Electronic Supplementary Material (ESI) for Soft Matter.

Supplementary information for "Deep Learning for Characterizing the Self-Assembly of Three-Dimensional Colloidal Systems"

Jared O'Leary,[a] Runfang Mao,[b] Evan J. Pretti,[b] Joel Paulson,[c] Jeetain Mittal,[b] and Ali Mesbah[a]

Department of Chemical and Biomolecular Engineering, University of California, Berkeley, Berkeley, CA 94720, USA. Tel: +1-510-642-7998; E-mail: mesbah@berkeley.edu

Department of Chemical and Biomolecular Engineering, Lehigh University, Bethlehem, PA, 18015, USA. Tel: +1-512-699-4643; Email: jem309@lehigh.edu

Department of Chemical and Biomolecular Engineering, The Ohio State University, Columbus, OH, 43210, USA.

S1. Basics of Multi-Flavored Colloids

One way to promote the self-assembly of colloidal particles is through functionalization of their surfaces with DNA. DNA-functionalized particles (DFPs) interact with each other through complementary Watson–Crick base-pairing interactions and have been used to assemble many superlattice structures [1-2].

Typically, selective binding between DNA molecules tethered on two different particles is achieved in one of two ways. First, complementary single-stranded DNA (ssDNA) may be grafted on different particles so that they bind through direct hybridization with each other. Alternatively, this may be done indirectly by grafting the same ssDNA on both particles, and then introducing a complementary linker that can hybridize with the strands on both particles. Consequently, the unlike pairs effectively attract each other, whereas non-complementary like pairs repel each other due to steric interactions. In both instances, the interactions between like and unlike DFPs are not entirely independent of each other [2-4].

As a means of achieving this independence, it has recently been suggested that particles can be functionalized with a blend of two types of DNA strands with complementary concentrations on each particle. These ‘‘multi-flavored’’ particles can exhibit a tunable attraction between the like particles while maintaining the interaction between unlike pairs. Indeed, this approach has been shown to induce the crystallization of equally sized particles into BCC, HCP, and FCC structures. In this instance, the like and unlike interactions may be tuned independently. However, each like interaction is not independent of the other because the relative concentrations of the two strands are fixed [2, 5-6].

S2. Interaction model of Multi-Flavored Colloids

The self-assembly trajectories are obtained from binary colloidal mixtures representing multi-flavored DNA functionalized particles (DFPs) for which the attractive interaction between A-type and B-type particles (i.e., E_{AA}, E_{BB} and E_{AB}) can be adjusted independently Fig. S2.1 shows the schematic representation of the multi-flavored DFPs and the pairwise interaction model used in molecular dynamics (MD) simulations for obtaining these trajectories. The functional form of pair interaction utilized in these simulations is of a Fermi-Jagla type, which has previously been successfully used to study the self-assembly process of DFPs both in two and three dimensions [2, 28-29]:

$$U(r) = \varepsilon'(\frac{\sigma'}{r-s})^n + \frac{a_1}{1+\exp\left[b_1(1-c_1)\right]} + \frac{a_2}{1+\exp\left[b_2(1-c_2)\right]}$$

The first term controls the particle core-core repulsions and the remaining two terms together control the soft repulsion and attraction of particle surfaces due to the steric and hybridization interaction of DNA sequences. Here, the potential depth of E_{AA}, E_{BB} and E_{AB} can be varied independently based on a_2 parameter. For interactions between A-type and B-type particles, the a_2 is set as $-1.3219\ \varepsilon$ to control a minimum potential depth ε of E_{AB}. a_2 is modified for E_{AA} and E_{BB} to give E_{AA}/E_{AB} and $E_{BB}/E_{AB} \in [0,1]$. The parameters used in Fermi-Jalge potentials are listed as follows:

$\varepsilon' = 10\ \varepsilon$	$\sigma' = 0.2\ \sigma$
$n = 36$	$s = 0.8\ \sigma$
$a_1 = 11.035\ \varepsilon$	$a_2 \in [-1.3219\ \varepsilon, 0]$
$b_1 = 404.40\ \sigma^{-1}$	$b_2 = 1044.5\ \sigma^{-1}$
$c_1 = 1.0174\ \sigma$	$c_2 = 1.0306\ \sigma$

Indeed, this self-assembly approach based on tuning E_{AA}, E_{BB} and E_{AB} has been shown to induce the crystallization of equally sized DFPs into diverse categories of lattices such as BCC, HCP, and FCC structures, which is suitable for the purpose to train the autoencoder.

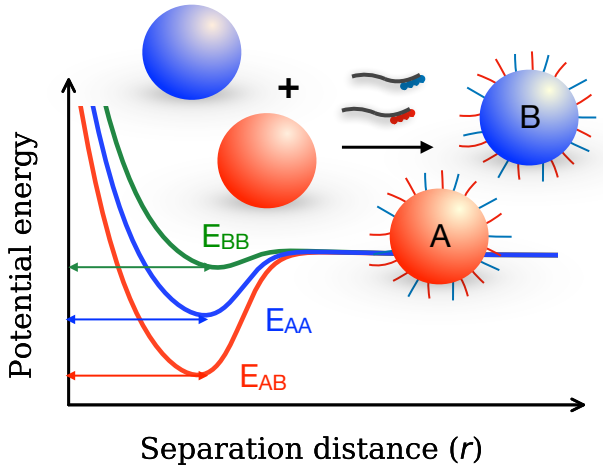

Figure S2.1. Schematic representation of multi-flavored DFPs and its effective pair potential model. Each of pairwise interaction strengths E_{AA}, E_{BB}, and E_{AB} can be manipulated experimentally by controlling the blending ratio of two different types of DNA sequences and it can be adjusted in simulations by changing the parameters of implicit Fermi-Jagla potential as illustrated above.

S3. Neighborhood Graph Construction

We employ the methodology described in [7-8] to obtain the neighbor list of topologically adjacent particles and subsequent neighborhood graph. The gist of the method is that the convex hull formed by the set of neighboring atoms describes the local structure around an atom. The convex hull is represented in the form of a neighborhood graph which is then used to classify the structure. The convex hull is determined from a Delaunay triangulation of the particle of interest and its first coordination shell (which is defined by its 18 nearest neighbors or half the inner shell atoms in FCC and HCP lattices). Because this method avoids the concept of bonds between particles and instead uses a geometry-based, fixed number of particles to establish the neighborhood, it is less sensitive to thermal fluctuations, density gradients, and anisotropy mentioned in the main text. Finally, this method includes the central particle in the neighborhood graph, which provides greater connectivity between neighbors and therefore greater distinction between structures in comparison to CNA and Steinhardt classification methods. Delaunay triangulation does yield inconsistent results at solid-vapor interfaces, however, as the method tends to connect far-away particles in order to create three-dimensional convex hulls. The authors in [7] use outlier detection techniques to filter these spurious results. As will be discussed in later sections, our proposed dimensionality reduction and classification techniques naturally filter such outliers effectively.

We evaluate the neighborhood graphs using the graphlet decomposition-based methodology of refs. [7-11], which has been successfully implemented for analyzing local structure in a variety of colloidal and biological networks. Graphlets are small, connected, non-isomorphic induced subgraphs of a larger network that contain some number of nodes, k. The k nodes in each graphlet are topologically distinguished by their individual automorphism orbits that account for the symmetries among the nodes in said graphlet. Each graphlet thus contains 1 to k-1 distinct automorphism orbits. The neighborhood graph is evaluated by computing the frequency of these orbits for a given neighborhood. For the purposes of this paper, each node is a particle within the neighborhood graph established by the Delaunay triangulation described above. We evaluate the neighborhood graph using graphlets with 2-5 nodes, as calculations involving larger graphlets quickly become intractable. Graphlets with 2-5 nodes display 73 different automorphism orbits. As a result, the local structure of each particle is quantified by a 73×1 vector (i.e., the neighborhood graph), where each entry in the vector refers to the frequency of an automorphism orbit. Following the procedure of [7-8] we additionally weigh the frequencies to account for the fact that the appearance of more complex automorphism orbits correlates with the appearance of simpler ones. Finally, each neighborhood graph is normalized such that its sum is unity.

S4. Conceptual Details Behind Autoencoder

An autoencoder is comprised of an *encoder* that constructs a low-dimensional representation of its input (i.e., the neighborhood graph in this case) and a decoder that reconstructs the input from the low-dimensional representation [12-13]. The encoding process is often lossy, meaning that part of the information is lost during the encoding process and cannot be recovered during decoding. Dimensionality reduction is thus accomplished by finding the encoder/decoder pair that keeps the maximum information when encoding and correspondingly has the minimum reconstruction error when decoding. Note that only the encoder is used to reduce dimensionality, while the decoder is used to find the encoder model that creates the best low-dimensional representation of the input data.

The encoder and decoder are deep feed-forward neural networks (see Fig. S3.1). These neural networks consist of multiple fully-connected layers that contain various numbers of nodes. Each node multiplies its input by a weight vector and feeds that product into a (generally nonlinear) activation function (e.g., hyperbolic tangent, sigmoid, rectified linear unit). Each neural network has an input layer, some number of middle or hidden layers, and an output layer. In this work, the input layer to the encoder is the neighborhood graph while its output is the low-dimensional representation of the neighborhood graph (also called the bottleneck layer). On the other hand, the input to the decoder is the bottleneck layer and the output is the reconstructed neighborhood graph.

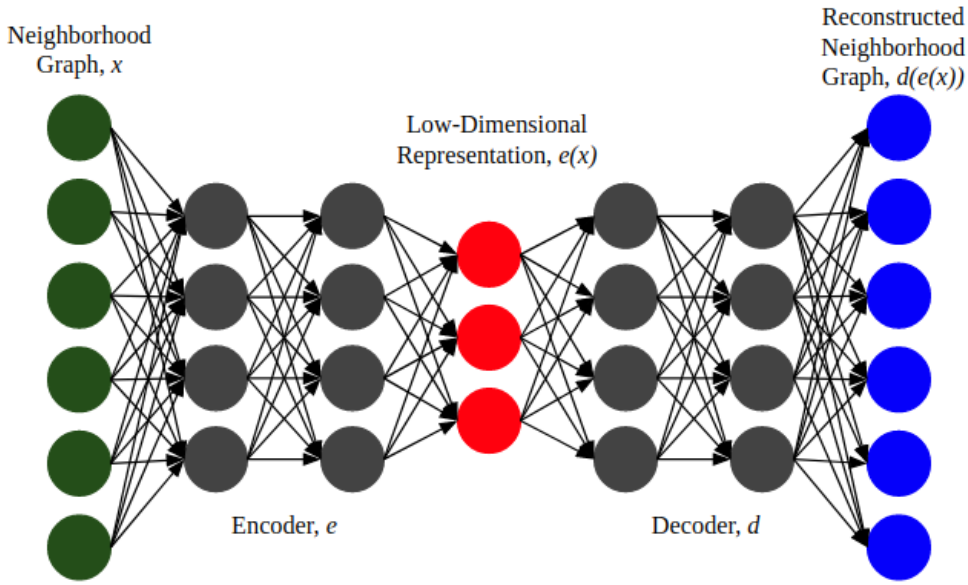

Figure S4.1. Autoencoder architecture. The encoder, *e*, compresses the neighborhood graph of a given particle (a 73 × 1 vector, *x*) into a low-dimensional representation *e(x)*. The *decoder*, *d*, reconstructs the given neighborhood graph from the low-dimensional representation. In this work, the encoder and decoder are deep neural networks with nonlinear activation functions that learn the encoding/decoding scheme that minimizes the reconstruction error of the decoder. This "optimal" encoder/decoder pairing is determined through an iterative training process, where the weights and biases within these neural networks are updated through gradient descent methods. Each circle represents a node within the neural network, and the arrows represent the connections between these nodes. The autoencoder input layer nodes are green, the autoencoder output layer nodes are blue, the hidden layer nodes are grey, and the bottleneck layer nodes are red.

For a given autoencoder architecture (i.e., number of nodes and layers with chosen activation functions), the "optimal" encoder/decoder scheme is found through an iterative training process. Here, a set of training data is fed to the autoencoder and gradient descent methods are used to update the encoder/decoder weights until the reconstruction loss is sufficiently minimized. Denote *E* and *D* as all possible encoder/decoder combinations (i.e., all possible values of the autoencoder weights), *x* as the neighborhood graph, $e(x; \lambda_e)$ as the encoder where λ_e denotes all encoder weights, and $d(e(x; \lambda_e); \lambda_d)$ as the decoder where λ_d denotes all decoder weights, and $J(x,(d(e(x; \lambda_e); \lambda_d))$$ as the decoder's reconstruction loss. The reconstruction loss is often formulated as the mean squared error (MSE) of the original and reconstructed neighborhood graphs. The process of finding the optimal encoder/decoder pair is mathematically represented below. Note that training the autoencoder can be thought of as a ``self-supervised" learning process, as training determines a (nonlinear) function that maps the neural network's inputs (i.e., the neighborhood graphs) to themselves (i.e., neighborhood graphs that are reconstructed from their low-dimensional representation).

$$(e^*, d^*) = \underset{(e,d)\in E\times D}{\operatorname{argmin}} \{J(x, (d(e(x; \lambda_e); \lambda_d))$$

Larger (autoencoder) neural networks (i.e., those with more nodes and/or layers) can find more complex relationships between their inputs and outputs, leading to a lower reconstruction loss

[14]. Larger autoencoders are especially prone to overfitting, however, as the autoencoder is solely trained to encode and decode with as little reconstruction loss as possible, no matter how the low-dimensional space is organized. This can manifest itself in the low-dimensional space lacking continuity (i.e., two close points in the latent space give two completely different decoded contents) and lacking completeness (i.e., certain points within the latent space provide non-physical responses once decoded). One way to overcome this problem is to introduce dropout regularization, which omits certain nodes at random gradient descent iterations in order to reduce the size of neuron weights and prevent co-adaptations of the training data [15].

Neural networks are generally trained using a 5-fold cross-validation methodology in which 60% of the sample data is used to train the model, 20% is used to validate model accuracy (i.e., test for model over-fitting), and 20% is used to evaluate model performance [14,16]. This approach to model training implicitly assumes that the sections of sample data chosen to train, validate, and test the model are fairly representative of the remaining sections (i.e., the data is fairly normally or evenly distributed with minimal outliers). However, the natural non-uniformity in the distances among neighborhood graphs and the fact the Delaunay triangulation creates inconsistent neighbor lists at solid-vapor interfaces render such assumptions invalid. As a result, we train the model with all the 4153 unique neighborhood graphs from the sample set. We then introduced dropout regularization during training to prevent over-fitting. The objective of training the autoencoder is to tune the model weights and biases to achieve the most accurate reconstruction of the neighborhood graphs for a given number of nodes and layers. We evaluate the performance of our encoder by classifying lattice structures in the low-dimensional space (Sections 3.2-3).

Autoencoder performance greatly depends on certain architectural choices, namely network size (i.e., number of nodes in each hidden layer and the bottleneck layer, and the total number of hidden layers), activation function choice (e.g., hyperbolic tangent, linear, rectified linear unit), batch size (i.e., the number of sample data points used for each weight update), and regularization strategy (e.g., norm on the cost function that regulates weight size, dropout regularization). Certain architectural choices are fairly standard or are informed by the training data. For example, choosing hyperbolic tangent activation functions for the hidden layers is a standard choice given the irregularity of the neighborhood graph data [17]. For the same reason, we choose a batch size equal to the size of the training data set. Because smaller batch sizes update the neural network weights after calculating losses from fewer training samples, implementing small batch sizes on data sets with irregular distributions can increase the reconstruction error of the autoencoder. Note that smaller batch sizes are often used throughout the literature because they generally lead to faster convergence during training datasets with Gaussian or nearly Gaussian distributions [26-27].

The remaining architectural choices include using Linear activation functions in the encoder/decoder output layers (which are standard choices regardless of the sample data properties). Dropout regularization is a standard technique to prevent the model from over-fitting the training data and MSE is a standard choice of loss function. The optimal network size is found by plotting autoencoder training loss as a function of network size and implementing the

“elbow method” in order to choose the network architecture with the best balance of computational cost and performance [18-19]. Note that the number of nodes in the bottleneck layer corresponds to the dimension of the low-dimensional space. Finally, the autoencoder was implemented using the Python library Keras (a TensorFlow API) [20-21].

The elbow plots referred to in the main text (Section 3.1) are below:

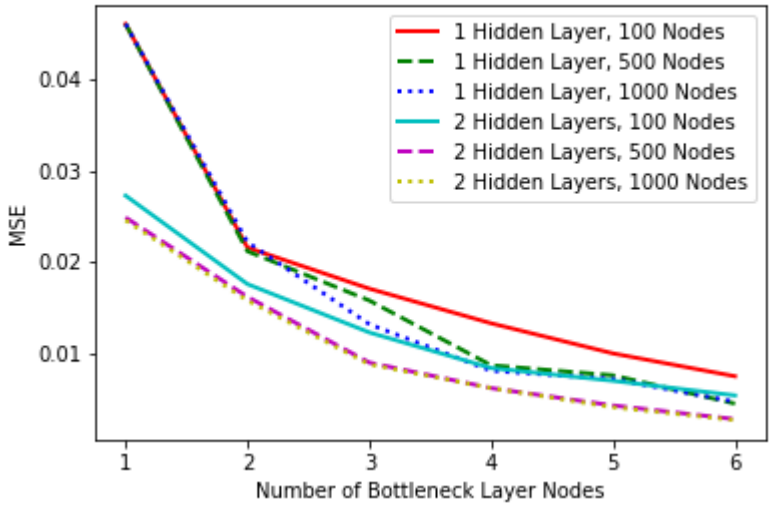

Figure. S4.2. Autoencoder architecture optimization. The autoencoder MSE is plotted against the number of nodes in the bottleneck layer (i.e., the length of the low-dimensional representation vector) for various network sizes. ``Elbows'' in these plots occur between 2 and 4 order parameters, indicating that 3 order parameters are likely sufficient to capture the essential information from the neighborhood graphs. The autoencoder with 2 hidden layers and 1000 nodes per hidden layer displays the (albeit marginally) lowest MSE.

S5. Conceptual Details Behind Relative Importance Analysis

One of the main advantages of autoencoders over diffusion maps is that autoencoders provide an exact analytical mapping from the high to low-dimensional spaces. This mapping allows us to assess the relative importance of each entry in the neighborhood graph via input perturbation and stepwise methods [22-26]. Relative importance is measured by the variation in MSE caused by perturbing samples in the training data set. Input perturbation methods add Gaussian white noise to neighborhood graph entries while stepwise methods replace all graph entry values with their mean. Graph entries that show the largest MSE variation are deemed the ``most important''. A mathematical representation of relative importance can be found below. Here, ΔE_k is the variation in MSE caused by the change applied to the kth neighborhood graph entry and the sum in the denominator runs over all entries from a neighborhood graph of dimension $N \times 1$.

$$RI_k = \frac{\Delta E_k}{\sum_{j=1}^{N} \Delta E_j}$$

The relative importance analysis reveals the influence each graph entry has on the quantification of local structure and even validates (or invalidates) the need for dimensionality reduction.

References

1. Macfarlane, R. J., Lee, B., Jones, M. R., Harris, N., Schatz, G. C., & Mirkin, C. A. (2011). Nanoparticle superlattice engineering with DNA. *science*, *334*(6053), 204-208.
2. Pretti, E., Zerze, H., Song, M., Ding, Y., Mahynski, N. A., Hatch, H. W., ... & Mittal, J. (2018). Assembly of three-dimensional binary superlattices from multi-flavored particles. *Soft Matter*, *14*(30), 6303-6312.
3. Nykypanchuk, D., Maye, M. M., Van Der Lelie, D., & Gang, O. (2007). DNA-based approach for interparticle interaction control. *Langmuir*, *23*(11), 6305-6314.
4. Xiong, H., van der Lelie, D., & Gang, O. (2009). Phase behavior of nanoparticles assembled by DNA linkers. *Physical Review Letters*, *102*(1), 015504.
5. Casey, M. T., Scarlett, R. T., Rogers, W. B., Jenkins, I., Sinno, T., & Crocker, J. C. (2012). Driving diffusionless transformations in colloidal crystals using DNA handshaking. *Nature communications*, *3*(1), 1-8.
6. Scarlett, R. T., Ung, M. T., Crocker, J. C., & Sinno, T. (2011). A mechanistic view of binary colloidal superlattice formation using DNA-directed interactions. *Soft Matter*, *7*(5), 1912-1925.
7. Reinhart, W. F., & Panagiotopoulos, A. Z. (2018). Automated crystal characterization with a fast neighborhood graph analysis method. *Soft matter*, *14*(29), 6083-6089.
8. Milenković, T., & Pržulj, N. (2008). Uncovering biological network function via graphlet degree signatures. *Cancer informatics*, *6*, CIN-S680.
9. Pržulj, N., Corneil, D. G., & Jurisica, I. (2004). Modeling interactome: scale-free or geometric?. *Bioinformatics*, *20*(18), 3508-3515.
10. Pržulj, N. (2007). Biological network comparison using graphlet degree distribution. *Bioinformatics*, *23*(2), e177-e183.
11. Hočevar, T., & Demšar, J. (2014). A combinatorial approach to graphlet counting. *Bioinformatics*, *30*(4), 559-565.
12. Baldi, P. (2012, June). Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning* (pp. 37-49).
13. Wang, Y., Yao, H., & Zhao, S. (2016). Auto-encoder based dimensionality reduction. *Neurocomputing*, *184*, 232-242.
14. Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, *113*, 54-71.
15. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929-1958.
16. Krogh, A., & Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. In *Advances in neural information processing systems* (pp. 231-238).
17. Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

18. Farina, M., Nakai, Y., & Shih, D. (2020). Searching for new physics with deep autoencoders. *Physical Review D*, *101*(7), 075021.
19. Salvador, S., & Chan, P. (2004, November). Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *16th IEEE international conference on tools with artificial intelligence* (pp. 576-584). IEEE.
20. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)* (pp. 265-283).
21. Chollet, F. (2018). Keras: The python deep learning library. *ascl*, ascl-1806.
22. Boattini, E., Dijkstra, M., & Filion, L. (2019). Unsupervised learning for local structure detection in colloidal systems. *The Journal of chemical physics*, *151*(15), 154901.
23. Yao, J., Teng, N., Poh, H. L., & Tan, C. L. (1998). Forecasting and analysis of marketing data using neural networks. *J. Inf. Sci. Eng.*, *14*(4), 843-862.
24. Scardi, M., & Harding Jr, L. W. (1999). Developing an empirical model of phytoplankton primary production: a neural network case study. *Ecological modelling*, *120*(2-3), 213-223.
25. Gevrey, M., Dimopoulos, I., & Lek, S. (2003). Review and comparison of methods to study the contribution of variables in artificial neural network models. *Ecological modelling*, *160*(3), 249-264.
26. Smith, S. L., Kindermans, P. J., Ying, C., & Le, Q. V. (2017). Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*.
27. Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*.
28. Song, M., Ding, Y., Zerze, H., Snyder, M. A., & Mittal, J. (2018). Binary superlattice design by controlling DNA-mediated interactions. *Langmuir*, *34*(3), 991-998.
29. Pretti, E., Zerze, H., Song, M., Ding, Y., Mao, R., & Mittal, J. (2019). Size-dependent thermodynamic structural selection in colloidal crystallization. *Science advances*, *5*(9), eaaw5912.