



HAL
open science

SE(3)-equivariant Graph Neural Networks for Learning Glassy Liquids Representations

Francesco Saverio Pezzicoli, Guillaume Charpiat, François P. Landes

► **To cite this version:**

Francesco Saverio Pezzicoli, Guillaume Charpiat, François P. Landes. SE(3)-equivariant Graph Neural Networks for Learning Glassy Liquids Representations. 2022. hal-03868206v1

HAL Id: hal-03868206

<https://inria.hal.science/hal-03868206v1>

Preprint submitted on 23 Nov 2022 (v1), last revised 17 Jan 2024 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

SE(3)-equivariant Graph Neural Networks for Learning Glassy Liquids Representations

Francesco Saverio Pezzicoli,¹ Guillaume Charpiat,¹ and François P. Landes¹

¹*Université Paris-Saclay, CNRS, INRIA, Laboratoire Interdisciplinaire des Sciences du Numérique, TAU team, 91190 Gif-sur-Yvette, France*

Within the glassy liquids community, the use of Machine Learning (ML) to model particles' static structure in order to predict their future dynamics is currently a hot topic. The actual state of the art consists in Graph Neural Networks (GNNs) [BKGB⁺20] which, beside having a great expressive power, are heavy models with numerous parameters and lack interpretability. Inspired by recent advances [TSK⁺18], we build a GNN that learns a robust representation of the glass' static structure by constraining it to preserve the roto-translation (SE(3)) equivariance. We show that this constraint not only significantly improves the predictive power but also allows to reduce the number of parameters while improving the interpretability. Furthermore, we relate our learned equivariant features to well-known invariant expert features, which are easily expressible with a single layer of our network.

PACS numbers:

Introduction

Understanding the nature of the dynamical glass transition is one of the most intriguing open problems in condensed matter. As a glass-forming liquid is cooled down towards its characteristic temperature (T_g), its viscosity increases by several orders of magnitude, while its structure, the spatial arrangement of the particles, does not show any obvious change. Finding the subtle changes in the structure that explain the huge variations in the dynamics (increased viscosity) seems like an obvious target for Machine Learning, which is a great tool for finding hidden patterns in data.

Aside from Machine Learning, historically and up to the present day, physicists have inspected how some well-defined physical quantities correlate with glassy dynamics [RSL14, GPS16, TT18, TTSR19, Ler20, LBVP22], in a manner akin to expert features (but without any learning). There are also direct pattern-recognition techniques [Cos11, KIG11, RK17, TTR17, RTS20] and a few unsupervised approaches [BMAM⁺20, PJC20], but the typical strategy is to learn a representation using supervised learning, i.e. using ML to map a given local structure to a target label that represents the local dynamics (i.e. a "local viscosity" measure). Since glasses are very heterogeneous when approaching the transition (a phenomenon called Dynamic Heterogeneity [CWCK⁺10, ALBB21]), a single temperature point already displays a broad variety of labels' values, and is generally considered to be enough data to learn a good representation.

The quest for this good representation has prompted many works, relying on various techniques. The idea of using ML for glasses was introduced by the pioneering works of Liu *et al.* [SLRR14, CSR⁺15, SCKL16, SCS⁺15], where a rather generic set of isotropic features (describing the density around the target particle) is fed to a binary classifier (a linear SVM) to predict the target particle's mobility. After considering only isotropic features, they marginally improved accuracy by using angular-aware expert features, measuring bond-angles

around the target particle. This simple yet robust technology alone yielded a number of physics-oriented works [STC⁺17, Sch17, CIS⁺17, SSCL17, LBD⁺20], which provide physical interpretations of the model's output value (named Softness). These and the other shallow learning approaches all rely on features that describe the neighborhood of a single particle to then predict this single particle's future: in all cases, there is no interaction between neighborhoods.

Independently from the glass literature, within other application fields dealing with particles living in 3D space, Deep Learning frameworks and in particular Graph Neural Networks (GNNs) were developed, with some success, quite early on (2016). Applications range from molecular properties predictions [KMB⁺16], to bulk materials properties prediction (MegNet) [CZY⁺19], or Density Functional Theory approximations [HSD⁺21], and most importantly for us, for glassy dynamics [BKGB⁺20]. Indeed, designing geometrical features to capture the slight variations in the geometry of a particle's neighborhood is a daunting task, and the lesson learned from Computer Vision and the advent of Convolutional Neural Networks (CNNs) is that meta design works better than design, that is, expert features designed by hand are not as informative as the ones learned by a neural network with a suitable architecture designed by an expert. This is the point of the Deep approach, and more specifically GNNs, that are designed to build aggregate representations of the nodes' neighborhoods. Although they are very effective, standard GNNs lack interpretability due to their structure and their high number of learnable parameters.

The coarse-graining effectively performed by GNNs was then mimicked (or distilled) in an expert-features approach, with the impressive result by Boattini *et al.* [BSF21] where it is shown that a simple Ridge Regression with $\sim O(1000)$ of rather generic, rotational-invariant basic features, perform equally well as the state of the art (GNNs, [BKGB⁺20]). The features consist for a part of node-wise rotational-invariant descriptors, and

for another, of these elementary features but averaged over neighboring nodes, over a small radius (mimicking a GNN aggregation step). There is currently some debate [BSF21, ABFS22] over whether Deep Learning can achieve better results than such expert features, and part of the community leans on this expert side.

Here, inspired by the fast-growing field of SE(3)-equivariant networks [TSK+18, BMS+21, WGW+18, BHvdP+22], we build a GNN that produces directional hidden representations that are rotation-equivariant, i.e. respect the symmetries of atom packings. In other words, we take the best of both worlds: we combine the basic idea of symmetry-aware features, already used with some success [BSF21], with the combinatorial or expressive power of Deep Learning. In practice, we surpass the state of the art for prediction of the dynamical propensity of 3D Kob-Anderson mixtures, at a reduced number of parameters and increased interpretability.

In the next section (sec. I) we define the type and amount of input and output data, and the task to be solved. We then introduce all the necessary tools to build a SE(3)-GNN, explaining how they apply to our specific case (sec. II). Equipped with these tools, we then describe the main characteristics of our network’s architecture, in sec. III. In section IV, we compare the performance of our network with the previously known state of the art results, showing it consistently outperforms previous methods, at a reduced cost. In the discussion (sec. V) we put our work in a broader perspective and outline directions for future work.

I. DATASET AND TASK

To probe the ability of our model to predict mobility, we adopt the dataset built by Bapst *et al.* in [BKGB+20]. It is obtained from molecular dynamics simulations of an 80:20 Kob-Anderson mixture of $N = 4096$ particles in a three-dimensional box with periodic boundary conditions, at number densities of $\rho \simeq 1.2$. Four state points (temperatures) are analyzed: $T = 0.44, 0.47, 0.50, 0.56$ (in dimensionless units). For each point, 800 independent configurations $\{\mathbf{x}_i\}_{i=1\dots N}$ are available, i.e. 800 samples (each sample represents N particles’ positions).

Ground truth labels, i.e. the quantities to be predicted, are the dynamical propensity [BJ07, BBB+07] of particles: for each initial configuration, 30 micro-canonical simulations are run independently, each with initial velocities sampled from the Maxwell-Boltzmann distribution. The propensity of particle i over a timescale τ is then defined as the average displacement over the 30 runs.

For each sample to be processed through the GNN, an input graph is built by taking particles as nodes and connecting them when the inter-atomic distance is less than $d_c = 2$ (in atomic potential units). The node features [45] only encode the particle type, here A or B. We

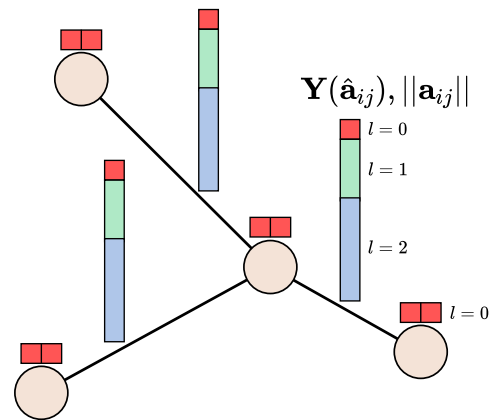


FIG. 1: **Input Graph with its input features.** Node features are the one-hot encoded particle types (invariant features, $l = 0$), and edge attributes \mathbf{a}_{ij} are split: the direction is embedded in Spherical Harmonics $Y(\hat{\mathbf{a}}_{ij})$ and the norm is retained separately.

use one-hot encoding, such that node features consist of $n_{type} = 2$ boolean variables. This generalizes trivially to mixtures with $n_{type} > 2$. The edges are directed, and edge (i, j) has for feature $\mathbf{a}_{ij} = (\mathbf{x}_j - \mathbf{x}_i)$, i.e. it stores the relative position of the particles (nodes) it connects. We show a sketch of our input graph with its node and edge features in Fig. 1.

Our task is the node-wise regression of the particles’ propensity $m_i \in \mathbb{R}$ (node label), restricted to the type-A particles (nodes).

II. HOW TO BUILD A GRAPH-CONVOLUTION EQUIVARIANT LAYER?

Graph Neural Networks Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \dots, n_v\}$ is the set of nodes v_i and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges e_{ij} , endowed with node features $\mathbf{h}_i \in \mathbb{R}^{c_v}$ and edge features $\mathbf{a}_{ij} \in \mathbb{R}^{c_e}$. GNNs operate on such graphs by updating node (and possibly edge) features through local operations on the neighborhood of each node. These operations are designed to be adaptive to different kinds of neighborhoods and respect node-index permutation invariance, which are the two key features of GNNs, as opposed to CNNs (for which the learned kernels must have fixed, grid-like geometry, and for which each neighboring pixel is located at a fixed relative position). In this work we deal with Graph Convolutional Networks (GCN), a subclass of GNNs. A GCN layer acts on node features as follows:

$$\mathbf{h}'(\mathbf{x}_i) = \sum_{j \in \mathcal{N}(i)} \kappa(\mathbf{x}_j - \mathbf{x}_i) \mathbf{h}(\mathbf{x}_j) \quad (1)$$

where $\mathcal{N}(i)$ is the neighborhood of node i . Here a position $\mathbf{x}_i \in \mathbb{R}^3$ is associated to each node and κ is a continuous convolution kernel which only depends on relative nodes’ positions. In this case, as for CNNs, the node update

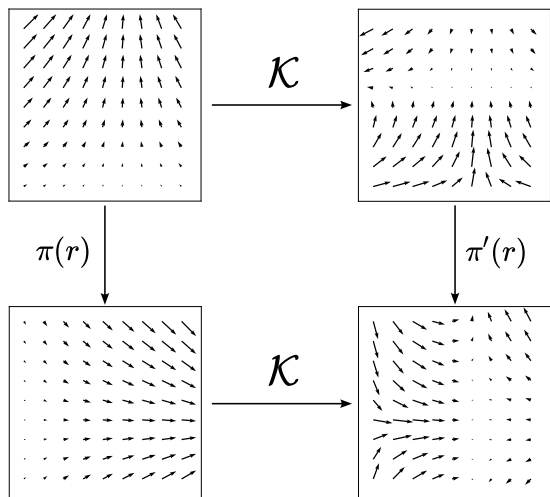


FIG. 2: **Equivariance to 2D rotations.** Simple case in which the input and the output fields have the same dimension. $\pi(r)$ represents the action of the rotation operator on the input field, $\pi'(r)$ on the output one. The mapping \mathcal{K} acts in an equivariant way, indeed it commutes with the rotation. More generally, the output field may be of a different dimension, hence the presence of $\pi'(r)$, which in the above case is equal to $\pi(r)$.

operation is translation-equivariant by construction. It is however not automatically rotation-equivariant.

Equivariance A layer of a network is said to be equivariant with respect to a group G if upon group action on the input, the output is transformed accordingly. One simple example is shown in Figure 2: it depicts a 2D vector field $\mathbf{f}(\mathbf{x})$ and a mapping \mathcal{K} that acts on it, $\mathcal{K}(\mathbf{f}) = \cos(\|\mathbf{f}\|) \cdot \hat{\mathbf{f}}$ (where $\hat{\mathbf{f}} = \mathbf{f}/\|\mathbf{f}\|$), generating the output field $\mathbf{f}'(\mathbf{x})$, which happens to live in the same space. \mathcal{K} is equivariant to 2D rotations: it operates only on the norm of the vectors, thus it commutes with the rotation operator.

As introduced in previous the example, we can represent the point cloud processed by the GNN as a vector field $\mathbf{h}(\mathbf{x}) = \sum_{i \in \mathcal{V}} \delta(\mathbf{x} - \mathbf{x}_i) \mathbf{h}_i$ with values in the vector space H , and the action of a layer as a mapping \mathcal{K} from one field to the updated one. To require that \mathcal{K} fulfils equivariance, we need to define how the group of interest acts on the vector space H through representations. Given a group G , a representation ρ is a mapping from group elements g to square matrices $\mathbf{D}^H(g)$ that respect the group structure. Essentially it tells how G acts on a specific space H . For example, the representation of the group of three-dimensional rotations $SO(3)$ on the 3-D Cartesian space is the usual rotation matrix $\mathbf{R}(r_{\alpha,\beta,\gamma}) = \mathbf{R}_x(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_z(\gamma)$. Then if we consider an element $tr \in SE(3)$ which is composed of a translation and a rotation, it will act on a vector field as follows:

$$\mathbf{h}(\mathbf{x}) \xrightarrow{\pi(tr)} \mathbf{D}^H(r)\mathbf{h}(\mathbf{R}^{-1}(r)(\mathbf{x} - \mathbf{t})) \quad (2)$$

The codomain (output domain) is transformed by the

representation of the rotation while the domain by the one of the inverse roto-translation. See Fig. 2 top left to bottom left for an example with 2D rotations. For more details and further examples, see [WGW⁺18].

Let us define equivariance. Let there be a mapping $\mathcal{K} : \mathbf{h}(\mathbf{x}) \rightarrow \mathbf{h}'(\mathbf{x})$ and $\mathbf{h} \in H$, $\mathbf{h}' \in H'$ with H, H' two vector spaces, \mathcal{K} is equivariant with respect to G if

$$\forall g \in G \quad \mathcal{K} \circ \pi(g) = \pi'(g) \circ \mathcal{K} \quad (3)$$

The input and output codomains H, H' do not need to be the same, and this is taken into account by the group representations $\mathbf{D}^H(g)$ and $\mathbf{D}^{H'}(g)$. A direct consequence of this definition is that invariance is a particular case of equivariance where $\mathbf{D}^{H'}(g) = \mathbb{I} \quad \forall g \in G$.

When dealing with $SE(3)$, the *only* invariant quantities are *scalar*, thus considering only invariant features significantly reduces the model's expressivity.

Equivariant features To enforce equivariance of layers, we work with equivariant features (also called *steerable features*), following the schema of steerable group convolutions [WGW⁺18, TSK⁺18, KLT18] (for theoretical insight, see Appendix-B of [BHvdP⁺22]). These features inhabit the space of irreducible representations of $SO(3)$, which is factorized into sub-spaces: $V_{SO(3)} = V_{l=0} \oplus V_{l=1} \oplus V_{l=2} \oplus \dots$. Each subspace, indexed by $l \geq 0$, is of size $2l + 1$ and transforms independently under rotations thanks to the action of Wigner-D matrices $\mathbf{D}^{(l)} \in \mathbb{R}^{(2l+1) \times (2l+1)}$. Coming to implementation, a feature is just a concatenation of different l -vectors: scalars ($l = 0$), 3-D vectors ($l = 1$), 5-D vectors ($l = 2$) and so on. Multiple pieces with the same l are also allowed, we address this multiplicity by referring to channels. For example we can have two $l = 0$ channels, and a single $l = 1$ channel, likewise for $l = 2$:

$$\mathbf{h}(\mathbf{x}) = \left(h_{c=0}^{(l=0)}(\mathbf{x}), h_{c=1}^{(l=0)}(\mathbf{x}), \mathbf{h}_{c=0}^{(l=1)}(\mathbf{x}), \mathbf{h}_{c=0}^{(l=2)}(\mathbf{x}) \right) \quad (4)$$

where $\mathbf{h} : \mathbb{R}^3 \rightarrow \mathbb{R}^k$ with $k = \sum_l n_c^{(l)} \cdot (2l + 1) = 2 \cdot 1 + 3 + 5 = 10$ with $n_c^{(l)}$ number of channels of type l . Rotation of these features is straightforward:

$$\mathbf{D}(r)\mathbf{h} = \begin{bmatrix} D^{(0)} & & & \\ & D^{(0)} & & \\ & & \mathbf{D}^{(1)} & \\ & & & \mathbf{D}^{(2)} \end{bmatrix} \begin{bmatrix} h_{c=0}^{(l=0)} \\ h_{c=1}^{(l=0)} \\ \mathbf{h}_{c=0}^{(l=1)} \\ \mathbf{h}_{c=0}^{(l=2)} \end{bmatrix} \quad (5)$$

The representation matrix is block-diagonal thanks to $SO(3)$ being decomposed in a direct sum, and scalars ($l = 0$) are invariant with respect to rotation ($D^{(0)} = \mathbb{I}$). The wording *steerable* becomes clear: upon rotation of the input coordinates, these features rotate accordingly, as when one steers the steering wheel, thus turning the wheels.

Spherical Harmonics (SH) To embed the three dimensional node and edge input data in an equivariant form, we use Real Spherical Harmonics $Y_m^l : \mathbb{S}^2 \rightarrow \mathbb{R}$.

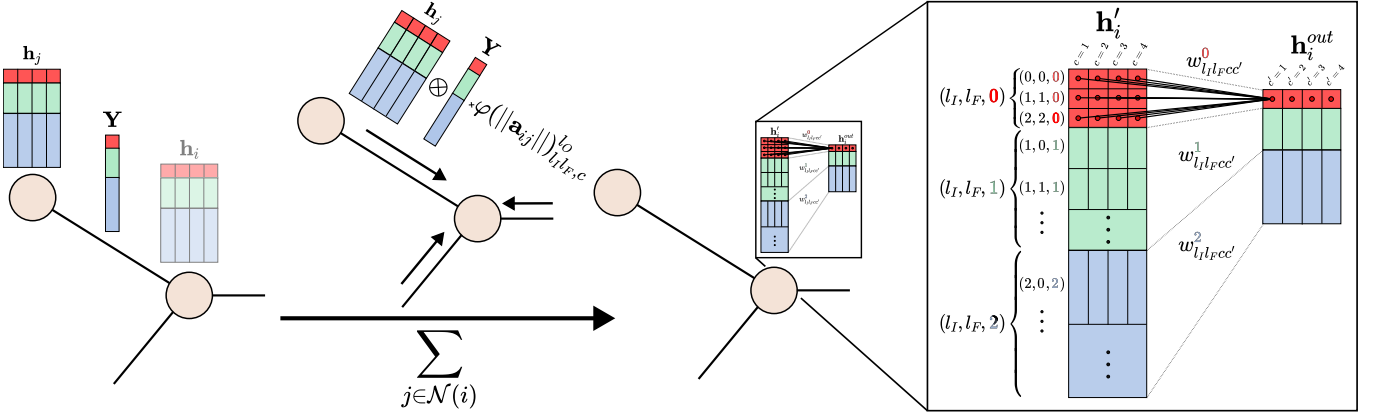


FIG. 3: **Overview of the convolution layer, summarizing Eqs. (10,11).** For each neighboring node, the node and edge features are combined (with C-G product) and multiplied by the learned radial filter φ . Before performing this operation, the one-hot encoded particle, are concatenated to \mathbf{h}_i by adding 2 $l = 0$ channels, but this is not shown in the figure for simplicity. Because multiple triplets come out of the C-G product, we obtain a much larger representation (left part of the inset at the right). This intermediate representation is narrowed down using a linear layer (one for each l_O).

They can be thought of as the generalization of Fourier modes (circular harmonics) to the sphere. Spherical Harmonics are indexed by the rotational order $l \geq 0$, which is reminiscent of the 1-D frequency, and by $m = -l, \dots, l$, which determines the spatial orientation. They form an orthonormal basis of $\mathbf{L}^2(\mathbb{S}^2)$, i.e. any real-valued function on the sphere $f : \mathbb{S}^2 \rightarrow \mathbb{R}$ can be Fourier Transformed to this SH basis:

$$f(\mathbf{n}) = \sum_{l=0}^{\infty} \sum_{m=-l}^l \hat{f}_m^l Y_m^l(\mathbf{n}) \quad (6)$$

$$\mathcal{F}(f)(l, m) = \hat{f}_m^l = \int_{\mathbb{S}^2} f(\mathbf{n}) Y_m^l(\mathbf{n}) d\mathbf{n} \quad (7)$$

where $\mathbf{n} = (\theta, \phi)$ represents a generic direction on the sphere. Here the coefficients \hat{f}^l are not real values but are $(2l + 1)$ -dimensional vectors (with components \hat{f}_m^l). The set of all coefficients $(\hat{f}^l)_{l=0, \dots}$ plays the same role as $\mathbf{h}(\mathbf{x})$ in Eq.(4), and each coefficient \hat{f}^l transforms according to a Wigner-D matrix $\mathbf{D}^{(l)}$: the SH embedding is thus equivariant.

In particular, the density of neighbor particles at a fixed distance r from the central one, $\rho_r(\mathbf{n}) = \sum_{j \in \mathcal{N}(i)} \delta(\mathbf{n} - \mathbf{n}_{ij}) \delta(r - r_{ij})$, is a real-valued function (distribution) on the sphere and can be decomposed into Spherical Harmonics. Furthermore, summing such decompositions at a number of $r \in [0, d_c]$, one obtains an equivariant representation of the field of density around a target particle in the ball of radius d_c (this is what our very first convolution performs, see lower).

Note that to make a fixed- r representation finite-dimensional, we need to choose a high-frequency cutoff value for the rotational order, $l = l_{max}$. Analogously to Fourier transforms on the circle, this way of decomposing and filtering out high-frequencies preserves the input signal better than most naive schemes (as e.g. a dis-

cretization of solid angles).

Clebsch-Gordan tensor product As said above, we do not want to restrict ourselves to invariant features, but to equivariant ones. For this, we need a way to combine feature vectors together, other than the dot product (which would produce invariant scalar features).

Analogously to the outer product for vectors of \mathbb{R}^3 , which is a bilinear operator $\otimes : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}^3$, the Clebsch-Gordan tensor product $\otimes_{l_1, l_2}^{l_O} : V_{l_1} \times V_{l_2} \rightarrow V_{l_O}$ is a bilinear operator that combines two $SO(3)$ steerable features of type l_1 and l_2 and returns another steerable vector of type l_O . It allows to maintain equivariance when combining equivariant features: consider $\mathbf{h}_1(\mathbf{x}) \in V_{l_1}$, $\mathbf{h}_2(\mathbf{x}) \in V_{l_2}$ and their C-G tensor product $\mathbf{h}'(\mathbf{x}) = (\mathbf{h}_1(\mathbf{x}) \otimes_{l_1, l_2}^{l_O} \mathbf{h}_2(\mathbf{x})) \in V_{l_O}$; if we apply a rotation r , inputs will be transformed by $\mathbf{D}^{(l_1)}(r)$, $\mathbf{D}^{(l_2)}(r)$ and the output by $\mathbf{D}^{(l_O)}(r)$, i.e. equivariance is fulfilled. Concretely, the tensor product is computed using Clebsch-Gordan coefficients $C_{(l_1, m_1), (l_2, m_2)}^{(l_O, m_O)}$ as

$$h_{m_O}^{l_O} = \sum_{m_1=-l_1}^{l_1} \sum_{m_2=-l_2}^{l_2} C_{(l_1, m_1), (l_2, m_2)}^{(l_O, m_O)} h_{m_1}^{l_1} h_{m_2}^{l_2} \quad (8)$$

We have $C_{(l_1, m_1), (l_2, m_2)}^{(l_O, m_O)} \neq 0$ only for $l_O \in [|l_1 - l_2|, l_1 + l_2]$, thus it is a sparse tensor product. In a more concise form we write:

$$\mathbf{h}^{l_O} = \mathbf{h}_1^{l_1} C_{l_1 l_2}^{l_O} \mathbf{h}_2^{l_2} \quad (9)$$

where $C_{l_1 l_2}^{l_O}$ is a $(2l_O + 1) \times (2l_1 + 1) \times (2l_2 + 1)$ tensor.

SE(3)-equivariant Graph Convolution Layer Using all the concepts introduced above, we can now define the explicit form of the convolution kernel κ of Eq. 1. Denoting the edge features $\mathbf{a}_{ij} = (\mathbf{x}_j - \mathbf{x}_i)$, the kernel factorizes the effect from the radial part $\|\mathbf{a}_{ij}\|$ and

the direction $\hat{\mathbf{a}}_{ij}$ of the input edge feature. Each Input rotational order l_I interacts with the rotational order of the Filter l_F to Output various rotational orders l_O . We have $\kappa(\mathbf{a}_{ij}) = \varphi(\|\mathbf{a}_{ij}\|)_{l_I l_F, c}^{l_O} \mathbf{Y}^{l_F}(\hat{\mathbf{a}}_{ij})$. The radial filters $\varphi_{l_I l_F, c}^{l_O}$ are implemented as Multi-Layer Perceptrons (MLPs, to be learned) that share some weights among triplets l_O, l_I, l_F and channels (details in Appendix A 1). Expliciting the C-G tensor product, Eq. 1 now reads:

$$\mathbf{h}_{i,c,l_I l_F}^{l_O} = \sum_{j \in \mathcal{N}(i)} \varphi(\|\mathbf{a}_{ij}\|)_{l_I l_F, c}^{l_O} \mathbf{Y}^{l_F}(\hat{\mathbf{a}}_{ij}) C_{l_I l_F}^{l_O} \mathbf{h}_{j,c}^{l_I} \quad (10)$$

This operation is depicted in Fig. 3 (left part). At this stage, operations are performed channel-wise, but $\mathbf{h}_{i,c}^{l_O}$ is a concatenation of all possible triplets, and as multiple combinations of l_I, l_F can contribute to a given l_O , it is larger than the original feature $\mathbf{h}_{i,c}$. For $l_{max} = 3$, there are 34 different triplets (instead of just 4 different values of l).

To go back to a reduced representation, we mix together the triplets that share the same output l_O with a linear layer. However, to let the various channel interact, we also perform channel mixing (also called self-interaction) with a linear layer. As linear layers combine linearly, this can be expressed as a single linear layer (see right part of Fig. 3):

$$\mathbf{h}_{i,c}^{out, l_O} = \sum_{l_I l_F, c'} w_{l_I l_F, cc'}^{l_O} \mathbf{h}_{i,c', l_I l_F}^{l_O} \quad (11)$$

where all operations are now performed node-wise and independently for each l_O . Note that this operation fulfills equivariance since only features with the same l_O are combined with weights that don't depend on m (all elements inside a vector are multiplied by the same factor) thus preserving rotation properties. At this point we are back to our expected node feature shape, and the convolution layer can be repeated (up to a few technical details like using Batch-Norm and adding the previous layer's representation to the newly computed one, see Appendix A 2).

Interpretation Here we want to insist that the first layer especially, and to some extent the next ones, have a very clear interpretation. Let us consider the first convolution, for which the input node features consist in two $l = 0$ channels, namely the one-hot encoding of the particle type. Since $l_I = 0$, the only non-zero C-G coefficients will be the ones at $l_O = l_F = 0, 1, 2, 3$. For simplicity of view, let us imagine that the radial filters $\varphi_{l_I=0, l_F, c}^{l_O=l_F}$ are not learned anymore, but consist of a Gaussian Radial Basis, with numerous ‘‘soft bins’’ (i.e. numerous channels). Then, the first layer's ($L = 0$) action is to convolve the particle density with our kernel, i.e. to build a high-dimensional embedding of the density field in the ball of radius d_c (each particle type is associated to a given density field). Then the channel mixing step Eq. (11) does not have to mix any triplet (since $l_O = l_F$) and thus simply mixes channels. Taking the norm of this

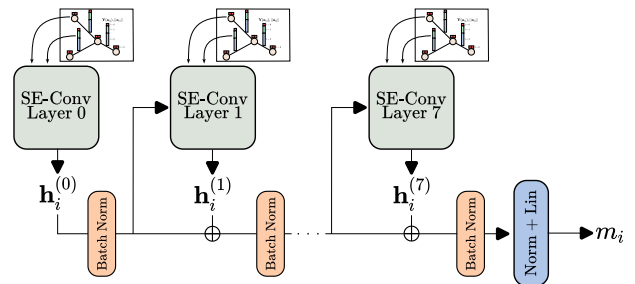


FIG. 4: **Overall Architecture.** Arrows connecting the embedded graph features to each convolution block show that the initial information (one-hot particle types and relative positions in SH and radial basis) is fed to each layer.

representation, one qualitatively obtains the expert descriptors described in [BSF21] (for an exact and more detailed derivation, see Appendix C). For a quantitative match however, one would need to use a much larger cut-off radius $d_c = 5$ for building the graph and a maximum rotational order of $l_{max} = 12$.

In our network, we build analogous intermediate features but we do not rush to compute the norm (the invariant feature), instead we remain at the equivariant level to combine them, keeping the computation of invariants as the last step of our network. We believe that this difference significantly contributes to making our model much more expressive than direct invariant expert features.

III. COMBINING EQUIVARIANT LAYERS

Network Our network is composed of embedding blocks for nodes and edges features followed by a series of SE(3)-equivariant convolutional layers interspersed with batch normalization connected in a Res-Net fashion and one output block, as shown in Fig. 4.

Here we provide a few insights on some key parts that are specific to SE(3)-aware networks. Further details about the architecture and the training procedure are available in Appendix A.

Batch Normalization As often in Neural Networks, we sometimes need to perform Batch Normalization to avoid the neuron's activation to take overly large values. However, using a usual batch normalization layer [IS15] separately on each entry of the hidden representations \mathbf{h} would kill the equivariance property. Thus a modified version is implemented and applied to node features [WGW+18, GSM+22]. The $l = 0$ features are invariant and thus can be processed as usual:

$$\frac{h^0 - \bar{h}}{\sigma} \beta + \gamma \quad (12)$$

where $\bar{h} = \langle h^0 \rangle$ and $\sigma^2 = \langle h^{0^2} \rangle - \langle h^0 \rangle^2$ with $\langle \cdot \rangle$ batch average computed with 0.5 momentum (keeping memory of previous batches) and β, γ are learned parameters. For

each piece of feature with $l \geq 0$, only the norm can be modified:

$$\frac{\|\mathbf{h}^l\|}{\sigma^l} \beta^l \quad (13)$$

with $\sigma^l = \sqrt{\langle \|\mathbf{h}^l\|^2 \rangle} / \sqrt{2l+1}$ and β^l learned parameter. In Fig. 4 we show where this Batch Norm is used.

Output Block After the last convolution layer, the ultimate output block performs two node-wise operations. First it extracts SE(3)-invariant features from the hidden representations. The norm of each directional ($l > 0$) feature $h^l = \sqrt{\sum_m h_m^l{}^2}$ is computed, then they are all concatenated together with the $l = 0$ features (already invariant). At this stage the network’s representation of the input is a vector of invariant features. The second operation is to feed this vector into a linear layer that outputs one real value, to predict the mobility.

We note that all the layers act linearly on the node features. The only non-linearities of the network are hidden in the implementation of the radial part of filters. This limited scope of non-linearities is unusual, and is needed to preserve equivariance (as pointed out above when we describe Batch Norm).

IV. EXPERIMENTS, RESULTS

Here we report the performance of our architecture on the glass data set mentioned above, and compare it with the previous State Of The Art (SOTA) performance model, namely the GNN presented in [BKGB⁺20], and the expert approach of [BSF21].

Experimental setup For each state point (4 different temperatures), we have 10 different tasks that correspond to the prediction of mobility at 10 different times. As in the works we compare with, we use the Pearson correlation coefficient as performance metric, which is invariant under shift and scale of the test labels distribution. Thus when comparing different tasks, we at least discount the trivial effect of the dispersion of labels’ values (the Loss, by comparison, is sensitive to the scale’s of labels’ distribution).

The network architecture and all hyper-parameter choices were optimized for a single task ($T = 0.44$ and longest timescale $\tau = 3 \times 10^4$), using only the train and validation sets. The resulting choices were applied straightforwardly to other tasks, thus preventing over-tuning of the hyper-parameters. The main architecture’s and hyper-parameters choices are the following. The number of convolution layers is $n_{(L)} = 8$. In the input layer, the visible representation has 2 channels of $l = 0$ features (the encoded particle type). At each layer $L > 0$ the internal or hidden representation $\mathbf{h}_i^{(L)}$ (for particle i at layer (L)) has 4 channels, with a maximum rotational order $l_{max} = 3$. These choices arise from striking a balance between over- and under-fitting, under our compute-time and memory budget constraints.

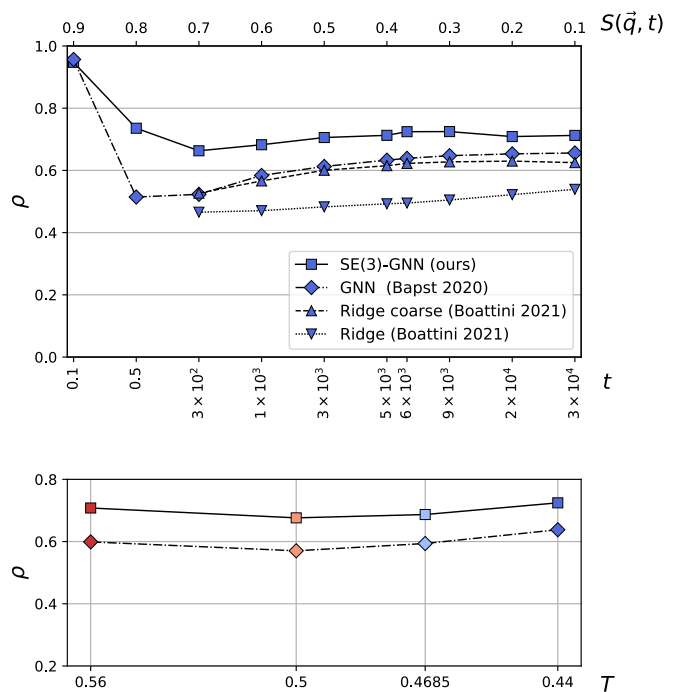


FIG. 5: **SE(3)-GNN outperforms conventional GNN across time scales and temperatures.** (top) Correlation coefficient ρ as a function of the time scale probed, for $T = 0.44$. We observe a dip at intermediate times, but less pronounced than SOTA. (bottom) Dependence on Temperature, for each state point $\tau = \tau_\alpha(T)$. Here the comparison with [BSF21] is not available. Overall, at all points studied our model has a higher correlation coefficient than the baseline, always by a significant margin.

Note that we perform a different train-test split with respect to [BKGB⁺20], which does not explicitly use a test set. Here, for each state point, 400 configurations are used for training, 320 for validation and 80 for the final test.

In Appendix A, we provide more details about the training of the model.

Results In Fig. 5 and table I, we show our main result, i.e. that our model consistently outperforms the SOTA [BKGB⁺20] by 5 to 10% depending on the task, in terms of test accuracy, despite having much fewer learnable parameters. For comparison, we also report results from the expert approach [BSF21], which nearly matches the GNN baseline.

The convolutional layers that we stack, as described in section III, effectively build a *representation* describing the environment of each particle, that is then regressed to our target label, the mobility. To probe the robustness (generalization ability) of this representation, we perform a transfer learning experiment across temperatures: we train the full model at a single temperature, then freeze the parameters in all layers except from the very last one. At all other temperatures we then re-train (the last layer

Model	$\tau = 1 \cdot 10^3$	$3 \cdot 10^3$	$5 \cdot 10^3$	$6 \cdot 10^3$	$9 \cdot 10^3$	$2 \cdot 10^4$	$3 \cdot 10^4$	Num. parameters
GNN (Bapst 2020)	0.584	0.613	0.633	0.639	0.647	0.653	0.656	70 721
SE(3)-GNN $l_{max} = 3$	0.683	0.706	0.713	0.725	0.725	0.709	$\in [0.7095, 0.7169]$	23 177
SE(3)-GNN $l_{max} = 1$	0.635	0.660	0.672	0.690	0.696	0.689	0.676	5 381
Ridge (Boattini 2021)	0.566	0.601	0.615	0.623	0.627	0.630	0.625	~ 1000

TABLE I: **SE(3)-GNN outperforms SOTA despite having a smaller number of parameters.** Numbers display the Pearson correlation coefficient ρ between predicted and true mobilities for different timescales τ at $T = 0.44$. For comparison, ridge regression from [BSF21] is also reported. Due to lack of time, the confidence interval is only available for one task (see Appendix B for details). Notice that our model is able to beat the SOTA also with $l_{max} = 1$ at a significantly reduced number of learnable parameters.

only) and test the model. Let’s recall that the operation performed just before our last layer is to compute the norm of the representation $\mathbf{h}^{(L=7)}$ to obtain a set of invariant features: for each $l = 1, 2, 3$ and each $c = 1, 2, 3, 4$ we compute the norm of $\mathbf{h}_{i,c}^{l,(L=7)}$, while for $l = 0$ the feature $\mathbf{h}_{i,c}^{l=0,(L=7)}$ is already invariant, so taking the norm is useless. Thus, the representation which summarizes a particle’s environment and is transferred across temperatures consists in exactly 16 values. The final (linear) layer simply performs linear regression between these 16 values and the target label, here the mobility. In Fig. 6 we show that the model generalizes well across temperatures (full lines), with an accuracy that decreases when the difference in train and test temperatures increases.

Furthermore we perform an even simpler check: without re-learning anything, we take a model trained at a temperature and test it directly at the other temperatures (dashed lines). Since the performance metric ρ is insensitive to global shift and scale of the labels’ distribution, there is no need to perform such global re-shift and re-scale when comparing values of ρ (when comparing output mobilities, such a re-scaling would be needed however). Here again we still obtain good performances, significantly better than the SOTA: compare with the Fig. 3d of [BKGB+20]. Overall, this shows that our learned representation is not strongly training-temperature-dependent, a desirable property in the perspective of learning a structural order parameter.

All our results rely on the embedding of the input data into the Spherical Harmonics basis and on the built-in equivariance of convolution layers. So one may expect the choice of the cutoff rotational order l_{max} to be crucial. In Fig. 7 we show it is indeed the case: we build architectures that vary only by their l_{max} value and measure the performance ρ in each. The biggest gap in performance is indeed observed between purely isotropic, scalar features ($l_{max} = 0$) versus directional ones ($l_{max} = 1$). Further increasing l_{max} provides a finer angular resolution, but we observe that the accuracy tends to saturate. We cannot go above $l_{max} = 3$ due to memory constraints: as the rotational order increases, the number of activations of neurons to keep track of grows exponentially with l_{max} (while it grows linearly with the number of edges, with the batch size, and the with size of the hidden layer in

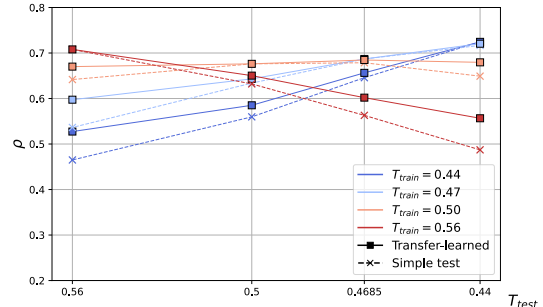


FIG. 6: **Transfer-learning between different temperatures.** Each model is trained once at one state point and tested on the remaining ones. The timescale of mobilities is fixed at $\tau = \tau_\alpha(T)$. Best generalization performances are observed for model trained at $T = 0.50$.

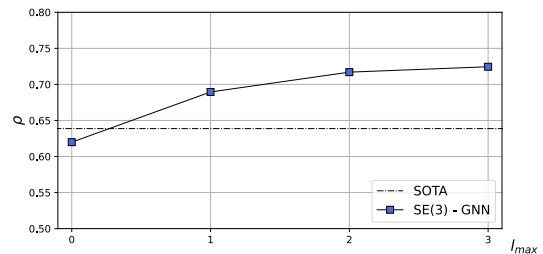


FIG. 7: **Rotational order matters.** Performance ρ as a function of l_{max} , the maximum rotational order retained in the Spherical Harmonics embedding of the edge feature. Results shown for $T = 0.44$ and $\tau = \tau_\alpha$

the radial MLP).

V. DISCUSSION

As is well known in physics, enforcing symmetries is key to reducing the number of degrees of freedom in the description of a physical problem. In the ML vocabulary, this translates as building features that respect the proper symmetries. We achieve this by combining two ingredients. First, the proper embedding by Spherical Harmonics builds a description of the neighborhood that is both easy-to-rotate and compact, since it is a decom-

position in rotational order, akin to frequency decomposition, a much more efficient representation than directly discretizing angles. Second, layers combine the hidden representations $\mathbf{h}_i^{(L)}$ in a way that preserves equivariance up to the last layer, a choice that compared with rushing to invariant features, guarantees maximum expressivity for the network. We believe that these two ingredients are key to building a good representation: we surpass the SOTA and achieve better generalization across tasks, while using much fewer learnable parameters.

While we cannot claim our network to be fully interpretable, our first hidden feature $\mathbf{h}_i^{(0)}$ is clearly interpreted as a representation of the field of density around node i , in a way that relates with the rotation-invariant features introduced in [BSF21].

The present work focuses on building a representation for glassy materials, but we would like to stress that progress in this area is intimately connected to progress made in other application areas, whenever the input data consists in particles living in 3D space (as in *ab initio* or effective potential approximations, crystalline materials or amorphous materials’ properties prediction), regardless of the precise task or output label. While each of these application tasks may need fine-tuning of the architecture to perform well, we believe that they are essentially different facets of the same problem, that is, efficient learning of representations for sets of particles in space.

Although we performed neural architecture search by hand somehow to achieve these results, we did not yet explore the space of possibilities thoroughly, and there is ample room for improvement. For instance, one could include attention mechanisms, or bottleneck-styled ar-

chitectures (varying $l_{max}^{(L)}$ across layers), or simply using larger l_{max} (would require multi-GPU training). Besides, one could mix different tasks, beyond the obvious multi-time and multiple particle-type regression and mixing of temperatures of the same glass-forming liquids: one could mix tasks between various glass-forming liquids and crystalline materials, together with other amorphous material’s input, thus requiring the backbone of the network to generalize even more strongly. This kind of pre-training strategy has been shown [LWL+21, FLL+22, ZXJ+22] to be effective to improve robustness.

Whatever improvements one could think of, we believe that the SE(3)-GNN is the right framework to be developed. Indeed, it seems in line with the recent history of neural architecture design: while the CNN architecture has been a game-changer thanks to its enforcement of translation-equivariance by design, the GNNs then enforced the node permutation-equivariance by construction, and SE(3)-GNNs now additionally enforce the rotation-equivariance, leveraging all the symmetries at hand.

Acknowledgments

We are thankful to [BKGB+20] for sharing their dataset publicly, a good practice that should be fostered in the physics’ community.

We are thankful to the e3nn library developers [GSM+22] for developing their E(3)-GNN library, which should give a huge boost to work in this area.

This work is supported by a public grant overseen by the French National Research Agency (ANR) through the program UDOPIA, project funded by the ANR-20-THIA-0013-01.

-
- [ABFS22] Rinske M. Alkemade, Emanuele Boattini, Laura Filion, and Frank Smalenburg. Comparing machine learning techniques for predicting glassy dynamics. *arXiv:2202.09173 [cond-mat]*, February 2022.
- [ALBB21] Francesco Arceri, François Landes, Ludovic Berthier, and Giulio Biroli. A statistical mechanics perspective on glasses and aging. *Encyclopedia of Complexity and Systems Science*, pages 1–68, 2021.
- [BBB+07] L. Berthier, G. Biroli, J.-P. Bouchaud, W. Kob, K. Miyazaki, and D. R. Reichman. Spontaneous and induced dynamic fluctuations in glass formers. I. General results and dependence on ensemble and dynamics. *The Journal of Chemical Physics*, 126(18):184503, May 2007.
- [BHvdP+22] Johannes Brandstetter, Rob Hesselink, Elise van der Pol, Erik J. Bekkers, and Max Welling. Geometric and Physical Quantities Improve E(3) Equivariant Message Passing. *arXiv:2110.02905 [cs, stat]*, March 2022.
- [BJ07] Ludovic Berthier and Robert L. Jack. Structure and dynamics in glass-formers: predictability at large length scales. *Physical Review E*, 76(4):041509, October 2007.
- [BKGB+20] V. Bapst, T. Keck, A. Grabska-Barwińska, C. Donner, E. D. Cubuk, S. S. Schoenholz, A. Obika, A. W. R. Nelson, T. Back, D. Hassabis, and P. Kohli. Unveiling the predictive power of static structure in glassy systems. *Nature Physics*, 16(4):448–454, April 2020.
- [BMAM+20] Emanuele Boattini, Susana Marín-Aguilar, Saheli Mitra, Giuseppe Foffi, Frank Smalenburg, and Laura Filion. Autonomously revealing hidden local structures in supercooled liquids. *Nature Communications*, 11(1):5479, December 2020.
- [BMS+21] Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P. Mailoa, Mordechai Korabluth, Nicola Molinari, Tess E. Smidt, and Boris Kozinsky. E(3)-Equivariant Graph Neural Networks for Data-Efficient and Accurate Interatomic Potentials. *arXiv:2101.03164 [cond-mat, physics:physics]*, December 2021. arXiv: 2101.03164.
- [BSF21] Emanuele Boattini, Frank Smalenburg, and Laura Filion. Averaging local structure to predict the dynamic propensity in supercooled liquids. *Physical Review Letters*, 127(8):088007, August 2021. arXiv: 2105.05921.
- [CIS+17] E. D. Cubuk, R. J. S. Ivancic, S. S. Schoenholz, D. J. Strickland, A. Basu, Z. S. Davidson, J. Fontaine, J. L. Hor, Y.-R. Huang, Y. Jiang, N. C. Keim, K. D. Koshigan, J. A. Lefever, T. Liu, X.-G. Ma, D. J. Maga-

- gnosc, E. Morrow, C. P. Ortiz, J. M. Rieser, A. Shavit, T. Still, Y. Xu, Y. Zhang, K. N. Nordstrom, P. E. Arratia, R. W. Carpick, D. J. Durian, Z. Fakhraai, D. J. Jerolmack, Daeyeon Lee, Ju Li, R. Riggleman, K. T. Turner, A. G. Yodh, D. S. Gianola, and Andrea J. Liu. Structure-property relationships from universal signatures of plasticity in disordered solids. *Science*, 358(6366):1033–1037, November 2017. ISBN: 3487716170192.
- [Cos11] Daniele Coslovich. Locally preferred structures and many-body static correlations in viscous liquids. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 83(5):1–8, 2011. arXiv: 1102.5663.
- [CSR⁺15] E. D. Cubuk, S. S. Schoenholz, J. M. Rieser, B. D. Malone, J. Rottler, D. J. Durian, E. Kaxiras, and A. J. Liu. Identifying structural flow defects in disordered solids using machine-learning methods. *Physical Review Letters*, 114(10):1–5, 2015. arXiv: 1409.6820.
- [CWCK⁺10] R. Candelier, A. Widmer-Cooper, J. K. Kummerfeld, O. Dauchot, G. Biroli, P. Harrowell, and D. R. Reichman. Spatiotemporal hierarchy of relaxation events, dynamical heterogeneities, and structural reorganization in a supercooled liquid. *Physical Review Letters*, 105(13):135702, September 2010. arXiv: 0912.0193 ISBN: 0031-9007\n1079-7114.
- [CYZ⁺19] Chi Chen, Weike Ye, Yunxing Zuo, Chen Zheng, and Shyue Ping Ong. Graph Networks as a Universal Machine Learning Framework for Molecules and Crystals. *Chemistry of Materials*, 31(9):3564–3572, May 2019.
- [FL19] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
- [FLL⁺22] Xiaomin Fang, Lihang Liu, Jieqiong Lei, Donglong He, Shanzhuo Zhang, Jingbo Zhou, Fan Wang, Hua Wu, and Haifeng Wang. Geometry-enhanced molecular representation learning for property prediction. *Nature Machine Intelligence*, 4(2):127–134, 2022.
- [GPS16] Sebastian Golde, Thomas Palberg, and Hans Joachim Schöpe. Correlation between dynamical and structural heterogeneities in colloidal hard-sphere suspensions. *Nature Physics*, 12(7):712–717, July 2016.
- [GSM⁺22] Mario Geiger, Tess Smidt, Alby M., Benjamin Kurt Miller, Wouter Boomsma, Bradley Dice, Kostiantyn Lapchevskiy, Maurice Weiler, Michał Tyszkiewicz, Simon Bätzner, Dylan Madisetti, Martin Uhrin, Jes Frelsen, Nuri Jung, Sophia Sanborn, Mingjian Wen, Josh Rackers, Marcel Rød, and Michael Bailey. Euclidean neural networks: e3nn. April 2022.
- [HSD⁺21] Weihua Hu, Muhammed Shuaibi, Abhishek Das, Siddharth Goyal, Anuroop Sriram, Jure Leskovec, Devi Parikh, and C. Lawrence Zitnick. ForceNet: A Graph Neural Network for Large-Scale Quantum Calculations. *arXiv:2103.01436 [cs]*, March 2021. arXiv: 2103.01436.
- [HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015.
- [IS15] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, March 2015.
- [KIG11] Aaron S. Keys, Christopher R. Iacovella, and Sharon C. Glotzer. Characterizing Structure Through Shape Matching and Applications to Self-Assembly. *Annual Review of Condensed Matter Physics*, 2(1):263–285, March 2011.
- [KLT18] Risi Kondor, Zhen Lin, and Shubhendu Trivedi. Clebsch-Gordan Nets: a Fully Fourier Space Spherical Convolutional Neural Network. *arXiv:1806.09231 [cs, stat]*, November 2018.
- [KMB⁺16] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design*, 30(8):595–608, August 2016.
- [LBD⁺20] François P. Landes, Giulio Biroli, Olivier Dauchot, Andrea J. Liu, and David R. Reichman. Attractive versus truncated repulsive supercooled liquids: The dynamics is encoded in the pair correlation function. *Physical Review E*, 101(1):010602, January 2020. arXiv: 1906.01103.
- [LBVP22] Matthias Lerbinger, Armand Barbot, Damien Vandembroucq, and Sylvain Patinet. Relevance of shear transformations in the relaxation of supercooled liquids. *Physical Review Letters*, 129(19):195501, 2022.
- [Ler20] Matthias Lerbinger. *Local shear rearrangements in glassy systems: from micromechanics to structural relaxation in supercooled liquids*. PhD thesis, Université Paris sciences et lettres, 2020.
- [LWL⁺21] Shengchao Liu, Hanchen Wang, Weiyang Liu, Joan Lasenby, Hongyu Guo, and Jian Tang. Pre-training molecular graph representation with 3d geometry. *arXiv preprint arXiv:2110.07728*, 2021.
- [PJC20] Joris Paret, Robert L. Jack, and Daniele Coslovich. Assessing the structural heterogeneity of supercooled liquids through community inference. *The Journal of Chemical Physics*, 152(14):144502, April 2020.
- [RK17] C. Patrick Royall and Walter Kob. Locally favoured structures and dynamic length scales in a simple glass-former. *Journal of Statistical Mechanics: Theory and Experiment*, 2017(2), 2017. arXiv: 1611.03314 Publisher: IOP Publishing.
- [RSL14] Jörg Rottler, Samuel S. Schoenholz, and Andrea J. Liu. Predicting plasticity with soft vibrational modes: From dislocations to glasses. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 89(4):1–6, 2014. arXiv: 1403.0922.
- [RTS20] C. Patrick Royall, Francesco Turci, and Thomas Speck. Dynamical Phase Transitions and their Relation to Thermodynamic Glass Physics. *arXiv:2003.03700 [cond-mat]*, March 2020. arXiv: 2003.03700.
- [Sch17] Samuel S. Schoenholz. Combining Machine Learning and Physics to Understand Glassy Systems. 2017. arXiv: 1709.08015.
- [SCKL16] Samuel S. Schoenholz, Ekin D. Cubuk, Efthimios Kaxiras, and Andrea J. Liu. The Relationship Between Local Structure and Relaxation in Out-of-Equilibrium Glassy Systems. *Proceedings of the National Academy of Sciences*, 114(2):263–267, January 2016. arXiv: 1607.06969.
- [SCS⁺15] Samuel S. Schoenholz, Ekin D. Cubuk, Daniel M. Sussman, Efthimios Kaxiras, and Andrea J. Liu. A structural approach to relaxation in glassy liquids. *Nature Physics*, 12(5):469–471, February 2015. arXiv: 1506.07772 ISBN: 1745-2473.
- [SLRR14] S. S. Schoenholz, A. J. Liu, R. A. Riggleman, and J. Rottler. Understanding plastic deformation in thermal glasses from single-soft-spot dynamics. *Physical Review X*, 4(3):1–11, 2014. arXiv: 1404.1403.
- [SSCL17] Daniel M. Sussman, Samuel S. Schoenholz, Ekin D.

- Cubuk, and Andrea J. Liu. Disconnecting structure and dynamics in glassy thin films. *Proceedings of the National Academy of Sciences*, 114(40):10601–10605, October 2017.
- [STC⁺17] Tristan A Sharp, Spencer L Thomas, Ekin D Cubuk, Samuel S Schoenholz, David J Srolovitz, and Andrea J Liu. Using machine learning to extract atomic energy barriers in polycrystalline materials. 1:1–5, 2017.
- [TSK⁺18] Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds. *arXiv:1802.08219 [cs]*, May 2018. arXiv: 1802.08219.
- [TT18] Hua Tong and Hajime Tanaka. Revealing Hidden Structural Order Controlling Both Fast and Slow Glassy Dynamics in Supercooled Liquids. *Physical Review X*, 8(1):011041, March 2018.
- [TTR17] Francesco Turci, Gilles Tarjus, and C. Patrick Royall. From Glass Formation to Icosahedral Ordering by Curving Three-Dimensional Space. *Physical Review Letters*, 118(21):1–5, 2017. arXiv: 1609.03044.
- [TTSR19] Hajime Tanaka, Hua Tong, Rui Shi, and John Russo. Revealing key structural features hidden in liquids and glasses. *Nature Reviews Physics*, 1(5):333–348, May 2019.
- [WGW⁺18] Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco Cohen. 3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data. *arXiv:1807.02547 [cs, stat]*, October 2018. arXiv:1807.02547 [cs, stat].
- [ZXJ⁺22] Zuobai Zhang, Minghao Xu, Arian Jamasb, Vijil Chenthamarakshan, Aurelie Lozano, Payel Das, and Jian Tang. Protein representation learning by geometric structure pretraining. *arXiv preprint arXiv:2203.06125*, 2022.
- [45] “Node features” is Machine Learning Vocabulary for “set of values associated to the node. Similarly for edge features.

Appendix A: Training details, Training curve and time/energy cost

Here we provide more details about our architecture, hyper-parameters and how the model is trained.

1. Radial MLP

As mentioned in the main text, the radial MLPs are the only part of the network with non-linearities. They implement the radial dependence of convolution filters $\varphi_{l_I l_F, c}^{l_O}(\|\mathbf{a}_{ij}\|)$, thus they take as input the norms of relative node positions. Before being fed to the MLPs each norm is expanded from a real value to an array through an embedding. Here we use a Bessel basis embedding:

$$B_n(r) = \sqrt{\frac{2}{r_c}} \frac{\sin(n\pi \frac{r}{r_c})}{r} \quad (\text{A1})$$

with $r_c = d_c = 2$ and $n = 1, 2, \dots, N_b = 10$ is the number of roots of each basis function. Other embeddings could be for instance a Gaussian basis (with cutoff), which

would act as a kind of smooth one-hot encoding of the value r . In practice, the Bessel basis has better generalization properties.

The embedded input is processed through an MLP with layers sizes $(N_b, 16, n_{comb})$ and ReLU non linearities. The output size n_{comb} is the number of possible triplets, times the number of channels. We also use BatchNorm (BN) [IS15] and Dropout (with rate $p = 0.5$) in this MLP to stabilize training and reduce overfitting. In summary, for each combination of triplets and channels (l_O, l_I, l_F, c) , we have a real output $\varphi_{l_I l_F, c}^{l_O} = \sigma(W_{n_{comb}, 16} \text{Dropout}(\text{BN}(\sigma(W_{16, 10} B(\|\mathbf{a}_{ij}\|))))))$, where the W 's are weight matrices and $\sigma(z) = \max(0, z)$. There are bias parameters, which are not displayed here. Note that up to the layer of 16 neurons, the MLP is the same for all triplets and channels, only the last linear layer introduces different weights for each combination. In total, the MLPs of our network (across all layers) account for a number of 18 936 learnable parameters: in each layer $L > 0$ we have one radial MLP of size $(10, 16, 144)$ with 2656 parameters, for the layer $L = 0$ the MLP is of size $(10, 16, 2 \times 4)$ with 344 parameters. The other main source of learnable parameters in the Network is the part of mixing the channels (right part of fig. 3), which accounts for 4064 learnable parameters: $144 \times 4 = 576$ for $L > 0$ layers and $8 \times 4 = 32$ for the $L = 0$ layer.

2. Overall Architecture

Steerable group convolution blocks are connected in a way similar to the widespread Res-Net [HZRS15] structure, which is known to speed up the training. The output of each layer is added to its own input and it is passed through one batch-normalization block to the next layer: see Fig. 4. After the final convolution, node features are passed to the output block acts where invariants are computed and combined through a linear layer. This architecture choice is found empirically to be the most stable at train time. Our architecture is built and trained in the framework of PyTorch Geometric [FL19] which handles all the generic graph operations. All the SE(3)-related operations (SH embedding, C-G tensor product, equivariant batch-normalization) are integrated in this framework thanks to the *e3nn* library [GSM⁺22].

3. Training strategy

In Fig. 8 we display one learning curve (as function of iterations, epochs). Each epoch is a sweep over the entire 400 samples dataset (each sample represents $N = 4096$ atoms).

For training, we use Adam optimizer with initial learning rate $\gamma = 10^{-3}$, moments $\beta_1 = 0.95, \beta_2 = 0.999$ and weight decay $\lambda = 10^{-7}$. We also add a learning rate

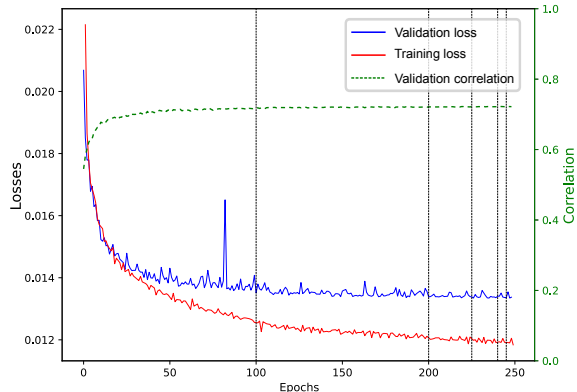


FIG. 8: **Loss and ρ vs epoch.** Training of our main model performed at $T = 0.44$, $\tau = 3 \times 10^4$ (longest timescale available). Vertical dashed lines locate the epochs at which the learning rate is divided by 2.

scheduler that divides γ by 2 at several epochs as shown by the vertical dashed lines in Fig. 8.

Each training of our model takes approximately 20 hours on a A100 Nvidia GPU. This represents approximately 4 kWh per training, and in France, an equivalent of 300 gCO₂ (we use a number of 75 gCO₂/kWh, after considering quickly EDF’s optimistic figure of 25 g/kWh or RTE’s more detailed daily numbers, which oscillate mostly between 50 and 100, depending on the season and time of day). We did not include the footprint of manufacturing of the GPU and other infrastructure, which is generally estimated to be one half of the IT-related CO₂ emissions (the other half being running the infrastructure).

Appendix B: Measures of the errors

There are two sources of uncertainty when measuring the accuracy of a given model (a model is a ML architecture).

The first, that one usually reports, comes from varying the initialization of the weights in the Neural Network (NN). Indeed, as gradient descent only finds a local minimum, the learned weights depend on the (random) initialization. To account for this variability, one typically re-trains the network multiple times, each time with a different initial random seed. Due to lack of time, we do not compute these error bars thoroughly at all timescales and temperatures, but we did a focused study at $T = 0.44$ and $\tau = 3 \times 10^4$ (longest timescale available), see Table II. From this, we conclude that the true accuracy lies approximately within the range $[0.7095, 0.7169]$ (a spread of 0.0074), i.e. the accuracy is 0.713 ± 0.004 .

The second source of uncertainty can come from the test set itself (being too small). A way to assess the

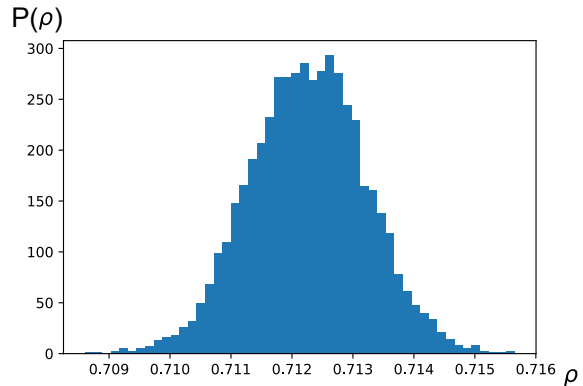


FIG. 9: **Distribution of accuracy over bootstrap samples.** Train/Test performed at $T = 0.44$ and $\tau = 3 \times 10^4$. Number of re-samplings $N_{res} = 5000$.

test set robustness, i.e. how confident we can be that our accuracy would be in the same range, for an ideal infinite test set, is to bootstrap the test set. We do not recall the classic bootstrap idea here. The test set is made of 80 configurations of $N = 4096$ atoms each, for which 80% are A particles (for which we regress), this results in $n = 80 * 4096 * 0.80 = 262144$ samples. We sample with replacement n out of these n samples, and measure ρ on this “new” test set. Repeating this operation 5000 times, we obtain the empirical distribution $P(\rho)$ shown in Fig. 9. From this, we conclude that for this particular seed the true accuracy is 0.712 ± 0.001 .

We find that the fluctuations coming initialization of the Network are very much comparable (same order of magnitude) to those due to the finite-size test set fluctuations (evaluated by bootstrap).

Appendix C: Reproducing expert features (Boattini 2021)

Here we show how a network with one SE(3)-convolution layer is able to reproduce some good expert features used for glassy liquids. We start by considering embedded node features $h_{i,c} = \delta_{t_i, (1-c)}$ where t_i is the type of particle i : we have only two channels at $l = 0$ that we extend from $n_{ch} = 2 \rightarrow n_{ch} = 2 * N_b$ just by replicating them N_b times. We will denote the replicated $h_{i,c}$ as $h_{i,c,r}$ since in spirit, each copy of the two channels (one-hot of the particle type) will correspond to one radial basis function. Then the convolution operation reads:

$$\mathbf{h}'_{i,c,r,l_I=0,l_F} = \sum_{j \in \mathcal{N}(i)} \varphi(\|\mathbf{a}_{ij}\|)_{l_I=0,l_F,c,r}^{l_F} \mathbf{Y}^{l_F}(\hat{\mathbf{a}}_{ij}) h_{j,c,r} \quad (\text{C1})$$

We choose a Gaussian radial basis function B (instead of Bessel): $\varphi(\|\mathbf{a}_{ij}\|)_{l_I=0,l_F,c,r}^{l_F} = B(\|\mathbf{a}_{ij}\|)_r$. Then if we

Seed number	1	2	3	4	5	6	7	8	9	10
ρ	0.7122	0.7095	0.7135	0.7099	0.7128	0.7120	0.7169	0.7117	0.7128	0.7165

TABLE II: **Accuracy across random initializations.** Each seed of the random number generator determines a different initial set of weights for the model. We report the test accuracy of each model independently trained at $T = 0.44$, $\tau = 3 \times 10^4$. Best and worst performance are highlighted.

focus on $l_F = 0$, since $Y^0(\hat{\mathbf{a}}_{ij}) = 1$ we have:

$$h'_{i,c,r}{}^0 = \sum_{j \in \mathcal{N}(i)} e^{-\frac{(\|\mathbf{a}_{ij}\| - r)^2}{2\delta}} \delta_{t_j, (1-c)} \quad (\text{C2})$$

which correspond to $G_i^{(0)}(r, \delta, s)$ in [BSF21]. Note that we require also no channel mixing at $l = 0$. For $l_F > 0$:

$$h'_{i,c,r}{}^{l_F} = \sum_{j \in \mathcal{N}(i)} \mathbf{Y}^{l_F}(\hat{\mathbf{a}}_{ij}) e^{-\frac{(\|\mathbf{a}_{ij}\| - r)^2}{2\delta}} \delta_{t_j, (1-c)} \quad (\text{C3})$$

which, after a channel mixing step that sums over c (mixing different particle types), correspond to $q_i^{(0)}(l, m, r, \delta)$ in [BSF21]. By computing invariants from these features through their norm, we recover exactly $q_i^{(0)}(l, r, \delta)$ from [BSF21]. By contrast, in our model we do not compute invariants after one layer, we keep equivariant features and let them interact over multiple layers in order to increase the expressivity.