Supplementary Information for:

Deep Learning for Automated Classification and Characterization of Amorphous Materials

Kirk Swanson,[1,2,a)] Shubhendu Trivedi,[3,4,b)] Joshua Lequieu,[2,5,c)] Kyle Swanson,[3,d)] and Risi Kondor[1,6,7,e)]

[1)]*Department of Computer Science, The University of Chicago, Chicago, IL 60637*

[2)]*Pritzker School of Molecular Engineering, The University of Chicago, Chicago, IL 60637*

[3)]*Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA 02139*

[4)]*Institute for Computational and Experimental Research in Mathematics, Brown University, Providence, RI 02903*

[5)]*Materials Research Laboratory, University of California, Santa Barbara, CA 93106*

[6)]*Department of Statistics, The University of Chicago, Chicago, IL 60637*

[7)]*Center for Computational Mathematics, Flatiron Institute, New York, NY 10010*

[a)]Electronic mail: swansonk1@uchicago.edu

[b)]Electronic mail: shubhendu@csail.mit.edu

[c)]Electronic mail: lequieu@mrl.ucsb.edu

[d)]Electronic mail: swansonk@mit.edu

[e)]Electronic mail: risi@cs.uchicago.edu

I. SIMULATION DETAILS

In each of the 20,000 simulations that we performed, the same initial configuration of particles is used as a starting point. This configuration is minimized using the FIRE minimization algorithm, which brings the configuration to its local potential energy minimum.[1] To do this we used an energy stopping tolerance of 1×10^{-10}, a force stopping tolerance of 1×10^{-5}, a maximum of 1,000 iterations of the minimizer, and a maximum of 1,000 force and energy evaluations. To ensure independence of the simulations, the particles in this configuration are then given initial velocities drawn from a Gaussian distribution with mean zero and with a total linear momentum of zero and a temperature of 2.0. The seeds used to initialize random particle velocities in each of the 10,000 simulations are generated using the Bash $RANDOM function, which returns a pseudorandom integer in the range 0 to 32,767 (see http://tldp.org/LDP/abs/html/randomvar.html).

Each simulation is then run at constant particle number, volume, and temperature (canonical NVT ensemble) with an integration timestep of $\Delta t = 0.005$ in Lennard-Jones units. For the duration of the simulation, we cool the system linearly from the initial temperature of 2.0 to a final temperature of 0.05 using a damping parameter of 0.5. The system is constrained to two dimensions during the simulation using the *enforce2D* command in LAMMPS, and the linear momentum is zeroed in each dimension at every timestep. Particle configurations (both scaled and unscaled coordinates) are recorded every 100,000 steps. Unscaled coordinates refer to the raw coordinates output by a simulation. Scaled coordinates refer to the raw coordinates normalized so that they lie within the unit interval [0, 1].

After the simulation steps are complete, we measure the system's inherent structure energy, also at intervals of 100,000 steps, by loading in the scaled configurations output from the simulation. To compute inherent structure energy, we again use the FIRE minimization algorithm as described above, this time with a maximum of 10,000,000 iterations and a maximum of 10,000,000 force and energy calculations.

Figure 1 shows comparisons of the average radial distribution functions of liquid and glass configurations in dataset 1, where the radial distribution function is defined as

$$g_{ij}(r) = \frac{N}{\rho N_i N_j} \sum_{m=1}^{N_i} \sum_{n=1}^{N_j} \langle \delta(\mathbf{r} - \mathbf{r}_{mn}) \rangle \qquad i, j \in \{A, B\}. \tag{1}$$

Here, N is the total number of particles, N_i is the total number of type i particles, N_j is the total

number of type j particles, ρ is the total density, δ is the delta functional, $\mathbf{r}_{mn}$ is the vector from particle m to particle n, and the average is taken over all vectors $\mathbf{r}$ with magnitude r. We calculated the radial distribution functions in LAMMPS using 500 histogram bins and a cutoff distance of $10\sigma_{AA}$. As expected, these radial distribution functions exhibit clear structural differences between the liquid and glass configurations, with the glasses having higher and sharper peaks than the liquids.

At $T = 0.55$ the 10,000 configurations from $t_{cool} = 2 \times 10^7$, which are liquids, have a mean inherent structure energy of -3.86698 with a standard deviation of 0.00382, while at $T = 0.05$ the 10,000 configurations from $t_{cool} = 2 \times 10^5$, which are glasses, have a mean inherent structure energy of -3.86744 with a standard deviation of 0.00288. These sets of configurations, therefore, have approximately the same energy.

We computed average radial distribution functions comparing the configurations from these glasses and liquids, which comprise dataset 6, shown in Figure 2. Note that the structures are much more similar than those shown in Figure 1.

II. IMAGE DATA FOR CNNS

We utilized CNNs to classify liquid and glass configurations by rendering them as images. For a given configuration, we load its scaled particle coordinates and particle types as a *numpy* array in Python and used the *matplotlib* package to save the configuration as a 250 x 250 pixel PNG image with 100 dots per inch. As noted at https://lammps.sandia.gov/doc/dump.html: *"Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom written to a dump file may be slightly outside the simulation box...atom coords are written in a scaled format (from 0 to 1)...an x value of 0.25 means the atom is at a location 1/4 of the distance from xlo to xhi of the box boundaries."* Upon inspection, most particles have coordinate values between 0 and 1, with only occasional exceptions slightly outside these bounds, e.g. values such as 1.0001. This does not affect our analyses. We remove the axes and the frame from the image so that only the particles are rendered in the image, without any additional artifacts. Type A particles are represented as orange dots and type B particles as blue dots using the *scatter* function, with a dot size of $s = 1$. The image is then slightly cropped in order to remove any unnecessary white space. In some images this slightly truncates particles at the edges of the image, but we found that this does not inhibit the performance of the CNN.

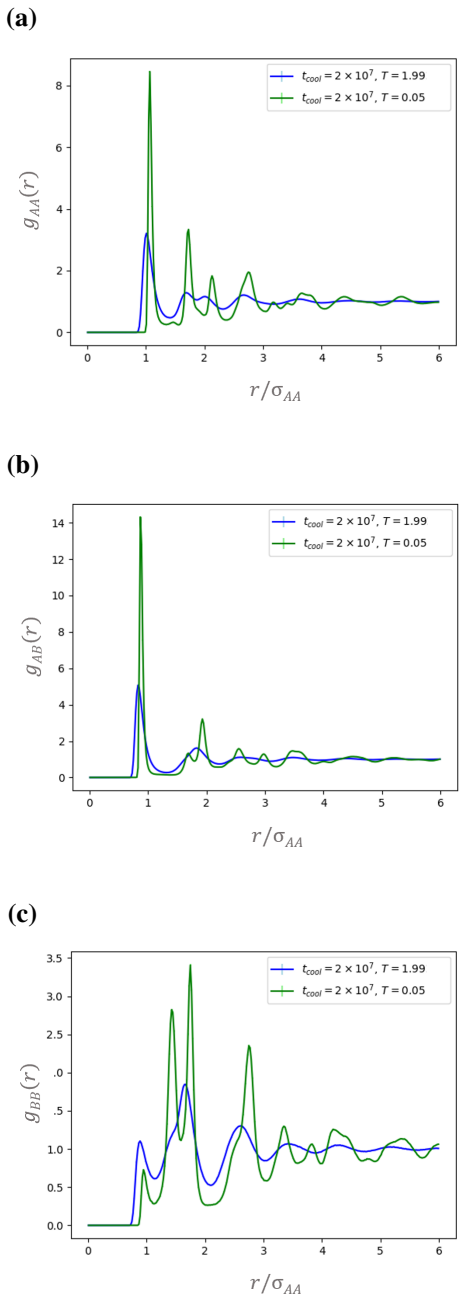

Figure 1: Average **(a)** A-A, **(b)** A-B, and **(c)** B-B radial distribution functions of 10,000 configurations at T = 1.99 and 10,000 configurations at T = 0.05 with $t_{cool} = 2 \times 10^7$ (dataset 1). Glasses (green) are clearly distinguishable from liquids (blue) because of their higher and sharper radial distribution function peaks.

Metadata representing these images for a dataset are saved in a JSON file.

During training the generator function loads a batch of training images into a *numpy* array by reading their file paths from the metadata JSON file. This function converts the liquid and glass labels into a one-hot vector representation, i.e. ([0, 1] for glasses and [1, 0] for liquids).

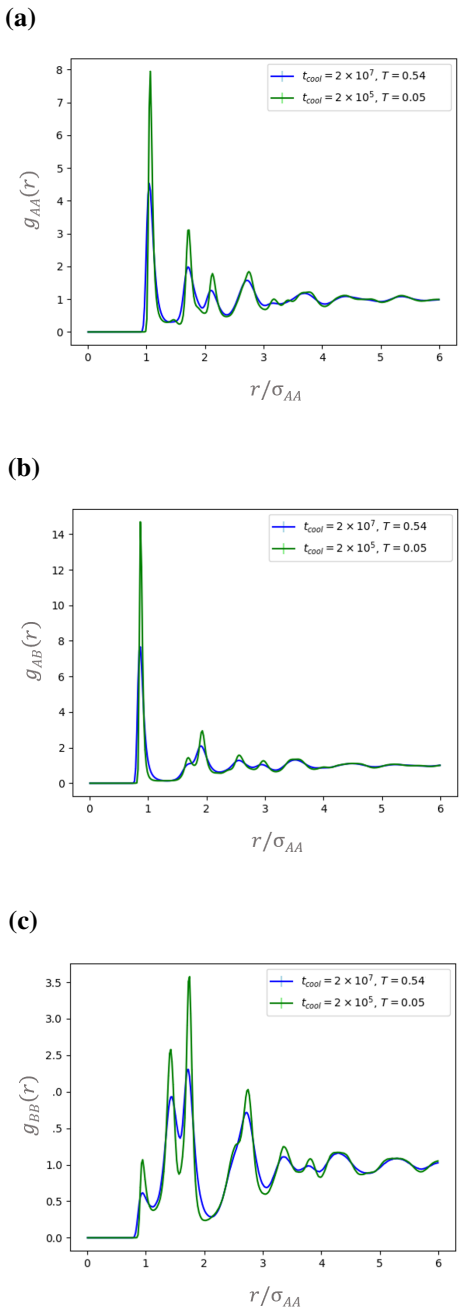

Figure 2: Average **(a)** A-A, **(b)** A-B, and **(C)** B-B radial distribution functions of 10,000 configurations at T = 0.55 with $t_{cool} = 2 \times 10^7$ and 10,000 configurations at T = 0.05 with $t_{cool} = 2 \times 10^5$ (dataset 6). Glasses (green) are distinguishable from liquids (blue) because of higher and sharper radial distribution function peaks, but the differences are much less severe than in Figure 1 as the glasses and liquids here have the same average inherent structure energy.

III. HYPERPARAMETER OPTIMIZATION FOR D-MPNNS

The *hyperopt* Sequential Model-based Global Optimization (SMBO) algorithms form a probabilistic model that maps hyperparameters to a probability of a score on the loss function, $P(y|x)$. This probabilistic model is called a surrogate. SMBO methods work by choosing the next set of hyperparameters to test on the loss function by selecting hyperparameters that perform best on the surrogate function.[2] As each set of hyperparameters is evaluated, the method updates the surrogate probability model in order to make increasingly well-informed guesses. After a specified number of iterations, the method suggests the optimal set of hyperparameters. The specific SMBO method that we use in this work is called a Tree-structured Parzen Estimator (TPE), which is thoroughly described in Bergstra *et al.* and which we follow closely here.[2] There are different ways of identifying which hyperparameters to select based on the surrogate model in an SMBO method, but one of the most effective is a metric called Expected Improvement, otherwise known as an "exploration-exploitation" criterion. Given a desired threshold value for the objective function, y^*, and some set of hyperparameters x, the Expected Improvement is given by

$$\mathrm{EI}_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y) P(y|x) dy. \tag{2}$$

The first factor in the integrand promotes values in regions that are likely to contain objective function minima (exploitation), while the second term promotes regions that have greater uncertainty (exploration). When this integral is positive, it means that the hyperparameter set x is expected to yield an improvement relative to the threshold value y^*.

In the TPE algorithm, instead of modeling the surrogate directly as $p(y|x)$, this method uses Bayes rule, $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$, to model $p(x|y)$ and $p(y)$ instead. $p(x|y)$ is broken down into $l(x)$ and $g(x)$, such that

$$p(x|y) := \begin{cases} l(x) & y < y^* \\ g(x) & y \geq y^*. \end{cases} \tag{3}$$

In other words, we create two different distributions for the hyperparameters: one where the objective function value is less than the threshold, $l(x)$, and one where the objective function value is greater than the threshold, $g(x)$. These non-parametric densities are constructed after some num-

ber K of evaluations of the objective function. y^* is chosen to be slightly greater than the best observed objective function score.

In this approach, the Expected Improvement is given by

$$\mathrm{EI}_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y) \frac{p(x|y)p(y)}{p(x)} dy, \tag{4}$$

which can be rearranged as

$$\mathrm{EI}_{y^*}(x) \propto \left(\gamma + \frac{g(x)}{l(x)}(1-\gamma) \right)^{-1}, \tag{5}$$

where $\gamma = p(y < y^*)$ (no specific $p(y)$ is necessary). So, the TPE works by drawing sample hyperparameters from $l(x)$, evaluating them in terms of $g(x)/l(x)$, and returning the set x that gives the best expected improvement value.

IV. ATTEMPTS TO INTERPRET CNNS

In an attempt to interpret the CNNs, we computed the feature maps produced by the first convolutional layer for several glass and liquid images that were correctly classified as such by the best network trained on dataset 1. Visualizing components of neural networks is gaining traction as a method for interpretation (for example, see https://distill.pub/2017/feature-visualization/). Visualizations of these feature maps for representative liquid and glass configurations are shown in Figure 3.

These visualizations are not clearly interpretable. Feature maps 3 (Figures 3(g) and 3(h)) and 6 (Figures 3(m) and 3(n)) are blank. Feature maps 1 (Figures 3(c) and 3(d)) and 2 (Figures 3(e) and 3(f)) appear to show some kind of texture pattern, with map 1 having bumps that are visually reminiscent of regions of the original images concentrated with type A (orange) particles. Feature maps 4 (Figures 3(i) and 3(j)) and 5 (Figures 3(k) and 3(l)) appear to highlight specific pixels in the image. Perhaps these correspond to specific particles from the original images, but they do not map back directly to pixels that correspond to specific particle locations.

We attempted a different method for interpretation based on our prior knowledge about local geometric structure in two-dimensional Kobb-Anderson binary mixtures. As evidenced in Reid *et*

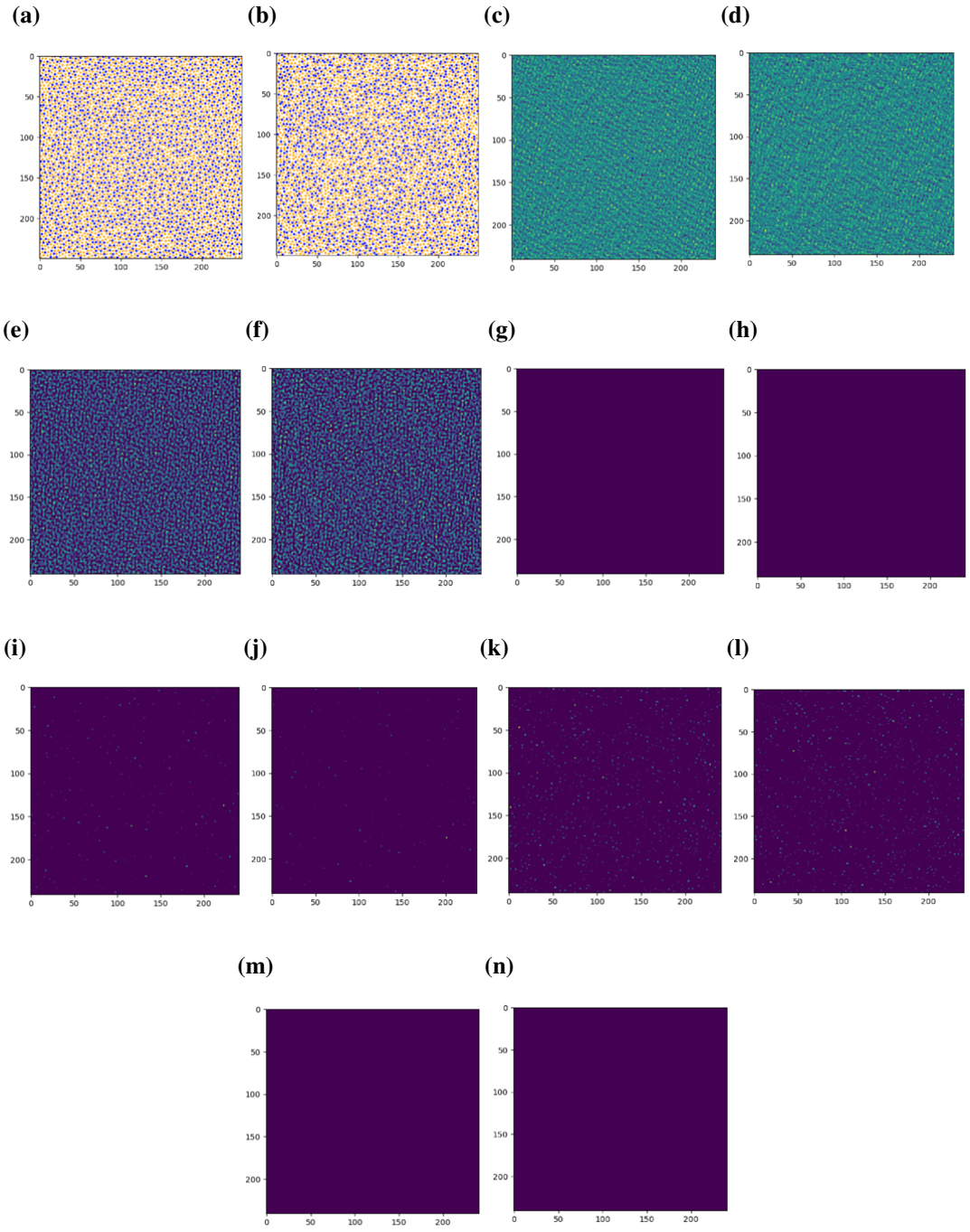

Figure 3: First convolutional layer feature map visualizations for representative liquid and glass configurations. The network used here is a model optimized on dataset 1. The glass and liquid images are taken from $t_{cool} = 2 \times 10^7$ data and were correctly classified by the network. The axes on all of these plots correspond to pixel number. **(a)** is the original glass configuration; **(b)** is the original liquid configuration; **(c)** is glass feature map 1; **(d)** is liquid feature map 1; **(e)** is glass feature map 2; **(f)** is liquid feature map 2; **(f)** is glass feature map 3; **(h)** is liquid feature map 3; **(f)** is glass feature map 4; **(h)** is liquid feature map 4; **(f)** is glass feature map 5; **(h)** is liquid feature map 5; **(f)** is glass feature map 6; **(h)** is liquid feature map 6.

al., the degree of five-fold symmetry in a liquid-cooled glass is much higher than that in liquids at higher temperatures and higher inherent structure energies.[3] One possibility is that the network has identified this geometric quantity as a means for classifying liquids and glasses and that the kernels in the CNN have been trained to identify local pentagonal arrangements of particles. As in Reid *et al.*, we consider a region of five-fold symmetry to be characterized by a type B (blue) particle surrounded immediately by a pentagon of type A (orange) particles. Perhaps, if the CNN is trained to correlate high concentrations of five-fold symmetry with glassy materials, an artificial image that is saturated with five-fold symmetry patterns will be classified by the network as a glass.

We tested this hypothesis by constructing artificial images imbued with five-fold symmetry as follows. First, we placed 1,512 (35% of 4,320) type B particles in a grid, evenly spaced, with the x- and y-axes ranging from 0 to 1. Each particle was then given a slight random displacement in the x- and y-directions from their initial placements, corresponding to a uniform random number in the range -1/195 to 1/195. These specific random displacement values were determined by visual inspection and trial and error. We then selected 500 of these type B particles to surround with pentagonal arrangements of type A (orange) particles. To attempt to replicate the five-fold symmetry clustering effect described in Hu *et al.*, we added pentagonal arrangements around type B particles iteratively and selected each subsequent type B particle to be a neighbor of a previously selected particle with a probability of 3/4.[4] Every time a pentagonal arrangement of type A particles was placed, we arranged the particles to be at a radial distance of 1/78 from the central type B particle and gave the pentagonal arrangement a random angular rotation selected uniformly from the range -2π to 2π. For any pair of particles within a distance of 1/90 of each other, we removed one of the particles to prevent overlaps. Figure 4(a) shows a representative image of the result.

The remainder of the type A particles are filled in randomly, again avoiding any overlaps, to produce a final result, shown in 4(b). We again used the dataset 1 network to classify several hundred examples of these artificial images. However, all of them were classified as liquids. We also constructed images with other n-fold symmetries and with random configurations of particles, but all were classified as liquids.

V. CNN AND D-MPNN CLASSIFICATION RESULTS

Figure 5 shows accuracy results for CNNs and D-MPNNs.

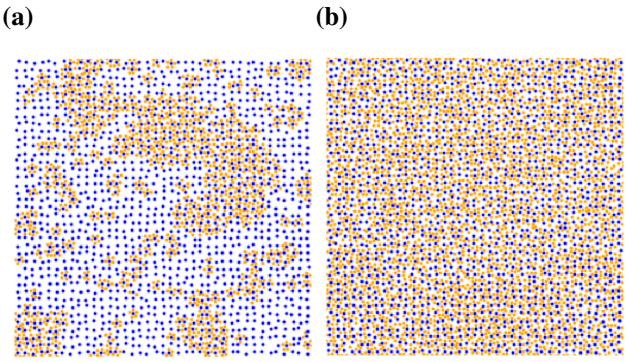

Figure 4: **(a)** Initial artificial pentagonal arrangements of type A and type B particles. **(b)** Final artificial five-fold symmetry configuration.

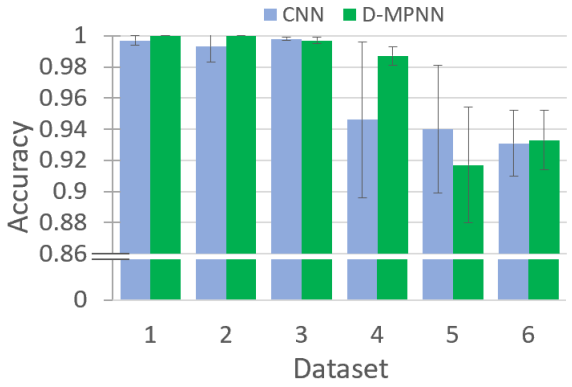

Figure 5: CNN and D-MPNN average classification accuracy for datasets 1 through 6 with error bars showing standard deviation. These average values were computed using the three-fold nested cross-validation scheme described in §2.5 in the main text.

VI. D-MPNN SELF-ATTENTION

Figure 6 shows self-attention visualizations for configurations in an outer test set of dataset 1 with $l_{win} = 0.3$. High attention edges were determined with a hard cutoff; upon inspection, there was a clear trough in the distribution of attention weights that separated those with small, almost negligible magnitudes and those with larger magnitudes. Figure 7 shows the graph-based metrics from Figure 10 in the main text plotted individually as a function of temperature.

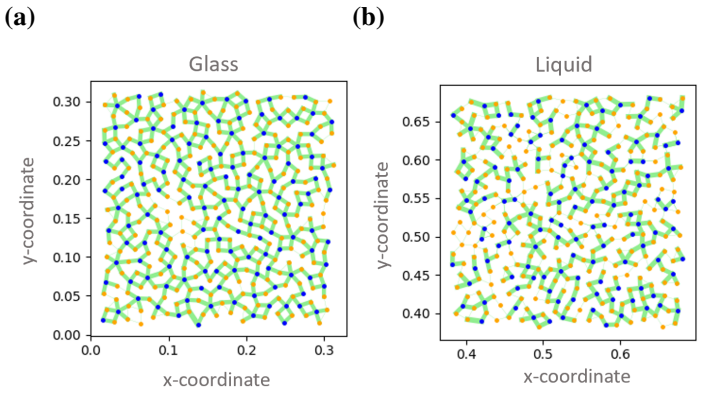

Figure 6: Self-attention visualizations, with the attention weights computed on the $\mathcal{T}-1^{th}$ step of message passing. All connected particles in the graph are joined with a green line whose width is proportional to the magnitude of the attention weight. Note that each connected pair of particles actually has two edges, because the graph is directed. Here, we visualize the edge with the higher weight. **(a)** Glass configuration with $l_{win} = 0.3$. **(b)** Liquid configuration with $l_{win} = 0.3$.

VII. TRAINING TIME

We trained the CNNs and D-MPNNs and performed hyperparameter optimization on NVIDIA Tesla V100 GPUs using Amazon Web Services. The training time for CNNs was consistently less than 7.5 milliseconds per data sample regardless of batch size, while the training time for D-MPNNs ranged from approximately 28 milliseconds to 260 milliseconds per data sample, depending on the values of the hyperparameters. Since one epoch of training corresponded to 16,000 data samples, the training time per epoch for CNNs was less than 2 minutes, while the training time per epoch for D-MPNNs ranged from approximately 7.5 minutes to 70 minutes. Since we trained each model for 10 epochs, training a CNN took approximately 20 minutes while training a D-MPNN ranged from approximately 75 minutes to 6.5 hours. Figure 8 shows D-MPNN training time for three representative sets of hyperparameters as a function of system size, which we varied by using different values of l_{win} while keeping all other hyperparameters fixed. We also found that including the self-attention step during training did not significantly increase training time. For example, including the self-attention step for the model II network described in Figure 8 with $l_{win} = 0.4$ increased the average training time from 31.38 min. to 32.58 min. The time to perform inference (predicting liquid versus glass), much less than the time for training, was less than 5 milliseconds per data sample for the CNN and ranged from approximately 30 milliseconds to 150

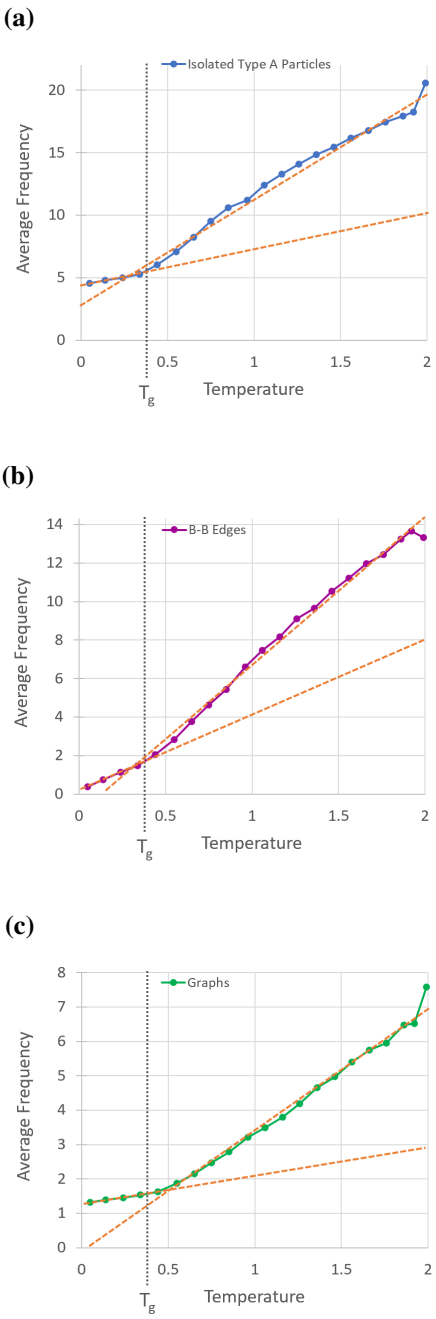

Figure 7: Average number of isolated type A particles **(a)**, edges connecting pairs of type B particles, **(b)**, and number of disjoint graphs **(c)** in configurations from simulations with $t_{cool} = 2 \times 10^7$ at a variety of temperatures. The dotted orange lines show predictions from the linear regression models for each of these curves, as described in §3 in the main text. The segments of the linear models above and below T_g are artificially extended to visually highlight the difference in slopes above and below T_g.

milliseconds for the D-MPNNs, depending on values of the hyperparameters.

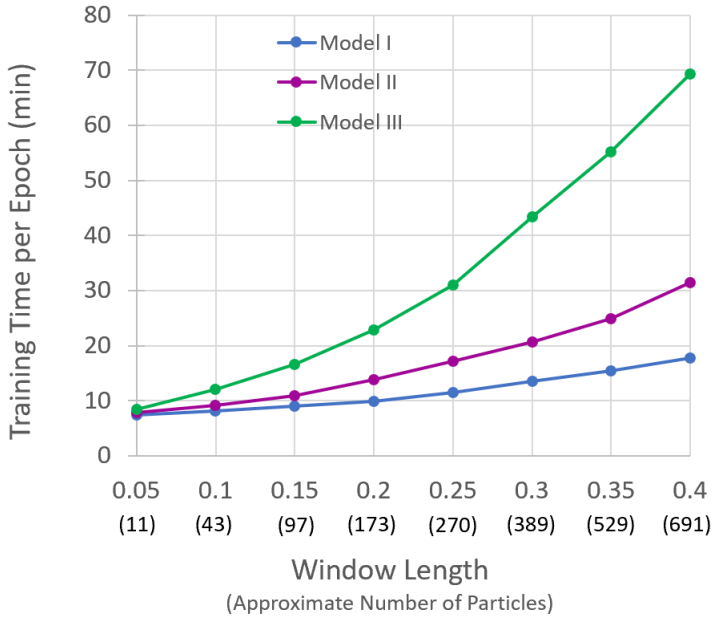

Figure 8: Training time for three representative sets of hyperparameters as a function of l_{win} (window length). Model I used $k = 1$ nearest neighbors, $h = 300$ hidden size, $\mathcal{T} = 2$ message passing steps, $f = 1$ feedforward layer, $p = 0.5$ dropout probability, and $n_b = 5$ batch size, which is the smallest model obtainable from the range of hyperparameters that we used during hyperparameter optimization. Model II used $k = 3$ nearest neighbors, $h = 1000$ hidden size, $\mathcal{T} = 4$ message passing steps, $f = 2$ feedforward layers, $p = 0.5$ dropout probability, and $n_b = 5$ batch size. Model III used $k = 5$ nearest neighbors, $h = 2400$ hidden size, $\mathcal{T} = 6$ message passing steps, $f = 3$ feedforward layers, $p = 0.5$ dropout probability, and $n_b = 5$ batch size, which is the largest model obtainable from the range of hyperparameters that we used during hyperparameter optimization. For each model at a given value of l_{win}, we report the average training time in minutes on an outer train set of dataset 1 in three independent trials. Standard deviations of training time, not shown here, were less than 1 minute.

REFERENCES

[1]E. Bitzek, P. Koskinen, F. Gahler, M. Moseler and P. Gumbsch, *Phys. Rev. Lett.*, 2006, **97**, 170201.

[2]J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl, *Algorithms for Hyper-Parameter Optimization*, Advances in Neural Information Processing Systems 24, 2011.

[3]D. R. Reid, I. Lyubimov, M. D. Ediger and J. J. de Pablo, *Nature Communications*, 2016, **7**, 13062.

[4]Y. C. Hu, F. X. Li, M. Z. Li, H. Y. Bai and W. H. Wang, *Nature Communications*, 2015, **6**, 8310.