anisotropy and broken symmetries in DNA-mediated nanoparticle crystallization. *Proc. Natl. Acad. Sci.*, 113 (38):10485–10490, 2016. doi: 10.1073/pnas.1611808113.

[18] Haixin Lin, Sangmin Lee, Lin Sun, Matthew Spellings, Michael Engel, Sharon C. Glotzer, and Chad A. Mirkin. Clathrate colloidal crystals. *Science*, 355(6328):931–935, 2017. doi: 10.1126/science.aal3919.

[19] Mary X. Wang, Jeffrey D. Brodin, Jaime A. Millan, Soyoung E. Seo, Martin Girard, Monica Olvera De La Cruz, Byeongdu Lee, and Chad A. Mirkin. Altering DNA-Programmable colloidal crystallization paths by modulating particle repulsion. *Nano Lett.*, 17(8):5126–5132, 2017. doi: 10.1021/acs.nanolett.7b02502.

[20] E. Edlund, O. Lindgren, and M. Nilsson Jacobi. Designing Isotropic Interactions for Self-Assembly of Complex Lattices. *Phys. Rev. Lett.*, 107(8):085503, 2011. doi: 10.1103/PhysRevLett.107.085503.

[21] E. Edlund, O. Lindgren, and M. Nilsson Jacobi. Using the uncertainty principle to design simple interactions for targeted self-assembly. *J. Chem. Phys.*, 139(2):024107, 2013. doi: 10.1063/1.4812727.

[22] Beth A. Lindquist, Ryan B. Jadrich, and Thomas M. Truskett. Communication: Inverse design for self-assembly via on-the-fly optimization. *J. Chem. Phys.*, 145(11):111101, 2016. doi: 10.1063/1.4962754.

[23] A.K. Soper. Empirical potential Monte Carlo simulation of fluid structure. *Chem. Phys.*, 202(2-3):295–306, 1996. doi: 10.1016/0301-0104(95)00357-6.

[24] Dirk Reith, Mathias Pütz, and Florian Müller-Plathe. Deriving effective mesoscale potentials from atomistic simulations. *J. Comput. Chem.*, 24(13):1624–1636, 2003. doi: 10.1002/jcc.10307.

[25] Tanmoy Sanyal and M. Scott Shell. Coarse-grained models using local-density potentials optimized with the relative entropy: Application to implicit solvation. *J. Chem. Phys.*, 145(3):034109, 2016. doi: 10.1063/1.4958629.

[26] Helgi I Ingólfsson, Cesar A Lopez, Jaakko J Uusitalo, Djurre H de Jong, Srinivasa M Gopal, Xavier Periole, and Siewert J Marrink. The power of coarse graining in biomolecular simulations. *Wiley Interdiscip. Rev. Comput. Mol. Sci.*, 4(3):225–248, 2014. doi: 10.1002/wcms.1169.

[27] M. Scott Shell. The relative entropy is fundamental to multiscale and inverse thermodynamic problems. *J. Chem. Phys.*, 129(14):144108, 2008. doi: 10.1063/1.2992060.

[28] Aviel Chaimovich and M. Scott Shell. Coarse-graining errors and numerical optimization using a relative entropy framework. *J. Chem. Phys.*, 134(9):094112, 2011. doi: 10.1063/1.3557038.

[29] Michael Engel, Pablo F. Damasceno, Carolyn L. Phillips, and Sharon C. Glotzer. Computational self-assembly of a one-component icosahedral quasicrystal. *Nat. Mater.*, 14(1):109–116, 2014. doi: 10.1038/nmat4152.

[30] Beth A. Lindquist, Ryan B. Jadrich, and Thomas M. Truskett. Assembly of nothing: equilibrium fluids with designed structured porosity. *Soft Matter*, 12(10):2663–2667, 2016. doi: 10.1039/C5SM03068D.

[31] R. B. Jadrich, B. A. Lindquist, and T. M. Truskett. Probabilistic inverse design for self-assembling materials. *J. Chem. Phys.*, 146(18):184103, 2017. doi: 10.1063/1.4981796.

[32] Beth A. Lindquist, Ryan B. Jadrich, William D. Piñeros, and Thomas M. Truskett. Inverse Design of Self-Assembling Frank-Kasper Phases and Insights Into Emergent Quasicrystals. *J. Phys. Chem. B*, 122(21):5547–5556, 2018. doi: 10.1021/acs.jpcb.7b11841.

[33] Spotswood D Stoddard and Joseph Ford. Numerical Experiments on the Stochastic Behavior of a Lennard-Jones Gas System. *Phys. Rev. A*, 8(3):1504–1512, 1973. doi: 10.1103/PhysRevA.8.1504.

[34] Joshua A. Anderson, Chris D. Lorenz, and A. Travesset. General purpose molecular dynamics simulations fully implemented on graphics processing units. *J. Comput. Phys.*, 227(10):5342–5359, 2008. doi: 10.1016/j.jcp.2008.01.047.

[35] Jens Glaser, Trung Dac Nguyen, Joshua A Anderson, Pak Lui, Filippo Spiga, Jaime A Millan, David C Morse, and Sharon C. Glotzer. Strong scaling of general-purpose molecular dynamics simulations on GPUs. *Comput. Phys. Commun.*, 192:97–107, 2015. doi: 10.1016/j.cpc.2015.02.028.

[36] John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaither, Andrew Grimshaw, Victor Hazlewood, Scott Lathrop, Dave Lifka, Gregory D Peterson, Ralph Roskies, J Ray Scott, and Nancy Wilkins-Diehr. XSEDE: Accelerating Scientific Discovery. *Comput. Sci. Eng.*, 16(5):62–74, 2014. doi: 10.1109/MCSE.2014.80.

[37] Carl S. Adorf, Paul M. Dodd, Vyas Ramasubramani, and Sharon C. Glotzer. Simple data and workflow management with the signac framework. *Comput. Mater. Sci.*, 146: 220–229, 2018. doi: 10.1016/j.commatsci.2018.01.035.

[38] Vyas Ramasubramani, Bradley D. Dice, Eric S. Harper, Matthew P. Spellings, Joshua A. Anderson, and Sharon C. Glotzer. freud: A Software Suite for High Throughput Analysis of Particle Simulation Data. *arXiv:1906.06317 [cond-mat, physics:physics]*, June 2019. arXiv: 1906.06317.

[39] Joshua A Anderson. fresnel is a python library for path tracing publication quality images of soft matter simulations in real time., 2019. URL `https://github.com/glotzerlab/fresnel`. Accessed on: 2019-09-18.

[40] Matthew Spellings and Sharon C Glotzer. Machine learning for crystal identification and discovery. *AIChE J.*, 64(6):2198–2206, 2018. doi: 10.1002/aic.16157.

[41] Frank H Stillinger. Phase transitions in the Gaussian core system. *J. Chem. Phys.*, 65 (10):3968–3974, 1976. doi: 10.1063/1.432891.

[42] Santi Prestipino, Franz Saija, and Paolo V. Giaquinta. Phase diagram of softly repulsive systems: The Gaussian and inverse-power-law potentials. *J. Chem. Phys.*, 123(14): 144110, 2005. doi: 10.1063/1.2064639.

[43] C. Domb. CXXX. The melting curve at high pressures. *London, Edinburgh, Dublin Philos. Mag. J. Sci.*, 42(334):1316–1324, 1951. doi: 10.1080/14786444108561387.

[44] William G. Hoover, Steven G. Gray, and Keith W. Johnson. Thermodynamic Properties of the Fluid and Solid Phases for Inverse Power Potentials. *J. Chem. Phys.*, 55(3):1128, 1971. doi: 10.1063/1.1676196.

[45] Brian B. Laird and A. D J Haymet. Phase diagram for the inverse sixth power potential system from molecular dynamics computer simulation. *Mol. Phys.*, 75(1):71–80, 1992. doi: 10.1080/00268979200100071.

[46] Rupal Agrawal and David A. Kofke. Thermodynamic and structural properties of model systems at solid-fluid coexistence. *Mol. Phys.*, 85(1):43–59, 1995. doi: 10.1080/00268979500100921.

[47] Mikhail Dzugutov. Glass formation in a simple monatomic liquid with icosahedral inherent local order. *Phys. Rev. A*, 46(6):R2984–R2987, 1992. doi: 10.1103/PhysRevA.46.R2984.

[48] Mikhail Dzugutov. Formation of a dodecagonal quasicrystalline phase in a simple monatomic liquid. *Phys. Rev. Lett.*, 70(19):2924–2927, 1993. doi: 10.1103/PhysRevLett.70.2924.

[49] J Roth and Alan R Denton. Solid-phase structures of the Dzugutov pair potential. *Phys. Rev. E*, 61(6):6845–6857, 2000. doi: 10.1103/PhysRevE.61.6845.

[50] Yu D. Fomin, N. V. Gribova, V. N. Ryzhov, S. M. Stishov, and Daan Frenkel. Quasibinary amorphous phase in a three-dimensional system of particles with repulsive-shoulder interactions. *J. Chem. Phys.*, 129(6):064512, 2008. doi: 10.1063/1.2965880.

[51] Mikael C. Rechtsman, Frank H. Stillinger, and Salvatore Torquato. Self-assembly of the simple cubic lattice with an isotropic potential. *Phys. Rev. E*, 74(2):021404, 2006. doi: 10.1103/PhysRevE.74.021404.

[52] Mikael C. Rechtsman, Frank H. Stillinger, and Salvatore Torquato. Synthetic diamond and wurtzite structures self-assemble with isotropic pair interactions. *Phys. Rev. E*, 75 (3):031403, 2007. doi: 10.1103/PhysRevE.75.031403.

[53] Avni Jain, Jeffrey R. Errington, and Thomas M. Truskett. Communication: Phase behavior of materials with isotropic interactions designed by inverse strategies to favor diamond and simple cubic lattice ground states. *J. Chem. Phys.*, 139(14):141102, 2013. doi: 10.1063/1.4825173.

[54] Fredrik H.M. Zetterling, Mikhail Dzugutov, and Sven Lidin. γ-Brass Crystallization in a Simple Monotomic Liquid. *MRS Proc.*, 643(0):K9.5, 2000. doi: 10.1557/PROC-643-K9.5.

[55] Mans Elenius, Fredrik H.M. Zetterling, Mikhail Dzugutov, Daniel C. Fredrickson, and Sven Lidin. Structural model for octagonal quasicrystals derived from octagonal symmetry elements arising in β-Mn crystallization of a simple monatomic liquid. *Phys. Rev. B - Condens. Matter Mater. Phys.*, 79(14):144201, 2009. doi: 10.1103/PhysRevB.79.144201.

[56] Frederick A Lindemann. Über die Berechnung molekularer Eigenfrequenzen. *Phys. Z*, 11:609–612, 1910.

[57] J. J. Gilvarry. The Lindemann and Grüneisen Laws. *Phys. Rev.*, 102(2):308–316, 1956. doi: 10.1103/PhysRev.102.308.

CHAPTER III

Analysis of Self-Assembly Pathways with Unsupervised Machine Learning Algorithms

This chapter is adapted from a manuscript which is currently in preparation for publication authored by me with contributions to the design and evaluation of the study by Timothy C. Moore and Sharon C. Glotzer and contributions to the technical implementation by Yannah J. U. Melle.

3.1 Introduction

A full and detailed mechanistic understanding of crystallization pathways would have enormous implications on our ability to develop novel materials with unique properties. The still remaining discrepancy between predicted nucleation and growth rates and those measured in experiment suggest that our models of the system and the crystallization process itself are still insufficient in capturing all relevant effects [1]. One particular ambiguity in studying crystallization is the identification of local motifs that particles participate in as they transform from fluid to solid. The advent of machine learning in the field of chemical physics provides an avenue to reduce such ambiguities and autonomously identify relevant particle environments, as we will show in this chapter.

Here we present the analysis of crystallization pathways of colloidal systems, which are especially useful, because simple model systems such as hard spheres and pairwise interacting potentials are amenable to study both theoretically [2–4] and experimentally [5, 6]. The fact that colloidal building blocks on the nano- and microscale can be artificially designed and fabricated and thus, in principle, be optimized for the design of self-assembly routes has sparked the imagination of the scientific community [7] and led to a plethora of computational and experimental studies [8–19].

One of the main challenges in studying the exact pathways of self-assembly is the ability to access relevant time and length scales. That holds true for both physical and computational experiments, but is especially problematic for the study of rare events including nucleation [20]. Rare events like these may have such a minuscule probability to occur on the time scales accessible via brute-force simulation, that even increases in orders of magnitude of available computational resources would not be sufficient. Rare event sampling techniques that fall into the general category of advanced sampling techniques, such as umbrella sampling [21], transition interface sampling [22], and forward flux sampling [23, 24], provide ways to study rare events but require more consideration and a better understanding of the underlying process compared to simple brute-force approaches. They also might have restrictions such as being only suitable to equilibrium or quasi-equilibrium processes.

Even with sufficient sampling, describing and identifying particle environments in a systematic, unbiased, and computationally feasible manner represents its own challenge. One particularly successful approach is to use Steinhardt bond orientational order parameters [25] based on spherical-harmonics, which we will from here on refer to as Steinhardt order parameters, or Q_l in short. Steinhardt order parameters capture symmetries that are broken by the particle's local environment in a rotationally invariant manner, which is especially important when we are concerned with the characterization and identification of local motifs before and during the crystallization process as opposed to characterizing bulk phases post-crystallization.

While useful, Steinhardt order parameters are often insufficient: For one, they require some heuristic on what neighbors to include in their computation. A simple heuristic is to select all points within a certain radius as neighbors. Another heuristic may be based on the number of n-nearest neighbors regardless of their distance. A more advanced approach is the determination of neighbors based on the topology of the local environment: The Polyhedra Template Matching (PTM) structure identification algorithm selects the number of neighbors based on the Voronoi cell surrounding a particular particle [26]. Similarly, the solid-angle based nearest-neighbor algorithm (SANN) also adjusts the number of neighbors for each particle individually based on reaching a threshold of the sum of the solid angle [27].

However, no matter the environment descriptor and environment selection heuristic applied, the use of order parameters to unambiguously identify structures and phases changes still requires the evaluation and comparison of distributions and arbitrary cut-offs. The design of suitable order parameters (OPs) could, in many instances, be better characterized as an art than a science [28]. However in the past few years the field has seen a significant shift towards the use of machine-learning based OPs [29–33], an approach that has the potential to greatly simplify and accelerate the search for suitable OPs for a new system or process.

In this work, we advance techniques for the autonomous identification of local environments present within a given system, and apply them to characterize the crystallization pathways of colloidal systems, including hard spheres, the Lennard-Jones fluid, and systems of point particles interacting via isotropic pair potentials (IPPs) designed to self-assemble specifically targeted complex crystal structures [34]. We use a descriptor composed of the bispectrum of the three-dimensional rotation group ($SO(3)$) that is a high-dimensional rotationally-invariant representation of a local particle environment. The bispectrum descriptor is similar to the Steinhardt order parameter based on spherical harmonics, but preserves more information about the environment than just its symmetry. We show that the high-dimensional descriptor is effective in devising an environment detection model through dimensionality reduction and a clustering-based unsupervised machine-learning workflow.

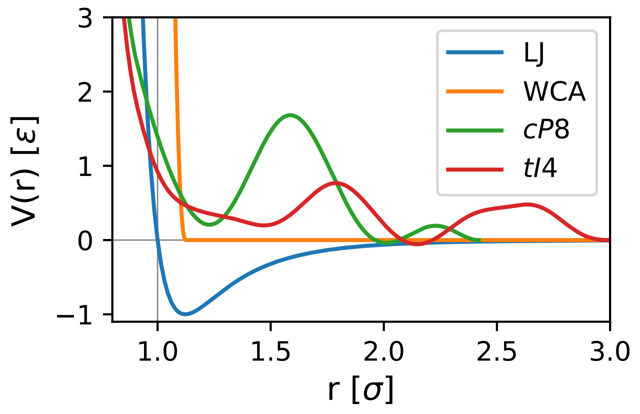


Figure 3.1: We analyzed the nucleation and growth crystallization pathways for systems of point particles interacting via IPPs. We studied two benchmark models (Lennard-Jones (LJ) and Weeks-Chandler-Andersen (WCA)) as well as two models that were specifically designed to assemble complex crystal structures ($cP8$ and $tI4$).

3.2 Theoretical Methods

3.2.1 Models

We studied the crystallization behavior of Weeks-Chandler-Andersen (WCA) spheres and systems of point particles interacting via the Lennard-Jones (LJ) potential and potentials optimized for the self-assembly of A15-type $cP8$-Cr_3Si ($cP8$) and β-tin $tI4$-Sn ($tI4$) crystal structures, all of which are plotted in Fig. 3.1. The models for $cP8$ and $tI4$ were obtained with an algorithm designed for the optimization of simple, short-ranged IPPs that will self-assemble complex crystal structures [34]. To our knowledge, detailed self-assembly pathways of these crystal structures using computational models have not been reported.

3.2.2 Unsupervised Machine Learning Workflow

To study the self-assembly pathways of models for colloidal self-assembly, we employ a workflow, illustrated in Fig. 3.2, that uses a generalized descriptor space as input and produces an environment classification as output. In this way we are able to assign to each particle at each sampled time step a label corresponding to exactly one environment group. The descriptor is in principle independent of the studied systems, but must be general enough to incorporate all important features that sufficiently define a local particle environment at least up to its first, but optionally also to its second, neighbor shell. At a minimum, for a system that assembles a specific target structure, the descriptor space must have features that capture the broken symmetries of that structure.

The "complete feature vector" has length D and is a concatenation of multiple descriptors with the assumption that the unsupervised machine learning model is going to be able to discern the relevant features automatically, and will discard all features that are irrelevant for a given system and pathway. The complete descriptor space is then an array with shape $N \times D$, where N corresponds to our sample size, which is spanned by the number of sampled particles and time steps.

After computing the descriptor space, we apply two dimensionality reduction steps to reduce the overall data size and thus computational demand and to prepare the data for the crucial clustering step. We first reduce the dimensionality from D to 20 by transforming the descriptor space with incremental Principal Component Analysis (PCA) [35, 36] and selecting the first 20 components with the largest variance and hence the most information. Our preliminary studies showed that the relative variance typically drops drastically after 10-20 components, which means that including more than that would not lead to more accurate results.

A subset corresponding to 1,000 randomly selected particles of the resulting data space embedding is then further reduced in dimensionality using the density-based and non-linear Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) algorithm [37]. The UMAP algorithm has natural advantages for our problem space. It reduces the dimensionality of the descriptor space computationally more efficiently than comparable algorithms such as as t-SNE [38], which is important because the descriptor space dimensionality is typically in the hundreds or thousand. Furthermore, UMAP optimizes the resulting manifold as much as possible to preserve distances between points within the higher-dimensional space. This means that the resulting topology and distances between points is related to the actual high-dimensional representation of the system and thus potentially informative and interpretable. The UMAP algorithm was applied with the number of neighbors parameter set to 30, the minimum distance set to zero, and with point distances computed using the "Manhattan" norm.[1] The "Manhattan" metric is advantageous compared to the Euclidean metric in preserving distances in higher dimensional spaces [39].

We reduced the descriptor space embedding to two dimensions for visualization and validation purposes, and to 10 dimensions for the actual unsupervised learning via a clustering algorithm. The lower-dimensional manifolds constitute a representation of the self-assembly process as described by the descriptor vectors, which naturally cluster into groups that may be associated with phase changes and particle environments present in the studied system. Concretely, we expect environments that are found within the same cluster to be more similar to each other than to environments in a different cluster.

We apply the HDBSCAN* clustering algorithm [40] to automatically identify clusters within the 10-dimensional descriptor space manifold. The HDBSCAN* algorithm represents an extension of the popular DBSCAN algorithm that is more robust against fluctuations in density and cluster size within the clustered space compared to standard DBSCAN.

Like DBSCAN, HDBSCAN* clusters points by their local neighborhood density, building up a graph of connected points referred to as a cluster hierarchy. However, unlike DBSCAN, which selects clusters within the hierarchy based on a global cut-off distance, HDBSCAN* merges connected nodes until the cluster associated with each node reaches a specified minimum cluster size. That means the effective cut-off range will be different for each cluster. We determined that a minimum cluster size within the range of 1 to 4% of the system size in combination with a minimum sample size of 1% of the system size is suitable for our application. We further used the "leaf" cluster selection mode, which means that HDBSCAN* selects all leafs within the merged cluster hierarchy, resulting in smaller and more homogeneous clusters.

[1]The Manhattan norm is also referred to as 1-norm.

As aforementioned, the manifold generation and its subsequent clustering was applied to a subset of 1,000 randomly selected particles for all or a subset of all sampled time steps. To predict labels for *all* particles at *all* time steps, we trained a linear Support Vector Machine (SVM) model on the embedded PCA descriptor space (excluding all points classified as noise) with a 3-to-1 training-validation split. The validation accuracy obtained for these models was typically close to or even above 98%. Training a separate model has the distinct advantage of avoiding the need to compute the manifold for the complete descriptor space as well as simplifying the labeling of new data from either the same system or possibly even a different system.

3.2.3 Descriptors

Within the scope of this study, we employed various descriptors based on the local neighborhood including bond distances and bond angles, as well as spherical-harmonics based descriptors such as Steinhardt order parameters and the bispectrum. These order parameters are implemented within the open-source pythia machine learning (ML) package [32, 41]. We found the spherical-harmonics based bispectrum descriptor to generally be most robust to universally describe the self-assembly pathway characteristics across all studied systems [42] compared to Steinhardt order parameter- and bond- or angle-based descriptors.

We also compared our results against the traditionally employed Steinhardt order parameters for comparison and validation purposes. While the bispectrum descriptor preserves much more information compared to the Steinhardt order parameters, the latter require only one dimension per symmetry-variant and neighborhood size. That is significantly less than the bispectrum descriptor, whose dimension is proportional to the cube of l_{max}, which is not a huge problem for the initial dimensionality reduction using PCA but vastly increases the required storage size if the descriptor is to be preserved. We find it encouraging that our machine learning algorithm easily handles descriptors with up to at least 5,368 dimensions, as is the case for the $cP8$ system presented later.

3.2.4 Environment characterization

The environments identified by our unsupervised machine learning workflow can be characterized through visual inspection as well as various order parameters, including the Local Density (LD), the Steinhardt order parameter for different values of l, and through their root-mean-squared deviation. For this we collect a sample of particles that are all classified to belong to the same environment and then compute the given order parameter such as the LD for that sample. This post-classification analysis serves as additional validation, since the original classification via the descriptor-based unsupervised machine learning model is independent of this step.

3.2.5 Data Management and Workflow Implementation

The computational workflow in general and data management in particular for this publication was primarily supported by the signac data management framework [43]. Simulations were performed with HOOMD-blue [44]. Trajectories were analyzed with freud [45]. The descriptors were calculated with pythia [41]. The rendered visualizations of particle systems were generated with OVITO [46]. Computations were carried out on the Oak Ridge National Laboratory (OLCF) Titan and Summit super computers as well as the University of Michigan Flux cluster.

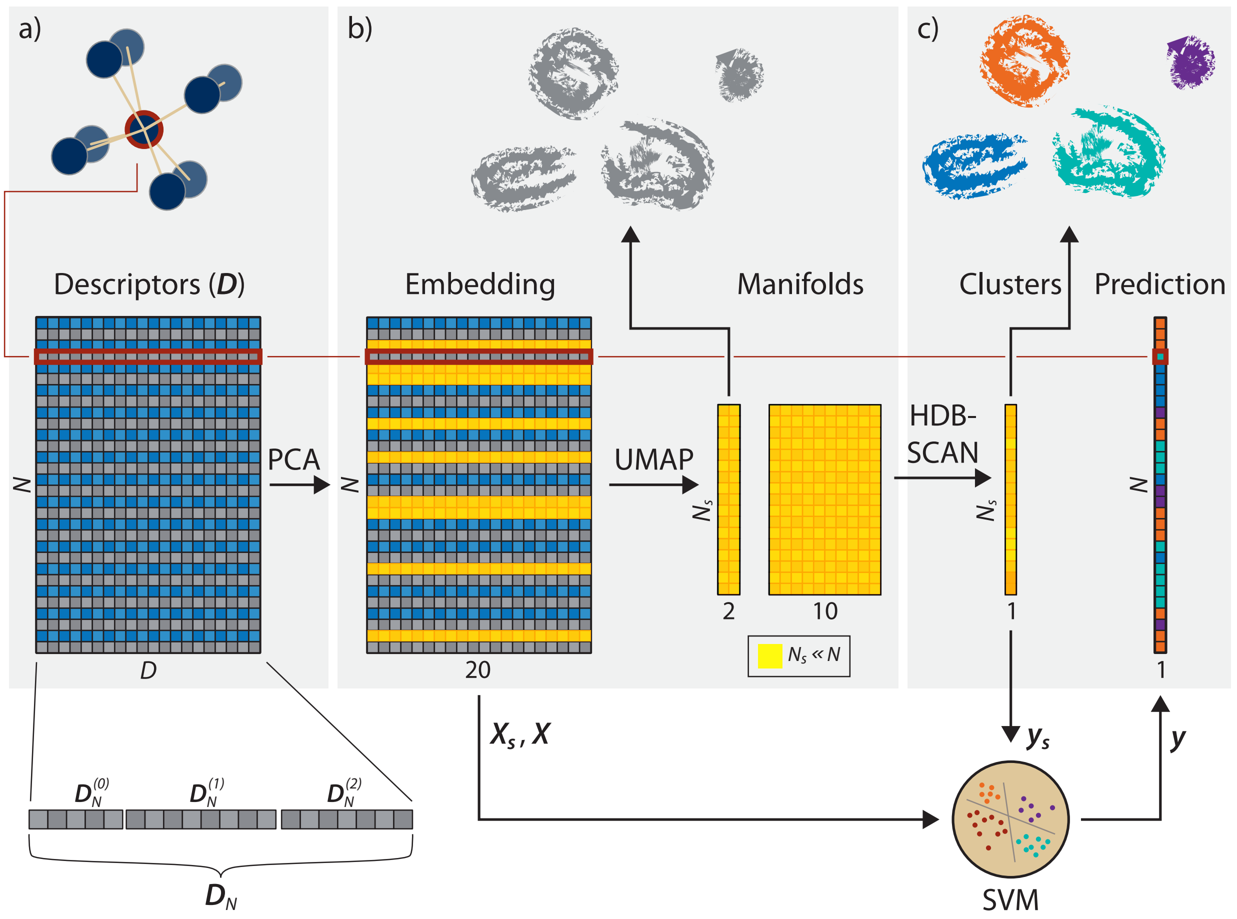


Figure 3.2: We employ an unsupervised machine learning workflow for the automated detection of environments within the crystallization pathway. Particle environments are first mapped to a descriptor space, which should capture the local environment of each particle in a rotationally invariant manner (a). This potentially high-dimensional descriptor space is then reduced to its first 20 principal components using a Principal Component Analysis (PCA) decomposition and subsequently reduced to 2 and 10 dimensions using the Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) algorithm (b). Environments are detected from the UMAP manifold using the HDBSCAN* clustering algorithm (c).

3.3 Results

3.3.1 Weeks-Chandler-Andersen (WCA)

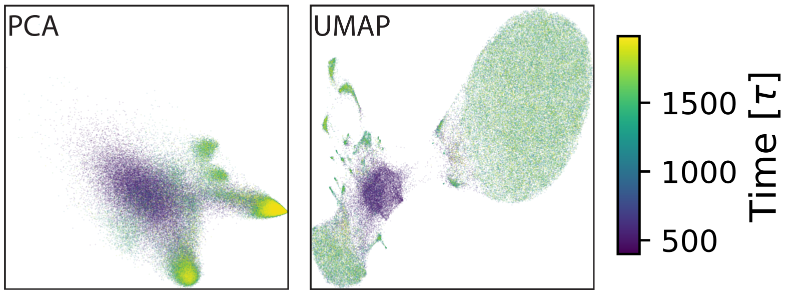
The self-assembly of colloidal hard spheres was simulated with the WCA model

$$V_{\mathrm{WCA}}(r) \begin{cases} 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} + \frac{1}{4} \right] & r \leq 2^{\frac{1}{6}}\sigma \\ 0 & r > 2^{\frac{1}{6}}\sigma \end{cases}, \tag{3.1}$$

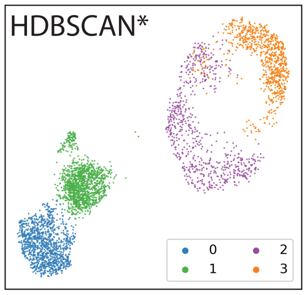
which approximates the interaction of physical hard spheres via a purely repulsive and short-ranged energy function. The self-assembly behavior of WCA spheres has been studied extensively [1, 3, 47] and was just recently analyzed with a similar unsupervised machine learning approach by Boattini et al. [33]. The system represents a suitable benchmark system for the validation of novel colloidal crystallization pathway analysis methods. We sampled the system with Molecular Dynamics (MD) in the NPT ensemble with a constant system size N, pressure P, and temperature T. The integration was carried out via a MTK barostat-thermostat [48–50], a time step of $dt = 0.001\tau$, a thermal coupling constant of $\tau_T = 0.01$, and a pressure coupling constant of $\tau_P = 0.1$. We initialized the system with a largely expanded box and at an elevated temperature of $\beta\epsilon = 40/3$ and then equilibrated at the same temperature to a pressure of $\beta P \sigma^3 = 30$, where β is the inverse temperature $1/kT$. At time $t = 400\tau$ the temperature was abruptly reduced to $\beta\epsilon = 40$.

We computed the bispectrum descriptor for 12 neighbors and, as described in the previous section, mapped the descriptor space with PCA and the UMAP algorithm from $D = 1342$ dimensions to 20, 10, and 2 dimensions (Fig. 3.3a). Although we visualize only the first two principal components, we apply the UMAP algorithm twice, once to generate a two-dimensional manifold for visualization and once to generate a higher-dimensional representation used for clustering. Both UMAP manifolds are inferred from the 20-dimensional PCA reduced embedding.

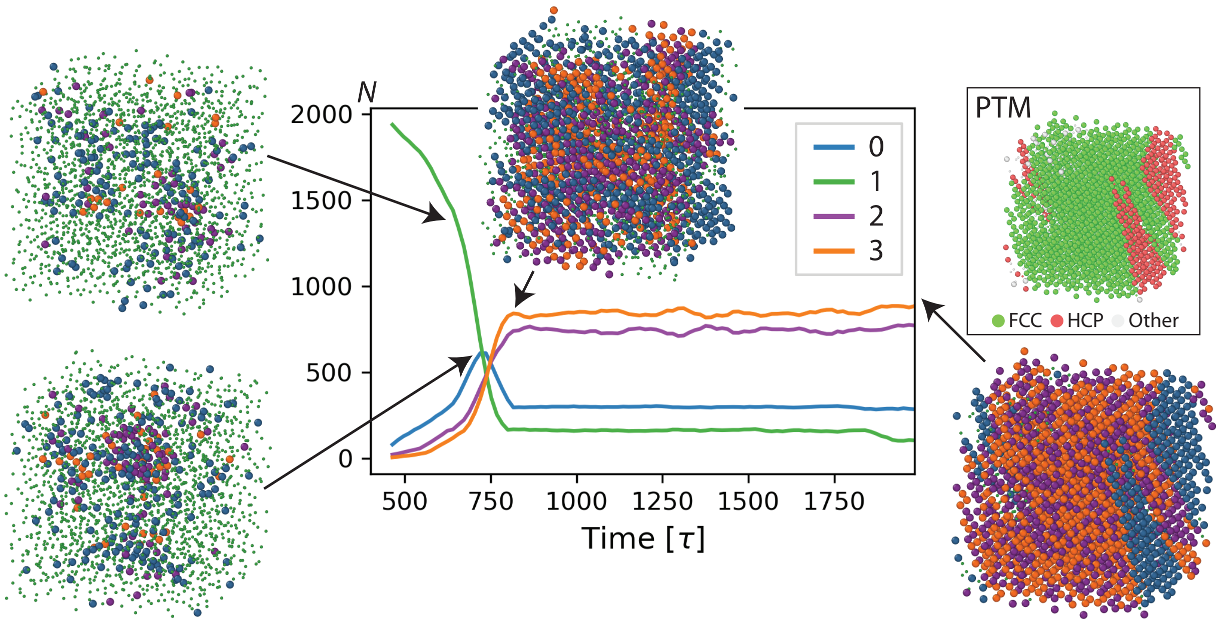
Each point in the plots in Fig. 3.3a represents one particle at a specific time step, which



(a) PCA embedding (left) and 2D-UMAP manifold (right)



(b) UMAP colored by environments



(c) Number of environments over time; fluid environment 1 is visualized with reduced size for clarity

Figure 3.3: The first two principal components of the descriptor space (a left), where each point represents one particle and its environment colored by time step, has a broad initial basin with two smaller competing basins at a later stage. In contrast to the PCA embedding, the two-dimensional UMAP manifold is optimized to represent the complete topology of the high-dimensional descriptor space and for this system displays better separation between unrelated environments (a right). We used the HDBSCAN* algorithm to cluster the 10-dimensional manifold and identify four environments (b). The supersaturated fluid is dominated by a liquid environment (1), with fluctuations of random hexagonal close packing (RHCP) environments (0). The RHCP environments present a precursor for the nucleation of the crystal nucleus, which is primarily made up of face-centered cubic (FCC) environments (2 and 3). We compared our identification of environments to those obtained with Polyhedra Template Matching (PTM) (c).

is signified in terms of its color from dark to bright. The location of each point on the two-dimensional graph represents its environment in terms of the descriptor vector mapped into two dimensions.

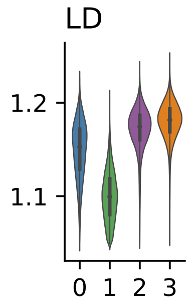
Simply by examining this lower-dimensional representation of the descriptor space, we can already infer some characteristics of the self-assembly pathway and the final phase:

1. There are two primary distinct environment groups within the system.
2. One group is present at both an early and a late stage of the simulation while the other one is only present at a later stage.
3. The environment present at both the early and late stages is more similar to the initial (disordered) fluid phase than it is to the second environment group.

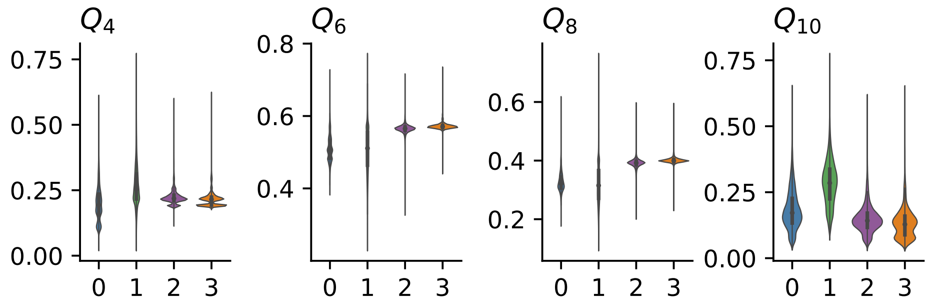
We then applied the HDBSCAN* clustering algorithm to automatically cluster the indiscriminate point cloud shown in Fig. 3.3a into the four clusters shown in Fig. 3.3b. The clustering algorithm clearly delineates between the two primary clusters, however it also splits these further into two smaller clusters that might not have been immediately obvious through simple visual inspection in two dimensions. As stated before, the clustering algorithm operates on the 10-dimensional reduced descriptor space, taking in more information about distances between points than visible to the eye.

Through visual inspection in combination with PTM, we identify the four principal environments to correspond to two phases, as shown in Fig. 3.3c. We identify the initially dominant environment to correspond to a disordered fluid (environment 1). The second most dominant phase consists of random fluctuations of RHCP (0), which find a local maximum around 750τ, playing a strong role in the formation of the crystal nucleus (Fig. 3.3c). Shortly after, the two FCC-like environments (2 and 3) become dominant. The remaining 15% of all particles identified within environment 0 persist in the form of a hexagonal close packing (HCP)-stacking fault.

The fluid-like environment (1) has a significantly lower median LD compared to all other



(a) Local Density



(b) Steinhardt order parameters Q_l

Figure 3.4: The Local Density (LD) order parameter is significantly lower for the fluid environment (2) compared to all other environments, with the HCP-like environment (0) in between the fluid and FCC (3 and 4) (a). The Steinhardt order parameter Q_l, here evaluated for $l = 4, 6, 8$, and 10 shows narrow distributions of Q_6 for both HCP (0) and FCC (3 and 4). We see that there is significant overlap for all environments, which supports that using Q_l alone in combination with hard cut-offs would make it difficult to reliably distinguish between the identified environments (b). The plots show the interquartile range of the data as a vertical bar, the median as a dot, and the whiskers extend to the full range of the data excluding all data points that are considered outliers, overlaid by a kernel density estimate of the distribution; please see the appendix for details.

environments, see Fig. 3.4a. The HCP-like environment (0) has also a slightly lower median LD compared to the two FCC-like environments (2 and 3), which are both very similar to each other. Both environments 3 and 4 are highly similar, however environment 4 has a slightly higher LD and lower Q_{10} Steinhardt order parameter (Fig. 3.4b) which means its local order is slightly higher.

Our simulation and analysis of the WCA system confirms observations made previously by Auer and Frenkel [51], Kawasaki and Tanaka [3], and recently by Ren et al. [1], that fluctuations of RHCP provide the necessary precursory conditions for the formation of the nucleus, which predominantly consists of FCC-like environments with higher Q_6 compared to HCP. Local body-centered cubic (BCC)-like environments do not play a role in the formation of the critical nucleus. Having now validated our methodology, we study increasingly complex systems in the following sub-sections.

3.3.2 Lennard-Jones (LJ)

The nucleation and growth characteristics of the LJ system have been extensively studied [52–54], however there are a few remaining questions. The nucleation rates determined from simulations are still orders of magnitude away from experimental measurements [55] and the exact composition of the nucleus in its early stage is still somewhat dependent on the chosen order parameter [20].

We sampled the crystallization behavior of 4,096 point particles interacting via the LJ potential

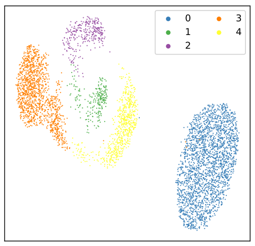
$$V_{\mathrm{LJ}}(r) \begin{cases} V(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] & r \leq r_{\mathrm{cut}} \\ 0 & r > r_{\mathrm{cut}} \end{cases} \tag{3.2}$$

with a cut-off range of $r_{\mathrm{cut}} = 3\sigma$ without shift at the cut-off using Forward Flux Sampling (FFS) at moderate supercooling with a temperature of $kT = 0.85\epsilon$ and a pressure of $P = 5.76\epsilon\sigma^{-3}$ in the NPT ensemble. The integration time step was set to $dt = 0.01\tau$, and the system was coupled to the environment with $\tau_T = 0.1$ and $\tau_P = 1$. Details on the implementation of the advanced sampling with FFS are described in the appendix.

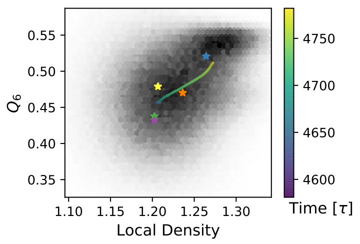
Similar to the WCA system, the pathway analysis of the LJ system reveals two dominant environment groups: Group one represents the environment of the final crystal structure FCC (0), and group two is made up of fluid- (1 and 2) and HCP-like (3 and 4) environments. The two fluid-like environments have low density and order, whereas the two HCP-like environments have similar density, but higher local order, see Fig. 3.5b. We can infer from the manifold that RHCP/HCP is overall more similar to the fluid than FCC (Fig. 3.5a).

The pathway as visualized in the probability density plot of LD vs. Q_6 (Fig. 3.5b) proceeds from the fluid (1 and 2) to RHCP/HCP (3 and 4) to FCC (0) and is characterized by simultaneous densification and ordering. There are two primary wells with a clear separation between FCC and all other environments, with the latter being shallower and broader.

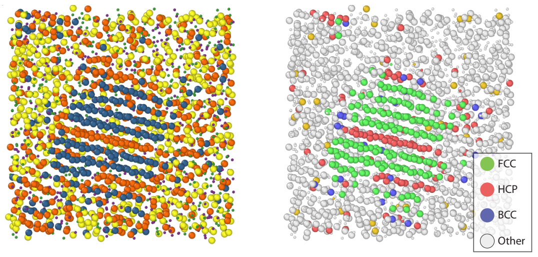
The critical crystal nucleus (see Fig. 3.5c) is primarily made up of FCC with one HCP



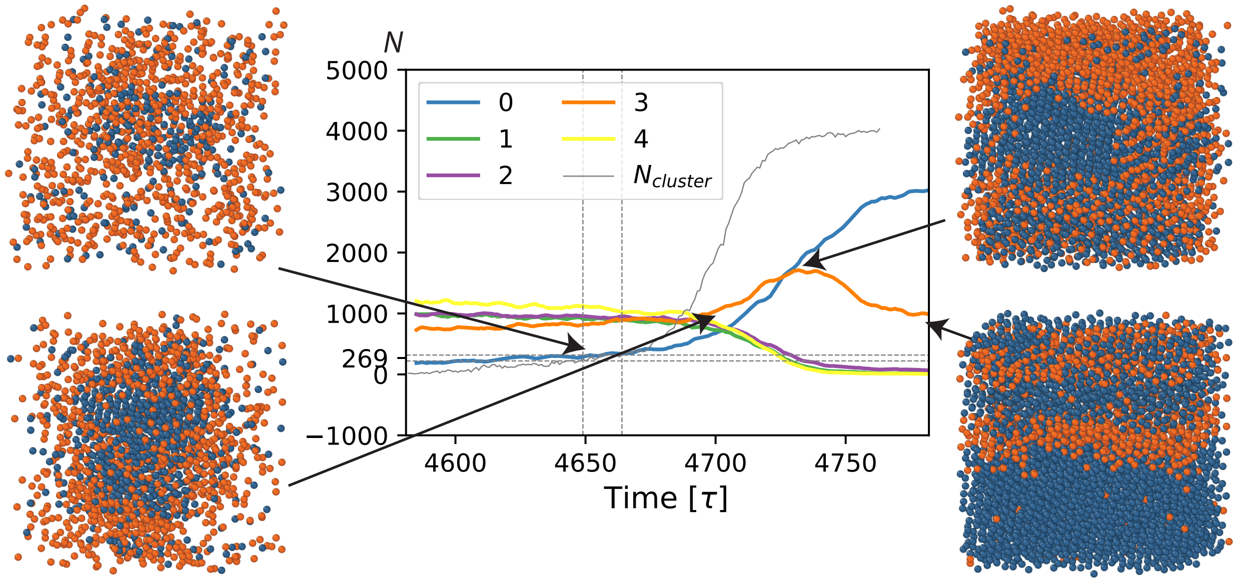
(a) UMAP colored by environments



(b) Probability Density



(c) Critical nucleus colored by environments (left) and PTM (right); envs. 1, 2, and 4 reduced in size for clarity



(d) Number of environments over time; only environments 0 and 3 are visualized for clarity

Figure 3.5: The UMAP representation of the Lennard-Jones (LJ) assembly pathway descriptor space (a) is divided into two major cluster groups with a clear separation of the FCC environment (0) on the right. The probability density diagram (b) features two wells with the deepest well located at the top coinciding with environment 0. We observe that the two HCP-like environments (3 and 4) are located halfway between the fluid environments (1 and 2). Environment 3 has a significant smaller LD compared to environment 4. The visualization of the critical nucleus (c) colored by environments on the left and with polyhedra template matching on the right shows that the HCP-like environments with higher density (3) are incorporated into the critical nucleus, while those with lower density (4) are primarily found in the bulk. The critical nucleus has a significantly higher concentration of FCC environments compared to the bulk, however there is a large number of spontaneous fluctuations of HCP order within the fluid at all times which then increase in density and finally transform into FCC (d).

stacking fault and wetted by HCP environments (3). Fluctuations of RHCP (environment 4) are present within the bulk, meaning that the system has overall very high HCP-like order at all times with a significant density gradient at the surface of the crystal nucleus.

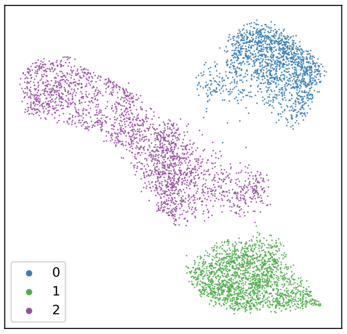
Looking at the crystallization over time, fluctuations of RHCP and HCP environments (3 and 4) are present at all times. However the increase in the number of particles with FCC environment (0) coincides with the nucleus reaching a critical size at 4650τ, which makes sense considering that the nucleus is primarily made up of particles with the FCC environment. The crystal growth phase is characterized by a simultaneous increase of both HCP (3) and FCC (0) environments with HCP being initially dominant and reaching a maximum at 4730τ at which point FCC becomes the dominant environment.

3.3.3 A15-type $cP8$-$\mathrm{Cr_3Si}$

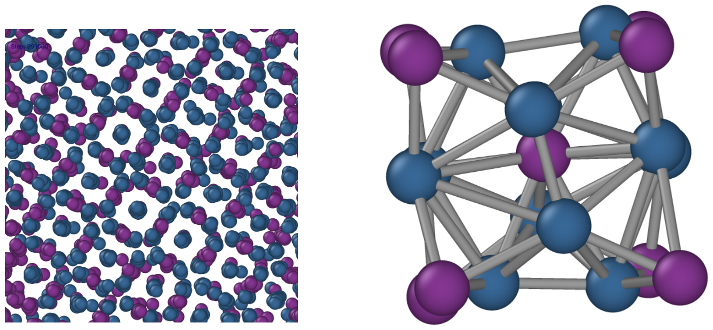
The crystallization of the $cP8$ structure was simulated with the pairwise interaction potential presented in Chapter II. The system was equilibrated with the Langevin integrator (time step $dt = 0.001\tau$) at an elevated temperature of $kT = 3.0\epsilon$ and a slightly expanded box for 100τ and then compressed to the final density for another 100τ. The temperature was then reduced abruptly to $kT = 1.0\epsilon$ at 400τ.

The descriptor for the $cP8$ structure was computed for 12, 13, 24, and 26 neighbors to include the first and second neighbor shells, which results in the identification of the three environments shown in Fig. 3.6a. These environments can be attributed to the disordered fluid (1), an intermittently dominant icosahedral environment (2) corresponding to Wyckoff site a, and an environment that is only present in the final crystal structure (0) and corresponds to Wyckoff site c (see Fig. 3.6b).

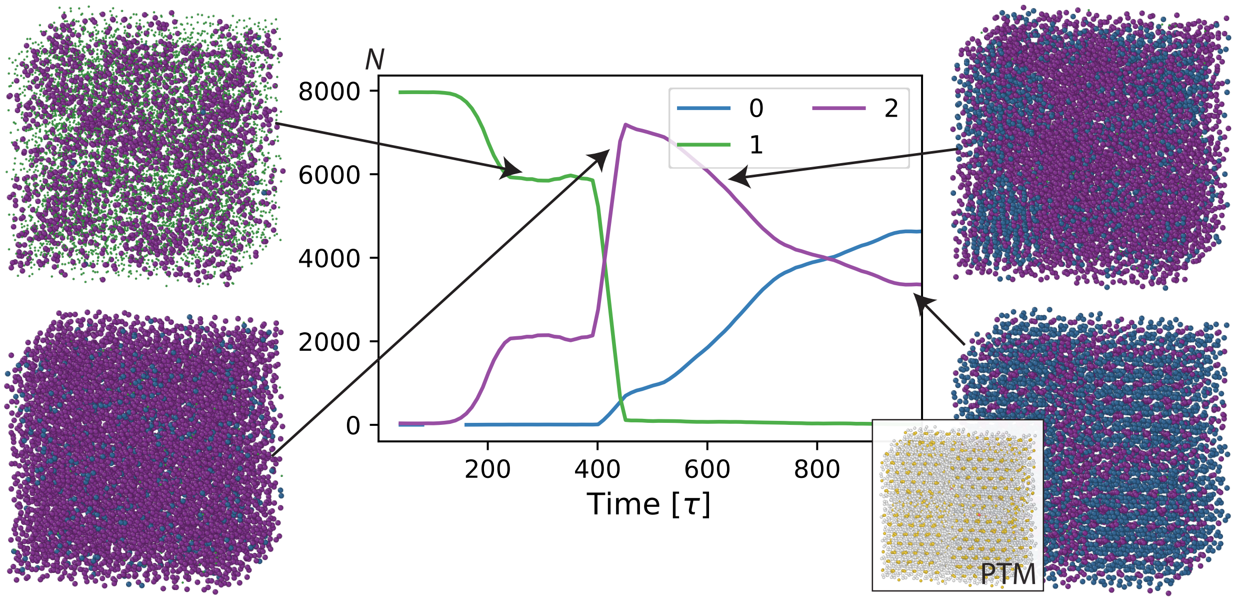
Roughly one-fourth of the particles were found to be within the icosahedral environment (2) prior to the quench, with a significant uptake to almost 100% immediately after the temperature quench (see Fig. 3.6c). These environments became incorporated into the crystal nucleus and the final crystal structure over a prolonged growth period. The final crystal



(a) UMAP colored by environments



(b) A cut of the final crystal structure (left) and one unit cell (right) colored by environments



(c) Number of environments over time; fluid environment 1 is visualized with reduced size for clarity

Figure 3.6: The *cP*8 UMAP descriptor space representation (a) has three primary clusters, a disordered fluid (2), an icosahedral environment (1) that corresponds to *cP*8 Wyckoff site a and a third environment only present in the ordered crystal structure (0), which corresponds to Wyckoff site c (b). The manifold topology suggests that the environments 2 and 1, *i.e.*, the icosahedral environment and the fluid are more closely related. Environment 2 reaches a brief local maximum right after the temperature quench at 400τ and then partially transitions to environment 0 (c). The PTM representation of the final crystal structure shows that environment 2 corresponds to an icosahedral motif (gold), while environment 0 is not recognized (white) (see inset).

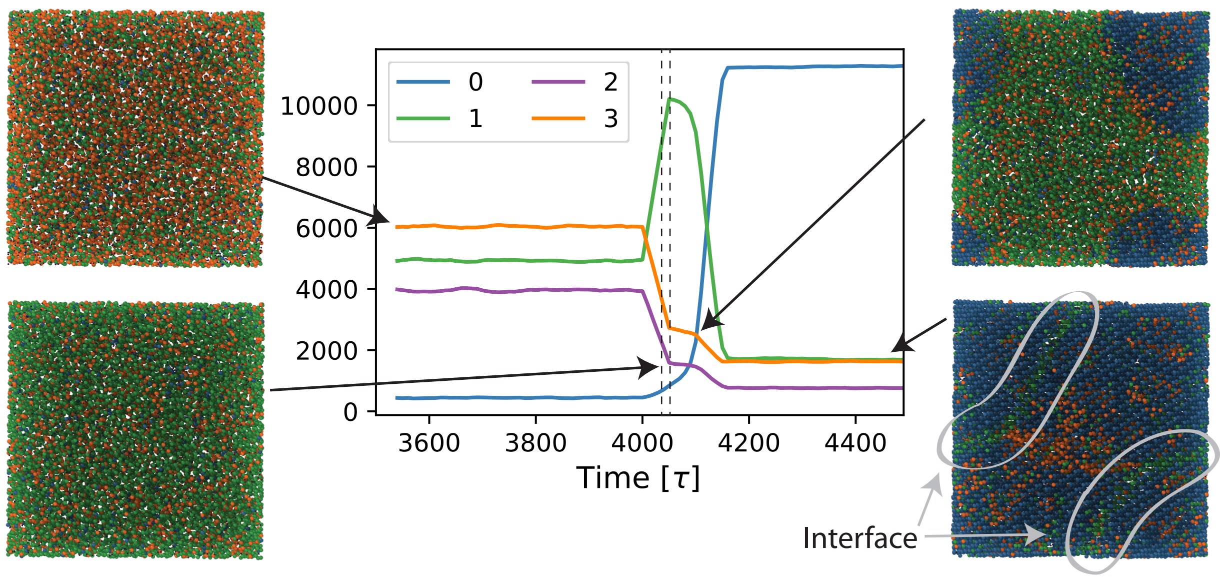


Figure 3.7: The $tI4$ crystallization pathway is characterized by precursor environment (1), which becomes dominant shortly after the temperature quench at 4000τ and is then consumed by the crystalline environment (0). The precursor environment persists in the form of a spherical defect with interfaces circled in the bottom right visualization.

structure has two defect planes, which are visually apparent by a reduced concentration of environment 0.

3.3.4 β-tin $tI4$-Sn ($tI4$)

The crystallization of the $tI4$ structure was simulated with the Langevin integrator ($dt = 0.001$) with the potential presented in Chapter II. The system, comprised of 14,400 point particles, was equilibrated at $kT = 3.0\epsilon$ for 4000τ, and then quenched to $kT = 0.1\epsilon$.

There are three environments present in significant amounts prior to the temperature quench: environments 1, 2, and 3. Environment 1 becomes dominant immediately after the temperature quench at 400τ only to be replaced shortly after by the crystalline environment 0. Environment 1 remains present in significant concentration localized at a defect in the form of a spherical shell (see Fig. 3.7).

The UMAP representation of the $tI4$ self-assembly descriptor space is interpretable both in terms of the relative distance between clusters and their relative location within the two-dimensional coordinate system. The clustering algorithm identifies four clusters within

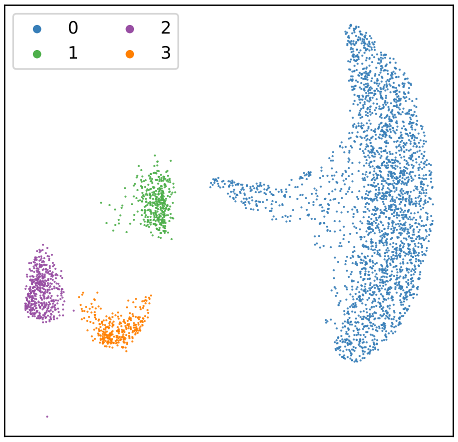


Figure 3.8: The UMAP representation of the $tI4$ descriptor space separates into two main cluster groups, which we can identify as pre- and post-crystallization from the time coding alone (not shown). The clustering algorithm identifies three distinct environments within the left group, which we later associate with the disordered fluid state (2 and 3) and a precursor environment (1).

the descriptor space, with two fluid environments (1 and 3), a precursor environment (1), and a crystalline environment (0) visualized in Fig. 3.8.

3.4 Conclusions

Using a general spherical harmonics-based descriptor in combination with an unsupervised ML workflow we studied the self-assembly of four interaction models that assemble a variety of crystal structures. The applied clustering algorithm autonomously identified groups of local structural motifs, here referred to as environments, that helped us to discriminate between a fluid and a solid phase as well as specific crystal structures, *e.g.*, FCC and HCP and related structures such as RHCP. The autonomous environment classification performed as well as established methods such as PTM in discriminating between fluid and solid phases and in identifying the final structure, interfaces, and defects.

The employed bispectrum descriptor takes multiple symmetries into account and was only adjusted with respect to the number of neighbors for a particular system. We used the coordination number of particles within the final crystal structure as a guideline on the number of neighbors selection, however future studies should use a more advanced neighbor selection heuristic, for example based on the SANN algorithm [27] or by employing Voronoi cell division [26].

All pathways show significant fluctuations of local structural order within the disordered bulk fluid phase prior to nucleation. This suggests that nucleation events are triggered by a combination of random fluctuations in both local order and density, providing further evidence for the hypothesis by Russo and Tanaka [56] that the supercooled bulk liquid is highly structured to lower its free energy compared to a completely isotropic bulk phase. We can further confirm their insight on the importance of the structure of the embryonic crystal nucleus on the final macro state, evidenced by our observation that stacking faults that are present even at a very early stage, for example in the LJ nucleus, persist even after substantial crystal growth and opportunity for defect healing (see Fig. 3.5c and Fig. 3.5d).

3.5 Appendix

3.5.1 Violin plots

Violin plots show the median of the data (dot), the interquartile range in form of a vertical box, and the complete range of the data including the minimum and maximum except for points that are considered outliers as the grey line. Outliers are defined as any point that extends 1.5 times the interquartile range below the first and above the third quartile. In addition, the graph also features a kernel density estimate (KDE) of the distribution, which makes it easier to visually asses the overall distribution of the data.

3.5.2 Forward Flux Sampling (FFS)

Forward Flux Sampling (FFS) is an advanced sampling technique originally presented by Allen et al. [24] which allows the sampling of pathways that have rare events in a way that is computational feasible, largely unbiased with respect to the chosen order parameter, and allows the study of non-equilibrium systems [23].

We studied the crystallization of the LJ system at moderate supercooling, which made it necessary to use advanced sampling techniques to observe nucleation events on a reasonable computational time scale. The nucleation rate of the WCA spheres as well as of the systems of point particles interacting via the $tI4$ and $cP8$ pairwise interactive potentials is comparatively higher at the chosen state points and could therefore be observed with brute-force sampling.

We employed the solid-liquid order parameter introduced by ten Wolde et al. [52] as implemented in the freud analysis package [45]. Using this order parameter, a particle is considered solid-like if and only if it has more than 10 nearest neighbors where the dot product

$$q_l = \bar{Q}^*_{lm}(i) \cdot \bar{Q}_{lm}(j) \tag{3.3}$$

exceeds the threshold of $q_l > 0.6$. The value for $Q_{lm}(i)$ is determined by evaluating

$$\bar{Q}_{lm}(i) = \frac{1}{N_b(i)} \sum_{j=1}^{N_b(i)} Y_{lm}(\boldsymbol{r_{ij}}) \tag{3.4}$$

over the N_b nearest neighbors of particle i, where $\boldsymbol{r}_{ij}$ denotes the bond vector between particle i and particle j and $Y_{lm}(\boldsymbol{r})$ are spherical harmonics. The number of nearest neighbors was determined using a ball-metric with a cut-off radius of $r_{\text{cut}} = 1.35\sigma$.

Bibliography

[1] Shang Ren, Yang Sun, Feng Zhang, Alex Travesset, Cai-Zhuang Wang, and Kai-Ming Ho. Calculation of critical nucleation rates by the persistent embryo method: application to quasi hard sphere models. *Soft Matter*, 14(45):9185–9193, 2018. doi: 10.1039/C8SM01415A.

[2] John D Weeks, David Chandler, and Hans C Andersen. Role of Repulsive Forces in Determining the Equilibrium Structure of Simple Liquids. *The Journal of Chemical Physics*, 54(12):5237–5247, 1997. doi: 10.1063/1.1674820.

[3] T. Kawasaki and H. Tanaka. Formation of a crystal nucleus from liquid. *Proceedings*

of the National Academy of Sciences, 107(32):14036–14041, 2010. doi: 10.1073/pnas.1001040107.

[4] John Russo and Hajime Tanaka. The microscopic pathway to crystallization in supercooled liquids. *Scientific Reports*, 2(1), 2012. doi: 10.1038/srep00505.

[5] U. Gasser, Eric R. Weeks, Andrew Schofield, P. N. Pusey, and D. A. Weitz. Real-Space Imaging of Nucleation and Growth in Colloidal Crystallization. *Science*, 292(5515): 258–262, 2001. doi: 10.1126/science.1058457.

[6] Peng Tan, Ning Xu, and Lei Xu. Visualizing kinetic pathways of homogeneous nucleation in colloidal crystallization. *Nat. Phys.*, 10(1):73–79, 2014. doi: 10.1038/NPHYS2817.

[7] Sharon C. Glotzer and Michael J. Solomon. Anisotropy of building blocks and their assembly into complex structures. *Nature Materials*, 6(8):557–562, 2007. doi: 10.1038/nmat1949.

[8] Zhiyong Tang, Zhenli Zhang, Ying Wang, Sharon C. Glotzer, and Nicholas A. Kotov. Self-Assembly of CdTe Nanocrystals into Free-Floating Sheets. *Science*, 314(5797): 274–278, 2006. doi: 10.1126/science.1128045.

[9] Christopher R. Iacovella, Aaron S. Keys, Mark A. Horsch, and Sharon C. Glotzer. Icosahedral packing of polymer-tethered nanospheres and stabilization of the gyroid phase. *Physical Review E*, 75(4):040801, 2007. doi: 10.1103/PhysRevE.75.040801.

[10] Zhenli Zhang, Zhiyong Tang, Nicholas A. Kotov, and Sharon C. Glotzer. Simulations and analysis of self-assembly of CdTe nanoparticles into wires and sheets. *Nano Letters*, 7(6):1670–1675, 2007. doi: 10.1021/nl0706300.

[11] Mirjam E. Leunissen, Rémi Dreyfus, Fook Chiong Cheong, David G. Grier, Roujie Sha, Nadrian C. Seeman, and Paul M. Chaikin. Switchable self-protected attractions in DNA-functionalized colloids. *Nature Materials*, 8(7):590–595, 2009. doi: 10.1038/nmat2471.

[12] Joel Henzie, Michael Grünwald, Asaph Widmer-Cooper, Phillip L. Geissler, and Peidong Yang. Self-assembly of uniform polyhedral silver nanocrystals into densest packings and exotic superlattices. *Nature Materials*, 11(2):131–137, 2011. doi: 10.1038/nmat3178.

[13] C. Knorowski, S. Burleigh, and A. Travesset. Dynamics and statics of DNA-programmable nanoparticle self-assembly and crystallization. *Physical Review Letters*, 106(21):3–6, 2011. doi: 10.1103/PhysRevLett.106.215501.

[14] Robert J. Macfarlane, Byeongdu Lee, Matthew R. Jones, Nadine Harris, George C. Schatz, and Chad A. Mirkin. Nanoparticle superlattice engineering with DNA. *Science*, 334(6053):204–208, 2011. doi: 10.1126/science.1210493.

[15] P. F. Damasceno, M. Engel, and S. C. Glotzer. Predictive Self-Assembly of Polyhedra into Complex Structures. *Science*, 337(6093):453–457, 2012. doi: 10.1126/science.1220869.

[16] Xingchen Ye, Jun Chen, Michael Engel, Jaime A Millan, Wenbin Li, Liang Qi, Guozhong Xing, Joshua E Collins, Cherie R Kagan, Ju Li, Sharon C. Glotzer, and Christopher B Murray. Competition of shape and interaction patchiness for self-assembling nanoplates. *Nature Chemistry*, 5(6):466–473, 2013. doi: 10.1038/nchem.1651.

[17] Bart de Nijs, Simone Dussi, Frank Smallenburg, Johannes D. Meeldijk, Dirk J. Groenendijk, Laura Filion, Arnout Imhof, Alfons van Blaaderen, and Marjolein Dijkstra. Entropy-driven formation of large icosahedral colloidal clusters by spherical confinement. *Nature Materials*, 14(1):56–60, 2014. doi: 10.1038/nmat4072.

[18] Michael Grünwald and Phillip L. Geissler. Patterns without Patches: Hierarchical Self-Assembly of Complex Structures from Simple Building Blocks. *ACS Nano*, 8(6): 5891–5897, 2014. doi: 10.1021/nn500978p.

[19] Mary X. Wang, Jeffrey D. Brodin, Jaime A. Millan, Soyoung E. Seo, Martin Girard, Monica Olvera De La Cruz, Byeongdu Lee, and Chad A. Mirkin. Altering DNA-Programmable colloidal crystallization paths by modulating particle repulsion. *Nano Letters*, 17(8):5126–5132, 2017. doi: 10.1021/acs.nanolett.7b02502.

[20] Gabriele C. Sosso, Ji Chen, Stephen J. Cox, Martin Fitzner, Philipp Pedevilla, Andrea Zen, and Angelos Michaelides. Crystal Nucleation in Liquids: Open Questions and Future Challenges in Molecular Dynamics Simulations. *Chemical Reviews*, 116(12): 7078–7116, 2016. doi: 10.1021/acs.chemrev.5b00744.

[21] Glenn M. Torrie and John P. Valleau. Monte Carlo free energy estimates using non-Boltzmann sampling: Application to the sub-critical Lennard-Jones fluid. *Chemical Physics Letters*, 28(4):578–581, 1974. doi: 10.1016/0009-2614(74)80109-0.

[22] Titus S. Van Erp, Daniele Moroni, and Peter G. Bolhuis. A novel path sampling method for the calculation of rate constants. *Journal of Chemical Physics*, 118(17):7762–7774, 2003. doi: 10.1063/1.1562614.

[23] Rosalind J. Allen, Chantal Valeriani, and Pieter Rein Ten Wolde. Forward flux sampling for rare event simulations. *Journal of Physics Condensed Matter*, 21(46), 2009. doi: 10.1088/0953-8984/21/46/463102.

[24] Rosalind J. Allen, Daan Frenkel, and Pieter Rein ten Wolde. Forward flux sampling-type schemes for simulating rare events: Efficiency analysis. *J. Chem. Phys.*, 124(19):194111, 2006. doi: 10.1063/1.2198827.

[25] Paul J. Steinhardt, David R. Nelson, and Marco Ronchetti. Bond-orientational order in liquids and glasses. *Physical Review B*, 28(2):784–805, 1983. doi: 10.1103/PhysRevB.28.784.

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

[27] Jacobus A. van Meel, Laura Filion, Chantal Valeriani, and Daan Frenkel. A parameter-free, solid-angle based, nearest-neighbor algorithm. *J. Chem. Phys.*, 136(23):234107, 2012. doi: 10.1063/1.4729313.

[28] James Sethna. *Statistical Mechanics: Entropy, Order Parameters, and Complexity.* OUP Oxford, April 2006. ISBN 978-0-19-856676-2.

[29] Lei Wang. Discovering phase transitions with unsupervised learning. *Phys. Rev. B*, 94 (19):195105, 2016. doi: 10.1103/PhysRevB.94.195105.

[30] J. Wang and A. L. Ferguson. Nonlinear machine learning in simulations of soft and biological materials. *Molecular Simulation*, 44(13-14):1090–1107, 2018. doi: 10.1080/08927022.2017.1400164.

[31] R. B. Jadrich, B. A. Lindquist, and T. M. Truskett. Unsupervised machine learning for detection of phase transitions in off-lattice systems. I. Foundations. *J. Chem. Phys.*, 149 (19):194109, 2018. doi: 10.1063/1.5049849.

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

[33] Emanuele Boattini, Marjolein Dijkstra, and Laura Filion. Unsupervised learning for local structure detection in colloidal systems. *arXiv:1907.02420 [cond-mat]*, 2019. arXiv: 1907.02420.

[34] Carl S. Adorf, James Antonaglia, Julia Dshemuchadse, and Sharon C. Glotzer. Inverse design of simple pair potentials for the self-assembly of complex structures. *The Journal of Chemical Physics*, 149(20):204102–204102, 2018. doi: 10.1063/1.5063802.

[35] Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11): 559–572, 1901. doi: 10.1080/14786440109462720.

[36] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441, 1933. doi: 10.1037/h0071325.

[37] Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv:1802.03426 [cs, stat]*, 2018. arXiv: 1802.03426.

[38] G. Hinton and Y. Bengio. Visualizing data using t-SNE. *Cost-sensitive Machine Learning for Information Retrieval*, 33:2579–2605, 2008.

[39] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Jan Van den Bussche, and Victor Vianu, editors, *Database Theory — ICDT 2001*, volume 1973, pages 420–434. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. ISBN 978-3-540-41456-8 978-3-540-44503-6. doi: 10.1007/3-540-44503-X_27.

[40] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *JOSS*, 2(11):205, 2017. doi: 10.21105/joss.00205.

[41] Matthew P. Spellings. A library to generate numerical descriptions of particle systems: glotzerlab/pythia, July 2019. URL `https://github.com/glotzerlab/pythia`. Accessed on: 2019-09-18.

[42] Risi Kondor. A novel set of rotationally and translationally invariant features for images based on the non-commutative bispectrum. *arXiv:cs/0701127 [cs.CV]*, 2007. arXiv: cs/0701127.

[43] Carl S Adorf, Paul M Dodd, Vyas Ramasubramani, and Sharon C Glotzer. Simple data and workflow management with the signac framework. *Computational Materials Science*, 146:220–229, 2018. doi: 10.1016/j.commatsci.2018.01.035.

[44] Joshua A. Anderson, Chris D. Lorenz, and A. Travesset. General purpose molecular dynamics simulations fully implemented on graphics processing units. *Journal of Computational Physics*, 227(10):5342–5359, 2008. doi: 10.1016/j.jcp.2008.01.047.

[45] Vyas Ramasubramani, Bradley D. Dice, Eric S. Harper, Matthew P. Spellings, Joshua A. Anderson, and Sharon C. Glotzer. freud: A Software Suite for High Throughput Analysis of Particle Simulation Data. *arXiv:1906.06317 [cond-mat, physics:physics]*, 2019. arXiv: 1906.06317.

[46] Alexander Stukowski. Visualization and analysis of atomistic simulation data with OVITO–the Open Visualization Tool. *Modelling Simul. Mater. Sci. Eng.*, 18(1):015012, 2009. doi: 10.1088/0965-0393/18/1/015012.

[47] L. Filion, R. Ni, D. Frenkel, and M. Dijkstra. Simulation of nucleation in almost hard-sphere colloids: The discrepancy between experiment and simulation persists. *J. Chem. Phys.*, 134(13):134901, 2011. doi: 10.1063/1.3572059.

[48] Glenn J. Martyna, Douglas J. Tobias, and Michael L. Klein. Constant pressure molecular dynamics algorithms. *J. Chem. Phys.*, 101(5):4177–4189, 1994. doi: 10.1063/1.467468.

[49] Mark E. Tuckerman, José Alejandre, Roberto López-Rendón, Andrea L. Jochim, and Glenn J. Martyna. A Liouville-operator derived measure-preserving integrator for molecular dynamics simulations in the isothermal–isobaric ensemble. *J. Phys. A: Math. Gen.*, 39(19):5629–5651, 2006. doi: 10.1088/0305-4470/39/19/S18.

[50] Tang-Qing Yu, José Alejandre, Roberto López-Rendón, Glenn J. Martyna, and Mark E. Tuckerman. Measure-preserving integrators for molecular dynamics in the isothermal–isobaric ensemble derived from the Liouville operator. *Chemical Physics*, 370(1): 294–305, 2010. doi: 10.1016/j.chemphys.2010.02.014.

[51] Stefan Auer and Daan Frenkel. Prediction of absolute crystal-nucleation rate in hard-sphere colloids. *Nature*, 409(6823):1020, 2001. doi: 10.1038/35059035.

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

[53] Pieter Rein ten Wolde, Maria J. Ruiz-Montero, and Daan Frenkel. Numerical calculation of the rate of crystal nucleation in a Lennard-Jones system at moderate undercooling. *J. Chem. Phys.*, 104(24):9932–9947, 1996. doi: 10.1063/1.471721.

[54] F. Turci, T. Schilling, M.H. Yamani, and M. Oettel. Solid phase properties and crystallization in simple model systems. *Eur. Phys. J. Spec. Top.*, 223(3):421–438, 2014. doi: 10.1140/epjst/e2014-02100-8.

[55] V. I. Kalikmanov, J. Wölk, and T. Kraska. Argon nucleation: Bringing together theory, simulations, and experiment. *J. Chem. Phys.*, 128(12):124506, 2008. doi: 10.1063/1.2888995.

[56] John Russo and Hajime Tanaka. Crystal nucleation as the ordering of multiple order parameters. *J. Chem. Phys.*, 145(21):211801, 2016. doi: 10.1063/1.4962166.

CHAPTER IV

Scientific Data and Workflow Management with Signac

This chapter is adapted from a publication in the *Journal of Computational Materials Science* in 2018 authored by me, Paul M. Dodd, Vyas Ramasubramani, and Sharon C. Glotzer. The article was assigned DOI:10.1016/j.commatsci.2018.01.035.

4.1 Introduction

Improved software [1–5] and increased resources available to computational researchers [6, 7] have led to significant increases in the quantities of data generated [8]. This makes a highly systematic data management approach crucial to preserving data provenance and ensuring reproducibility. To address this problem, researchers often employ data organization practices such as using human-readable file-naming conventions. Although such solutions address the problem at a superficial level, they suffer from numerous drawbacks with respect to efficiency and flexibility. Here, we introduce `signac`, named after Paul Signac (see Fig. 4.1), a simple and robust framework for the management of complex and heterogeneous data spaces as well as the efficient implementation of workflows. Data spaces managed with `signac` are immediately searchable and sharable.

The capabilities of `signac` are best illustrated by example. Consider a typical, albeit trivial, research task in which we are given data about the pressure, volume, and temperature of a noble gas and wish to develop a simple theory to explain these data. As a first hypothesis,

Figure 4.1: The Pointillist style was invented by Paul Signac (1863-1935) and Georges Seurat (1895-1891) and describes paintings in which images are composed from collections of individual dots, each containing a single color. This style serves as a metaphor for `signac`'s data model, in which the data is dependent on both individual data points *and* their position within the larger parameter space. The painting underlying this artistic illustration *Cassis, Cap Lombard* was created by Paul Signac in 1889 and is owned by the Gemeentenmuseum in Den Haag.

we might test Boyle's law, $pV = \text{const.}$, by iterating over values of p and storing the corresponding values for V in text files named for those values of p. Upon finding that the data appears to be temperature-dependent, we then could choose to test a more general equation, $pV = NkT$.

We are now faced with a dilemma: how do we efficiently adapt our data space for this extension? We could provide the existing files with new names incorporating temperature, but this could quickly become intractable if we had to further increase the complexity of our equation of state. Alternatively, we might determine that storing data in a (relational) database would be a more flexible solution to accommodate any future schema changes; however, that could be much less efficient for a generally file-based workflow and could introduce a significant bottleneck in downstream data processing and analysis.

The `signac` framework resolves this by abstracting away the details of file-based data storage while simultaneously functioning like a lightweight, semi-structured database. Using `signac`, files are directly stored on the file system *along with the associated metadata* in a well-defined storage layout. The metadata is parsed and indexed on-the-fly whenever we use `signac`'s interface to access and search for data. By using `signac` to manage the data in the above example, the tasks of adding a parameter such as temperature and searching for data associated with a particular p, T pair can both be easily realized with only a few commands.

This chapter is organized as follows. First, the general design principles of `signac` are presented. We then delve into greater detail about how the core `signac` functionality is implemented in keeping with these principles, followed by a more in-depth comparison to closely related solutions. Finally, the practicality of this system is then demonstrated through numerous examples indicating how `signac` can be used to manage a variety of disparate, heterogeneous data sets.

4.2 Overview

4.2.1 Design

In the following section we lay out the core design principles behind `signac`, which necessitates making a clear distinction between the `signac` *framework* and the `signac` *application*. The primary focus of this chapter is the `signac` application (henceforth simply `signac`), which implements the core data management functions discussed throughout this chapter. The `signac` framework is a collection of applications and modules that are built on top of the core `signac` application, such as the `signac-flow` application, which will be introduced in Section 4.3.3.

At its core, `signac` is a database built directly on top of the file system, leveraging the many advantages of direct file system access while also providing functions to efficiently index and search the data space. As a database system, `signac` makes only one central assumption: that all data may be discretized within a high-dimensional parameter space (see

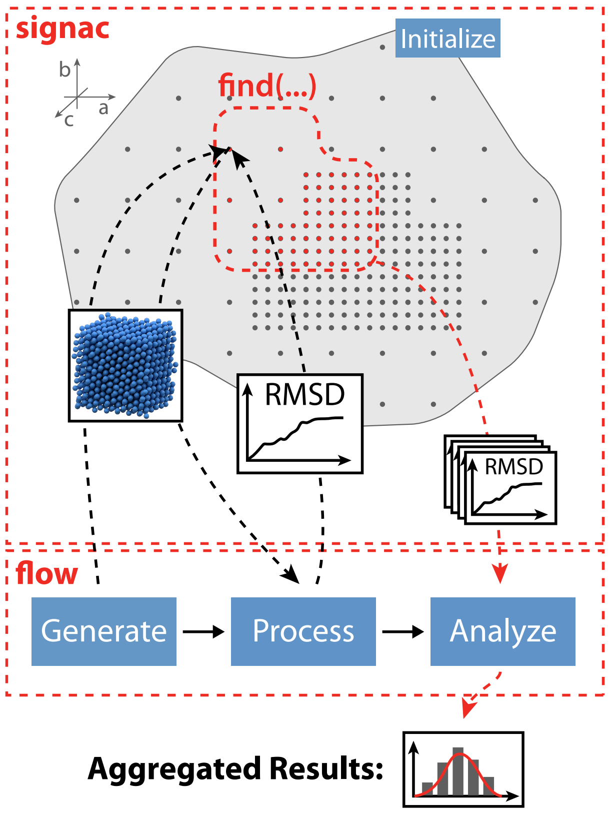


Figure 4.2: This conceptual example demonstrates how we manage and operate on a data space using the `signac` and the `signac-flow` applications. We use `signac` to *initialize* a discrete data space (represented by dark grey dots), where each dot represents a discrete data point and may be associated with anything from a single number to a large set of data. The data space is coordinated within a higher-dimensional parameter space (light grey shape), in this case spanned by the three vectors a, b, and c. Manipulations of the data space (addition, modification, or removal of data), can be divided into *operations*, where each operation must be a function of one or more data points. The operations shown in the example deposit and extract data (dashed arrows) and are organized into a specific workflow using `signac-flow`. Specifically, after initialization, we first *generate* particle configurations, then *post-process* these configurations to extract the root-mean squared displacement (RMSD). Finally, we aggregate results *via* the *analysis* of a subset of our data space that we *find* using a `signac` search query. This example shows the clear division of responsibilities between the different applications. The signac application manages and provides access to the data space and allows us to perform complex search queries. The `signac-flow` application assists in the definition and execution of reproducible workflows comprised of individual data space operations.

Fig. 4.2). Once the user provides the parameters and associated data, `signac` is responsible for managing both the storage of data and its association with the parameters through the maintenance of metadata files encoded in the open JavaScript Object Notification (JSON) format. Through this division, `signac` can ensure both data integrity and searchability.

The database functions of `signac` are modeled after those provided by well-tried database management systems (DBMS) such as MongoDB [9] or MySQL [10]. Typically, such DBMS are very efficient when it comes to the execution of complex query and aggregation operations; however, there are two main issues that render these tools suboptimal for managing the large amounts of (binary) data typically generated by massively parallelized scientific applications within high-performance computing (HPC). First, unless a database is specifically set up to handle peak loads originating from many instances (potentially numbering in the thousands) hitting them in parallel, reading and writing to files distributed on the file system will usually scale more efficiently. Setting up a partitioned or replicated database system to handle higher loads is non-trivial, and this task becomes even more complicated if we care about proper authentication and authorization among different nodes. Secondly, data may need to be serialized for ingestion into the database, which may pose another performance bottleneck, particularly if the data are large binary files.

With `signac`, files are managed directly on the file system and performance is mainly determined by the latency and scalability of the file system. This technique fully exploits the existing file systems on supercomputers, which are commonly designed to process highly parallel, computationally intensive input/output (I/O) operations, thereby avoiding all the above mentioned issues while also allowing for the immediate execution of the previously mentioned query routines.

The `signac` data model assumes that all data associated with a particular computational investigation is part of the same high-dimensional data space and therefore adheres to roughly the same semi-structured schema. Each such investigation is called a `signac` *project*, and the associated data is stored in a special directory, the project *workspace* (Fig. 4.3). The data associated with any given set of parameters within the project's data space is sorted into a distinct subdirectory within the workspace along with a JSON file containing the associated metadata. In the introductory example, each p, T pair represents a point within the larger parameter space, so the data associated with each pair would be stored in a distinct directory within the workspace along with a file containing the corresponding pressure and temperature.

This storage mechanism not only enables efficient on-the-fly indexing, it also ensures that parsing a `signac` managed data space is straightforward even without `signac` since the parameters associated with the data are stored at the same location. In practice, however, `signac` users can ignore these details since the software abstracts away the internal representation of the data space. As described in `signac`'s public documentation,[1] `signac` enables users to easily access the high-level information required to interpret the data space without ever inspecting the filesystem directly.

None of this relieves the user of the burden of documenting their data spaces, *i.e.*, describing explicitly the processes used to generate the data from the provided parameters; this procedure is facilitated by using `signac-flow`. Combined with proper documentation of these processes, however, the use of `signac` ensures that a data space is fully interpretable even for individuals who did not create it.

This interpretability is critical because it makes the data accessible to anyone, even individuals not using `signac` in their own workflows. There is strong evidence that well-maintained public databases, such as the Protein Data Bank (PDB) [11], the Cambridge Structural Database (CSD) [12, 13], The Materials Project [8] or ImageNet [14] have a significant positive impact on their respective fields. Promoting an open data culture among researchers within one or across multiple organizations will likely result in similar positive synergistic effects. The simplicity of `signac` is designed to facilitate this open data culture, because it lowers the barrier to adopt a standardized data storage layout, even for small data

[1]https://www.signac.io

spaces and simple workflows that do not necessarily warrant a more sophisticated solution. A data set managed with `signac` that is uploaded to a repository such as the materials data facility[2] (the National Institute of Standards and Technology (NIST) and the Center for Hierarchical Materials Design (CHiMaD)) or the NOMAD repository[3] (funded by the European Union) is immediately easier to parse, access, and search. A repository interface could be set up to directly support `signac`, which would allow users to search the data by metadata directly. Furthermore, any standardization of metadata tracking facilitates the curation and export of data to public databases such as the NRELMatDB[4] or the materials data base[5] managed by NIST, since converting an existing schema is easier than starting from scratch.

All of `signac`'s core functions are enabled through a highly efficient, *on-the-fly* indexing of the data space. For all higher-level functions, such as data searching and data selection, this indexing process is completely transparent to the user. As a result, `signac` maintains an extremely low barrier to entry, enabling new users to take immediate advantage of basic data management functions. Meanwhile, more advanced users can access `signac`'s full range of capabilities (including detailed control over indexing) for the implementation of complex data-driven workflows.

To remain lightweight and focused, `signac` does not attempt to solve *all* data management concerns. For example, we assume that infrastructure-related issues such as the setup of and access to a distributed file system are better addressed and solved by systems such as the Integrated Rule-Oriented Data System (iRODS) [15] or GLOBUS [16], both of which have a different scope than `signac`.

[2]https://www.materialsdatafacility.org/
[3]https://repository.nomad-coe.eu/
[4]https://materials.nrel.gov/
[5]https://materialsdata.nist.gov/

4.2.2 Workflow

In order to support generic file-based workflows, the `signac` data model makes minimal assumptions about how these workflows generate and operate on the data; `signac` manages the file paths, but the underlying files are stored directly on the file system without modification or serialization. This design ensures that existing tools may interact with a `signac` data repository without the need to serialize or convert existing file formats, an advantage shared by solutions like datreant [17]. Conversely, this design distinguishes `signac` from more domain-specific solutions that make certain assumptions about data schema and format, such as DCMS [18] and the AiiDA infrastructure [19]. See Section 4.4.1 for a more detailed discussion. Hence, a `signac` workspace can be written to or read from outside the context of any broader workflow, and this framework can be used irrespective of how the data is generated or what must be done to process it as long as it is file-based. In other words, whether data is generated through the evaluation of a single equation, or by means of compute-intensive molecular dynamics simulations, `signac` is used in exactly the same way.

While `signac` itself is workflow agnostic, the development of robust workflows operating on data and their reproducible execution is a central component in any scientific investigation. To facilitate this process for users of `signac`, the `signac-flow` package provides users with a flexible set of tools to implement workflows operating on `signac` data spaces (see Section 4.3.3).

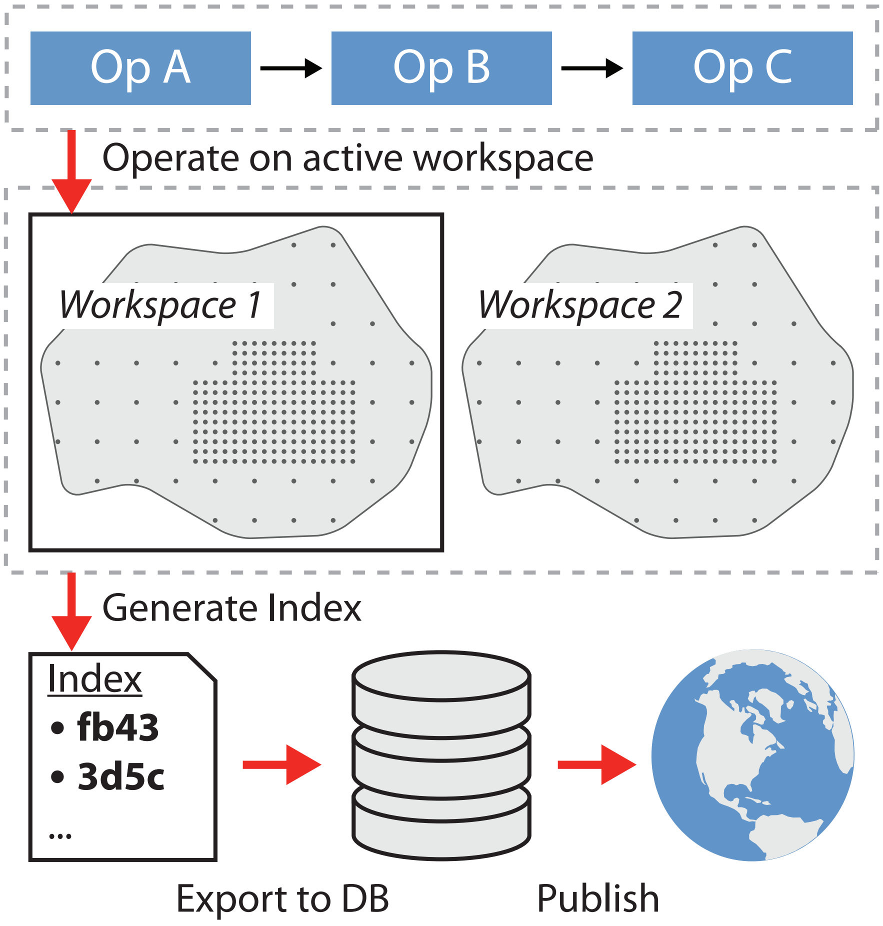


Figure 4.3: The `signac` application manages a particular data space (illustrated in Fig. 4.2) by allocating it to a distinct *workspace* (grey shaded space) on the file system. Data space operations (blue shaded boxes) used for the curation of data are always operating on one specific *active* workspace (black frame). Information about state points, data location and data format may be compiled into an index using `signac`. The index can be used for searching, aggregation, and even direct access to data. The index as well as the data itself, can be exported into a database, which is especially useful for the purpose of making data available to a wide range of subscribers, such as the general public.

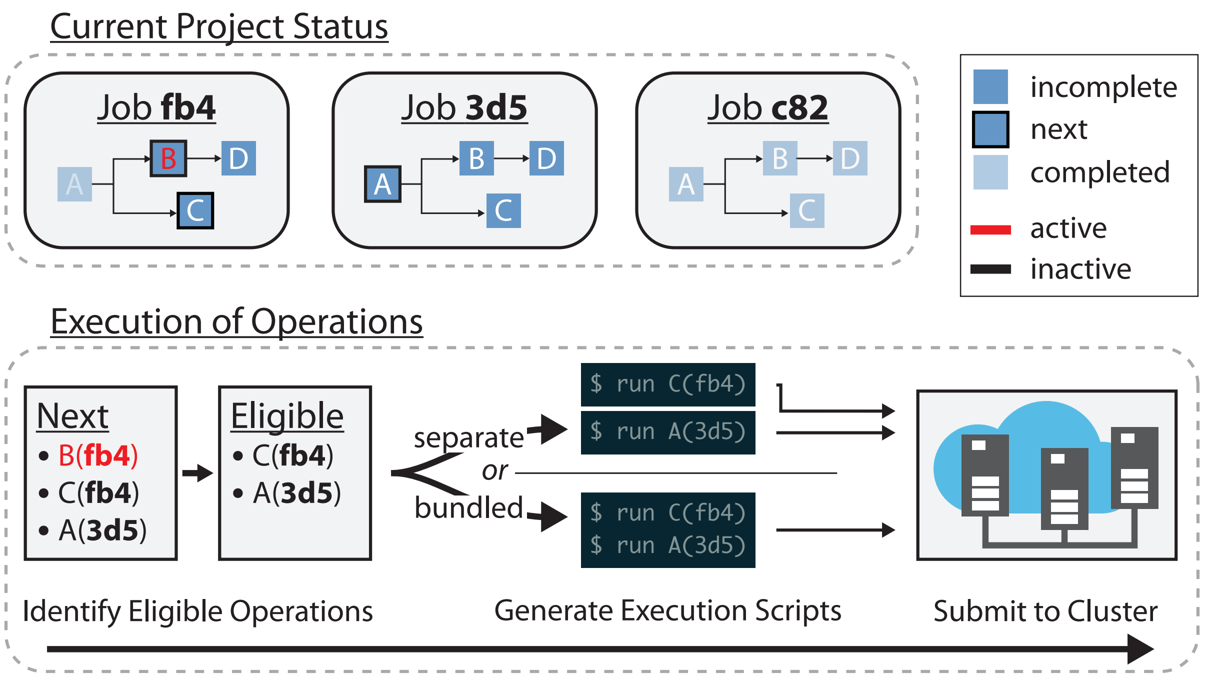


Figure 4.4: In order to track and execute workflows on a `signac` workspace, `signac-flow` FlowProjects track the status of each job (top). This status tracking includes information about which operations have been completed for a given job, which operations are next in line to run, and which operations are incomplete but are not ready to run due to unfulfilled dependencies upstream in the workflow. The progression of each job through the workflow is always known to the FlowProject, as is whether a particular job-operation pair is *active*, *i.e.*, is currently executed on a high-performance computing cluster or is queued for execution. This information is used to determine which job-operation pairs are *eligible* for submission to the cluster scheduler; pairs that are already queued or active are not resubmitted (bottom). For maximal flexibility, the execution of job-operations may be *bundled* prior to submission, *e.g.*, enabling the execution of large numbers of compute-light operations on a single node in serial or parallel.

4.3 Implementation

4.3.1 Software Architecture

The core `signac` data management application, as well as the rest of the `signac` framework, are implemented in Python and tested for versions 2.7.x and 3.x. They are designed to be used in high-performance computing (HPC) environments, and hard requirements besides the Python interpreter are avoided. We employ continuous integrated testing to ensure high interoperability between all main applications. Documentation is generated via the Sphinx documentation tool [20] and made available online.[6]

Although the primary interface is Python-centric, most core `signac` functionality is available through a command-line interface (CLI) to simplify the integration of workflows that are not Python-oriented. Metadata is encoded in the open standard JSON format, which is largely human-readable and can be easily parsed in most programming languages. Relying on a simple, open format ensures that the data remains accessible even without `signac`. Furthermore, the JSON format is internally used by many non relational (NoSQL) database management systems (DBMS), allowing an effortless integration of `signac` with these systems.

4.3.2 Software Components

The main data management functions of the `signac` framework are implemented as part of the core `signac` application. This application is designed with modularity in mind, enabling its extensibility *via* the implementation of additional components of the `signac` framework. This layered structure minimizes the interdependence of higher-level components, making the system more robust against architectural changes [21]. Besides the main application, we have implemented various other tools to augment the `signac` ecosystem such as the `signac-dashboard`, a web application to search and visualize `signac` data spaces in the

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

browser.

In this section, we first describe the three primary functions of the core application: data storage and searching, which simplifies the maintenance and access of complex and heterogeneous data spaces; indexing, which enables efficient advanced post-processing and analysis routines; and database integration, which allows the export of indexes and data to external databases. We then demonstrate this framework's extensibility in Section 4.3.3, where we discuss the `signac-flow` application that we have developed for the management of workflows utilizing `signac`'s data management capabilities.

4.3.2.1 Project Data Management

The data management component is the central component of the `signac` model. The framework supports all typical data management related processes, including data curation, data manipulation, and analysis, by providing a consistent and homogeneous interface for data access and storage within the workspace. The workspace itself is project agnostic, *i.e.* the particular workspace associated with a project may be swapped in and out at any time, and workspaces can be divided and merged as depicted in Fig. 4.3.

The main challenge of reliable long-term storage of data is to ensure the proper association of data and metadata. To surmount this obstacle, the `signac` application calculates a short numeric hash value from the full parameter metadata to generate a unique address, the *signac id*, which is a concise representation of the full state point. The signac id serves as the primary index and constitutes the basis for the file system path within the workspace where associated data is stored. A JSON-encoded copy of the parameter metadata is saved within these paths, which ensures that this association can be trivially identified. The use of a standard format such as JSON ensures that access to the data is not dependent on `signac`.

This methodology bypasses numerous issues common to file system-based workflows. As the output of a hash function, the signac id is both short and non-ambiguous, making it a unique, reliable, and indexable address of the data in all contexts. The signac id can also encode effectively arbitrary complexity, circumventing file naming limitations inherent to most file systems while maintaining great flexibility.

4.3.2.2 Indexing and Database Integration

The internal index that `signac` generates to support its main functions is exposed to the user *on demand*. This can be used to simplify the mapping between different, possibly heterogeneous storage devices, such as a file system and a database system. For example, we could use `signac` to generate files on the file system and execute post-processing routines on the data, and then export the data index into a database that is accessible to a wider group of data subscribers.

To facilitate integration, the current implementation supports export routines for the MongoDB NoSQL database, but in principle any database system that provides a Python driver could be integrated in the future. We chose to initially support MongoDB because its internal data structure is already based on the JSON format and because we consider the semi-structured NoSQL approach more flexible and intuitive to researchers, who are used to dynamic schemas rather than the more rigidly defined table schema used in relational DBMS. Using MongoDB also enables users to leverage tools built for the MongoDB ecosystem for data inspection and manipulation, *e.g.*, `Studio 3T` [22].

The indexes in `signac` are generated by one or more *crawlers*, which for our purposes are defined as any functions that generate a series of JSON documents. In general, the index needs to contain the metadata associated with the data and all information required to allow access to the data. In the specific case of a file system index, this is the metadata and information about file locations and formats. The system is designed for simple customization, *e.g.*, for the extraction of additional metadata from the data (deep indexing). The `signac` application provides templates for crawlers specialized to crawl file systems and generate indexes.

The data processing and index creation steps are intentionally decoupled in `signac`, allowing easy indexing of pre-existing data. This approach is enormously powerful in providing a single homogeneous data interface for new and existing data, particularly because crawlers can be used to index data spaces not generated by `signac`. These indexes can be used to make data accessible to individual researchers within and across organizations, whether or not `signac` was used for their curation.

4.3.3 Implementation of workflows with `signac-flow`

Although `signac` is designed to be workflow agnostic, it is very important for computational scientists to maintain a well-defined workflow that interacts in predictable ways with data. To ease the development of computational workflows using `signac`, we developed the `signac-flow` package, which offers users the ability to design complex workflows around `signac` managed data spaces (illustrated in Fig. 4.4). There are three critical elements of `signac-flow`: *jobs*, each of which represents the data associated with a single parameter combination; *operations*, which are sets of procedures acting on jobs; and *FlowProjects*, which are collections of operations encapsulating a complete workflow associated with a `signac` data space. Note that FlowProjects, which correspond to a single workflow, are distinct from `signac` projects, which correspond to a particular data space. The `signac-flow` package supports multiple FlowProjects acting on a single `signac` project to allow the implementation of multiple distinct workflows on the same data space. An example of where this might be useful would be to create separate FlowProjects to perform coarse-grained and atomistic molecular dynamics simulations of the same system to extract different sets of information.

To convert our original ideal gas workflow into a `signac-flow` FlowProject, we could define an `IdealGasEquationOfState` FlowProject with a single operation responsible for calculating the volume from the parameters. If we desired, we could then easily define additional operations for, *e.g*, the computation of the free energy of the gas. For more complex workflows, the sequence is controlled by a series of pre- and post-conditions for each operation that determine the next set of operations that should be executed. The FlowProject is entirely self-contained, relying on `signac` to store and manage the generated data.

The `signac-flow` package is also designed to facilitate working with compute clusters. For this purpose, we define a *job-operation* as an atomic task consisting of a FlowProject operation acting on a specific job. The FlowProject interface enables the packaging of sets of job-operations into cluster jobs by automatically generating the requisite job scripts; each cluster job can consist of an arbitrary number of job-operations running either in serial or in parallel. At the time of writing, FlowProjects support submission to both Slurm and Torque PBS clusters, generating job scripts on-the-fly after detecting the types of job schedulers present on a given cluster. The `signac-flow` package allows users to configure their default submission behavior, both globally and on the level of a single FlowProject. In addition, users working in cluster environments with specific requirements, such as submitting only to a specific partition, can encapsulate this information into specific Python modules that `signac-flow` can be configured to recognize, making it easier for users to share common configuration information. By providing simple and transparent APIs for cluster submission, `signac-flow` enables users to streamline the large-scale execution of data space operations in cluster environments.

4.4 Practicality and Scalability

To assess the practicality and scalability of our implementation, specifically with respect to existing comparable solutions, we evaluated the following key metrics:

1. Efficiency of setting up a new workflow for an existing tool set.
2. Time needed to determine the data space size.
3. Time needed to iterate through the data space.
4. Time needed to search and select data sets.

Since the first item is difficult to *quantify*, we instead attempt to demonstrate how easily *any* scriptable tool operating on input and output files may be integrated into a `signac`- and `signac-flow`-based workflow by means of the examples laid out in Section 4.5. The remainder of this section is dedicated to a more in-depth comparison of `signac` with alternative solutions, including benchmarks for the last three items in direct comparison with datreant, which we identified as the most comparable tool in both scope and approach.

4.4.1 Comparable Solutions

The `signac` philosophy entails leaving the development of data schemas and workflows largely up to the user, removing the need for specialized input scripts and output parsers. In this way, `signac` substantially differs from domain-specific tools such as AiiDA [19] or pylada-light [23], which impose strict data and workflow restrictions. We believe that this relaxed structure decreases the barrier for integrating new tools and developing new workflows; however, we also recognize that this less standardized approach increases the chance of user error during the implementation and execution of workflows.

In the realm of workflow management, the FireWorks open-source tool [24] stands out as a particularly mature and feature-rich option. Its feature set largely overlaps with the one provided by `signac-flow`, and in addition it offers more advanced job management and monitoring capabilities. These additional features are supported by a MongoDB database on the back-end. In contrast, `signac-flow` relies purely on `signac` to store all runtime and scheduling related metadata.

Integrating FireWorks and `signac` simply involves using `signac` to manage the data space while specifying and executing workflows through the FireWorks interface, similar to how `signac-flow` is currently integrated with `signac`. Yet, there is a caveat: FireWorks' data storage layout is strongly coupled to its execution model, to the extent that Fireworks' documentation explicitly discourages users from manually controlling data storage locations.[7] The tools operate on two different philosophies when it comes to storage layout management,

[7]https://materialsproject.github.io/fireworks/controlworker.html

which poses a barrier for integration.

The Sumatra tool [25] allows users to keep a detailed "automated electronic lab notebook"[8] of operations executed on a specific data space. It is not a job manager in the sense of FireWorks or `signac-flow`, but primarily focuses on ensuring that computational research is reproducible. We found it to integrate very well with `signac` and `signac-flow` operations, enabling users to keep better track of which operations have been executed, a feature which `signac-flow` currently lacks.

The software we found to be most similar to `signac` in core scope and functionality is datreant.core [17], which enables users to associate specific directories with searchable metadata. Just like `signac`, datreant.core is largely domain agnostic, does not require a central server, and performs distributed data management directly on the file system in distinct directories that are associated with searchable metadata.

However, there are also some key differences. First, datreant.core is even more agnostic than `signac` with respect to the general workflow, *e.g.*, there is no need to confine data within a single project entity. Instead, multiple directories may be dynamically organized in *bundles*, which loosely correspond to a `signac` workspace but need not share a common root directory. These bundles can be searched and grouped by metadata, just like `signac` jobs. Furthermore, datreant.core has no concept of a unique identifier like the *signac id*, so the user is still required to choose a directory name for each data set. While this methodology might provide more flexibility in defining a general storage layout and make it easier to combine different data spaces, we contend that it would make it harder for novices to overcome the habit of encoding metadata in file paths, reducing the homogeneity and flexibility of the overall data space. Finally, datreant.core employs file locking mechanisms to ensure that metadata may be safely manipulated in parallel from multiple processes. While that might be advantageous under some circumstances, in practice file locks do not work reliably on the network file systems commonly employed in HPC environments, rendering this feature a

[8]http://neuralensemble.org/sumatra/

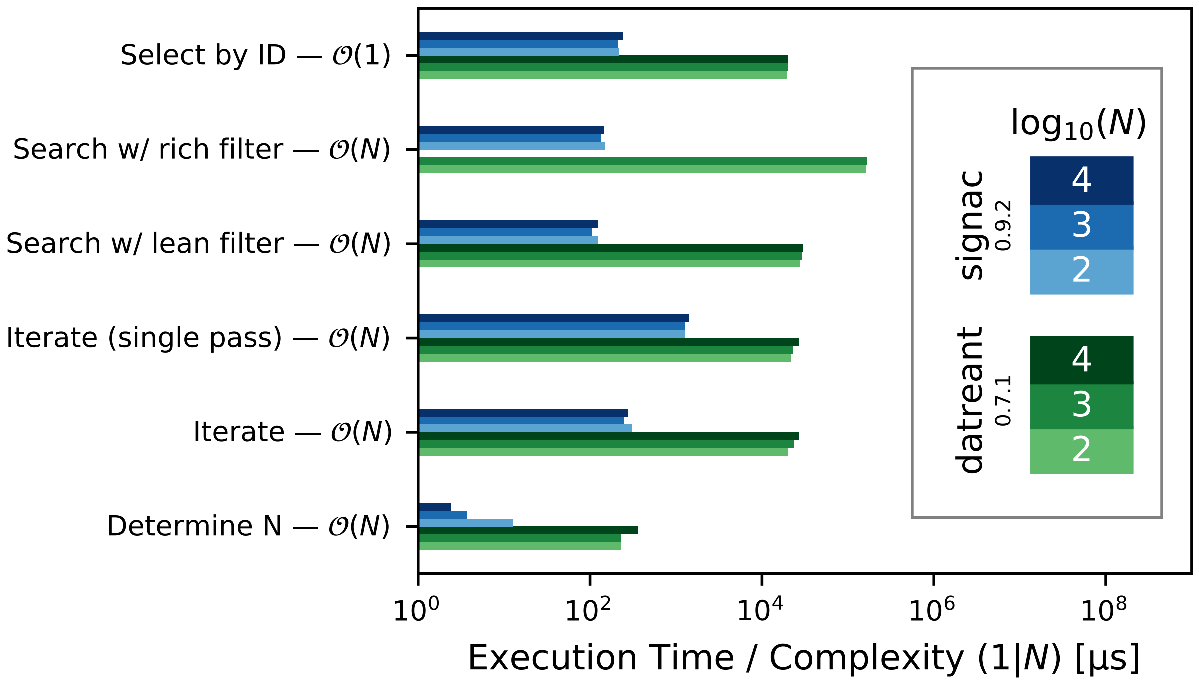


Figure 4.5: We measured the time required for the execution of a set of data space operations as a function of the number of directories N with `signac` and datreant. All tests were executed with Python 3.6 on a network file system; reported values are the minimum of 3 independent test sessions, where each one is averaged over 10 runs within one session, except for the 4th, which was run only once per session; the 2nd test category was aborted for datreant at $N = 10^4$ due to very long execution time. All values are normalized by the expected complexity, *i.e.*, they must be multiplied with the respective order to obtain absolute values for a specific data space size.

liability in cases where it would be most needed. For this reason, the `signac` implementation avoids any reliance on file locks. Overall, we have found datreant to be the most comparable existing solution for the core problems `signac` aims to solve.

4.4.2 Benchmarks

Since datreant.core most closely corresponds to `signac`'s scope and approach for data management, we used it as a quantitative benchmark for the performance and scalability of our implementation. Concretely, we measured the time each tool required

1. to select a single data set by known id,

2. to search and select with a rich filter (many keys),
3. to search and select with a lean filter (one key),
4. for the first iteration through the meta data space within one session,
5. for multiple iterations within one session,
6. to determine the data space size N.

We then used this data to estimate the time complexity of each operation with respect to N. In `signac`, we expect all but the first category to run with a time complexity of $\mathcal{O}(N)$, since the largest bottleneck is likely to be the initial parsing of all metadata within one session. A *selection by known id* should be constant time $\mathcal{O}(1)$. The results of these measurements are plotted in Fig. 4.5.

We are able to show that within our test environment[9] a data space of $N = 1,000$ directories and approximately 1 kB of metadata per directory,[10] all operations, even those that scale linearly with the data space, are executed nearly instantaneously on a human time scale. For example, the first iteration through the complete metadata space within one session requires on the order of 1 ms per directory. It is important to point out that none of these operations are in any way affected by the number or size of data files within those directories since they only interact with the JSON metadata files.

While both `signac` and datreant show very similar scaling behavior, we can clearly show that `signac` is at least one order of magnitude faster than datreant in all tested categories despite implementing very similar concepts. The maximum practical data space size – at which users perceive the system response time (SRT) acceptable for complex tasks – is therefore much larger.

Comparing our time measurements on a network file system (~1 ms per directory for start-up) with the guidelines laid out by Doherty and Sorenson [26], operating on data spaces

[9]A workstation with 20 Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz cores running Gentoo Linux (4.9.34).

[10]That corresponds roughly to 10 keys of one character associated with 100 character long values.

with up to 300 directories would be perceived as instantaneous (<300 ms), 1,000 directories as immediate (<1 s) and up to 5,000 directories as transient (<5 s). Larger data spaces with up to 300,000 directories may still be acceptable, but will require multitasking and/or additional feedback on the progress to not break the user flow.

In summary, while the only hard cap on the data space size is the file system and main memory storage capacity, interactive work may be significantly impaired by prolonged session start-up times for data spaces with more than 300,000 directories. In this case users would be advised to aggregate the working set of data prior to interactive work. We consider data spaces with up to 10,000 directories very practicable for interactive work even on network file systems. All the code to generate these benchmarks is open source and available online.[11]

4.5 Examples

In this section we introduce two representative conceptual examples that demonstrate how to incorporate `signac` into computational workflows. The first one is in reference to the case presented in Section 4.1, the evaluation of the equation of state of an ideal gas. The second is a molecular dynamics study of the Lennard-Jones potential, which is slightly more involved, but also more realistic.

For brevity, some commands are omitted or shortened; however, fully functional examples, including additional demonstrations for DFT and GROMACS, can be found online.[12] All Python examples are tested for Python version 3.5.

4.5.1 Ideal Gas Example

This is a minimal demonstration for carrying out the example described in the introduction. We intend to calculate and store the volume V of an ideal gas within the three-dimensional parameter space spanned by p, N, and kT.

[11]https://github.com/glotzerlab/signac-benchmarks
[12]https://github.com/glotzerlab/signac-examples

We start by creating an empty directory for our project and initializing the signac project:

```
$ mkdir idg_eos
$ cd idg_eos
$ signac init IdealGasEOS
```

The project initialization creates a small configuration file within the current directory to mark it as the project's root directory.

4.5.1.1 Minimal Ideal Gas Example

For our most basic demonstration, we implement a Python script to calculate and store the volume in `signac`'s built-in JSON storage for each state point of interest:

```
1 import signac
2
3 project = signac.get_project()
4
5 for p in 0.1, 1.0, 10.0:
6     sp = {"p": p, "N": 1000, "kT": 1.0}
7     job = project.open_job(sp)
8     V = job.sp.N * job.sp.kT / job.sp.p
9     job.doc.V = V
```

First, we import the `signac` Python package (l. 1). Then we obtain a handle on the project (l. 3), which is the interface for accessing and manipulating the project's data space. To calculate the phase diagram — here as a function of pressure — we simply iterate over p (l. 5) and construct the full state point `sp` associated with each data point (l. 6).

This state point is passed into the `project.open_job()` function, which returns a *job handle* that represents this specific data point (l. 7). The volume is calculated from the state point variables associated with the job, which we access *via* the `job.sp` property (l. 8). Being a single number, the volume naturally lends itself to being stored in a very lightweight format. Here, we leverage the `job.doc` property of `signac` jobs, which provides a lightweight, persistent, and immediately searchable JSON storage option associated with each `signac` job (l. 9). However, we could store the data just as well in a file with a format of our choosing, as will be shown in the next example.

Once the data space is initialized, we can immediately start searching it. For example, to find all state points, where *p is greater or equal than 1.0*, we would execute:

```
jobs = project.find_jobs({"p.$gt": 1.0})
```

The `jobs` variable is the result cursor that we can use to iterate over all jobs that match the given criterion. We can execute the same kind of queries directly on the command line:

```
$ signac find p.\$gt 1.0
```

In this case the ids of all matching jobs will be output for further processing. The query language supports a variety of operators, including but not limited to arithmetic and logical operators, and represents a subset of the MongoDB query language, making it easy to transition between the two systems. More details can be found in the online documentation.

4.5.1.2 Ideal Gas with a Bash Terminal Script

In many cases parts of our workflow will rely on precompiled programs or other scripts that can be interfaced on the command line, but not directly through Python. For example, we might have a program called `idg`, that accepts parameters N, kT, and p as the first through third argument and outputs the resulting volume V, *e.g.*:

```
$ idg 1000 2.0 1.0
2000.0
```

The `signac` application provides a command-line interface (CLI) to simplify the integration of such tools. The following example script replicates the first example, but in bash instead of Python and the volume is stored in a file called `V.txt` instead of the job document.

```
#!/bin/bash
N=1000
```

```
3 kT=1.0
4 for p in 0.1 1.0 10.0; do
5   SP={\"N\": $N, \"kT\": $kT, \"p\": $p}"
6   WS=`signac job -wc "$SP"`
7   ./idg $N $kT $p > $WS/V.txt
8 done
```

After storing parameters as constants at the beginning of the script (l. 2-3), we again iterate over the variable of interest (l. 4) and construct the full state point `SP` in JSON formatting.[13] (l. 5). We then provide the state point as argument to the `signac job -wc` command, which creates the corresponding job and returns the full workspace path `WS` (l. 6). Finally, we execute the `idg` program and pipe its output into the `V.txt` file within the job's workspace (l. 7). This approach reliably couples the job's data and the parameters used to generate them.

An alternative approach for the incorporation of command line tools is the construction of the required bash commands within a Python script:

```
import signac
from subprocess import run

IDG = "./idg {job.sp.N} {job.sp.kT} {job.sp.p}"\
      ">{job.ws}/V.txt"

project = signac.get_project()

for p in 0.1, 1.0, 10.0:
    sp = {"N": 1000, "kT": 1.0, "p": p}
    job = project.open_job(sp).init()
    if not job.isfile("V.txt"):
        run(IDG.format(job=job), shell=True)
```

This approach can be more flexible, especially in cases where users are already familiar with

[13]The JSON format expects all keys to be enclosed in double quotes, which need to be escaped within the bash script We recommend using here-docs for larger state point definitions.

Python. The crucial point is that input parameters and location of the output data are always automatically and unambiguously linked.

4.5.2 Molecular Dynamics with HOOMD-blue

Similar to the first example, we again calculate the equation of state of a gas, this time using molecular dynamics with a Lennard-Jones potential. This means that instead of merely evaluating a single analytic function, we need to set up initial and boundary conditions of the simulated system, load the interaction potential, define the simulation protocol, and possibly store significant amounts of output data.

4.5.2.1 Basic example

For this example we will use the `HOOMD-blue` [2, 27, 28] particle simulation toolkit which provides a native Python interface. This means we can interface with the `signac` project directly within the input script. If there was no Python interface, we would follow the approach shown in the previous (CLI) example (Section 4.5.1.2).

```
import signac
import hoomd
import hoomd.md

def setup_and_simulate(job):
    # [...] Setup initial conditions
    hoomd.md.integrate.langevin(kT=job.sp.kT, seed=job.sp.seed, ...)
    hoomd.dump.gsd(filename="trajectory.gsd", period=2e3, ...)
    hoomd.run(steps=1e4)

project = signac.get_project()

for kT in 0.1, 1.0, 2.0:
    sp = {"kT": kT, "seed": 42}
    with project.open_job(sp) as job:
```

```
16          setup_and_simulate(job)
```

We start by importing all required packages (l. 1-3) and continue by defining a function for the execution of our simulation as function of the job (l. 5). We skip HOOMD-specific commands needed for the setup of the simulation, but lines 7 and 8 show how we use the `job.sp` interface to directly set the simulation parameters.

The iteration over the data space (l. 13) and the definition of the full state point (l. 14) are analogous to the previous examples. Instead of wrapping all input and output filenames wherever they appear (such as in line 9), we use `signac`'s built-in context manager to change into the job's workspace for all commands that are within the scope of the `with` clause (l. 15). That means `signac` will change into the correct directory for the duration of the execution of the `setup_and_simulate()` function and return to the previous directory after completion.

In this example, the data space operations that we execute are still very simple: simulations are executed sequentially by iterating over the variable of interest, kT. However, for more complex workflows, especially those involving more compute-intensive operations, it is advantageous to break things up into smaller steps that can be executed in parallel and possibly be submitted to an HPC cluster. One possible approach for doing so is shown in the next example, utilizing the the previously introduced `signac-flow` application (see Section 4.3.3).

4.5.3 Workflow management with `signac-flow`

While users are encouraged to integrate the `signac` data management application into existing workflows or develop new ones that fit their specific applications, here we demonstrate the use of the `signac-flow` application for the rapid development of workflows for users that are so inclined. The application is quite general, and is simply designed around the sequential or parallel execution of operations in well-defined order. Splitting the overall workflow into such self-contained operations increases flexibility and reproducibility and is especially beneficial for larger studies.

We demonstrate the concept by adapting the previous example. First, we move the initialization logic into a separate script to *initialize* the data space prior to executing any data space operations:

```python
# init.py
import signac

project = signac.init_project("LJ-EOS")

for kT in (0.1, 1.0, 2.0):
    sp = {
        "kT": kT, "seed": 42,
        "epsilon": 1.0, "sigma": 1.0,
        "r_cut": 3.0}
    project.open_job(sp).init()
```

This initializes the complete data space with the essential parameters required for the execution of our molecular dynamics simulations.

Second, we split the `setup_and_simulate()` step into `setup()` and `simulate()`. These two operations are defined within an `operations.py` module:

```python
# operations.py
import hoomd
import hoomd.md

def setup(job):
    """Setup the initial conditions"""
    hoomd.init.create_lattice(unitcell=hoomd.lattice.sc(a=1.0), n=16)
    hoomd.dump.gsd(filename=job.fn("init.gsd"), ...)

def simulate(job):
    """Execute MD simulation"""
    with job:
        hoomd.init.read_gsd("init.gsd")
```

```
14          # [...]
15          lj = hoomd.md.pair.lj(r_cut=job.sp.r_cut, ...)
16          lj.pair_coeff.set(
17              "A", "A",
18              epsilon=job.sp.epsilon,
19              sigma=job.sp.sigma)
20          hoomd.md.integrate.langevin(kT=job.sp.kT, seed=job.sp.seed, ...)
21          hoomd.dump.gsd("trajectory.gsd", period=2e3, ...)
22          hoomd.run(tsteps=1e6)
23          job.document["step"] = hoomd.get_step()
24
25  if __name__ == "__main__":
26      import flow
27      flow.run()
```

The last three lines (l. 25-27) leverage `signac-flow`'s function to equip this module with a command line interface that allows us to execute all operations directly from the command line:

```
$ python operations.py setup
$ python operations.py simulate
```

To further automate the execution of operations and their submission to an HPC cluster, we can implement a workflow as part of a FlowProject as described in Section 4.3.3. The workflow is defined by adding operations to the FlowProject class with the `@operation` function decorator. Each operation can be associated with pre- and post-conditions to determine their order of execution. An operation is eligible to be executed when all pre-conditions are met and at least one of the post-conditions is not met. The execution conditions associated with each operation are implemented as methods, which are then passed as arguments to the `@pre` and `@post` function decorators.

For this example, we would want to execute the `setup` operation first, and then assuming that was successful, the `simulate` operation. A simple condition for successful setup would therefore be the existence of the `init.gsd` file, which contains the system's initial configuration, so we set that as the `post` condition *via* the `initialized` function. We also keep track of the *simulation progress* by storing the current simulation time step within the persistent JSON storage associated with the job (`job.document`).

```
# project.py
import flow

class Project(flow.FlowProject):
    pass

@Project.label
def initialized(job):
    return job.isfile("init.gsd")

@Project.operation
@Project.post(initialized)
def setup(job):
    # ...

@Project.label
def simulated(job):
    return job.doc.step >= 1e6

@Project.operation
@Project.pre.after(setup)
@Project.post(simulated)
def simulate(job):
    # ...

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

In addition to clearly defining the status of each individual operation, this FlowProject implementation also describes all valid sequences of operations. For example, because the `setup` operation has no pre-conditions, it is the only operation eligible for execution immediately after data space initialization. Since the FlowProject encapsulates this logic, we can trivially execute our workflow by leveraging the FlowProject's *run* capabilities, which take the simple run functionality from our `operations.py` script one step further. Rather than specifying operations to run, we can now simply execute `$ python project.py run`, which will automatically run the next eligible operation for each job. To submit these job-operations to a job scheduler on a HPC cluster, we could instead use `signac-flow`'s submission tool by typing `$ python project.py submit`. Note that the operations of this workflow are here defined as Python functions, but could just as well be an arbitrary terminal command, *e.g.*:

```
@Project.operation
@flow.cmd
def idg(job):
    return "./idg {job.sp.N} {job.sp.kT} {job.sp.p}".format(job=job)
```

4.6 Conclusions

The development of `signac` is motivated by the increased need for the management of heterogeneous and complex data spaces in computational materials science, specifically in work requiring HPC resources. Researchers in computational fields are frequently required to manage such data spaces and account for the various issues associated with this task. The `signac` framework provides non-intrusive solutions to many data management and workflow challenges in environments scaling from desktops to HPC clusters. The simple file-centric data model and the use of standard file formats such as JSON ensure easy access and portability of both the data and the associated workflows. This portability is particularly critical for sustainable long-term storage, since it allows the use of `signac` without tying users to future use of the platform or specific file formats in order to be able to access the data. The indexing functionality eases the transition from data acquisition to curation and analysis, and the simplicity of export to databases allows the integration of existing DBMS into HPC workflows. These functions allow `signac` to combine the advanced metadata handling capabilities of modern DBMS while also providing the performance of file system-based solutions. By providing a lightweight, high-performance solution to common data management and workflow challenges in HPC, the `signac` framework frees researchers from solving these problems themselves and enables more effective and efficient scientific research.

Acknowledgments

We would like to thank all contributors to the development of the framework's components, J.A. Anderson, M.E. Irrgang and P. Damasceno for fruitful discussion, feedback and support, and B. Swerdlow for his contributions and feedback and coming up with the name. Finally, we would like to thank all early adopters that provided feedback and thus helped in guiding and improving the development process. Development and deployment supported by MICCoM, as part of the Computational Materials Sciences Program funded by the U.S. Department of Energy, Office of Science, Basic Energy Sciences, Materials Sciences and Engineering Division, under Subcontract No. 6F-30844. Project conceptualization and implementation supported by the National Science Foundation, Award # DMR 1409620.

Bibliography

[1] Steve Plimpton. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *J. Comput. Phys.*, 117(1):1–19, 1995. doi: 10.1006/jcph.1995.1039.

[2] Joshua A. Anderson and Sharon C. Glotzer. The development and expansion of HOOMD-blue through six years of GPU proliferation. *arXiv:1308.5587*, 2013.

[3] Joshua A. Anderson, Eric Jankowski, Thomas L. Grubb, Michael Engel, and Sharon C. Glotzer. Massively parallel Monte Carlo for many-particle simulations on GPUs. *J. Comput. Phys.*, 254: 27–38, 2013. ISSN 00219991. doi: 10.1016/j.jcp.2013.07.023.

[4] Joshua A Anderson, M. Eric Irrgang, and Sharon C Glotzer. Scalable Metropolis Monte Carlo for simulation of hard shapes. *Comput. Phys. Commun.*, 204:21–30, 2016. doi: 10.1016/j.cpc.2016.02.024.

[5] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1-2:19–25, 2015. ISSN 23527110. doi: 10.1016/j.softx.2015.06.001.

[6] Michael Shirts and Vijay S. Pande. Screen Savers of the World Unite! *Science*, 290(5498): 1903–1904, 2000. ISSN 00368075. doi: 10.1126/science.290.5498.1903.

[7] John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaither, Andrew Grimshaw, Victor Hazlewood, Scott Lathrop, Dave Lifka, Gregory D Peterson, Ralph Roskies, J Ray Scott, and Nancy Wilkins-Diehr. XSEDE: Accelerating Scientific Discovery. *Comput. Sci. Eng.*, 16: 62–74, 2014. doi: 10.1109/MCSE.2014.80.

[8] Anubhav Jain, Shyue Ping Ong, Geoffroy Hautier, Wei Chen, William Davidson Richards, Stephen Dacek, Shreyas Cholia, Dan Gunter, David Skinner, Gerbrand Ceder, and Kristin A. Persson. Commentary: The Materials Project: A materials genome approach to accelerating materials innovation. *APL Mater.*, 1:011002, 2013. doi: 10.1063/1.4812323.

[9] MongoDB, Inc. MongoDB, 2016. URL `https://www.mongodb.com/`. accessed on 2017/09/29.

[10] Oracle Corporation. MySQL, 2016. URL `https://www.mysql.com`. accessed on 2017/09/29.

[11] H M Berman, John Westbrook, Zukang Feng, Gary Gilliland, T N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. The Protein Data Bank. *Nucleic Acids Res.*, 28(1):235–242, 2000. ISSN 13624962. doi: 10.1093/nar/28.1.235.

[12] Frank H. Allen. The Cambridge Structural Database: a quarter of a million crystal structures and rising. *Acta Crystallogr. Sect. B Struct. Sci.*, 58(3):380–388, 2002. ISSN 0108-7681. doi: 10.1107/S0108768102003890.

[13] Colin R. Groom and Frank H. Allen. The Cambridge Structural Database in Retrospect and Prospect. *Angew. Chemie Int. Ed.*, 53:662–671, 2014. ISSN 14337851. doi: 10.1002/anie.201306438.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conf. Comput. Vis. Pattern Recognit.*, pages 248–255. IEEE, 2009. ISBN 978-1-4244-3992-8. doi: 10.1109/CVPR.2009.5206848.

[15] The iRODS Consortium. Integrated Rule-Oriented System (iRODS), 2016. URL `http://irods.org`. accessed on 2017/09/29.

[16] Ian Foster. Globus online: Accelerating and democratizing science through cloud-based services. *IEEE Internet Comput.*, 15(3):70–73, 2011. ISSN 10897801. doi: 10.1109/MIC.2011.64.

[17] D L Dotson, S L Seyler, M Linke, R J Gowers, and O Beckstein. datreant: persistent, pythonic trees for heterogeneous data. In S Benthall and S Rostrup, editors, *Proceedings of the 15th Python in Science Conference*, pages 51–56, Austin, TX, 2016.

[18] Anand Kumar, Vladimir Grupcev, Meryem Berrada, Joseph C Fogarty, Yi-Cheng Tu, Xingquan Zhu, Sagar a Pandit, and Yuni Xia. DCMS: A data analytics and management system for molecular simulation. *J. Big Data*, 2(1):9, 2014. ISSN 2196-1115. doi: 10.1186/s40537-014-0009-5.

[19] Giovanni Pizzi, Andrea Cepellotti, Riccardo Sabatini, Nicola Marzari, and Boris Kozinsky. AiiDA: automated interactive infrastructure and database for computational science. *Comput. Mater. Sci.*, 111:218–230, 2016. ISSN 09270256. doi: 10.1016/j.commatsci.2015.09.013.

[20] G. Brandl and the Sphinx team and The Pocoo Team. Sphinx Documentation, 2016. URL `http://www.sphinx-doc.org`. accessed on 2017/09/29.

[21] Robert Martin. The Clean Architecture, 2012. URL `https://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html`. accessed on 2017/09/29.

[22] 3T Software Labs GmbH. Studio 3T, 2017. URL `https://studio3t.com`. accessed on 2017/09/29.

[23] Mayeul d'Avezac. pylada-light documentation, 2017. URL `http://pylada.github.io/pylada-light`. accessed on 2017/12/18.

[24] Anubhav Jain, Shyue Ping Ong, Wei Chen, Bharat Medasani, Xiaohui Qu, Michael Kocher, Miriam Brafman, Guido Petretto, Gian-Marco Rignanese, Geoffroy Hautier, Daniel Gunter, and Kristin A. Persson. Fireworks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, 27(17):5037–5059, 2015. ISSN 1532-0634. doi: 10.1002/cpe.3505.

[25] A P Davison. Automated capture of experiment context for easier reproducibility in computational research. *Comput. Sci. Eng.*, 14:48–56, 2012. doi: 10.1109/MCSE.2012.41.

[26] Rina A Doherty and Paul Sorenson. Keeping users in the flow: Mapping system responsiveness with user experience. *Procedia Manufacturing*, 3(Supplement C):4384–4391, 2015. doi: 10.1016/j.promfg.2015.07.436.

[27] Joshua A. Anderson, Chris D. Lorenz, and A. Travesset. General purpose molecular dynamics simulations fully implemented on graphics processing units. *J. Comput. Phys.*, 227(10):5342–5359, 2008. ISSN 00219991. doi: 10.1016/j.jcp.2008.01.047.

[28] Jens Glaser, Trung Dac Nguyen, Joshua A Anderson, Pak Lui, Filippo Spiga, Jaime A Millan, David C Morse, and Sharon C. Glotzer. Strong scaling of general-purpose molecular dynamics simulations on GPUs. *Comput. Phys. Commun.*, 192:97–107, 2015. ISSN 00104655. doi: 10.1016/j.cpc.2015.02.028.

CHAPTER V

On the Development of Scientific Software in Academic Research Groups

This chapter is adapted from a publication in *Computing in Science & Engineering* in 2019 as part of a special issue on *Accelerating Scientific Discovery With Reusable Software* authored by me and Vyas Ramasubramani with equal contributions and co-authored by Joshua A. Anderson and Sharon C. Glotzer. The article was assigned DOI:10.1109/MCSE.2018.2882355.

5.1 Introduction

As computational science continues to play a rapidly growing role across nearly all scientific disciplines, quality scientific software becomes ever more critical to research productivity [1]. The development of such software is impeded by the fact that most scientific software is developed by non-experts for whom the primary product is not the software, but the resulting science. The existing incentive structure, which is geared towards rapid publication of scientific results, tends to motivate the creation of single-use software. In addition to being highly targeted at a specific application, single-use software often also disregards established software development best practices such as proper version control and documentation, impeding its reuse for future studies and ultimately hindering long-term scientific progress.

Software solutions for many-body simulations present a useful case study of this development pattern. Over time, the range of scales and physical phenomena of interest have led to the creation of numerous packages with overlapping yet distinct feature sets. Despite this fragmentation, however, certain packages have attracted much larger user communities than others, suggesting a slow progression towards specific reusable solutions.

We present an approach, *lazy refactoring*, for accelerating this progression while ensuring that code development is always targeted at an *immediate* scientific objective. This approach—especially suited for researchers, who need to reconcile sustainable software development with the need to make immediate scientific progress—advocates that initial development should always lead to single-use code, but this code is refactored into a reusable solution *as soon as* two further uses for it are found [2]. We discuss how software standards such as modular design and well-defined interfaces, tools such as VCS and CI, and proper code documentation help facilitate lazy refactoring. With this context, we offer a concrete methodology to help determine when and how to write new code.

As an example of lazy refactoring, we present the development progression of software developed by the Glotzer Group at the University of Michigan. We are a diverse, collaborative research group comprised of 30+ graduate students, post docs, and research scientists from chemical engineering, materials science, physics, and other disciplines. Roughly half our members join the group with little to no simulation experience, and very few have developed professional quality code prior to joining. As such, we are a representative sample of the computational research community within academia. We show how lazy refactoring evolved alongside our development efforts and demonstrate its use to create a domain-specific, reusable, and loosely integrated software stack. We close with a brief discussion of how we train group members to foster the presented techniques.

5.2 Developing computational solutions

The goal of lazy refactoring is to minimize the total resources used on tool development for a computational problem within the context of a preexisting *software ecosystem*, which refers to the set of all available software. While much of this code will come in the form of clearly defined, well-documented, and consistently maintained software *packages*, the ecosystem will also contain harder to reuse special-purpose codes that may nevertheless be useful beyond their original intent. This second kind of software, commonly referred to as a *prototype*, generally lacks clean interfaces and documentation, and does not have explicit names or release versions. While packages are easily reusable, prototypes offset this benefit by being faster to implement and requiring less effort to maintain. Borrowing from well-established agile software development techniques [3] such as extreme programming (XP) [4], lazy refactoring ensures that the ability to respond rapidly to changing requirements and maintaining working software are prioritized over planned design and comprehensive documentation [5]. One critical difference here is that the programmer of scientific software aimed at solving a particular scientific problem is usually also the only customer.

In principle, total development efforts are minimized by using existing code whenever possible, and, when new code is required, expending effort to make code reusable only when reuse is expected. From this perspective, solving a computational problem starts by decomposing it into components that each require a software solution. This process helps identify which parts are already solved within the ecosystem, which ones can be addressed by adapting existing tools, and which ones require entirely new code. Then, the missing components can be assessed to determine which ones merit developing reusable packages and which ones should be developed as prototypes. We use the term *adapter* to refer to code whose primary purpose is to interface between other packages.

In practice, however, making an optimal decision about when to develop reusable code is generally impossible. The problem scope may be ill-defined and future project plans change, invalidating total resource estimates and making it difficult to determine which components are worth developing as packages. In this light, it is clear that developing reusable code packages any time reuse is foreseen is often an inefficient use of resources.

Instead, we recommend the more agile lazy refactoring approach, in which *all* missing code is developed as a prototype in the most convenient manner possible, *e.g.*, in Python [6,7]. Refactoring of this code should always be motivated by its imminent application to new problems, but refactoring should be preferred over new prototype development once such applications are identified. Specifically, we advocate following the Rule of Three: a prototype should be refactored as soon as a *third* application is found. We illustrate this approach in Figure 5.1, where there are two alternatives to address the additional software needs of Problem B after previously solving Problem A. If no prototypes exist, the problem should be solved with rapid development of prototypes $\mathbf{x}$ and $\mathbf{y}$, tailored to the specific problem (*Alternative A*). If two or more prototypes already exist, then the gap should be closed by refactoring these into package $\mathbf{d}$ (*Alternative B*). We argue that, on average, this approach minimizes total software development effort since the resources required for the design, implementation and maintenance of software packages are only expended once the future reuse potential is sufficiently apparent.

The advantages of lazy refactoring are manifold:

1. Total resource investment into solving one problem is minimized.

2. Refactoring a prototype into a package is usually easier than writing a package from scratch because it postpones developing interfaces and defining the scope until the problem is better understood.

3. The package can be validated against the original prototype.

4. The decision to refactor can account for actual projects that arise rather than attempting to estimate future reusability.

Lazy refactoring does carry significant risks: key personnel involved with the original development may no longer be available for refactoring, or the prototype may grow so large and opaque that refactoring itself presents a significant barrier. Therefore, the Rule of Three should be applied rigorously such that refactoring occurs in a timely manner and the original developers can be heavily involved in the refactoring process. This method requires that some quality controls be imposed on even initial prototypes to ensure that refactoring remains possible.

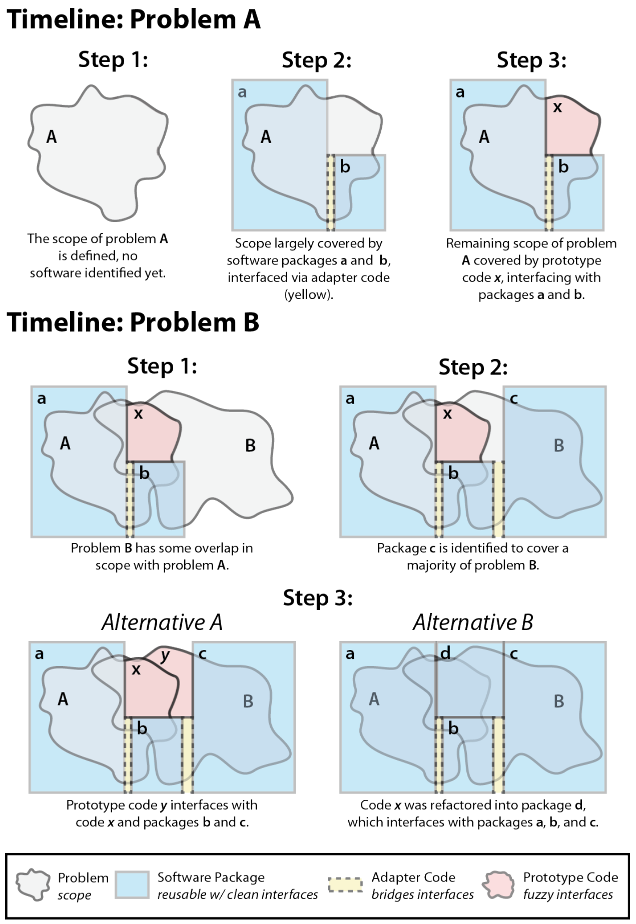

Timeline: Problem A
Step 1:
Step 2:
Step 3:
A
a
b
x
The scope of problem A is defined, no software identified yet.
Scope largely covered by software packages a and b, interfaced via adapter code (yellow).
Remaining scope of problem A covered by prototype code x, interfacing with packages a and b.
Timeline: Problem B
Step 1:
Step 2:
B
c
Problem B has some overlap in scope with problem A.
Package c is identified to cover a majority of problem B.
Step 3:
Alternative A
Alternative B
y
d
Prototype code y interfaces with code x and packages b and c.
Code x was refactored into package d, which interfaces with packages a, b, and c.
Problem scope
Software Package reusable w/ clean interfaces
Adapter Code bridges interfaces
Prototype Code fuzzy interfaces

Figure 5.1: This figure illustrates the code development progression associated with solving two related computational problems. Problem **A** is solved using packages **a** and **b** in combination with some adapter code and the prototype code base ***x***. Problem **B** is determined to overlap with problem **A**, and large portions of it are already solved by packages **b** and **c**. There are two alternatives for addressing the remaining software needs for problem **B**. The first alternative is to reuse ***x*** and develop prototype ***y*** to fill the gap (*Alternative A*). The second approach is to refactor prototype ***x*** into package **d**, which interfaces with packages **a**, **b**, and **c** and fully solves problem **B** (*Alternative B*).

5.3 Principles, Tools and Practices

In this section, we expound upon several principles, development tools and best practices that aid in effectively applying lazy refactoring. As we will show, these principles, tools, and practices are also useful barometers for evaluating preexisting software solutions, as will be shown later.

5.3.1 Principles

Modularity, the core design principle that we advocate, is a well-known standard enshrined in the UNIX philosophy that naturally leads to the development of tools with clearly delimited scopes and well-defined Application Programming Interfaces (APIs), each of which confer significant benefits. Code with limited scope can be developed more quickly, and it may preclude further time expenditures if the initial work proves sufficient for the task at hand. The API-driven design model also ensures that different components can be easily used independently of one another, making it easier to generalize and refactor specific pieces or integrate them with other tools. This feature is especially crucial in science because specific research applications often require combining previously unrelated, highly domain-specific software.

The first step towards modularity is defining a clear division between software components. Defining these divisions immediately suggests the appropriate scopes for each component, the specific interoperability requirements, and the necessary API to meet these requirements. In Section 5.5 we show how our group's stack is largely composed of modular packages with well-defined APIs that employ standard data formats and operational paradigms for easy integration.

Our other guiding principle is the public release of software source code. Although some code may be privately maintained to preserve a competitive advantage in the short term, the need to validate scientific results and the need to integrate disparate software tools make the ability to view source code more valuable than ever. These are strong incentives for open-source scientific software development.

5.3.2 Tools and Best Practices

We employ a number of tools and best practices to ensure that our code adheres to the principles expressed in the previous section. Version Control Systems (VCSs), which enable the robust management of a project's evolution over time, are especially important in scientific code development due to its collaborative nature. VCSs enable scientists to work in parallel while maintaining stable, working versions of code at all times. This collaborative process is much better suited to the decentralized model of Distributed Version Control Systems (DVCSs), such as Git, than to the centralized approaches of traditional VCSs like Subversion (SVN). The benefits of a controlled, shareable central repository can be recovered by hosting code on public repositories such as GitHub, Bitbucket, or GitLab.

Some code quality checks can be automated using code linters, such as Flake8 for Python, splint for C, or Cppcheck for C++. Higher level checks can solicit input by using, for instance, pull request-based workflows to require human reviews before new code is added to an existing code base. Code should include unit and integration tests, which increase confidence in code correctness and reduce the likelihood of introducing breaking changes. Code should also employ Continuous Integration (CI) [8], the automatic execution of tests to ensure that any errors or issues are caught before spreading to stable code versions. Many CI services, such as Jenkins, CircleCI, and Bitbucket Pipelines, integrate with repository hosting services and are free to publicly available repositories, further incentivizing open-source development.

To maximize reusability, software must be widely disseminated and painless to install. At the very least, source code should be publicly available, but standardizing modes of distribution through, *e.g.*, package managers, is the preferred mode of making the software readily available. For example, most open-source packages published by our group are distributed *via* the PyPI and Anaconda cloud repositories. For distribution on specialized High Performance Computing (HPC) environments, we currently host Docker images on the Docker container hub, which we deploy using Singularity on environments such as the TACC Stampede2, PSC Bridges, and SDSC Comet clusters available through XSEDE [9].

Software must come with effective documentation, which includes both guidance for high-level usage and more detailed information on the package's various components such as its API.

Documentation quality can be greatly improved by taking into account user feedback, which can be obtained through surveys and focus group sessions along with more asynchronous mechanisms like issue trackers, mailing lists, and chat rooms. As with CI, online documentation hosting services such as ReadTheDocs are free for open-source software to simplify the publishing of high-quality documentation.

Finally, software must be appropriately licensed. Terms and conditions for using unlicensed code may be unclear, so software (whether open- or closed-source) must have licenses that permit reuse. For open-source software, for instance, the Open Source Initiative (OSI) approves certain licenses as providing sufficiently free usage and modification. Our group typically licenses software under the BSD 3-clause or the MIT license.

5.4 Applying lazy refactoring

While we have described an agile approach to code development for a research problem, to this point our formulation has been purposely abstract, providing only a conceptual description and high-level guidelines. We now describe the practical application of lazy refactoring to solving a particular problem. Using the ideas espoused by the previous section as a guide, we identify specific attributes by which existing software may be assessed and that should be incorporated into any new software. We then present a process for rigorously and systematically determining exactly when and how much new code development is merited, using the identified attributes to assess the usability and integrability of external code bases. Note that the needs of a typical project will evolve over its lifetime, and will therefore require applying this heuristic multiple times.

5.4.1 Critical Attributes

Any software that is considered for integration into a problem solution workflow should be assessed according to the following critical attributes:

1. Scope: Does it solve the problem at hand?

2. Integrability: How well does it integrate into the existing software stack, including external tools upon which this stack relies, and how much work would be required for integration?

3. Stability: Does it have a well-defined, static API, is it largely error-free, and is it likely to be maintained into the foreseeable future, or at the very least, over the project lifetime?

4. Security: Does it pose any security risks?

These attributes are discussed in detail in the following subsections.

5.4.1.1 Scope

In order for software to be useful, it must solve the problem at hand. Although this statement appears trivial, its simplicity hides some important nuances. A complete solution must satisfy, for instance, scalability and performance requirements. These factors are context-dependent; for example, simulations may need to scale to leadership-class HPC platforms and run fast even for systems with millions of particles. Such considerations may disqualify a candidate code base that solves the problem in a limited set of cases.

5.4.1.2 Ease of integration

The next step is determining how easily the software can be integrated into existing workflows. As discussed earlier, license compatibility is a significant factor in this determination and may be a reason to prefer open-source software. Open-source software has the additional benefit that the code can be inspected and adapted if needed (see Section 5.4.2.2).

In addition to license compatibility, software compatibility also encompasses modularity, data types, and file formats. In order for different components of a software pipeline to work together, they must communicate in some common language, *i.e*, they must be able to exchange data in a standardized manner. For example, all numerical analysis software implemented in Python should make use of NumPy [6,7], which is the *de facto* standard for arrays of numerical data within the Python ecosystem.

5.4.1.3 Stability

High quality software must also be sufficiently stable. This means that API and code paths are set, further dependencies are unlikely to be introduced, and existing features are provided with the implicit or explicit promise that they will continue to work. Furthermore, stable packages

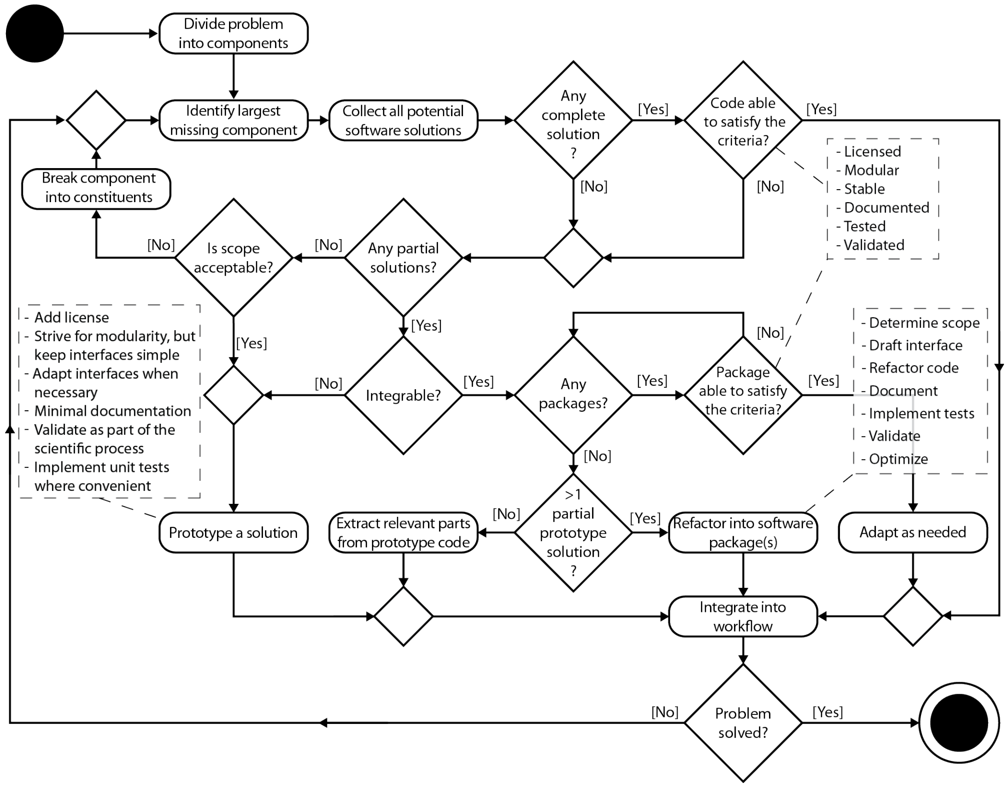

Divide problem into components
Identify largest missing component
Collect all potential software solutions
Any complete solution ?
[Yes]
Code able to satisfy the criteria?
[Yes]
[No]
[No]
- Licensed
- Modular
- Stable
- Documented
- Tested
- Validated
Break component into constituents
[No]
Is scope acceptable?
[No]
Any partial solutions?
[Yes]
[Yes]
[No]
- Add license
- Strive for modularity, but keep interfaces simple
- Adapt interfaces when necessary
- Minimal documentation
- Validate as part of the scientific process
- Implement unit tests where convenient
[No]
Integrable?
[Yes]
Any packages?
[Yes]
Package able to satisfy the criteria?
[Yes]
- Determine scope
- Draft interface
- Refactor code
- Document
- Implement tests
- Validate
- Optimize
[No]
Prototype a solution
Extract relevant parts from prototype code
[No]
>1 partial prototype solution ?
[Yes]
Refactor into software package(s)
Adapt as needed
Integrate into workflow
[No]
Problem solved?
[Yes]

Figure 5.2: This flow chart depicts the core elements of a basic decision tree for the development of workflows that solve a specific computational problem. In summary, we break down the problem into individual components that may or may not have existing solutions. Existing solutions are evaluated for use, and where appropriate, prototype solutions are either written or existing prototypes are refactored into packages. The decision on what solutions to include is largely based on the criteria outlined in "Applying Lazy Refactoring", which serve as basic guidelines for evaluating existing solutions for their integrability into the proposed workflow.

must be well-supported and have good documentation, and they should have active and responsive developers. Additional indicators of stability are active mailing lists, issue trackers, and forums.

5.4.1.4 Security

Last but not least is the consideration of security risks posed by potential external software solutions. For instance, any software that requires a privileged execution context may be problematic, especially in HPC environments. Additionally, software that communicates with internet servers, including cloud computing services, may be unsuitable for handling sensitive data. These services are commonly not compliant with rules and regulations implemented by the Health Insurance Portability and Accountability Act (HIPAA) in the U.S. and similar legislation in other countries.

5.4.2 General Heuristic

We now formulate a decision tree to guide the software-related decisions involved in solving a scientific problem (see Figure 5.2). This process involves decomposing a problem into components that can be solved sequentially. A critical part of the process is the usage of preexisting *partial* solutions, which are often originally developed as single-use prototypes and are ripe for refactoring. Although such refactoring can be difficult, eventually it will pay dividends as the quality of the software ecosystem increases and software becomes easier to refactor and adapt. NumPy is one example of how two partial solutions, Numarray and Numeric, were consolidated into an improved package that shares both their strengths.

In some circumstances, writing partially redundant software may be justifiable. Git is one example of a highly successful package that was developed despite the existence of other VCSs because those tools failed to support highly distributed, branch-heavy workflows. Any such development, however, should be driven by a clear shortcoming of existing software.

5.4.2.1 Checklist for software integration

Once potential complete or partial solutions have been identified, they must be individually checked to see whether the following criteria can be satisfied:

- ☐ Ensure that the software is appropriately licensed to allow integration and modification.
- ☐ Ensure that the architecture and interfaces are designed for easy integration.
- ☐ Determine whether the software is actively maintained and whether code or documentation contributions would be accepted.
- ☐ Check whether the API is stable to estimate future maintenance effort.
- ☐ Confirm that all interfaces, the architecture and—in case modifications are required—the source code are sufficiently documented.
- ☐ Establish that the majority of the code base is properly tested and validated.

In many cases, software that does not satisfy these criteria can be brought into compliance with sufficient modification, either by the user (*e.g.* by writing documentation, adding tests, or maintaining a fork) or by contacting the maintainer (*e.g.* to add a license). If such modification ultimately proves insufficient to satisfy these criteria, however, the software should be discarded as a solution for the problem at hand. Note that all criteria outlined above extend to dependencies as well, so code bases with many dependencies must effectively clear a higher bar.

5.4.2.2 Guidelines for package adaptation

For packages that provide a near-complete solution but require some adaptation to completely address the relevant part of the problem, we have developed the following guidelines:

1. Plan development: Unless it conflicts strongly with the protection of intellectual property, obtain feedback on the proposed modifications and extensions from the current package maintainer.

2. Testing: Before modifying any code, implement any missing tests to ensure adequate coverage of code paths that may be modified.

3. Validation: Add integration tests as needed to completely verify the correctness of the original software when applied to the problem at hand.

4. Modularity: Refactor the code base such that all parts that require modification are mostly separated from those that do not.

5. Adaptation: Write only as much code as is needed to address the problem at hand without aiming for too much generality.

5.4.2.3 Steps for refactoring prototypes into packages

If there are no applicable packages, but there are *multiple* existing prototypes, then these should be refactored into a package. Before refactoring, all related prototypes and packages should be analyzed for interfaces, overlaps, strengths and weaknesses, with a special focus on relevant packages that failed the tests of the checklist for software integration. Such software, which may have failed due to, *e.g.*, licensing issues, likely contains a great deal of expertise and know-how that is invaluable for the design of any new package. It may also be worthwhile to seek additional information on these packages through public forums or issue trackers, or by reaching out to developers or experienced users for further insight. The new package may then be implemented according to the following steps:

1. Determine the scope and the software architecture. Identify the core logic and then build layers around it for additional functionality such that dependencies only point inwards as described in the *The Clean Architecture* [10] principle.

2. Draft the end user interface before implementation. Ensure that the most prominent use cases are supported by the drafted interface design.

3. Replace the prototype code with the package step by step within the original applications. Ensuring that the new code produces identical results at each step is an important part of the validation process.

4. Document all functions and interfaces well such that a user can understand and use the package without needing to consult any of the developers.

5. Write unit tests for all core functions and add integration tests for the main applications of the code.

6. After ensuring function and correctness, optimize critical code paths (*and only those paths*) if needed.

5.4.2.4 Guidelines for developing a new prototype solution

If new software is required, we impose the following minimal standards on the resulting prototype code to simplify its future refactoring if needed:

1. License: Any code base should be licensed. Such a license may, for instance, explicitly permit internal reuse and require acknowledgment of the original author. To simplify matters, a research group could agree on a general license that is assumed to apply to all code that is

not explicitly licensed otherwise. Organization-wide policies regarding intellectual property will apply.

2. Modularity: Code should be developed as modularly as possible to simplify potential refactoring at a later stage. However, this modularity should not come at the expense of simple, problem specific interfaces that streamline solutions to the problem at hand.

3. Stability: All code should be part of a version-controlled repository. Interfaces can be changed whenever convenient.

4. Documentation: The code should have enough internal documentation to allow developers to modify it as well as sufficient API documentation for prospective users familiar with the code base.

5. Validation: The code base should be validated as part of the research process, ideally against known results.

6. Testing: Unit tests are not necessary unless they aid in the development process.

5.5 The Glotzer Group Software Stack

We now trace the growth of the core software developed within our group. Although our initial development followed a far less structured pattern than lazy refactoring, the challenges we encountered and the experiences we accumulated informed the guidelines that we now follow. By describing our internal software ecosystem (illustrated in Figure 5.3), we aim to motivate our approach and provide a blueprint for how to use it to create a powerful, sustainable, and integrated software stack for domain-specific computational research.

For brevity, we restrict ourselves to software that is directly involved with the generation, organization and transformation of data. Therefore, we omit discussion of, for instance, all software used for producing illustrations or presentations. We also avoid discussing operating systems aside from noting that we primarily use UNIX-like systems such as various Linux distributions and Mac OS X.

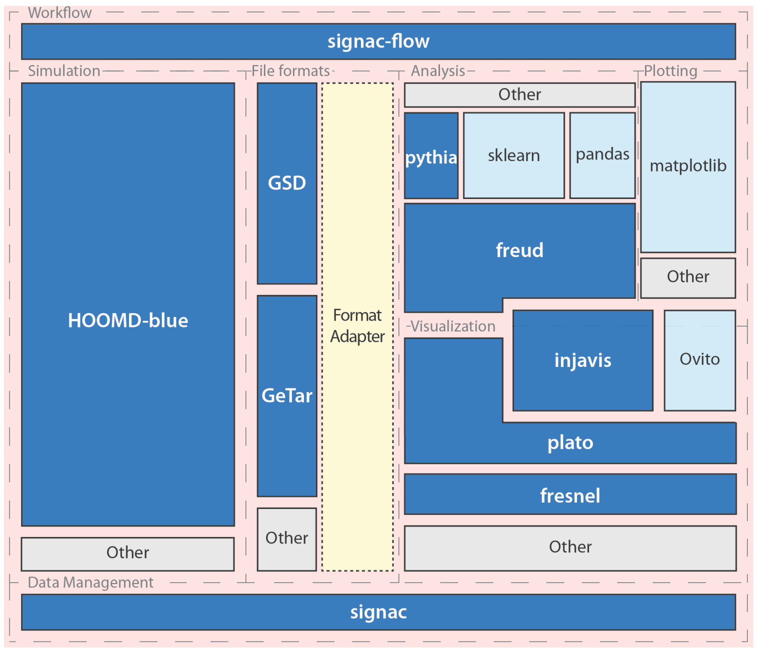


Figure 5.3: We illustrate here the relationships between the various components of our software stack. Software packages are grouped according to their overarching functions. Interactions are denoted by adjacency. Software that is developed within our research group is shown in dark blue with white labels. Workflows for computational projects are typically organized using signac-flow, which links together simulation, analysis, and plotting. The vast majority of our simulation needs are provided by HOOMD-blue, but we occasionally utilize other tools, for instance to perform atomistic simulations of proteins. Which visualization toolkit is used depends on the specific simulation, and occasionally also on whether the toolkit has the required analysis capabilities built-in. More generally, the problem at hand dictates the appropriate analysis software packages. We use various plotting tools, especially Matplotlib, to visualize the evolution of both raw quantities (such as total system energy) as well as the outputs of more complex analyses. All data generated throughout this process, including raw simulation trajectories as well as the outputs from complex analyses, is stored and managed using signac.

5.5.1 Simulation

The genesis of our lazy refactoring approach lies in the consolidation of “single-use codes” written for various particle simulation techniques into a package called HOOMD-blue [11, 12]. HOOMD-blue evolved from HOOMD [11], a general purpose Molecular Dynamics (MD) program developed for bead-spring polymer models that was the first MD package to run completely on Graphics Processing Units (GPUs). HOOMD-blue exemplifies the paradigm of growth by refactoring and incorporation of external code. Today, HOOMD-blue runs many flavors of MD and MC simulations on both CPUs and GPUs, and it supports a rich variety of phenomena that were not originally envisioned. These features were all developed separately for specific research purposes, and the original implementations underwent many revisions before incorporation into a public release of HOOMD-blue.

Consolidating our development efforts has reduced development time, increased reproducibility, and improved code quality, allowing us to improve upon our original prototypes. Moreover, we have incorporated lessons learned from modifying preexisting toolkits that were not modular enough to allow easy modification or integration with our workflows. HOOMD-blue's highly structured, object-oriented design maximizes ease of modification, enabling the continued integration of new tools such as the Hard Particle Monte Carlo (HPMC) [13] module from a pre-existing code base [14].

With internals written in C++ and CUDA, HOOMD-blue is one of the fastest particle simulation tools available; it is one of NVIDIA's official benchmarks for new GPU hardware. Unlike toolkits such as Large-scale Atomic/ Molecular Massively Parallel Simulator (LAMMPS) [15] or the GROningen Machine for Chemical Simulations (GROMACS) [16], which use a specialized file format for simulation configuration, HOOMD-blue offers a full-featured Python interface for greater flexibility and ease of integration with other tools. An open-source toolkit, HOOMD-blue has benefited greatly from numerous contributions from its user base; at the time of writing, 29 developers external to the group have contributed to HOOMD-blue. HOOMD-blue also uses a number of external code bases, including: pybind11 for exporting C++ classes to Python; CUDA, cub and Thrust to utilize GPUs; the Message Passing Interface (MPI) to scale across multiple nodes; LLVM for Just-in-time (JIT) compilation; the Eigen linear algebra library; and cereal to serialize data for communication.

5.5.2 Data Analysis

For generic data analysis tasks, our group makes liberal usage of existing software, such as components of the SciPy ecosystem. In addition to NumPy, which we use extensively and have incorporated into many of our own open-source packages, we also use other SciPy tools for, *e.g*, optimization and working with spatial data. Jupyter [17] notebooks are central to our workflow, and we typically use the Matplotlib library [18] for plotting. For machine learning tasks, we typically develop methods using scikit-learn, Keras, and TensorFlow.

For analyses more specific to particle simulations, however, we found that many of the standard analysis methods we use—including correlation functions, order parameters, and more—had only been implemented in various prototype codes. Eventually, we collected our prototypes into freud [19], a Cython [20], C++, and Threading Building Blocks (TBB)-accelerated tool that, like HOOMD-blue, provides a transparent Python API to a high-performance back-end. While many methods have been refactored and added to freud, the package maintains a consistent API across these methods. Tasks that do not fit in freud have been placed in other, more specialized tools.

5.5.3 Visualization

Our simulation visualization codes have undergone the most consistent process of repeated refactoring of prototypes and partial solutions. Our simulations typically generate long trajectories of complex systems that require powerful software and hardware for effective visualization. HOOMD-blue provides output in formats that can be interpreted by well-known simulation visualization packages such as VMD [21], PyMol, and OVITO [22], but these tools have up until recently been generally incapable of visualizing anisotropic particles, which require interpretation of their orientations in addition to their positions. Therefore, we used injavis, a previously developed Java visualization and analysis tool that represented particle data using XYZ-coordinates. Our group members extended this software to account for particle orientations, and we enabled HOOMD-blue to provide output data in this format. As a largely Graphical User Interface (GUI)-centric application, however, its analysis capabilities can be difficult to integrate into scripted workflows.

The goal of refactoring the functions in injavis into a more scriptable tool was a driving force for the development of freud. To leverage freud's analysis capabilities for visualization, we developed a visualization toolkit called plato that outsources its analysis functions to freud. This division makes plato easier to adapt for new visualization tasks, resulting in a more modular, maintainable software infrastructure. In addition to plato, we developed fresnel to generate publication quality images using a GPU-accelerated path tracing engine.

5.5.4 Data and Workflow Management

A comprehensive simulation study typically involves conducting many simulations, often using HPC clusters to achieve meaningful system dimensions and time scales. Although HOOMD-blue scripts accommodate user-defined parameters, the user remains responsible for reliably associating these parameters with the simulation outputs and subsequent analyses. Many group members developed prototype solutions , but they suffered from numerous drawbacks with respect to scalability, flexibility, and interpretability.

To resolve these problems, we developed signac [23], an open-source framework for constructing complex workflows on large, heterogeneous data spaces. Inspired by these prototypes, signac uses well-formed JavaScript Object Notation (JSON) parameter files to associated data files with their identifying parameters, providing a robust and full-featured database interface to data stored directly on the filesystem. The system is designed for the high performance filesystems inherent to HPC environments and supports the highly parallel file I/O operations required for, *e.g.*, MPI-enabled HOOMD-blue simulations. Since all data and metadata are stored directly on the filesystem, they can also be easily transferred using tools like 'rsync' or GLOBUS [24]. The signac framework includes signac-flow, a tool for managing and automating complex workflows operating on a signac data space. Workflows designed with signac-flow are immediately portable to HPC environments, making it possible to design and test workflows on local resources and then immediately submit them to a cluster scheduler.

5.5.5 Software Integration

The smooth functioning of our overall pipeline also depends on a number of smaller packages that perform more limited but equally important tasks. A good example of applying lazy refactoring is rowan [25], an open-source Python package for quaternion operations. A standard method for representing particle orientations in 3D, quaternions are used throughout our code base; however, individual packages have historically each had their own implementations. This fragmentation strongly suggested the need for standardization, particularly because many individuals also reimplemented these methods for their own, more *ad hoc* uses. Although quaternion packages already existed within the Python ecosystem, they suffered from numerous drawbacks with respect to performance and flexibility that made them unsuitable for incorporation into our code bases. rowan is the result of refactoring our own quaternion codes into a single package providing a unified API for working with quaternions at a uniform level of generality appropriate for our needs.

5.6 Training and Support

Our group has built an infrastructure around our software to provide users with training and information on how to take maximal advantage of our ecosystem. In addition to the documentation associated with each of our packages, we have created comprehensive integrated documentation that explains how to utilize our stack as part of a cohesive workflow, including a crash course to incrementally acquaint new users with this stack. We use Slack to maintain internal chat rooms for instant technical support, specialized discussions of science or software, and coordinating development.

We also aim to make our software useful to the broader community. We publish documentation through ReadTheDocs, distribute software *via* Anaconda and PyPI, and have published papers on some of our packages to further publicize them. We have also presented our software at the Scientific Computing with Python (SciPy) conference [26], the annual American Physical Society (APS) March meeting, the triennial Foundations of Molecular Modeling and Simulation Conference (FOMMS), and the annual meeting of the American Institute of Chemical Engineers (AIChE), to name a few. In addition to these presentations, which are one avenue for feedback, we also use surveys and focus groups to gather user feedback. Biannual hackathons where all group members participate in improving our code help to improve the distribution of code ownership.

5.7 Conclusions

Advances in computational science are heavily dependent on the ongoing evolution of the scientific software ecosystem. Although quickly produced *ad hoc* solutions may seem sufficiently expedient in the moment, scientific progress can often be accelerated by a judicious use of existing software. Consequently, time spent properly designing and developing reusable software solutions may prove worthwhile if the reuse potential is identified sufficiently early in the development process.

Properly estimating the future reuse potential of a piece of code is a daunting task, however, requiring a degree of foresight that is often impossible in scientific applications. Moreover, assessing the existing software options when embarking on a new project is far from trivial.

To address the need for a systematic approach to these problems, we have outlined a near optimal approach to determining when and how to develop reusable software that works well for us. The lazy refactoring approach is designed to balance sustainably improving the scientific software landscape with making immediate scientific progress. It does this by advocating for individual researchers to

1. evaluate existing software for its reuse potential prior to any code development,
2. adapt existing code bases for the problem at hand,
3. refactor existing code bases into proper packages whenever there are more than two use cases,
4. develop rapidly evolving prototype code strictly focused on solving the problem at hand in all other cases.

Lazy refactoring is particularly well suited to the academic research group. To illustrate its application in this context, we have shown how it has been applied over time by our research group to develop a suite of independent but highly interoperable and powerful tools targeted at a specific class of scientific problems. Through this, we provided a concrete example of how lazy refactoring may be used to significantly accelerate net scientific output in any subfield of computational science.

5.8 Acknowledgments

We would like to thank all contributors to the group's code base, including all members of the Glotzer group as well as the numerous external contributors to our software. We would like to thank Michael Engel for his development of injavis, some of which was done while in the Glotzer group. Development and deployment of signac supported by MICCoM, as part of the Computational Materials Sciences Program funded by the U.S. Department of Energy, Office of Science, Basic Energy Sciences, Materials Sciences and Engineering Division, under Subcontract No. 6F-30844.

Development and deployment of some aspects of HOOMD-blue and freud supported by the National Science Foundation, Award # DMR 1409620.

Bibliography

[1] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. Chue Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson, "Best Practices for Scientific Computing," PLoS Biol., vol. 12(1): e1001745, 2014.

[2] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, "Refactoring: Improving the Design of Existing Code," Addison-Wesley Professional, 1999.

[3] T. Dyb and T. Dingsyr, "Empirical studies of agile software development: A systematic review," Inf. Softw. Technol., vol. 50(9-1), pp. 833-859, 2008.

[4] L. Lindstrom and R. Jeffries, "Extreme Programming and Agile Software Development Methodologies," Inf. Syst. Manag., vol. 21(3), pp. 41-52, Jun. 2004.

[5] M. Fowler, and J. Highsmith, "Manifesto for Agile Software Development," http://agilemanifesto.org/, 2001 (accessed on 2018-09-29).

[6] T. E. Oliphant, "Python for Scientific Computing," Comput. Sci. Eng., vol. 9, no. 3, pp. 10-20, 2007.

[7] K. J. Millman and M. Aivazis, "Python for Scientists and Engineers," Comput. Sci. Eng., vol. 13(2), pp. 9-12, 2011.

[8] M. Fowler, "Continuous Integration," https://martinfowler.com/articles/continuousIntegration.html, 2006 (accessed on 2018-10-01).

[9] 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," Comput. Sci. Eng., vol. 16(5), pp. 62-74, 2014.

[10] R. Martin, "The Clean Architecture," https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html, 2012 (accessed on 2018/10/10)

[11] J. A. Anderson, C. D. Lorenz, and A. Travesset, "General purpose molecular dynamics simulations fully implemented on graphics processing units," J. Comput. Phys., vol. 227(10), pp. 5342-5359, 2008.

[12] 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," Comput. Phys. Commun., vol. 192, pp. 97-107, 2015.

[13] J. A. Anderson, M. E. Irrgang, and S. C. Glotzer, "Scalable Metropolis Monte Carlo for simulation of hard shapes," Comput. Phys. Commun., vol. 204, pp. 21 - 30, 2016.

[14] J. A. Anderson, E. Jankowski, T. L. Grubb, M. Engel, and S. C. Glotzer, “Massively parallel Monte Carlo for many-particle simulations on GPUs,” J. Comput. Phys., vol. 254, pp. 27 - 38, 2013.

[15] S. Plimpton, “Fast Parallel Algorithms for Short-Range Molecular Dynamics,” J. Comput. Phys., vol. 117(1), pp. 1-19, 1995.

[16] H. Berendsen, D. van der Spoel, and R. van Drunen, “GROMACS: A message-passing parallel molecular dynamics implementation,” Comput. Phys. Commun., vol. 91(1-3), pp. 43-56, 1995.

[17] F. Prez and B. E. Granger, “IPython: A System for Interactive Scientific Computing,” Comput. Sci. Eng., vol. 9(3), pp. 21-29, 2007.

[18] J. D. Hunter, “Matplotlib: A 2D Graphics Environment,” Comput. Sci. Eng., vol. 9(3), pp. 90-95, 2007.

[19] V. Ramasubramani, Bradley D. Dice, Eric S. Harper, Matthew P. Spellings, Joshua A. Anderson, Sharon C., “freud: A Software Suite for High Throughput Analysis of Particle Simulation Data,” arXiv:1906.06317 [cond-mat, physics:physics], 2019.

[20] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, “Cython: The Best of Both Worlds,” Comput. Sci. Eng., vol. 13(2), pp. 31-39, 2011.

[21] W. Humphrey, A. Dalke, and K. Schulten, “VMD - Visual Molecular Dynamics,” J. Mol. Graph., vol. 14, pp. 33- 38, 1996.

[22] A. Stukowski, “Visualization and analysis of atomistic simulation data with OVITO — the Open Visualization Tool,” Model. Simul. Mater. Sci. Eng., vol. 18(1), p. 015012, 2010.

[23] C. S. Adorf, P. M. Dodd, V. Ramasubramani, and S. C. Glotzer, “Simple data and workflow management with the signac framework,” Comput. Mater. Sci., vol. 146, pp. 220-229, 2018.

[24] I. Foster, “Globus online: Accelerating and Democratizing Science through Cloud-Based Services,” IEEE Internet Comput., vol. 15(3), pp. 70-73, 2011.

[25] V. Ramasubramani, Sharon C. Glotzer, “rowan: A Python package for working with quaternions,” J. Open Source Softw., 3(27), p. 787, 2018.

[26] V. Ramasubramani, C. S. Adorf, P. M. Dodd, B. D. Dice, and Sharon C. Glotzer, “signac: A Python framework for data and workflow management,” Proc. 17th Python Sci. Conf., pp. 91-98, 2018.

CHAPTER VI

Conclusions and Outlook

This dissertation research work was guided by the inverse design problem, which can be summarized as :

> What building blocks do we need to assemble materials with specifically targeted properties?

The properties of materials in the colloidal realm and in the absence of chemistry is largely determined by their structure, *i.e.*, the broken symmetries within the material. Structure is an emergent property that requires many individual building blocks to come together. Developing a method to design building blocks that will self-assemble a specifically targeted structure was therefore the primary motivation of the first part of my research on self-assembly.

In Chapter II of my dissertation, I presented the Fourier-filtered relative entropy minimization (FF-REM) method, a procedure for the optimization of isotropic pair potentials (IPPs) for the self-assembly of crystal structures. We were able to show that we can use the same procedure to find potentials for simple structures such as simple cubic ($cP1$) or body-centered cubic ($cI2$), but also more complex ones, *e.g.*, diamond ($cF8$), A15-type $cP8$-Cr_3Si ($cP8$), σ-phase $tP30$-CrFe ($tP30$), and clathrate-I $cP54$-K_4Si_{23} ($cP54$).

A crucial part of this work was to develop an algorithm that would modulate the Hamiltonian in such a way that its free energy landscape has a minimum at the targeted configuration while remaining sufficiently smooth. The smoothness criterion is what enables the self-assembly with sufficiently high yields.

This was achieved by deriving the algorithm within the general framework of relative entropy minimization directly in Fourier space. The mathematical formulation and implementation in Fourier space is more natural when working with configurations with long-range periodicity. Optimizing potentials directly in Fourier space also enables us to apply a filter function in a simple and elegant manner, which ensures that solutions are constrained in complexity. Many features in the form of local extrema make it more difficult to map the function to a physical particle.

An interesting opportunity to explore with respect to this work would be the optimization of IPPs that have two independent state points with two different crystal structures. That would allow us to, for example, design particles that self-assemble multiple different crystal structures as a function of temperature and pressure and have a well-defined freezing point.

The second part of my research on self-assembly is focused on understanding the nature and mechanisms of the self-assembly process. Chapter III is a study on crystallization pathways of representative benchmark and other model systems. We use a machine learning (ML)-driven approach in determining a holistic depiction of the pathway in the form of a manifold representation of the descriptor space. The descriptor space is a feature space that *describes* local particle environments over time. Essential components of that space are uncovered using Principal Component Analysis (PCA) and the manifold with reduced dimensionality is generated using the Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) algorithm.

The UMAP manifold is a collection of points, where each point represents one particle and its environment at a specific slice of its trajectory. This representation already gives us new insight into the assembly process. Phase changes are usually immediately apparent in the form of a large separation between different clusters. The number of smaller clusters and their connections is directly correlated to the number of local motifs present within the system.

All studied systems show significant local structure in the liquid, supporting the hypothesis by Kawasaki and Tanka [1] that liquids are highly structured to lower their free energy compared to a completely isotropic fluid. Furthermore, we observe that crystal nucleation is mediated by environments that are generally closer to the liquid structure. This is an example of two-step nucleation and in support of Ostwald's step rule which states that structures with a free energy closer to the liquid will form first even if they are not generally stable.

We observe two-step nucleation and growth with hexagonal close packing (HCP) forming earlier than face-centered cubic (FCC) for both the Weeks-Chandler-Andersen (WCA) and Lennard-Jones (LJ) systems. It would be very interesting to contextualize these phenomena within the theoretical framework laid out by James et al. [2]. They show that finite-size fluctuations play a crucial role in phase transitions, especially with respect to two-step nucleation. This means that fluctuations of local motifs facilitate the formation of crystal nuclei even if these motifs do not represent a metastable bulk phase.

This work could be substantially extended and its generality improved by using better heuristics for the determination of neighbors. For example, Voronoi-cell based algorithms [3] adapt the number of neighbors for each particle individually and could be implemented in freud [4].

The second part of my dissertation is focused on the technical aspects of computational work. I presented in Chapter IV the `signac` framework, which is a software framework for data and workflow management. Developing this software and facilitating its adoption presented its own set of challenges. Some of these challenges related to the development of software within an academic environment are discussed in Chapter V.

The original motivation of the `signac` project was to develop a software that would enable effective sharing of research data among my immediate peers. One specific driver of this was to make the data set related to the publication of Damasceno et al. on the self-assembly of convex polyhedra in Science 2012 [5] more accessible for continued research. Generating large data sets such as this one requires a lot of computational resources. Making data reusable and more sharable means that these resources can be used for other scientific goals and duplication of efforts is avoided.

However, very early on, I realized that software can only be one part of a solution to this problem. The second – and probably more important – part is related to data sharing culture. There are essentially three pillars needed to make data sharable and reusable:

- Access privilege – One must have the authorization to access and use the data.
- Technical access – One must have physical access to the data, *e.g.*, through the internet.
- Provenance – One needs to have sufficient context to understand what the source of the data is and how it was curated.

Many grants have stipulations that require researchers to have a data management plan and make their data publicly accessible. However, that usually only applies to data related to publications. Data that is never published is therefore most likely also never reused.

Providing technical access to data can be facilitated, for example, by using shared network file systems or by uploading data to internet repositories such as the University of Michigan Deep Blue Repository,[1] Zenodo,[2] or the Open Science Framework.[3] However, these repositories usually have data set size limitations well below the typical size of data sets within our field.

Finally, keeping track of the data provenance is a real challenge for highly exploratory and dynamic studies. Particle simulations within the soft matter field are typically highly customized to the problem at hand both in terms of the model and the simulation protocol. Developing standardized and extendable schemas that can capture all of these details is therefore very difficult.

One alternative to standardized schemas is to keep a copy of the source code that was used to generate the data in combination with all parameters used for a specific data set. The latter is the approach we take with `signac`. Users are encouraged to use their own metadata schema in combination with their own source code, however `signac` standardizes how these two components are combined. That makes it significantly easier to understand the general structure of the workflow and determine the provenance of a particular data set.

One crucial design goal for `signac` was to ensure that the implementation of workflows and the management of data spaces is so simple that the system can be easily used during all stages of a computational study. That means during early exploration and at production scales. This is important to avoid breaks and unnecessary redundancy in the workflow implementation, but also to facilitate the reuse of data sourced from preliminary studies that might never get published otherwise.

While `signac` has been a success when measured in terms of user adoption and recognition by organizations such as NumFOCUS,[4] there are also still a lot of challenges that need to be overcome. Users still face obstacles with respect to the implementation of inter-project workflows,

[1]https://deepblue.lib.umich.edu/

[2]https://zenodo.org

[3]https://osf.io

[4]The `signac` project is officially affiliated with the NumFOCUS organization as of this year.

i.e., workflows that operate on and curate data from multiple different projects. Similarly, workflows implemented with `signac-flow` that operate on more than one data set, *e.g.*, to compute collective statistics, are not yet directly supported.

The `signac` project has grown into a mature open-source project in the past couple of years, which means that many internal and external contributors have helped extend the packages' functionality, their documentation, and their exposure to the public. The increased size of the project has forced the core maintainer team to shift their attention from pure software development to the establishment of strategies to effectively integrate contributions and guide new contributors. I am confident that this transition facilitated by the increased involvement of the scientific community outside of our research group will vastly increase the longevity of the project.

Bibliography

[1] T. Kawasaki and H. Tanaka. Formation of a crystal nucleus from liquid. *Proceedings of the National Academy of Sciences*, 107(32):14036–14041, 2010. doi: 10.1073/pnas.1001040107.

[2] Daniella James, Seamus Beairsto, Carmen Hartt, Oleksandr Zavalov, Ivan Saika-Voivod, Richard K. Bowles, and Peter H. Poole. Phase transitions in fluctuations and their role in two-step nucleation. *J. Chem. Phys.*, 150(7):074501, 2019. doi: 10.1063/1.5057429.

[3] Jacobus A. van Meel, Laura Filion, Chantal Valeriani, and Daan Frenkel. A parameter-free, solid-angle based, nearest-neighbor algorithm. *J. Chem. Phys.*, 136(23):234107, 2012. doi: 10.1063/1.4729313.

[4] Vyas Ramasubramani, Bradley D. Dice, Eric S. Harper, Matthew P. Spellings, Joshua A. Anderson, and Sharon C. Glotzer. freud: A Software Suite for High Throughput Analysis of Particle Simulation Data. *arXiv:1906.06317 [cond-mat, physics:physics]*, June 2019. arXiv: 1906.06317.

[5] P. F. Damasceno, M. Engel, and S. C. Glotzer. Predictive Self-Assembly of Polyhedra into Complex Structures. *Science*, 337(6093):453–457, 2012. doi: 10.1126/science.1220869.