A Hitchhiker's Guide to Geometric GNNs for 3D Atomic Systems

Alexandre Duval, Simon V Mathis, Chaitanya K Joshi, Victor Schmidt, Santiago Miret, Fragkiskos D. Malliaros, Taco Cohen, Pietro Liò, Yoshua Bengio, Michael Bronstein

A Hitchhiker's Guide to Geometric GNNs for 3D Atomic Systems

Alexandre Duval*
Mila, Université Paris-Saclay†
alexandre.duval@mila.quebec

Simon V. Mathis*
University of Cambridge, UK
simon.mathis@cl.cam.ac.uk

Chaitanya K. Joshi*
University of Cambridge, UK
chaitanya.joshi@cl.cam.ac.uk

Victor Schmidt*
Mila, Université de Montréal
schmidtv@mila.quebec

Santiago Miret
Intel Labs

Fragkiskos D. Malliaros
Université Paris-Saclay†

Taco Cohen
Qualcomm AI Research‡

Pietro Liò
University of Cambridge, UK

Yoshua Bengio
Mila, Université de Montréal

Michael Bronstein
University of Oxford, UK

Abstract

Recent advances in computational modelling of atomic systems, spanning molecules, proteins, and materials, represent them as *geometric graphs* with atoms embedded as nodes in 3D Euclidean space. In these graphs, the geometric attributes transform according to the inherent physical symmetries of 3D atomic systems, including rotations and translations in Euclidean space, as well as node permutations. In recent years, *Geometric Graph Neural Networks* have emerged as the preferred machine learning architecture powering applications ranging from protein structure prediction to molecular simulations and material generation. Their specificity lies in the inductive biases they leverage — such as physical symmetries and chemical properties — to learn informative representations of these geometric graphs.
In this opinionated paper, we provide a comprehensive and self-contained overview of the field of Geometric GNNs for 3D atomic systems. We cover fundamental background material and introduce a pedagogical taxonomy of Geometric GNN architectures: (1) invariant networks, (2) equivariant networks in Cartesian basis, (3) equivariant networks in spherical basis, and (4) unconstrained networks. Additionally, we outline key datasets and application areas and suggest future research directions. The objective of this work is to present a structured perspective on the field, making it accessible to newcomers and aiding practitioners in gaining an intuition for its mathematical abstractions.

*Equal first authors.
†Full affiliation: Université Paris-Saclay, CentraleSupélec, Inria.
‡Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

Contents

1 Introduction **6**

2 Preliminaries **8**

2.1 Graphs and Graph Neural Networks 8

2.2 Geometric Graphs 9

3 From GNNs to Geometric GNNs **11**

3.1 Input preparation 11

3.2 Learning representations of atoms 13

3.2.1 Embedding block: initialising latent representations 14

3.2.2 Interaction blocks: learning geometric and relational features 14

3.3 Output block: making predictions 15

4 Invariant Geometric GNNs **16**

5 Equivariant Geometric GNNs **19**

5.1 Equivariant GNNs with Cartesian tensors 20

5.2 From Cartesian tensors to spherical tensors – reducible to irreducible representations 26

5.3 Equivariant GNNs with spherical tensors – Irreducible representations 31

5.3.1 An example spherical EGNN 31

5.3.2 Optimisations and improvements. 35

6 Unconstrained Geometric GNNs **37**

7 Applications **40**

7.1 Tasks 40

7.1.1 Property Prediction 40

7.1.2 Interatomic Potentials for Molecular Dynamics Simulation 41

7.1.3 Generative Modeling and Design 42

7.1.4 Structure Prediction 42

7.2 Datasets 43

8 Conclusion and Future Research Directions **46**

8.1 To what extent should physics and symmetry be ‘baked in’ to Geometric GNNs? . 46

8.2 How to construct geometric graphs? 47

8.3 How to scale up Geometric GNNs? 48

A Lexicon **62**

A.1 Geometric vocabulary 62

A.2 Group theory 63

A.3 Data structures 64

A.3.1 (Small) Molecules . . . 64
A.3.2 Proteins . . . 65
A.3.3 Solid-State Materials . . . 66
A.4 Periodic boundary conditions . . . 66
A.5 Quantum chemistry background . . . 67
A.6 Energy conservation . . . 68
A.7 GNN architectural details . . . 68
A.7.1 Message Passing . . . 68
A.7.2 Activation function . . . 69
A.7.3 Basis functions . . . 69
A.7.4 Examples of architecture . . . 70
A.8 Expressive power of Geometric GNN . . . 71
A.9 On inductive biases . . . 72

B An opinionated history of methods **73**

C Data **74**
C.1 Data splits . . . 74
C.2 Examples of predicted properties . . . 75
C.3 Atom-type rescaling . . . 76

Notation

We assume the reader is familiar with basic machine learning terminology and common neural network architectures such as multi-layer perceptrons (MLPs). To keep the text clear and concise for readers with varying levels of prior knowledge, we include explanations of certain concepts in Appendix A. These concepts are marked with a ❖ symbol.

Key notations

Throughout this paper, we use an intuitive *visual grammar* to help readers separate key concepts mentally: (1) Bold characters represent collections (lists) of objects of the same type and have channel indices; (2) Underlined characters are learnable; (3) Characters with an arrow on top have a geometric meaning; (4) Geometric characters may carry component/node/channel indices as subscript and representation indices as superscript for higher tensor types.

This high-level visual grammar gives rise to the following notation:

- **Scalars**: Scalar quantities (simple numbers) are denoted by lowercase Latin letters $s \in \mathbb{R}$.
- **Geometric vectors**: Vector quantities with a geometric meaning carry an arrow on top to emphasise their geometric significance. In this work, all geometric vectors❖ are assumed to lie in a 3D space: $\vec{\mathbf{v}} \in \mathbb{R}^{b\times 3}, \vec{x} \in \mathbb{R}^3$. When evident from the context, we refer to geometric vectors as *vectors*.
- **Geometric tensors**: Higher-order tensors❖ with geometric meaning are denoted by uppercase letters with an arrow on top $(\vec{T}, \ldots)$. To explictly distinguish spherical tensors $\vec{T}^{(l)}$ we use a bracketed l superscript with l indicating the type of the tensor❖ . For Cartesian tensors❖ $\vec{T}^{[\mathfrak{c}]}$ of type $\mathfrak{c}$, we use a square-bracket superscript instead.
- **Lists of quantities**: To represent lists of multiple quantities of the same type, we print characters in bold. For example, $\mathbf{a}$ is a list of scalar quantities, $\vec{\mathbf{v}}$ is a list of geometric vectors, and $\vec{\mathbf{T}}^{(l)}$ is a list of spherical tensors of type l. For matrices of scalars (e.g. the adjacency matrix), we use bold uppercase letters $\mathbf{A}$. Its entries are written a_{ij} and row vectors $\mathbf{a}_i$.
- **Learnable quantities**: Learnable quantities are marked with an underline. For example $\underline{a}$ is a learnable scalar, $\underline{\mathbf{a}}$ is a list of learnable scalars, and $\underline{\mathbf{W}}$ is a learnable weight matrix.
- **Node indices**: We use i, j, k, l to denote specific node indices. For example, $\vec{v}_i$ is a geometric vector at node i. Directed edges are denoted as tuples (i, j).
- **Channel indices**: We use c to denote channel indices for lists or matrices of objects. For example, $\vec{\mathbf{v}}_{c_1}$ is the c_1-th geometric vector in $\vec{\mathbf{v}}$. Channel indices serve to distinguish the feature dimensions associated with the same atom, like atom type, atomic mass and electronegativity.
- **Component indices**: To refer to the dimensions representing the different components or features of a data point (e.g. dimension of geometric vectors), we use q for Cartesian and m for spherical geometric tensors. For instance, $\vec{T}^{[\mathfrak{c}]}_{q_1}$ is a component of a Cartesian tensor while $\vec{Y}^{(l)}_m$ is a component of a spherical tensor.

Special symbols

- $\mathcal{G}$ is used to refer to a graph with vertex set $\mathcal{V}_\mathcal{G}$ and edge set $\mathcal{E}_\mathcal{G}$.
- $\mathfrak{G}$ refers to the group $\mathfrak{G}$, e.g. $\mathfrak{G} = \mathrm{SO}(3)$.
- $\sigma(\cdot)$ denotes a non-linear activation function $\sigma : \mathbb{R} \to \mathbb{R}$. When applied to a list of objects $\mathbf{u}$, $\sigma(\mathbf{u})$ is understood to act channel-wise.
- $f(\cdot)$ is a general function, often representing a neural network.
- $\underline{\mathbf{W}}$ stands for the learnable weights of a neural network.
- $\psi(\cdot)$ stands for any basis function (e.g. radial basis, bessel function). See Appendix A.7.3.
- $\odot$ denotes the channel-wise product.

- $\otimes$ denotes the tensor product.
- $||$ denotes concatenation.
- $\mathbf{R}$, $\mathbf{P}$, $\vec{t}$ denote a rotation matrix, a permutation matrix and a translation vector, respectively.
- $\measuredangle ijk = \measuredangle(\vec{x}_{ij}, \vec{x}_{jk})$ and $\measuredangle ijkl = \measuredangle(\vec{x}_{ij}, \vec{x}_{jk}, \vec{x}_{kl})$ denote bond angle and dihedral (torsion) angles respectively.
- $a, b, c, d, e, n, \ldots$ are scalar quantities. a, b are often used for space dimensions (e.g. number of features) and d for the Cartesian space dimension ($d = 3$ since we focus on 3D atomic systems). c is used as the scalar cutoff value to create a graph from a point cloud. n, e point to the number of nodes and edges in a graph.
- $(\mathbf{A}, \mathbf{S}, \vec{\mathbf{v}}, \vec{\mathbf{x}}, \vec{\mathbf{c}}, \vec{\mathbf{o}})$ denotes the graph $\mathcal{G}$ with adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, scalar feature matrix $\mathbf{S} \in \mathbb{R}^{n \times a}$, atom positions $\vec{\mathbf{x}} \in \mathbb{R}^{n \times d}$, geometric feature vector $\vec{\mathbf{v}} \in \mathbb{R}^{n \times b \times d}$. For periodic graphs, we also include unit cell $\vec{\mathbf{c}} \in \mathbb{R}^{d \times d}$ and cell offsets $\vec{\mathbf{o}} \in \{-1, 0, 1\}^{e \times d}$ parameters. Nodes-labels or graph-labels are written $y \in \mathbb{R}$ or $\vec{y} \in \mathbb{R}^{n \times 3}$.
- $\mathcal{N}_i$ refers to the neighbors of node i in the graph $\mathcal{G}$. $\mathcal{N}_i = \{j \in \mathcal{V}_\mathcal{G} \setminus \{i\} \text{ s.t. } \mathbf{a}_{ij} \neq 0\}$.
- $\mathbf{h}_i, \mathbf{m}_{ij}, \vec{\mathbf{m}}_{ij}, \vec{x}_{ij}, d_{ij}, \hat{x}_{ij}$ relate to message passing. $\mathbf{h}_i$ refers to atom i's hidden representation, $\mathbf{m}_{ij} \in \mathbb{R}^a$ to the scalar message from node j to node i and $\vec{\mathbf{m}}_{ij} \in \mathbb{R}^{a \times d}$ to the geometric message. We use the superscript $\bullet^{(t)}$ to denote the t^{th}message passing layer or iteration. $\vec{x}_{ij} = \vec{x}_i - \vec{x}_j$ is the relative position or displacement between two nodes, $d_{ij} = ||\vec{x}_{ij}||$ the distance separating them, and $\hat{x}_{ij} = \vec{x}_{ij}/||\vec{x}_{ij}||$ the unit directional vector.

Clarification

We use the words *vector*, *matrix* and *tensor* from the mathematical perspective, which is distinct from the common usage of these words in the wider machine learning literature. Tensors are **not** simply multidimensional arrays of numbers. Instead, the word *tensor* signifies that the object, in addition to being representable via a multidimensional array of numbers, also has certain properties and follows transformations in multidimensional spaces. In machine learning language, our word *tensor* therefore refers to multidimensional arrays of numbers which follow certain transformation rules. Similarly, *lists* denote ordered collections, not computer science data structures.

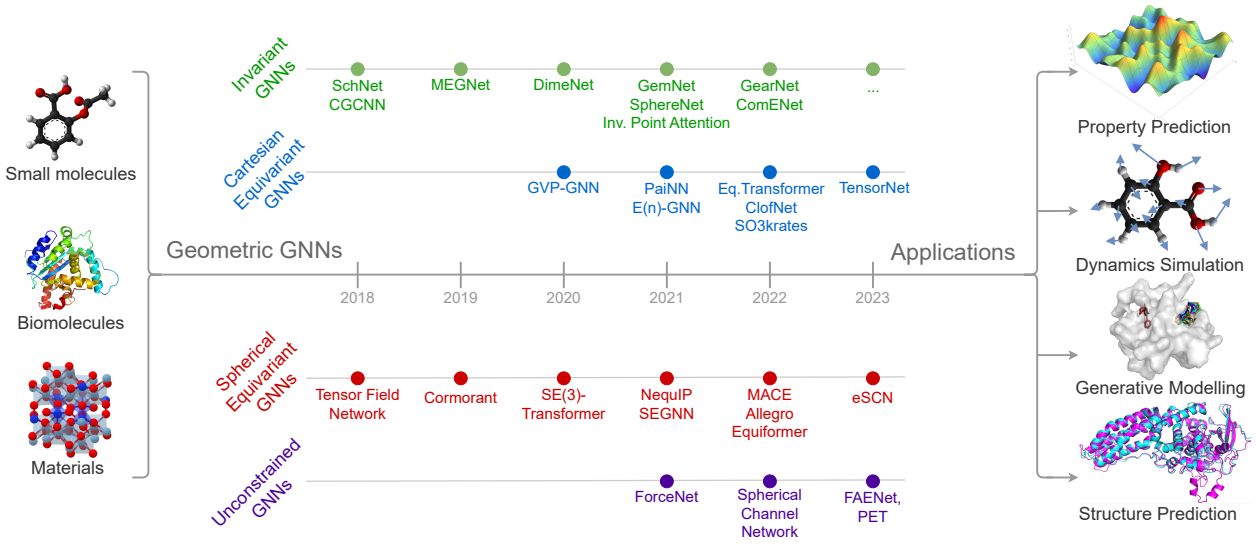


Figure 1: **Timeline of key Geometric GNNs for 3D atomic systems**, characterised by the type of intermediate representations within layers[4]. This survey presents a self-contained overview of the Geometric GNN architectures and their applications in modeling 3D atomic systems.

1 Introduction

Graphs are a powerful and general mathematical abstraction. They can represent complex relationships and interactions across fields as diverse as social networks, recommendation systems, molecular structures, and biological interactomes. Graph Neural Networks (GNNs) [Scarselli et al., 2008, Kipf and Welling, 2017, Veličković et al., 2018] are the current state-of-the-art machine learning methods for processing graph data and making predictions over nodes, edges or at the global graph level. GNNs learn latent representations of nodes through *message passing* [Gilmer et al., 2017], which enables the model to extract information about the local subgraph around each node. The Transformer architecture [Vaswani et al., 2017] for natural language processing can also be viewed as a type of GNN, where the nodes are words and the graph is assumed to be fully connected [Joshi, 2020].

Graphs are purely topological objects, in the sense that they specify only how entities (nodes) are connected, but not their spatial layout ('geometry'). For example, a social network represents friendship relations between people, but not where these people live. *Geometric graphs* are a type of graphs where nodes are additionally endowed with geometric information pertaining to the physical world, such as their spatial positions. A prototypical example we consider in this paper are molecules: the nodes represent atoms embedded in 3D Euclidean space with scalar attributes (e.g. atom type) and geometric attributes (e.g. position, velocity, or forces). Both are essential to accurately model the properties of a physical system. Because these properties are independent of the chosen reference frame, the geometric attributes are typically either invariant (independent) or equivariant (changing in the same way) under symmetry groups acting on them. Consider the illustration in Figure 2: molecular properties such as the potential energy of an isolated molecule remain the same no matter how we rotate or translate the molecule in space; it is thus *invariant* to Euclidean transformations. On the other hand, rotating or translating the molecule will lead to an equivalent transformation of the directional forces acting on each atom; atomic forces are *equivariant* to Euclidean transformations.

Thus, unlike generic graph data, geometric graphs contain additional attributes with known transformation behaviours under physical symmetries. GNNs which do not take physical symmetries into account are considered ill-suited to model geometric graphs, as treating geometric attributes in the same manner as standard node features would no longer retain their physical meaning and transformation semantics [Bronstein et al., 2021, Bogatskiy et al., 2022].

[4]This is a partial selection of representative architectures; an exhaustive list is provided on Github.

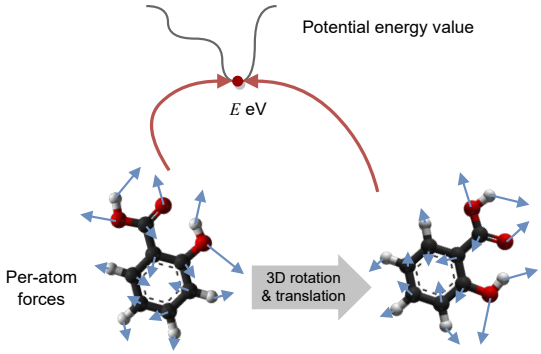


Figure 2: **Physical symmetries of geometric graph attributes.** Rotating or translating a molecule in 3D Euclidean space will lead to an equivalent transformation of the directional forces acting on each atom. On the other hand, molecular properties such as the potential energy are invariant to global rotations or translations of the system. Geometric GNNs explicitly account for physical symmetries and transformation behaviours when modeling 3D atomic systems, while standard GNNs do not.

Geometric Graph Neural Networks are an emerging class of GNNs for modeling geometric graphs constructed from 3D atomic systems. Geometric GNNs learn latent representations which enforce the appropriate physical symmetries on geometric attributes, enabling the model to better capture both geometric and relational structure in 3D systems. Geometric GNNs are the core architecture behind recent applications in protein structure prediction [Jumper et al., 2021], protein design [Dauparas et al., 2022], molecular simulation [Batzner et al., 2022] and materials discovery [Zitnick et al., 2020].

Given the progress in Geometric GNNs for 3D atomic systems, summarised in Figure 1, newcomers to the field often find themselves lost in the current 'zoo' of different models. This hitchhiker's guide aims to provide a comprehensive and pedagogical overview of the Geometric GNN modeling pipeline, describing all the architectural building blocks, highlighting key conceptual ideas, and outlining impactful future directions. Our primary goal is to serve as a guide for both newcomers and experienced researchers alike to navigate the exciting field of geometric graph learning.

The rest of the paper is organized as follows:

- Section 2 provides all necessary **background materials** for geometric graphs and Geometric GNNs, including explanations of key conceptual ideas. We aim to provide a solid foundation for understanding the subsequent content.
- Section 3 describes **all components of the Geometric GNN pipeline** such as input creation, embedding, interaction, and output blocks. We detail the variations within each component, allowing readers to grasp the intricacies and design choices involved.
- Section 4, 5, and 6 introduce a **novel taxonomy** that categorizes Geometric GNNs into four distinct families of methods: invariant, equivariant with Cartesian tensors, equivariant with spherical tensors, and unconstrained. This taxonomy offers a nuanced classification of existing architectures and establishes links between the different families.
- Section 7 explores various **datasets** and **applications** of Geometric GNNs, guiding the selection of evaluation methodology.
- Section 8 concludes the survey by identifying key areas for **future research**, shedding light on untapped opportunities in the field.

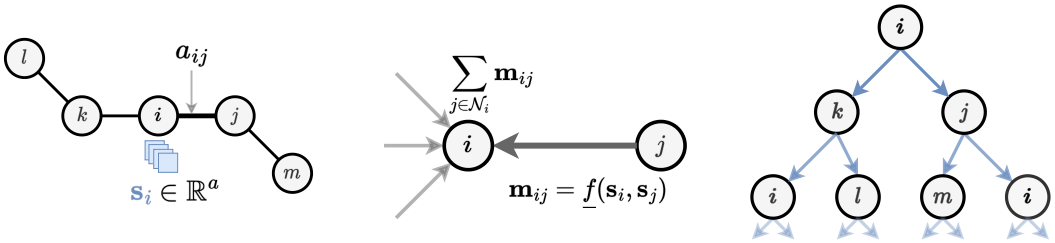
Additionally, the appendix contains definitions, refreshers, and complementary information on various topics. The accompanying Github repository offers an exhaustive list of Geometric GNNs and datasets along with their key properties, which we hope the community will keep up-to-date.

2 Preliminaries

2.1 Graphs and Graph Neural Networks

Graphs. Graphs are used to model complex and interconnected systems in the real world, ranging from molecules and knowledge graphs to social networks and recommendation systems. Formally, an attributed graph $\mathcal{G} = (\mathbf{A}, \mathbf{S})$ is a set $\mathcal{V}$ of n nodes connected by edges, as shown in Figure 3a. $\mathbf{A}$ denotes an $n \times n$ adjacency matrix where each entry $a_{ij} \in \{0, 1\}$ indicates the presence or absence of an edge connecting nodes i and j. The matrix of *scalar* features $\mathbf{S} \in \mathbb{R}^{n \times a}$ stores attributes $\mathbf{s}_i \in \mathbb{R}^a$ associated with each node i. For example, in molecular graphs (2D), each node contains information about the atom type (e.g. hydrogen, carbon), and edges represent bonds among atoms.

Typically, the nodes in a graph have no canonical or fixed ordering and can be shuffled arbitrarily, resulting in an equivalent shuffling of the rows and columns of the adjacency matrix $\mathbf{A}$. Thus, accounting for permutation symmetry is a critical consideration when designing machine learning models for graphs. One can also consider more complex definitions of a graph, including multi-relational graphs or higher-order topological variants such as hypergraphs, but we will proceed with a basic definition.



(a) An attributed graph (b) Message passing (c) GNN computation tree

Figure 3: **Graphs and Graph Neural Networks.** (a) Graphs model a set of entities as nodes, with edges denoting relationships and structure among them. (b) GNNs build latent representations of graph data through message passing operations, where each node performs learnable feature aggregation from its local neighbourhood. (c) Stacking L message passing layers enables GNNs to send and aggregate information from L-hop subgraphs around each node.

Graph Neural Networks. Graph Neural Networks (GNNs) [Goller and Kuchler, 1996, Sperduti and Starita, 1997, Gori et al., 2005, Scarselli et al., 2008] are bespoke neural networks for graph data that incorporate permutation symmetry. In recent years, modern variants of GNNs have emerged as the architecture of choice for machine learning with large-scale and real-world graph data [Kipf and Welling, 2017, Veličković et al., 2018]. GNNs build actionable node representations through message passing operations [Gilmer et al., 2017, Battaglia et al., 2018] where each node updates its feature vector by aggregating features from its local neighbourhood $\mathcal{N}_i$ in the graph. In simpler terms, neighbouring nodes (or edges) exchange information and influence each other's embedding update. Thus, node features represent the local sub-graph structure around the node and stacking several message passing layers propagates node features beyond local neighbourhoods.

Node features $\mathbf{s}_i$ are updated from iteration t to $t+1$ in three steps. (1) Compute "messages" between the node of interest i and each of its neighbour $\mathcal{N}_i$ in the graph, via a learnable MSG function; (2) Aggregate all messages coming from $\mathcal{N}_i$ via a fixed permutation-invariant aggregation operator $\oplus$ (e.g. sum, mean); (3) Update the representation of node i via a learnable function UPD, typically using both aggregated messages and its own representation as input. In practice, MSG and UPD are neural networks whose definition has been the focus of much of GNN methodology research. Formally, the message passing GNN paradigm is expressed as:

$$\begin{aligned} \mathbf{m}_{ij}^{(t)} &= \underline{\text{MSG}}\left(\mathbf{s}_i^{(t)}, \mathbf{s}_j^{(t)}\right) \\ \mathbf{s}_i^{(t+1)} &= \underline{\text{UPD}}\left(\mathbf{s}_i^{(t)}, \bigoplus_{j \in \mathcal{N}_i} \mathbf{m}_{ij}^{(t)}\right) \end{aligned} \tag{1}$$

The features derived in the final iteration L, i.e. the last message passing layer, are mapped to graph-level, node-level or edge-level predictions via a permutation-equivariant readout. For both node-level

and edge-level tasks, we can learn a shared classifier (e.g. MLP) on node/edge representations, $\mathbf{s}_i^{(L)}$ or $f(\mathbf{s}_i^{(L)}, \mathbf{s}_j^{(L)})$, where f is any function, e.g. a simple concatenation. For graph regression or classification tasks, we first need to derive a graph representation from learned node representations $\{\mathbf{s}_i^{(L)}\}$, using a permutation-invariant readout function $\bigoplus_{i \in \mathcal{V}} \mathbf{h}_i^L$. This is called *graph pooling*[5]. Then, we can learn a classification or regression head over the resulting flattened vector.

Applications of GNNs. GNNs have demonstrated their utility across a wide range of applications, ranging from recommendation systems [Hamilton et al., 2017], social networks [Monti et al., 2019, Benamira et al., 2020], transportation networks [Derrow-Pinion et al., 2021], weather forecasting [Lam et al., 2023], and, perhaps most importantly, for accelerating and augmenting scientific discovery [Zhang et al., 2023, Wang et al., 2023a]. This survey focuses on the later, introducing the family of graphs and GNNs powering recent advances in modeling atomic systems in 3D space, including molecular dynamics simulation [Batzner et al., 2022], protein folding [Jumper et al., 2021] and design [Dauparas et al., 2022], as well as materials discovery [Zitnick et al., 2020].

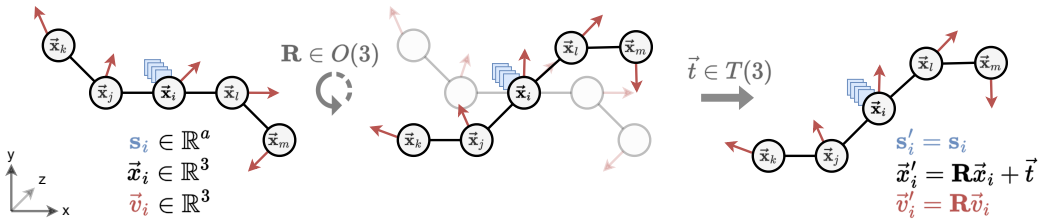


Figure 4: **Geometric graphs and Euclidean symmetries.** Geometric graphs embedded in 3D Euclidean space model systems with both geometry and relational structure, such as molecules and materials. The geometric attributes transform along with Euclidean transformations of the system: (1) The group of rotations $\mathrm{SO}(3)$, or rotations and reflections $\mathrm{O}(3)$, acts on the vector features $\vec{\mathbf{v}}$ and coordinates $\vec{\mathbf{x}}$; and (2) The translation group $\mathrm{T}(3)$ acts on the coordinates $\vec{\mathbf{x}}$. Scalar features remain invariant to Euclidean transformations. Note that this setup generalises to multiple vector features $\vec{\mathbf{v}}$ or higher-order tensor type features.

2.2 Geometric Graphs

Geometric graphs. Geometric graphs are used to model systems containing both relational structure and geometry embedded in d-dimensional Euclidean space (for most real-world applications, $d = 3$D space). As illustrated in Figure 4, a geometric graph $\mathcal{G} = (\mathbf{A}, \mathbf{S}, \vec{\mathbf{v}}, \vec{\mathbf{x}})$ is an attributed graph with scalar features $\mathbf{S}$ that is also decorated with geometric attributes: node coordinates $\vec{\mathbf{x}} \in \mathbb{R}^{n \times d}$ and (optionally) b vector features $\vec{\mathbf{v}} \in \mathbb{R}^{n \times b \times d}$, sometimes denoted $\vec{\mathbf{v}}$ for simplicity.

In biochemistry and material science, geometric graphs have to be constructed from the underlying point cloud $(\mathbf{S}, \vec{\mathbf{v}}, \vec{\mathbf{x}})$, which constitutes the typical input data of a set of atoms in 3D space. For example, molecules are often represented as a set of atoms/nodes which contain information about the atom type (a scalar feature) and its 3D spatial position (the coordinates), as well as other geometric vector quantities such as velocity or forces. Nodes are generally connected by edges using a predetermined radial cutoff distance c, such that the adjacency matrix is defined as $a_{ij} = 1$ if $\|\vec{x}_i - \vec{x}_j\|_2 \leq c$, or 0 otherwise, for all $a_{ij} \in \mathbf{A}$. In contrast, the 2D molecular graph representation from Section 2.1 does not provide any information about the spatial location or geometric attributes of a molecule. Conventional procedures for geometric graph creation beyond radial cutoffs are described in Section 3.1.

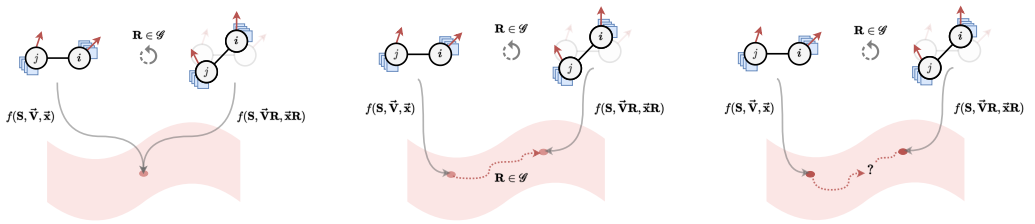
Permutation and Euclidean symmetries. As illustrated in Figure 2, the key factors distinguishing geometric graphs from standard graphs are the transformation behaviours of the geometric attributes under Euclidean symmetries. Geometric attributes are symmetric under physical transformations of the system: translations, rotations, and sometimes reflections, while scalar features remain invariant or unchanged. The following symmetry groups are relevant for geometric graphs (see Figure 4):

[5]This operation is similar to the pooling layers commonly found in CNNs: their goal is to coarsen representations and aggregate information from all the node features into a single feature for the entire graph.

- **Permutation.** The permutation group over n elements P_n acts via a permutation matrix $\mathbf{P}$ on the graph attributes as $\mathbf{P}\mathcal{G} := (\mathbf{PAP}^\top, \mathbf{PS}, \mathbf{P}\vec{\mathbf{v}}, \mathbf{P}\vec{\mathbf{x}})$. This entails shuffling the ordering of rows of the feature tensors and follows directly from the fact that a graph has no canonical ordering of its nodes.
- **Rotation (and reflection)** The group of rotation $\mathrm{SO}(d)$ or rotations and reflections $\mathrm{O}(d)$, denoted interchangeably by $\mathcal{G}$, acts via an orthogonal transformation matrix $\mathbf{R} \in \mathcal{G}$ on the vector feature $\vec{\mathbf{v}}$ and on the coordinates $\vec{\mathbf{x}}$ as $\mathbf{R}\mathcal{G} := (\mathbf{A}, \mathbf{S}, \vec{\mathbf{v}}\mathbf{R}, \vec{\mathbf{x}}\mathbf{R})$. Vector features and coordinates are physical quantities measured from an arbitrary frame of reference, so rotating the frame of reference implies an equivalent rotation of these quantities. On the other hand, scalar features are generally denoting categorical information (such as the atom type of a node) that remains unchanged or invariant under rotations. Whether equivariance to reflections is desired or not depends on the application, as explained in Appendix A.2.
- **Translation.** The group of translations $\mathrm{T}(d)$ acts via a translation vector $\vec{t} \in \mathrm{T}(d)$ on the coordinates $\vec{\mathbf{x}}$ as $\vec{x}_i + \vec{t}$ for all nodes i. The coordinates of each node are determined with respect to a single point in space (called the origin), so translating the origin leads to an equivalent translation of the coordinates. Importantly, translations do *not* impact the vector features at each node as their values are always determined relative to the coordinate of that particular node.

Opinion

The interplay between discrete (permutation) and continuous (Euclidean) symmetries makes the modeling of geometric graph data very vibrant, bringing together mathematical ideas from graph theory, topology, geometry, functional analysis, and quantum mechanics. Since node permutation equivariance is handled in traditional GNNs, we focus on Euclidean symmetries (specific of 3D atomic systems) in this work. A short refresher on group theory with a focus on geometric graphs is provided in Appendix A.2.



(a) $\mathcal{G}$-invariant function (b) $\mathcal{G}$-equivariant function (c) $\mathcal{G}$-unconstrained function

Figure 5: **Invariant, equivariant, and unconstrained functions.** The output of $\mathcal{G}$-invariant functions remains unchanged regardless of transformations applied to the input. $\mathcal{G}$-equivariant functions, on the other hand, exhibit transformations in the output that are equivalent to the transformations in the input. Finally, $\mathcal{G}$-unconstrained functions do not have predictable or known transformations of the output when the input undergoes transformations.

Functions on geometric graphs. Before describing GNNs specialised for geometric graphs, we first define three classes of functions that are used to construct Geometric GNN layers. Following the Geometric Deep Learning blueprint [Bronstein et al., 2021], we denote the action of a group $\mathcal{G}$ on a space X by $\mathfrak{g} \cdot x$, for $\mathfrak{g} \in \mathcal{G}$ and $x \in X$. If $\mathcal{G}$ acts on spaces X and Y, we say:

- A function $f : X \to Y$ is *$\mathcal{G}$-invariant* if $f(\mathfrak{g} \cdot x) = f(x)$, *i.e.* the output remains unchanged under transformations of the input, as shown in Figure 5a.
- A function $f : X \to Y$ is *$\mathcal{G}$-equivariant* if $f(\mathfrak{g} \cdot x) = \mathfrak{g} \cdot f(x)$, *i.e.* a transformation of the input must result in the output transforming correspondingly , as shown in Figure 5b.
- A function $f : X \to Y$ which is not *$\mathcal{G}$-invariant* nor *$\mathcal{G}$-equivariant* is referred to as *$\mathcal{G}$-unconstrained*. The transformation of the input results in an unknown change in the output, as shown in Figure 5c.

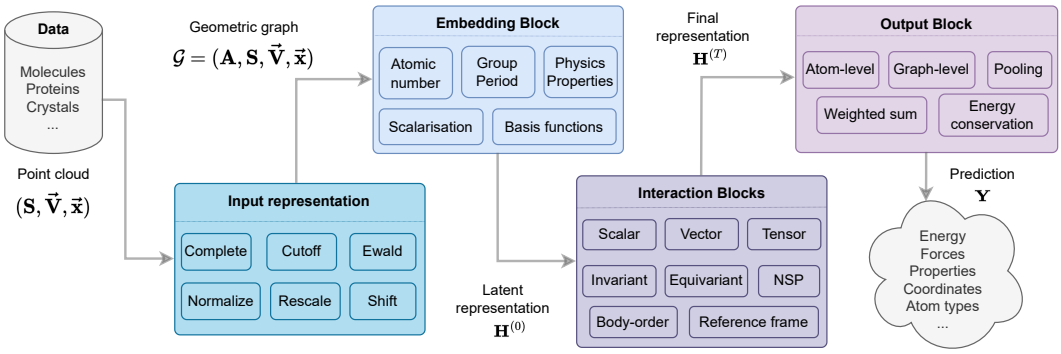


Figure 6: **Common Geometric GNN inference pipeline for 3D atomic systems.** The input representation phase (Section 3.1) involves the creation of a geometric graph $\mathcal{G}$ from the given point cloud $(\boldsymbol{S}, \vec{\mathbf{v}}, \vec{\mathbf{x}})$ data, which often depends on application-specific pre-processings (e.g. materials, small molecules, proteins). The Embedding block learns a representation for each node/edge of the graph, which is updated by Geometric GNN layers in repeated Interaction Blocks (Section 3.2). Finally, the Output block (Section 3.3) computes node-level, edge-level or graph-level predictions. The key distinctions between Geometric GNNs essentially lie in the Interaction block, where the message passing scheme varies significantly, mainly depending on how data symmetries are enforced.

3 From GNNs to Geometric GNNs

Traditional Graph Neural Networks (GNNs) are not well-suited for tasks involving geometric graphs, primarily due to their inability to predict real-world quantities while adhering to physical symmetries. For example, the energy of an atomic system remains unchanged no matter how the 3D system is rotated or translated. In contrast, Geometric GNNs are designed to capitalize on the symmetries inherent in these systems, incorporating them into the core of the model. This can be seen as an *inductive bias*[❖] that is built into the model architectures. In general, by confining the scope of learnable functions to desirable ones, these models ensure predictions align with the principles of physics, which in turn enhances generalization and data efficiency throughout the learning process.

Typically, making predictions on 3D atomic systems using Geometric GNNs involves a specific way (1) to represent the problem, (2) to learn meaningful atom embeddings, and (3) to predict desired physical quantities from the learned representations. In subsequent sections, we describe each part of the pipeline, represented in Figure 6.

3.1 Input preparation

The minimal 'raw' data for an atomic system is typically a set of atom types $(\mathbf{S})$ and positions in 3D space $(\vec{\mathbf{x}})$, i.e. a 3D point cloud $(\mathbf{S}, \vec{\mathbf{x}})$. The geometric graph is constructed from the underlying point cloud to model pairwise interactions among the atoms, and other physical descriptors and attributes are attached to the nodes and edges to prepare the input representation into Geometric GNNs.

Note that this survey uses the terms 'atoms' in an atomic system and 'nodes' in the corresponding geometric graph interchangeably. Geometric GNNs typically operate on a subset of all input atoms. For instance, Hydrogen atoms are generally ignored for computational efficiency when modeling small molecules as well as biomolecules. For larger systems such as proteins and nucleic acids, models may operate at different levels of granularity, such as only using the alpha Carbon atoms to represent an entire residue.

Graph representation. Given the pivotal role of atomic interactions in determining a system's properties, constructing a geometric graph, i.e. an adjacency matrix $\mathbf{A}$, from the 3D point cloud becomes a natural direction to pursue. Indeed, this approach then enables the design of a Geometric GNN that effectively captures both the topological and feature-related information of the system. Up to now, various strategies to construct this geometric graph have been explored in the literature (see Figure 7):

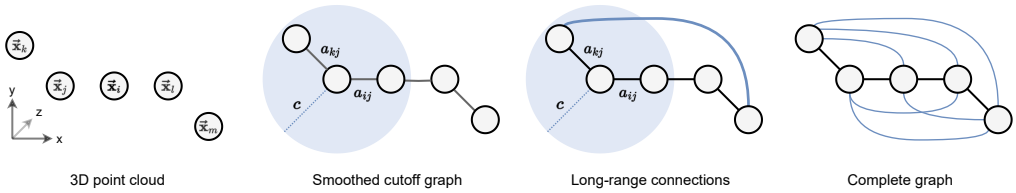


Figure 7: **From point clouds to geometric graphs**. The original 3D point cloud is transformed into a representative geometric graph. Examples include cutoff graphs, long-range connections, and complete graphs.

- **Via a complete graph** where every atom is connected to every other atom, capturing all potential atomic interactions including pairwise dependencies and potential long-range effects. This representation is motivated by the physical principle that atoms in a system can interact with each other to some degree. Using a complete graph (with pairwise distances as edge weights) allows for a comprehensive analysis of an atomic system and has been the preferred solution on small molecules (MD17 [Duvenaud et al., 2015], QM9 [Ramakrishnan et al., 2014]). However, it is computationally demanding and leads to unnecessary complexity, especially in large systems like proteins, which is why the approaches below are often preferred. It should be noted however that transformer-based architectures exist which are capable of handling large graphs [Brehmer et al., 2023].
- **Via a cutoff graph**, where there exists an edge between any two atoms if their relative distance $d_{ij} = ||\vec{x}_i - \vec{x}_j||$ is below a certain threshold "cutoff" distance c, expressed in Angstrom (e.g., $c = 6$Å) [Schütt et al., 2017, Gasteiger et al., 2021, Thomas et al., 2018].

$$\boldsymbol{A}_{ij} = \begin{cases} 1 \text{ if } d_{ij} \leq c \\ 0 \text{ otherwise.} \end{cases} \tag{2}$$

  By focusing on local interactions, the cutoff graph facilitates a deeper understanding of the system's behaviour while reducing the computational overhead. This approach aligns with physical and chemical constraints, explicitly enforcing locality as an inductive bias since atoms that are too far apart generally have negligible interactions. Stacking several GNN layers may enable to capture long-range dependencies. Cutoff graphs is the most widespread approach at present. It is sometimes combined with k nearest neighbours techniques to ensure that nodes have the same degree, constructing a regular graph.
- **Via a smooth cutoff graph** to ensure a smooth energy landscape and well-behaved force predictions, particularly for molecular dynamics (MD) simulations. Using a traditional cutoff graph for MD would imply that a small change in the position of a single atom could result in a large change in energy prediction, i.e. a very steep gradient in the energy landscape. This happens because from one frame to another, an atom can move from outside the cutoff to inside the cutoff, most likely breaking simulations since forces would be unbounded. To alleviate these jumps in the regression landscape, Unke and Meuwly [2019] proposed using a smoothed cutoff graph using the cosine function, where distances $d_{ij} = ||\vec{x}_i - \vec{x}_j||$ are smoothed out in the following way:

$$\boldsymbol{A}_{ij} = \begin{cases} \frac{1}{2}\left(\cos\left(\frac{\pi d_{ij}}{c}\right) + 1\right) \text{ if } d_{ij} \leq c \\ 0 \text{ otherwise.} \end{cases} \tag{3}$$

  Note that the adjacency matrix is no longer discrete and each edge's value is utilised inside the message passing scheme to weight atoms' contributions.
- **Long-range connections**. While cutoff graphs leverage locality as a useful inductive bias, this impedes learning long-range interactions such as electrostatics and van der Waals forces. To address this drawback, in addition to short-range interactions modelled by cutoff distance, [Kosmala et al., 2023] propose to incorporate long-range interactions using a non-local Fourier space scheme limiting interactions via a cutoff on frequency. It is particularly useful for systems with charged particles where the electrostatic interactions need to be taken into account, as well as for periodic structures containing diverse atoms.

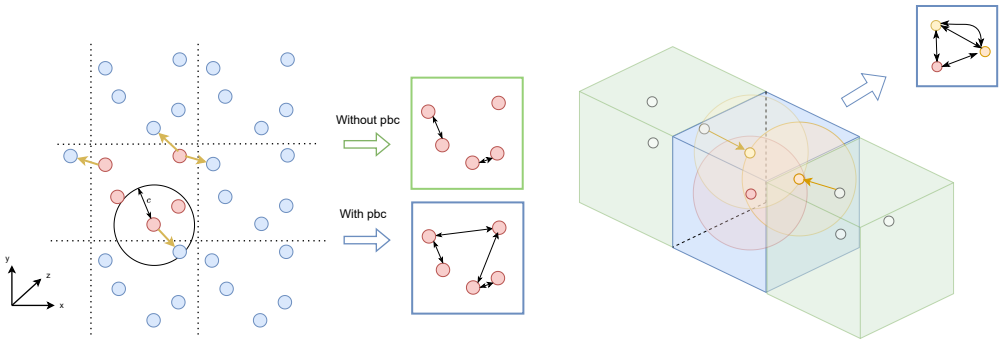


Figure 8: **Cutoff graph with periodic boundary conditions**. Without considering the repetition of the atomic pattern, one would neglect meaningful atomic interactions. With pbc, on the other hand, we notice that the bottom-most atom and top-right atom (in a single cell, red for instance) are, in fact, close enough to be connected *through* the boundary.

Alternate ways to incorporatie long-range interactions include sampling random connections weighted by a heuristically determined probability, e.g. the inverse of the distance [Ingraham et al., 2022] which has proven effective when used within generative models.

Opinion

While modeling atomic systems using a complete graph seems to be the most faithful representation of reality, the (smooth) local cutoff is a powerful inductive bias for modeling intermolecular interactions, which are mostly localised. In this case, long-range interactions may be captured by stacking several message passing layers. Alternatively, manually adding long-range dependencies to the cutoff graph reduces complexity (i.e. fewer layers needed) and may mitigate potential *over-squashing* issues.

Periodic boundary conditions in crystals. While molecules simply consist of a set of 3D points in space, easily representable using a finite graph, *crystals* are modelled to be infinite periodic structures whose repeating pattern is called a *unit cell* . To account for these infinite repetitions of the same substructures, we represent a single unit cell but take into consideration adjacent cells in all directions using *periodic boundary conditions* (PBC), depicted in Figure 8. In short, distances under PBC are expressed as $d_{ij} = ||\vec{x}_{ij} + \vec{o}_{ij} \cdot \vec{\mathbf{c}}||$, where $\vec{\mathbf{c}}$ is the 3D unit cell and $\vec{o}_{ij}$ is the cell offset parameter specifying pairwise atom proximity in neighbouring cells (i.e. a 3D vector in $\{0, 1, -1\}^3$ associated with edge (i, j)). Whether periodic boundary conditions are sufficient to handle crystal systems, or if alternative approaches are worth exploring, is an active area of research [Kaba and Ravanbakhsh, 2022, Yan et al., 2022].

Data pre-processing and augmentation. In addition to the graph creation, performing data pre-processing steps is sometimes useful to ensure accurate and efficient training. This includes standard techniques such as feature normalization, centering the coordinates to the origin, or target rescaling; as well as more domain-specific techniques like atom-type rescaling, defined in Appendix C.3. Finally, to improve the robustness of models, practitioners often consider multiple representations of the same sample during training, e.g. various Euclidean transformations [Hu et al., 2021] or corrupted versions where noise have been added to atom positions [Godwin et al., 2021]. For biomolecules, adding random Gaussian noise may improve model robustness to small changes in atomic coordinates due to crystallography artefacts [Dauparas et al., 2022].

3.2 Learning representations of atoms

Once the geometric graph is defined, the main objective is to learn meaningful atom representations. This is handled by the Embedding and Interaction blocks, which respectively initialise learnable latent representations of each atom and iteratively update them using Geometric GNN layers.

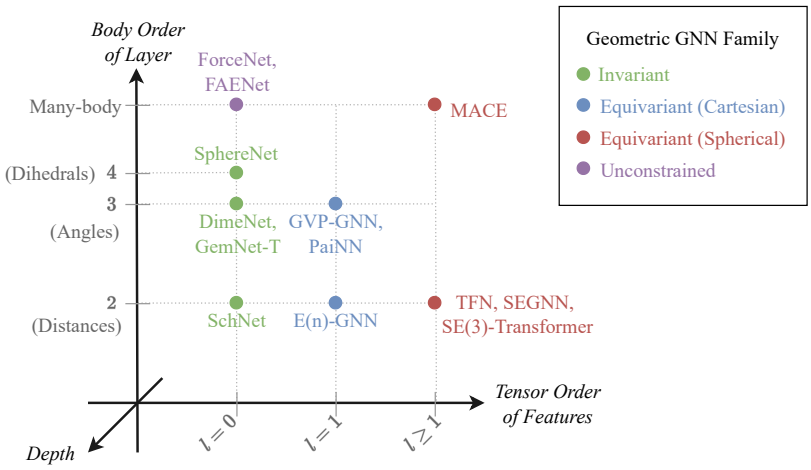


Figure 9: **Geometric GNN families, categorised by measures of expressive power** in terms of representing geometric (sub-)graphs: (1) *Body order* refers to the number of atoms involved in constructing local neighbourhood features; (2) *Tensor order* determine the granularity of capturing directional information as well as invariance or equivariance properties, and (3) *Depth* controls the extent to which geometric information is propagated beyond local neighbourhoods. Provably enforcing symmetries acts as a constraint on the expressivity of Geometric GNNs. Unconstrained Geometric GNNs 'break' the associated expressivity bottleneck of the axes by not being bound to enforce strict equivariance of intermediate features.

3.2.1 Embedding block: initialising latent representations

The Embedding block incorporates *independent* information about each atom in the geometric graph (without considering who it is connected to). This is typically accomplished by learning distinct embeddings $\mathbf{s}_z$ for each chemical element since the atomic number is always provided as part of the scalar feature matrix $\mathbf{S}$ of the geometric graph.

Especially for quantum chemistry tasks, Hu et al. [2021] and Duval et al. [2022] demonstrated the advantage of learning additional embeddings for the group/period of each atom ($\mathbf{s}_p$, $\mathbf{s}_g$) as well as incorporating known physical properties ($\mathbf{s}_f$, e.g., atomic mass, electronegativity). Such information can be easily derived from the atomic number of each atom. For biomolecules, practitioners typically consider only the alpha Carbon atoms to represent an entire residue for efficiency, and pre-compute geometric quantities such as displacement vectors and torsion angles to initialise node representations [Jamasb et al., 2023].

The embedding block creates learnable atom representations, where the different embeddings are concatenated together to form the initial scalar representation $\mathbf{s}^{(0)}$ at layer 0 for each atom:

$$\mathbf{s}^{(0)} = \mathbf{s}_z \| \mathbf{s}_p \| \mathbf{s}_g \| \mathbf{s}_f. \tag{4}$$

3.2.2 Interaction blocks: learning geometric and relational features

Incorporating geometric and relational information about how the atoms in a system interact with one another is the critical function of Geometric GNNs. As previously discussed, directly treating atomic coordinates as scalar quantities in standard GNNs would violate the equivariance of the model's features under Euclidean transformations of the input system. Therefore, the primary focus of Geometric GNNs has been to devise efficient and expressive approaches for encoding and processing geometric graphs while respecting physical symmetries. Most architectures explicitly incorporate these constraints into the model architecture, thereby constraining the GNN's function space to symmetry-preserving functions [Schütt et al., 2017, Satorras et al., 2021a, Thomas et al., 2018]. Recent approaches have also explored alternative methods towards symmetries, such as data augmentation [Hu et al., 2021] or frame-based techniques [Duval et al., 2023].

Interaction blocks of Geometric GNN are iteratively applied to the initial atom representations to build latent representations that capture geometric information from the local neighbourhood of each

atom via message passing. At a high level, the functioning of Interaction blocks can be broadly expressed as follows. They update scalar (and vector) features from layer t to $t+1$ via learnable message and update functions, $\underline{\text{MSG}}$ and $\underline{\text{UPD}}$, respectively, as well as a fixed permutation-invariant operator $\bigoplus$ (e.g. mean, sum):

$$\boldsymbol{s}_i^{(t+1)}, \vec{\mathbf{v}}_i^{(t+1)} := \underline{\text{UPD}}\left(\left(\boldsymbol{s}_i^{(t)}, \vec{\mathbf{v}}_i^{(t)} \right), \bigoplus_{j \in \mathcal{N}_i} \underline{\text{MSG}} \left(\boldsymbol{s}_i^{(t)}, \boldsymbol{s}_j^{(t)}, \vec{\mathbf{v}}_i^{(t)}, \vec{\mathbf{v}}_j^{(t)}, \vec{x}_{ij} \right) \right), \tag{5}$$

where $\vec{x}_{ij} = \vec{x}_i - \vec{x}_j$ denote relative position vectors. Note that *$\mathcal{G}$-invariant GNN layers* do not update vector features and only aggregate scalar quantities from local neighbourhoods. For instance, invariant GNNs may consider updates of the following form:

$$\boldsymbol{s}_i^{(t+1)} := \underline{\text{UPD}} \left(\boldsymbol{s}_i^{(t)}, \bigoplus_{j \in \mathcal{N}_i} \left(\underline{\text{MSG}}(\boldsymbol{s}_i^{(t)}, \boldsymbol{s}_j^{(t)}, \vec{x}_{ij}) \right) \right). \tag{6}$$

One of the key contributions of this survey is to categorise Geometric GNN architectures into four distinct families: (1) Invariant, (2) Equivariant in Cartesian basis, (3) Equivariant in spherical basis, and (4) Unconstrained models[6]. We will describe each architecture family in subsequent sections, and provide a concise opinionated history of methods in Appendix B.

3.3 Output block: making predictions

Once we have obtained meaningful latent representations of each atom using the Embedding and Interaction blocks, the Output block is used to make task-specific predictions for which the model is trained. In general, most predictive tasks involve outputs at the node level (e.g. forces acting on each atom) or global graph level (e.g. potential energy of the entire system). Moreover, node-level outputs can be either invariant or equivariant to physical transformations of the system, while global outputs are generally invariant. Therefore, Geometric GNN pipelines have different task-specific heads for each prediction level (graph, node, edge) and prediction type (invariant, equivariant).

Node-level predictions are obtained from the final node representations from the interaction blocks by passing the individual representations through multi-layer perceptrons (MLP) for invariant tasks, and equivariant variations of MLPs [Jing et al., 2020, Schütt et al., 2021a] for equivariant tasks. When graph-level predictions are required, the node representations from the interaction blocks are mapped to graph-level predictions via a permutation-invariant readout function $f : \mathbb{R}^{n \times a} \to \mathbb{R}^b$. This is usually a simple sum or averaging operation ($b = a$), but weighted averaging or hierarchical pooling approaches may also be effective [Duval et al., 2022]. Particularly in molecular dynamics tasks, it is common to concatenate all intermediate node representations when making predictions instead of considering only the final features [Schütt et al., 2017, Batatia et al., 2022a].

Energy conservation . It is worth highlighting how atomic forces[7] are predicted by Geometric GNNs for molecular dynamics. There are two main approaches in the literature: (1) computing atomic forces as the negative gradient of the energy with respect to atom positions $\vec{F}_i = -\frac{\partial E}{\partial \vec{x}_i}$ (i.e. following the formal definition from physics), or (2) using a separate neural network to predict the forces directly from atom representations. Computing forces as the gradient of the energy guarantees energy-conserving forces, which is a highly desirable feature when using Geometric GNNs to run molecular simulations since it ensures the stability of simulations and retains the ability to reach local minima [Chmiela et al., 2017]. However, Kolluru et al. [2022] demonstrated the significant computational burden associated with this approach. Compared to training a separate force prediction head, energy conservation increases memory usage by 2-4$\times$ and leads to a drop in modeling performance on several datasets (particularly the large-scale OC20 and OC22 datasets [Chanussot et al., 2021, Tran et al., 2023]). Whether models should enforce energy conservation is an open research question which probably depends on the task at hand [Fu et al., 2023]. We will expand on this discussion in Section 8 where we describe promising future directions.

[6] Unconstrained GNNs do not strictly enforce symmetries via their architecture, but generally attempt to learn approximate symmetries via data augmentation or canonicalization.

[7] Atom-wise 3D vectors representing the forces currently applied on each atom by the rest of the system.

4 Invariant Geometric GNNs

Key idea

Invariant GNNs leverage 3D geometric information by pre-computing informative scalar quantities between atoms, such as pairwise distances, triplet-wise angles, and quadruplet-wise torsion angles, and using learned latent representations of these quantities during message passing. Since these input scalar quantities are invariant to Euclidean transformations, the intermediate representations and predictions of these models are guaranteed to be invariant.

Overview. Invariant GNNs aim to learn atomic representations and make predictions that are guaranteed to be invariant to 3D Euclidean transformations of the system, including the group of rotation $\mathrm{SO}(3)$ or rotations and reflections $\mathrm{O}(3)$—denoted interchangeably by $\mathcal{G}$—and the translation group $\mathrm{T}(3)$. Enforcing translation invariance is straightforwardly done by: (1) centring input point clouds to the origin by subtracting the centre of mass from each atom coordinate; and (2) operating on relative displacements instead of raw coordinates.

One way to enforce $\mathcal{G}$-invariance is to avoid directly processing quantities that depend on the frame of reference. The reason for that is intuitive: if we let GNNs freely process any geometric quantity like a standard scalar quantity, every time our GNN is given as input a slightly transformed version of the same system, it may make a distinct prediction whereas the system's underlying properties are unchanged. Hence, $\mathcal{G}$-invariant GNN layers extract and aggregate invariant scalar quantities from atomic coordinates. These quantities are computed by *scalarising*[8] geometric quantities that are guaranteed to not change with Euclidean transformations of atomic systems. For instance, computing relative distances between atoms is a scalarisation of the geometric information $\vec{\mathbf{x}}$, and is invariant to translations, rotations and reflections.

These scalar features $\mathbf{S}$ are updated from iteration t to $t+1$ via learnable message ($\underline{\textsc{Msg}}$) and update ($\underline{\textsc{Upd}}$) functions as part of a standard message passing framework similar to Equation (1):

$$\boldsymbol{s}_i^{(t+1)} := \underline{\textsc{Upd}}\left(\boldsymbol{s}_i^{(t)}\,,\ \bigoplus_{j\in\mathcal{N}_i}\left(\underline{\textsc{Msg}}(\boldsymbol{s}_i^{(t)}, \boldsymbol{s}_j^{(t)},\ \vec{x}_{ij})\right)\right). \tag{7}$$

Depending on the task and implementation, the message from node i to node j may contain arbitrarily long dependencies through the graph. For instance, it could contain an aggregation over neighbours of j. Thus, in the more general form of Equation (7) provided below, $\underline{\textsc{Msg}}$ takes $\vec{\mathbf{x}}$[9] and $\mathbf{S}^{(t)}$ as arguments.

$$\boldsymbol{s}_i^{(t+1)} := \underline{\textsc{Upd}}\Big(\boldsymbol{s}_i^{(t)}\,,\ \bigoplus_{j\in\mathcal{N}_i}\underline{\textsc{Msg}}\big(\mathbf{S}^{(t)},\ \vec{\mathbf{x}},\ i,\ j\big)\Big). \tag{8}$$

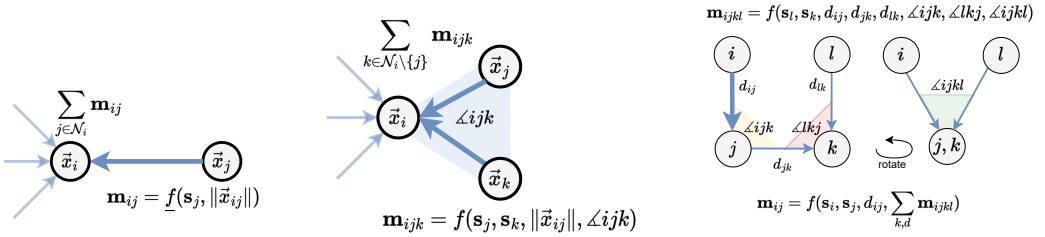
Distance-based invariant GNNs. SchNet [Schütt et al., 2018] was one of the first invariant GNN models and uses relative distances $\|\vec{x}_{ij}\|$ between pairs of nodes, encoded by a learnable Radial Basis Functions ($\underline{\psi}$, i.e. an RBF with a two-layer MLP), to encode local geometric information, as shown in Figure 10a. Each SchNet layer performs a continuous convolution to combine the encoded distance information (i.e. the filter) with neighbouring atom representations (via element-wise multiplication $\odot$). This creates a message which is propagated along graph edges, enabling SchNet to effectively capture and integrate local structural features in molecular systems:

$$\begin{aligned}\mathbf{s}_i^{(t+1)} &:= \mathbf{s}_i^{(t)} + \sum_{j\in\mathcal{N}_i}\underline{\mathbf{f}}_1\left(\mathbf{s}_j^{(t)}, \|\vec{x}_{ij}\|\right)\\ &:= \mathbf{s}_i^{(t)} + \sum_{j\in\mathcal{N}_i}\mathbf{s}_j^{(t)}\odot\underline{\psi}(\|\vec{x}_{ij}\|)\end{aligned} \tag{9}$$

As a result, SchNet efficiently utilizes both atom-identity-related and geometric information during message passing, making it an efficient and simple-to-understand tool for processing geometric

[8] A term used colloquially in the community for extracting scalar (i.e. invariant) components from a combination of geometric vector or tensor quantities.

[9] Remember: $\mathbf{A}$ is computed from atomic positions so $\vec{\mathbf{x}}$ contains the information about adjacency.



(a) SchNet (b) DimeNet (c) GemNet

Figure 10: **Invariant GNN message passing.** $\mathcal{G}$-invariant layers extract and propagate local scalar geometric quantities such as distances (SchNet), bond angles (DimeNet), and torsion angles (GemNet) which are guaranteed to be invariant to Euclidean transformations.

graphs. Other architectures which pioneered the use of GNNs for 3D atomic systems also relied on distance-based invariant message passing, including CGCNN [Xie and Grossman, 2018] and PhysNet [Unke and Meuwly, 2019].

However, distance-based invariant GNNs are not sufficiently expressive at modeling higher-order geometric invariants. As SchNet relies on atom distances within a cutoff value, it cannot differentiate between atomic systems that have the same set of atoms and pairwise distances among them but differ in higher-order geometric quantities such as bond angles (refer to Appendix A.8). This well-known limitation of low *body order*❖ invariant descriptors of atomic representations is well known in the broader molecular modeling community [Bartók et al., 2013], and continues to inform improvements to Geometric GNN architecture design [Pozdnyakov and Ceriotti, 2022, Joshi et al., 2023a].

Going beyond distances with many-body scalars. To address the lack of geometric expressivity of distance-based message passing, recent invariant GNNs [Shuaibi et al., 2021, Wang and Zhang, 2022, Wang et al., 2022a, 2023b] focus on incorporating higher body order scalar quantities beyond pairwise interactions (e.g. from triplets and quadruplets of atoms). Two pedagogical examples of architectures following this trend are explained subsequently.

DimeNet [Gasteiger et al., 2020], as illustrated in Figure 10b, employs continuous convolutions whose filter combines both pairwise distances $d_{ij} = \|\vec{x}_{ij}\|$ and bond angles $\measuredangle ijk = \measuredangle(\vec{x}_{ij}, \vec{x}_{ik})$ between triplets of atoms (i.e. 3-body order). DimeNet operates on local reference frames $(i, j \in \mathcal{N}_i, k \in \mathcal{N}_j\setminus\{i\})$ defined at each atom, which enables computing spatial angles between pairs of neighbours:

$$\boldsymbol{s}_i^{(t+1)} := \sum_{j\in\mathcal{N}_i} \underline{f_1}\Big(\boldsymbol{s}_i^{(t)},\ \boldsymbol{s}_j^{(t)}, d_{ij},\ \sum_{k\in\mathcal{N}_j\setminus\{i\}} \underline{f_2}\left(\boldsymbol{s}_j^{(t)},\ \boldsymbol{s}_k^{(t)},\ d_{ij},\ \measuredangle ijk\right)\Big) \tag{10}$$

The updated scalar features are $\mathcal{G}$-invariant since geometric information is only exploited via relative distances and angles[10], both of which remain unchanged under the action of $\mathcal{G}$. Nonetheless, there exist well-known edge cases of pairs of point clouds which are the same up to distances and angles [Pozdnyakov et al., 2020].

Thus, GemNet [Gasteiger et al., 2021] turned to 4-body order scalarization, additionally extracting torsion angles between groups of four atoms using local reference frames, denoted $\measuredangle ijkl = \measuredangle(\vec{x}_{ij}, \vec{x}_{kd}) \perp \hat{x}_{jk}$ (see Figure 10c). However, moving to higher body order scalarization of geometric information becomes computationally expensive. For each atom i, GemNet message passing must consider all direct neighbours $j \in \mathcal{N}_i$, 2-hop neighbours $k \in \mathcal{N}_j$ and 3-hop neighbours $l \in \mathcal{N}_k$:

$$\boldsymbol{s}_i^{(t+1)} := \sum_{j\in\mathcal{N}_i} \underline{f_1}\Bigg(\boldsymbol{s}_i^{(t)}, \boldsymbol{s}_j^{(t)}, d_{ij}, \sum_{\substack{k\in\mathcal{N}_j\setminus\{i\},\\ l\in\mathcal{N}_k\setminus\{i,j\}}} \underline{f_2}(\boldsymbol{s}_k, \boldsymbol{s}_l, d_{kl}, d_{ij}, d_{jk}, \measuredangle ijk, \measuredangle jkl, \measuredangle ijkl)\Bigg). \tag{11}$$

[10]Note that the distances and angular information are computed only once as a data pre-processing step. They are then leveraged inside each message passing layer.

To improve scalability, practical GemNet variants such as GemNet-OC [Gasteiger et al., 2022] are often restricted to 3-body scalars. SphereNet [Liu et al., 2022]and ComENet [Wang et al., 2022a] introduced efficient method for extracting 4-body angles within local neighbourhoods, avoiding the need to loop through all 3-hop neighbours. However, as noted in the SphereNet paper, this localised approach has known failure cases where local scalars up to 4-body angles are the same across two geometric graphs, but the systems differ in terms of non-local, higher-order scalars such as dihedral angles. Thus, the precise body order of scalars at which all geometric graphs can be uniquely identified remains an open question [Joshi et al., 2023a].

Opinion

The GemNet paper claims to show that 'GNNs with directed edge embeddings and two-hop message passing are indeed universal approximators for predictions that are invariant to translation, and equivariant to permutation and rotation'. This has led to a misconception in the community that invariant architectures operating on distances, angles, and torsions angles are universal or complete.

However, we believe that the GemNet architecture is *not* universal in the general case for sparse graphs. Theorems 2 and 3 in the GemNet paper prove the universality of spherical Networks for fully connected graphs based on their equivalence to Tensor Field Networks (which were proven to be universal by Dym and Maron [2020] when operating on full connected graphs *and* using infinite tensor rank for equivariant features). However, 4-body message passing in the final GemNet-Q architecture sacrifices universality guarantees by departing from spherical Networks and operating on a discretization of representations in the directions of each atom's neighbours instead of using spherical harmonics. The paper states that 'depending on the discretization scheme the resulting mesh might not provide a universal approximation guarantee'.

Additionally, GemNet-T, the more practical version of GemNet which is also the basis of GemNet-OC, performs 3-body message passing similar to DimeNet on radial cutoff graphs, which is not universal due to known counterexamples [Pozdnyakov et al., 2020]. This was also highlighted by the first author of GemNet in informal communication.

Summary. Overall, invariant GNNs' reliance on a precomputed procedure to scalarise geometric information is both a blessing and a curse. Models such as SchNet or DimeNet can be very efficient and simple baselines for modeling 3D atomic systems. However, improving the expressivity of invariant GNNs results in increasingly complex architectures (see Figure 25) as precomputing and incorporating higher body order invariants necessitates expensive accounting of higher-order tuples [Li et al., 2023]. Additionally, it is worth mentioning that the outputs of invariant GNNs are restricted to invariant predictions, thus lacking generalisation. One can also obtain equivariant predictions with invariant GNNs, but only through specific operations on their invariant predictions, such as taking the gradient of the energy with respect to atom positions to obtain atomic forces.

5 Equivariant Geometric GNNs

Overview. As explained in the previous section, invariant GNNs pre-compute a set of *local invariants* in each neighbourhood before performing message passing. While this can be efficient, it is also restrictive. A key limitation of invariant GNNs is that the set of local invariants is fixed and has to be determined prior to message passing.

How could we overcome this limitation and instead allow the network to *learn* its set of invariants which could (1) be more suited to the task at hand, and (2) whose complexity could be controlled by performing more or fewer message passing steps?

In this section, we investigate a family of GNNs, which we will refer to as *equivariant GNNs* (EGNNs), that fulfil these two goals and additionally allow the prediction of equivariant quantities. These models do not pre-compute local invariants but instead perform message passing in a way such that the hidden features at each layer are equivariant to symmetry transformations of the input. In other words, if we were to, say, rotate the input graph, the hidden features at each layer would also be rotated correspondingly. This is in contrast to invariant GNNs, where the hidden features at each layer would remain the same. We will see that by using equivariant GNN layers, the network can build up its own set of invariants *on the fly*, as it performs message passing, and more message-passing layers can lead to more complex invariants which contain information from multiple, possibly non-local neighbourhoods.

The crux to make equivariant GNNs work is to perform *diligent accounting* of how each hidden feature in each layer has to transform in order to remain equivariant. This accounting amounts to the following basic intuitions[11]:

1. **Types:** Each feature is associated with a *type* which tells us how that feature changes under symmetry transformations[12].
2. **Addition of types:** When adding features through component-wise addition of lists of numbers that represent them, we need to make sure that we only add features of the same type. This ensures that the sum of the features has the same type.
3. **Multiplication of types**: When we multiply features with each other, we can multiply features of *different types*, but we need to keep track of how the product transforms. As we shall see, the complication is that the naive component-wise product of two lists of numbers which represent two features, each of which transforms equivariantly individually, will generally transform differently than each of its factors[13]. To take this into account, we cannot just perform component-wise multiplication of equivariant feature types. Instead, we use a special product called the *tensor product* , which, for our purposes, can loosely be seen as a generalization of multiplication that takes into account how the product transforms.
4. **Non-linear operations on types:** Non-linear operations need to be handled with care. Why is this the case? For the relevant intuition, you may think of non-linear operations in terms of their Taylor expansion, which incorporates products of features. If higher-order products of features yield different types (as described in the previous point (3) on the multiplication rule), we cannot add them together by the addition rule in point (2). Importantly, the nonlinearities in machine learning are usually understood to be applied component-wise. But this often breaks equivariance for all but scalar representations. For this reason, nonlinearities are most often applied to scalar-type features only.

[11]These intuitions are formalised in the well-studied mathematical sub-field of representation theory. There are many great treatises on representation theory available (e.g. Zee [2016], chapter 2). Instead of repeating them and giving you the theory "top-down", we follow a different route and try to help you build an intuition as to why this is relevant in machine learning and how you would arrive at some basic tenants of representation theory from the "bottom-up". For those interested in exploring formal mathematics, we will provide the official mathematical terms for key concepts we discuss in footnotes so that you may look them up in standard references in your own time. For all mathematicians, we note that we will assume all representations we are working with are orthogonal.

[12]Mathematically, the *type* of a feature tells us which linear representation of the symmetry group the feature vector transforms in. The possible types depend on the symmetry group, e.g. rotations in 3D, and classifying all possible types of a given group is one of the main tasks in representation theory. Luckily for us, all types for rotations in 3D are well known, but this is not the case for some other groups.

[13]For instance, $\mathbf{R}(\vec{\mathbf{u}} \odot \vec{\mathbf{v}}) \neq \mathbf{R}\vec{\mathbf{u}} \odot \mathbf{R}\vec{\mathbf{v}}$ in general.

How then should we define a *type*? And which multiplication rules between these types do we then need to follow to ensure the multiplication rule (3)?

There are many ways of choosing types (which in turn determine the multiplication rules). Below, we discuss two of them: (1) using Cartesian coordinates and (2) using spherical coordinates[14]. We will start with the Cartesian formulation, which we expect to be more intuitive for most readers and then transition to the spherical formulation which has a natural relationship to rotations and reflections in 3D.

5.1 Equivariant GNNs with Cartesian tensors

Key idea

Cartesian EGNNs model atomic interactions in Cartesian coordinates and restrict the set of possible operations on geometric features to preserve equivariance. They often update (and combine) both scalar and vector messages in parallel.

Scalar-Vector GNNs. Let us start simple and consider two familiar types of features only: *scalars* and *vectors*. An example of a scalar is the distance between two nodes or the angle between two vectors: it is an object that does not change under transformations in the symmetry group $\mathcal{G}$, in our case under rotations, reflections and translations. A vector (e.g. atomic forces), in contrast, transforms under these operations in the familiar way: $\vec{v} \mapsto \mathbf{R}\vec{v} + \vec{t}$ where $\mathbf{R}$ is a rotation or reflection matrix and $\vec{t}$ is a translation vector. In the didactic discussion below, we will ignore translations and reflections and focus on the group of rotations only. Reflections and translations do not require novel insights and can be dealt with fairly easily once we understand how to deal with rotations[15].

What multiplication rules make sense, given we have these two types of features? Let us think about multiplication operations between scalars and vectors that we are already familiar with. We know that the product of two scalar-types will give us a scalar-type again and that the product of a scalar-type and a vector-type will simply give us a scaled vector-type[16]. Finally, we can 'multiply' two vector-types via the dot-product to get a scalar-type[17]. Notice that taking the norm $||\vec{v}||$ of a vector is just a special case of the dot-product: it amounts to taking the dot-product of a vector with itself (followed by a square root).

So, given scalar and vector feature types, as well as the three multiplication rules above, which GNNs can we build? The messages and update rules would have to be of the form:

$$\mathbf{m}_{ij}^{(t)}, \vec{\mathbf{m}}_{ij}^{(t)} := \underline{\text{MSG}}\big(\boldsymbol{s}_i^{(t)}, \mathbf{s}_j^{(t)}, \vec{\mathbf{v}}_i^{(t)}, \vec{\mathbf{v}}_j^{(t)}, \vec{x}_{ij}\big) \qquad \text{(Message)} \qquad (12)$$

$$\boldsymbol{s}_i^{(t+1)}, \vec{\mathbf{v}}_i^{(t+1)} := \underline{\text{UPD}}\left((\boldsymbol{s}_i^{(t)}, \vec{\mathbf{v}}_i^{(t)}), \bigoplus_{j \in \mathcal{N}_i} (\boldsymbol{m}_{ij}^{(t)}, \vec{\mathbf{m}}_{ij}^{(t)})\right) \qquad \text{(Update)} \qquad (13)$$

with a scalar message $\boldsymbol{m}_{ij}^{(t)}$ and a vector message $\vec{\mathbf{m}}_{ij}^{(t)}$. The most general messages with these operations could then be constructed by the following operations (suppressing the superscript (t) for

[14]This corresponds to an example of dealing with reducible and irreducible representations respectively. These terms will become clearer when we talk about the relations between Cartesian and spherical tensors.

[15]Global translations can be dealt with easily because we are normally only interested in invariant features with respect to translations. This can be achieved for example through zero-centring the point cloud by subtracting the centre of mass, or by working exclusively with displacement vectors between nodes. In both cases, a global translation $\vec{t}$ drops out in the subtraction $\vec{v}_1 - \vec{v}_2 \mapsto (\vec{v}_1 + \vec{t}) - (\vec{v}_2 + \vec{t}) = \vec{v}_1 - \vec{v}_2$. We will speak more about reflections in Section 5.3.

[16]While the notations "scalar-type" and "vector-type" may seem like an unnecessary burden at this point, we want to emphasize that thinking constantly about the types of the objects being manipulated is paramount in this section.

[17]You might rightly wonder about the cross-product here. Wouldn't that give us a valid type too? In fact, the cross-product of two vectors would yield a pseudo-vector, which transforms differently under point reflections at the origin than a vector would and is, therefore, a different type. A vector would flip sign under point reflections while the cross product's sign remains unchanged (try it for yourself!). Hence, it is not in our multiplication table. We will return to products such as the cross-product slightly later and see that it effectively corresponds to a special case of the tensor product.

readability and letting $\mathbf{m}_i$, $\vec{\mathbf{m}}_i$ denote the aggregated message for node i from $\mathcal{N}_i$, right before the actual update):

$$\mathbf{m}_i := \underline{f_1}(\mathbf{s}_i, \|\vec{\mathbf{v}}_i\|) + \sum_{j \in \mathcal{N}_i} \underline{f_2}\left(\mathbf{s}_i, \mathbf{s}_j, \|\vec{x}_{ij}\|, \|\vec{\boldsymbol{v}}_j\|, \vec{x}_{ij} \cdot \vec{\mathbf{v}}_j, \vec{x}_{ij} \cdot \vec{\mathbf{v}}_i, \vec{\mathbf{v}}_i \cdot \vec{\mathbf{v}}_j\right) \tag{14}$$

$$\begin{aligned}\vec{\mathbf{m}}_i := \underline{f_3}(\mathbf{s}_i, \|\vec{\mathbf{v}}_i\|) \odot \vec{\mathbf{v}}_i + \sum_{j \in \mathcal{N}_i} \underline{f_4}\left(\mathbf{s}_i, \mathbf{s}_j, \|\vec{x}_{ij}\|, \|\vec{\boldsymbol{v}}_j\|, \vec{x}_{ij} \cdot \vec{\mathbf{v}}_j, \vec{x}_{ij} \cdot \vec{\mathbf{v}}_i, \vec{\mathbf{v}}_i \cdot \vec{\mathbf{v}}_j\right) \odot \vec{\mathbf{v}}_j \\ + \sum_{j \in \mathcal{N}_i} \underline{f_5}\left(\mathbf{s}_i, \mathbf{s}_j, \|\vec{x}_{ij}\|, \|\vec{\boldsymbol{v}}_j\|, \vec{x}_{ij} \cdot \vec{\mathbf{v}}_j, \vec{x}_{ij} \cdot \vec{\mathbf{v}}_i, \vec{\mathbf{v}}_i \cdot \vec{\mathbf{v}}_j\right) \odot \vec{x}_{ij},\end{aligned} \tag{15}$$

where $\underline{f_1}$ to $\underline{f_5}$ are learnable, possibly non-linear, functions and $\odot$ denotes element-wise multiplication[18]. Special cases of the "most general" equations above give rise to a whole host of published architectures [Jing et al., 2020, Schütt et al., 2021a, Satorras et al., 2021b, Thölke and De Fabritiis, 2022, Du et al., 2022, Le et al., 2022, Morehead and Cheng, 2022].

For example, in PaiNN [Schütt et al., 2021b] interaction layers aggregate scalar and vector features via learnt filters conditioned on the relative distance, as shown in Figure 19a:

$$\boldsymbol{m}_i^{(t)} := \boldsymbol{s}_i^{(t)} + \sum_{j \in \mathcal{N}_i} \underline{f_1}\left(\boldsymbol{s}_j^{(t)}, \|\vec{x}_{ij}\|\right) \tag{16}$$

$$\vec{\mathbf{m}}_i^{(t)} := \vec{\mathbf{v}}_i^{(t)} + \sum_{j \in \mathcal{N}_i} \underline{f_2}\left(\boldsymbol{s}_j^{(t)}, \|\vec{x}_{ij}\|\right) \odot \vec{\mathbf{v}}_j^{(t)} + \sum_{j \in \mathcal{N}_i} \underline{f_3}\left(\boldsymbol{s}_j^{(t)}, \|\vec{x}_{ij}\|\right) \odot \vec{x}_{ij}. \tag{17}$$

TorchMD-Net [Thölke and De Fabritiis, 2022], an equivariant transformer-based GNN, extends the above message passing layer to attention by choosing the $\underline{f}$'s appropriately. E-GNN [Satorras et al., 2021b] and GVP-GNN [Jing et al., 2020] also fall within this paradigm. The update step applies a gated non-linearity [Weiler et al., 2018] on the vector features, which learns to scale their magnitude using their norm concatenated with the scalar features:

$$\boldsymbol{s}_i^{(t+1)} := \boldsymbol{m}_i^{(t)} + \underline{f}_4\left(\boldsymbol{m}_i^{(t)}, \|\vec{\mathbf{m}}_i^{(t)}\|\right), \qquad \vec{\mathbf{v}}_i^{(t+1)} := \vec{\mathbf{m}}_i^{(t)} + \underline{f}_5\left(\boldsymbol{m}_i^{(t)}, \|\vec{\mathbf{m}}_i^{(t)}\|\right) \odot \vec{\mathbf{m}}_i^{(t)}. \tag{18}$$

The updated scalar features are both $\mathcal{G}$-invariant and $T(d)$-invariant because the only geometric information used is the relative distances, while the updated vector features are $\mathcal{G}$-equivariant and $T(d)$-invariant as they aggregate $\mathcal{G}$-equivariant, $T(d)$-invariant vector quantities from the neighbours.

These *scalar-vector* GNNs achieve good performance and are relatively fast by avoiding expensive operations. Obtaining them required us to manually define and exploit the multiplication rules between scalars and vectors that we already knew. But are these all the possible multiplication-like operations that exist in geometric operations? As we shall see, these scalar-vector GNNs are but special cases of a much broader possible design space[19].

Higher tensor-types and the tensor product. We just saw that we can understand many of the published equivariant GNNs as special cases of scalar-vector type GNNs in which we restrict ourselves to only two types of features: scalars and vectors. Why stop there? We could also create other types of features which transform differently. For example, if we have two (possibly identical) vectors $\vec{v}$ and $\vec{w}$, we could create a matrix from them by assigning the components $\mathbf{M}_{ij} = \vec{v}_i \vec{w}_j$. Under a global rotation $\mathbf{R}$, this matrix transforms as $\mathbf{M} \mapsto \mathbf{R}\mathbf{M}\mathbf{R}^\top$, or in component notation[20]:

$$\mathbf{M}_{ij} \mapsto \sum_{m=1}^{3} \sum_{n=1}^{3} \mathbf{R}_{im} \mathbf{R}_{jn} \mathbf{M}_{mn}.$$

[18] If it is unclear to you why this is the most general form, think of it this way: first, as stated above, component-wise non-linearities ($\underline{f_i}$) can only be applied to scalars so they can only work from all possible scalar quantities at our disposal, namely scalar features and dot products; second we decompose self-interactions and neighbourhood interactions; lastly in the case of $\vec{\mathbf{m}}_i$ we compose those non-linearities with all the *vector* features at our disposal. Note that we do indeed maintain equivariance of vector features because of the distributivity of the matrix-vector product: $\mathbf{R}\vec{v} + \mathbf{R}\vec{u} = \mathbf{R}(\vec{u} + \vec{v})$.

[19] What we call scalar-vector-based models here are sometimes referred to as low tensor rank models elsewhere, for reasons that will become apparent shortly.

[20] It is a good exercise to verify this by hand once.

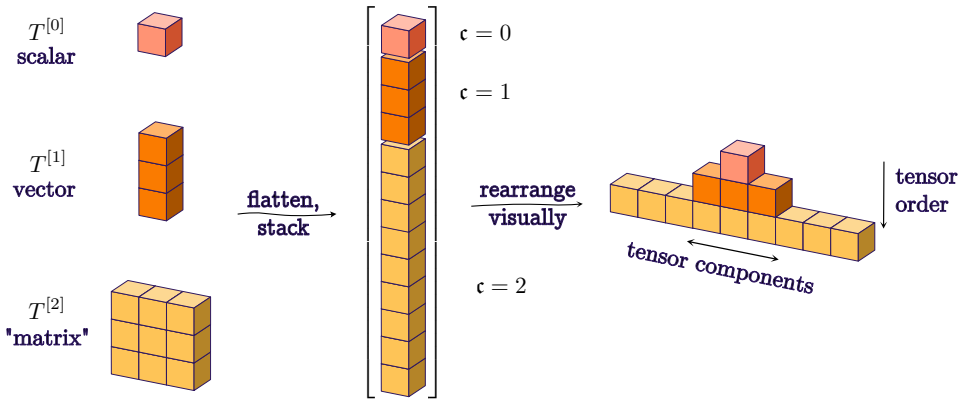


Figure 11: Cartesian Tensors, represented visually. The left column shows a few familiar objects that behave as Cartesian tensors do under rotations: scalars, vectors, matrices. When we flatten them (central column), it becomes apparent that higher-order Cartesian tensors can be re-interpreted as vectors in a higher-dimensional linear space. A matrix, for example, can be flattened into a vector in a 9-dimensional space. On the right, we group the tensors together into a pyramid to visually disentangle the tensor component and tensor order axis.

Importantly, if we construct two matrices this way, any sum or indeed linear combination of so constructed matrices will continue to transform as stated above. Further, as we have just seen, this matrix-like transformation is a different transformation than that of the vector types $\vec{v}$ or $\vec{w}$, which each transforms as $\vec{v} \mapsto \mathbf{R}\vec{v}$. We have therefore discovered a *new type*[21]!

In the same way, we created the "matrix" type above, we can also create objects of yet other types, which transform differently than matrices. For example, the object comprised of the components[22] $\boldsymbol{T}_{ijk} = \vec{v}_i \vec{v}_j \vec{w}_k$ would transform as

$$\boldsymbol{T}_{ijk} = \sum_{l=1}^{3} \sum_{m=1}^{3} \sum_{n=1}^{3} \mathbf{R}_{il} \mathbf{R}_{jm} \mathbf{R}_{kn} \boldsymbol{T}_{lmn}.$$

By its definition, this object transforms differently than a vector-type $\vec{v}$ or matrix-type $\mathbf{M}$. It transforms with the help of 3 copies of a rotation matrix and has $3^3 = 27$ components.

We shall call *Cartesian tensors* $\vec{T}^{[\mathfrak{c}]}$ the new types that we can build by multiplying each combination of components of $\mathfrak{c}$ vectors, illustrated in Figure 11. We assign $\mathfrak{c}$ indices to enumerate them and call this number the rank of the tensor. Channel indices are ignored without loss of generality, as shown in Figure 12. The matrix $\mathbf{M}_{ij}$ that we constructed above is therefore a rank-2 type Cartesian tensor $\vec{M}^{[2]}$ and the object $\boldsymbol{T}_{ijk}$ is a rank-3 type Cartesian tensor $\vec{T}^{[3]}$. A vector $\vec{v} = \vec{V}^{[1]}$ could be considered a rank-1 type Cartesian tensor, although for consistency and ease of notation, we will continue to refer to vectors via the simpler notation $\vec{v}$.

Mathematically, the above way of creating new types with more indices from existing ones is called taking the *tensor product*[23] (denoted as $\otimes$) of types we had previously, which is why we called these new types tensors. As we just saw, it gives us a way to build tensor types of higher and higher rank, i.e. more and more indices. We can naturally apply the tensor product not just to vectors but any

[21] We use the word *type* here as we imagine our readers to mostly be computer scientists to whom this term will be more familiar. Mathematically, our types correspond to different representations of the relevant group, i.e. $O(3)$ or $SO(3)$ here.

[22] You may think of $\boldsymbol{T}$ as a $3 \times 3 \times 3$ cube.

[23] The tensor product can also be defined abstractly as a map from two input linear spaces to a third one (the tensor product space) that satisfies what is called the *universal property*, which loosely says that for every bilinear map on the original two linear spaces there is a unique, corresponding linear map in the third linear space. It turns out this construction is unique up to a (unique) isomorphism and its practical instantiation gives the tensor product we "re-discovered" above.

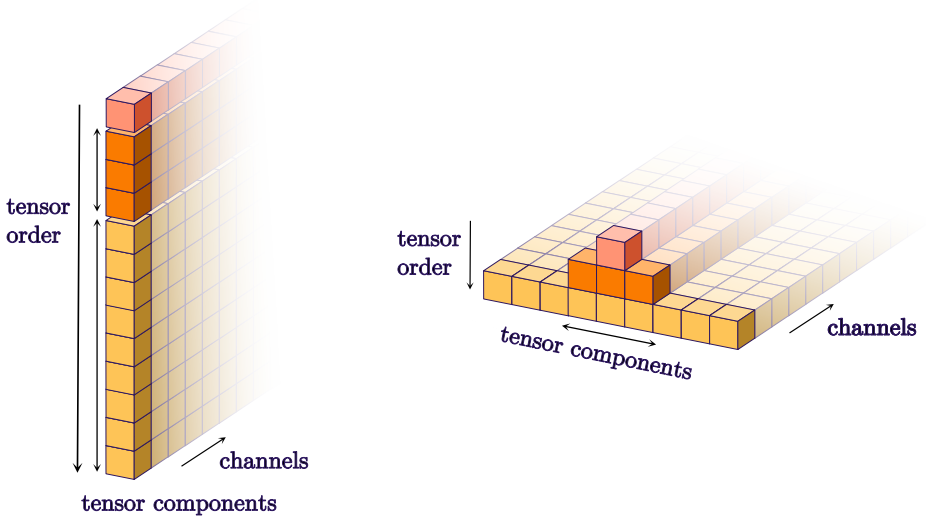


Figure 12: For simplicity we omit the channel axis in most of our illustrations and the channel index in our equations. In all contemporary machine learning models, you would carry an implicit channel index. The channel axis has no bearing on the transformation behaviour, which is why we omit it without loss of generality.

other Cartesian tensors[24] as well. In the new notation:

$$\vec{M}^{[2]} = \vec{v} \otimes \vec{w} \qquad \text{component wise: } \vec{M}^{[2]}_{ij} = \vec{v}_i \vec{w}_j \tag{19}$$

$$\vec{T}^{[3]} = \vec{u} \otimes \vec{v} \otimes \vec{w} \qquad \text{component wise: } \vec{T}^{[3]}_{ijk} = \vec{u}_i \vec{v}_j \vec{w}_k \tag{20}$$

$$\vec{U}^{[5]} = \vec{x} \otimes \vec{y} \otimes \vec{T}^{[3]} \qquad \text{component wise: } \vec{U}^{[5]}_{ijklm} = \vec{x}_i \vec{y}_j \vec{T}^{[3]}_{klm} \tag{21}$$

A Cartesian tensor of a given rank $\mathfrak{c}$ is a well-defined type according to the type intuition (1), because it consistently transforms in a predictable way under rotations and reflections, and linear combinations of tensors of the same type also yield a tensor of that type[25].

To write down these transformation rules and deal with tensors more generally, it is convenient to use a notational convention called *Einstein summation*. In *Einstein summation* notation, repeated indices are understood to be summed over and we therefore drop the explicit summation symbol. Let's see a few examples and write down the ways the Cartesian tensors we constructed above transform:

Full notation:	Einstein summation notation:	
$\sum_{i=1}^{3} \vec{v}_i \vec{w}_i$	$\vec{v}_i \vec{w}_i$	(23)
$\sum_{m=1}^{3} \sum_{n=1}^{3} \mathbf{R}_{im} \mathbf{R}_{jn} \vec{M}^{[2]}_{mn}$	$\mathbf{R}_{im} \mathbf{R}_{jn} \vec{M}^{[2]}_{mn}$	(24)
$\sum_{l=1}^{3} \sum_{m=1}^{3} \sum_{n=1}^{3} \mathbf{R}_{il} \mathbf{R}_{jm} \mathbf{R}_{kn} \vec{T}^{[3]}_{lmn}$	$\mathbf{R}_{il} \mathbf{R}_{jm} \mathbf{R}_{kn} \vec{T}^{[3]}_{lmn}$	(25)
$\sum_{i=1}^{3} \sum_{j=1}^{3} \sum_{k=1}^{3} \sum_{l=1}^{3} \sum_{m=1}^{3} \mathbf{R}_{ri} \mathbf{R}_{sj} \mathbf{R}_{tk} \mathbf{R}_{ul} \mathbf{R}_{vm} \vec{U}^{[5]}_{ijklm}$	$\mathbf{R}_{ri} \mathbf{R}_{sj} \mathbf{R}_{tk} \mathbf{R}_{ul} \mathbf{R}_{vm} \vec{U}^{[5]}_{ijklm}$	(26)

[24] Mathematically, these are simply vectors in a higher-dimensional linear space that is the tensor product of two lower dimensional linear spaces. The reason we call them tensors is mostly because we keep multiple indices for them around, because this makes it easy to address how they transform via standard rotation matrices, but there are many different but equivalent "viewpoints" you can take, illustrated in Figure 11.

[25] Mathematically, these two properties mean that the tensor product linear space satisfies the definition a linear representation of the rotation and reflection group $O(3)$.

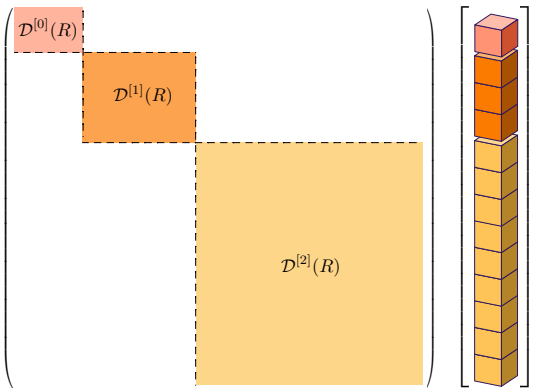


Figure 13: Illustration of the transformation of a compound tensor composed of $\mathfrak{c} = 0, 1, 2$ components when seen as a column vector in a higher dimensional space. The transformation matrix is block-diagonal and the blocks $\mathcal{D}^{[0]}(R) = 1$ and $\mathcal{D}^{[1]}(R) = R$ are familiar. $\mathcal{D}^{[2]}(R)$ corresponds to a transformation matrix that performs $\cdot \mapsto R \cdot R^\top$ on the flattened, 9 dimensional tensor $T^{[2]}$. These matrices represent rotations on rank $\mathfrak{c}$ Cartesian tensors.

As we can see, Einstein summation helps express the transformation rules more concisely. Unless otherwise specified and until the end of this section, *we will assume Einstein summation*.

Just as above, we can now investigate which multiplication rules are allowed, in the sense that they allow us to do the diligent accounting of multiplication rule (3) given a certain set of tensor types that we wish to use in our model. With the tensor product, we have found a way to multiply two Cartesian tensors of ranks $\mathfrak{c}_1$ and $\mathfrak{c}_2$ and obtain a new Cartesian tensor of rank $\mathfrak{c}_1 + \mathfrak{c}_2$, with $3^{(\mathfrak{c}_1+\mathfrak{c}_2)}$ components. But could we also "multiply" two higher-order types together to get a type of the same or lower-order rank?

We already know one operation that goes from two higher-rank tensors to a lower rank: the dot product, which turns two vectors into a scalar:

$$\vec{V}^{[0]} = \vec{T}^{[1]}_{\mathbf{n}} \vec{U}^{[1]}_{\mathbf{n}} \qquad \text{(Einstein summation!)}.$$

We can see that dimensions 1 and 2 in $\vec{T}^{[1]}_{\mathbf{n}} \vec{U}^{[1]}_{\mathbf{n}}$ have been *contracted* into dimension 0 of $\vec{V}^{[0]}$.

It turns out that performing a generalised dot product-like operation called *contraction* gives us a way to equivariantly generate tensors of lower ranks from higher ranks by summing over pairs of dimensions.

Imagine we have a rank-3 Cartesian tensor $\vec{T}^{[3]}$ and a rank-5 tensor Cartesian tensor $\vec{U}^{[5]}$, then the *contracted* combination along the pairs of dimensions $(1:3)$ and $(2:5)$:

$$\vec{V}^{[4]}_{ijkl} = \vec{T}^{[3]}_{\mathbf{nmn}} \vec{U}^{[5]}_{i\mathbf{m}jkl}$$

will transform as a rank-4 Cartesian tensor. To see that $\vec{V}^{[4]}$ will indeed transform predictably under rotations, we use the fact that rotation and translation matrices are orthonormal: $(\mathbf{R}_{ia}\mathbf{R}_{ja})_{ij} = (\mathbf{R}_{ia}\mathbf{R}^\top_{aj})_{ij} = (\mathbf{R}\mathbf{R}^\top)_{ij} = (\mathbb{1})_{ij} = \delta_{ij}$, with $\mathbb{1}$ the identity matrix, δ_{ij} the Kronecker delta[26] and using Einstein notation. Using this identity and the transformation rules for $\vec{T}^{[3]} \otimes \vec{U}^{[5]}$ from above, $\vec{V}^{[4]}$ transforms as:

[26]The Kronecker delta is 1 if $i = j$ and 0 otherwise.

$$
\begin{aligned}
\vec{V}^{[4]}_{ijkl} &= \vec{T}^{[3]}_{nmn}\vec{U}^{[5]}_{imjkl} = \delta_{no}\delta_{mp}\vec{T}^{[3]}_{nmo}\vec{U}^{[5]}_{ipjkl} \qquad \text{(Replacing repeated indices by a } \delta \text{ and a new index)} \\
&\mapsto \left(\mathbf{R}_{n'n}\mathbf{R}_{o'o}\delta_{no}\right)\left(\mathbf{R}_{m'm}\mathbf{R}_{p'p}\delta_{mp}\right)\left(\mathbf{R}_{n'n}\mathbf{R}_{o'o}\mathbf{R}_{m'm}\mathbf{R}_{p'p}\mathbf{R}_{i'i}\mathbf{R}_{j'j}\mathbf{R}_{k'k}\mathbf{R}_{l'l}\vec{T}^{[3]}_{nmo}\vec{U}^{[5]}_{ipjkl}\right) \\
&= \left(\mathbf{R}_{n'n}\mathbf{R}_{o'n}\right)\left(\mathbf{R}_{m'm}\mathbf{R}_{p'm}\right)\left(\mathbf{R}_{i'i}\mathbf{R}_{j'j}\mathbf{R}_{k'k}\mathbf{R}_{l'l}\vec{T}^{[3]}_{n'm'o'}\vec{U}^{[5]}_{ip'jkl}\right) \\
&= \delta_{n'o'}\delta_{m'p'}\mathbf{R}_{i'i}\mathbf{R}_{j'j}\mathbf{R}_{k'k}\mathbf{R}_{l'l}\vec{T}^{[3]}_{n'm'o'}\vec{U}^{[5]}_{ip'jkl} \\
&= \mathbf{R}_{i'i}\mathbf{R}_{j'j}\mathbf{R}_{k'k}\mathbf{R}_{l'l}\vec{T}^{[3]}_{n'm'n'}\vec{U}^{[5]}_{im'jkl} \\
&= \mathbf{R}_{i'i}\mathbf{R}_{j'j}\mathbf{R}_{k'k}\mathbf{R}_{l'l}\vec{V}^{[4]}_{ijkl}.
\end{aligned}
$$

This is the transformation of a rank 4 Cartesian tensor, so $\vec{V}^{[4]}$ is indeed a rank 4 Cartesian tensor! We have thus found a consistent way to generate lower-rank Cartesian tensors by contracting higher-rank tensors, for example from the tensor product of Cartesian tensors. We denote the contraction operation as $\mathfrak{C}$ and write

$$
\vec{V}^{[0]} = \mathfrak{C}_{(1:2)}[\vec{T}^{[1]} \otimes \vec{U}^{[1]}] := \vec{T}^{[1]} \otimes_{(1:2)} \vec{U}^{[1]}, \tag{27}
$$

$$
\vec{V}^{[4]} = \mathfrak{C}_{(1:3)(2:5)}[\vec{T}^{[3]} \otimes \vec{U}^{[5]}] := \vec{T}^{[3]} \otimes_{(1:3)(2:5)} \vec{U}^{[5]}, \tag{28}
$$

to signify the contraction of dimension 1 with 3 (n with n in $\vec{T}^{[3]}$), and 2 with 5 (m in $\vec{T}^{[3]}$ with m in $\vec{U}^{[5]}$) of the intermediate rank 8 Cartesian tensor $\vec{T}^{[3]} \otimes \vec{U}^{[5]}$ that results from the tensor product.

Let us step back and reflect on what we have learned. We saw that scalars and vectors are not the only types with which information can transform consistently with rotations and reflections. Indeed, we found many examples of Cartesian tensors of different ranks that transform differently to scalars and vectors. And with the (contracted) tensor product and the notion of Cartesian tensors, we now have the tools to build new, higher-rank types and contract them to lower-rank types which transform consistently under rotations and translations. By keeping with the rules of diligent accounting set out at the start of Section 5, we can use these Cartesian tensor types and the above operations to build equivariant GNNs with higher-order tensor types[27]. However, in the literature higher-order Cartesian tensor GNNs are not common, mostly because of the exponential cost in memory that comes with creating and storing many tensors of higher ranks[28].

Instead of building a Cartesian GNN, let us turn to two natural questions that arise with the tools to build and contract Cartesian tensors at hand. Discussing these questions will lead us to the common class of equivariant GNNs with higher tensor types used in the current machine learning literature.

1. **Exhaustiveness**: How do we know these Cartesian tensor types capture all possible types of transformation for rotational symmetries, or could there be some types that are constructed in yet a different way?
2. **Usefulness**: Does using higher-rank tensors and contracting them down to scalars (sometimes colloquially referred to as *scalarisation*) yield any new information about our point cloud? In particular, can we obtain any new invariants that cannot be built from scalars and vectors via the interactions in the scalar-vector GNNs we saw above?

To answer these questions, let's switch gears and utilize a well-established mathematical result regarding the representation theory of the rotation and rotation group $SO(3)$. This will lead us to the concept of spherical tensors, which are tensor types corresponding to the irreducible representations of $SO(3)$. More details will be explained in the next section.

[27] The creation of higher-order Cartesian tensors and their contraction to lower order tensors here mostly serves didactic purposes. If one were to build a Cartesian tensor-based GNN one would need to include asymmetric contractions, in addition to the symmetric contractions introduced in the main text, to reach generality. We omit this here and instead make the link to spherical tensors and harmonics, which form the backbone of many current equivariant GNNs.

[28] A rank $\mathfrak{c}$ Cartesian tensor in 3D has $3^{\mathfrak{c}}$ components

5.2 From Cartesian tensors to spherical tensors – reducible to irreducible representations

For the rotation group $SO(3)$, the fundamental building blocks into which all types can be decomposed are known. Such LEGO-block-like types are called irreducible representations, or *irreps* for short. In our language, irreps correspond to tensor types that can be combined[29] to create any other possible tensor type – including any Cartesian tensor that we can come up with via the rules in Section 5.1. Conversely, any other type that is not an irrep can be decomposed into its irrep components. The irreps of $SO(3)$, can be numbered by non-negative integers $l = 0, 1, 2, \dots$ and we will refer to them as *spherical tensor* types $\vec{T}^{(l)}$[30]. A spherical tensor of type l is $2l+1$-dimensional, that is it can be represented as a list of $2l+1$ components. All other (finite-dimensional)[31] representations of $SO(3)$ are fully reducible into these irreps[32]. These results are well known in the mathematical subfield of representation theory, but their proofs are beyond the scope of this hitch-hike[33].

Rather than provide proofs, we will continue with examples to build intuition and connect the theory to the machine learning practice. We said irreps are the atomic building blocks of any possible representation and we can decompose more complicated representations (types) into their atomic components. Let us see an example of this and decompose an arbitrary rank-2 Cartesian tensor $\vec{T}^{[2]}$ into irreps (visually illustrated in Figure 14):

$$
\begin{aligned}
\vec{T}^{[2]} &= \begin{bmatrix} T_{xx} & T_{xy} & T_{xz} \\ T_{yx} & T_{yy} & T_{yz} \\ T_{zx} & T_{zy} & T_{zz} \end{bmatrix} \\
&= \frac{T_{xx}+T_{yy}+T_{zz}}{3} \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} && (l=0) \\
&+ \begin{bmatrix} & \frac{T_{xy}-T_{yx}}{2} & \frac{T_{xz}-T_{zx}}{2} \\ -\frac{T_{xy}-T_{yx}}{2} & & \frac{T_{yz}-T_{zy}}{2} \\ -\frac{T_{xz}-T_{zx}}{2} & -\frac{T_{yz}-T_{zy}}{2} & \end{bmatrix} && (l=1) \\
&+ \begin{bmatrix} T_{xx} & \frac{T_{xy}+T_{yx}}{2} & \frac{T_{xz}+T_{zx}}{2} \\ \frac{T_{xy}+T_{yx}}{2} & T_{yy} & \frac{T_{yz}+T_{zy}}{2} \\ \frac{T_{xz}+T_{zx}}{2} & \frac{T_{yz}+T_{zy}}{2} & T_{zz} \end{bmatrix} - \frac{T_{xx}+T_{yy}+T_{zz}}{3} \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} && (l=2) \\
&= \overbrace{\frac{T_{xx}+T_{yy}+T_{zz}}{3}}^{\lambda_1} \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} && (l=0) \\
&+ \overbrace{\frac{T_{yz}-T_{zy}}{2}}^{\lambda_2} \begin{bmatrix} & & \\ & -1 & 1 \\ & & \end{bmatrix} + \overbrace{\frac{T_{zx}-T_{xz}}{2}}^{\lambda_3} \begin{bmatrix} & & -1 \\ & & \\ 1 & & \end{bmatrix} + \overbrace{\frac{T_{xy}-T_{yx}}{2}}^{\lambda_4} \begin{bmatrix} & 1 & \\ -1 & & \\ & & \end{bmatrix} && (l=1) \\
&+ \overbrace{\frac{2T_{xx}-T_{yy}-T_{zz}}{3}}^{\lambda_5} \begin{bmatrix} 1 & & \\ & -1 & \\ & & \end{bmatrix} + \overbrace{\frac{T_{xy}+T_{yx}}{2}}^{\lambda_6} \begin{bmatrix} & 1 & \\ 1 & & \\ & & \end{bmatrix} + \overbrace{\frac{T_{xz}+T_{zx}}{2}}^{\lambda_7} \begin{bmatrix} & & 1 \\ & & \\ 1 & & \end{bmatrix} \\
&+ \overbrace{\frac{T_{yz}+T_{zy}}{2}}^{\lambda_8} \begin{bmatrix} & & \\ & & 1 \\ & 1 & \end{bmatrix} + \overbrace{\frac{2T_{zz}-T_{xx}-T_{yy}}{3}}^{\lambda_9} \begin{bmatrix} & & \\ & -1 & \\ & & 1 \end{bmatrix} && (l=2)
\end{aligned}
$$

In the above equations, missing matrix entries represent 0's and the $l = 0, 1, 2$ components are color-coded for clarity. Each component corresponds to a subspace of the 9-dimensional linear space that contains $\vec{T}^{[2]}$. To understand how these components relate to quantities we already know,

[29] By combined we here mean taking the direct sum $\oplus$, the mathematical definition of concatenation, and performing a change of basis.

[30] Notice that we use parentheses () to explicitly denote spherical tensor types as opposed to [], which denoted Cartesian tensor types.

[31] This decomposition result also holds for many infinite dimensional representations, e.g. for $L^2(SO(3))$ via the Peter-Weyl Theorem [Zee, 2016].

[32] Representations of compact Lie groups, such as $SO(3)$ or $O(3)$, are fully reducible [Zee, 2016].

[33] Zee [2016], chp. 4 is a good textbook reference.

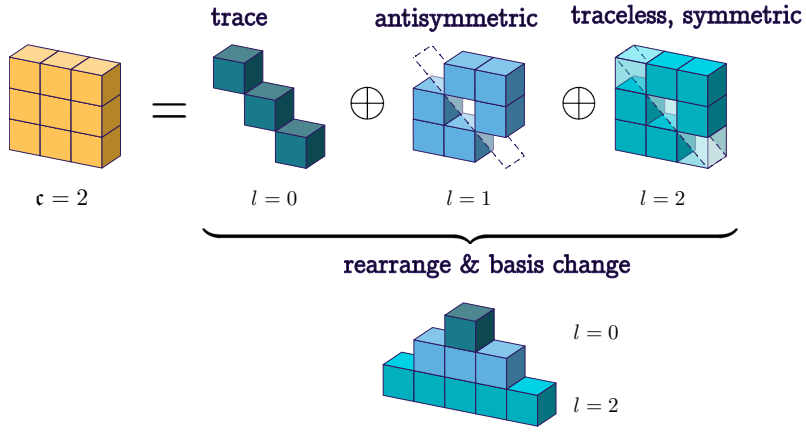


Figure 14: Visual illustration of the decomposition of a $\mathfrak{c} = 2$ Cartesian tensor into spherical tensors. The 9 dimensional Cartesian vector decomposes into a 1 dimensional $l = 0$ (scalar), a 3 dimensional $l = 1$ (vector) and a 5 dimensional $l = 2$ component. This decomposition corresponds to a change of basis in the 9 dimensional space in which $\vec{T}^{[2]}$ lives and the change of basis equations are given in the main text.

assume for a moment that $\vec{T}^{[2]} = \vec{v} \otimes \vec{w}$ is the result of a tensor product of two vectors. In this case $T_{xy} = v_x w_y, T_{yx} = v_y w_x$ and so on. The $(l = 0)$ part in the decomposition has one component (λ_1) and is therefore one-dimensional. λ_1 corresponds to the trace of $\vec{T}^{[2]}$ and it remains unchanged under global rotations. In the case where $\vec{T}^{[2]}$ is the tensor product of two vectors, the $(l = 0)$ component holds the same information as the dot-product $\vec{v} \cdot \vec{w}$ which is, of course, invariant:

$$\vec{v} \cdot \vec{w} = v_x w_x + v_y w_y + v_z w_z = 3\lambda_1$$

The $(l = 1)$ part is the asymmetric part of the matrix and it corresponds to the cross-product space. To see this, assume $\vec{T}^{[2]} = \vec{v} \otimes \vec{w}$, then:

$$\vec{v} \times \vec{w} = \begin{bmatrix} v_y w_z - v_z w_y \\ v_z w_x - v_x w_z \\ v_x w_y - v_y w_x \end{bmatrix} = \begin{bmatrix} \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix} \tag{29}$$

and we observe that the coefficients in the decomposition of the $(l = 1)$ component of $\vec{v} \otimes \vec{w}$ coincide with those of the cross product $\vec{v} \times \vec{w}$! Therefore the $(l = 1)$ part is three-dimensional and transforms under rotations just as a vector would, as can be seen from Equation (29).

Finally, the $(l = 2)$ component of $\vec{T}^{[2]}$ corresponds to a traceless, symmetric matrix and from the basis decomposition above we can see that it is 5 dimensional, with coefficient λ_5 to λ_9. Unfortunately, the $(l = 2)$ subspace does not have as easy of a relation to quantities we have already seen – but from the relation between the irreps to the spherical harmonics, which we will see shortly, we can gain the intuition that $l \geq 2$ irreps capture ever higher frequency information and therefore increasingly fine-grained angular information.

To summarize, we have seen that the 9-dimensional rank-2 Cartesian tensor can be decomposed into a 1D, 3D and 5D part which correspond to the $l = 0, 1, 2$ irreps respectively

$$3 \otimes 3 = 1 \oplus 3 \oplus 5. \tag{30}$$

In this slightly loose notation, the $\oplus$ symbol denotes the direct sum of a 1D, 3D and 5D linear space. This amounts to a concatenation in machine learning lingo.

More generally, we can decompose *any* Cartesian tensor into irreps of various l, albeit with slightly more complicated decompositions than the above. And as we have seen in the above example, the special cases $l = 0$ and $l = 1$ correspond to familiar scalar-type and vector-type information respectively and transform under global rotations as scalars or vectors would.

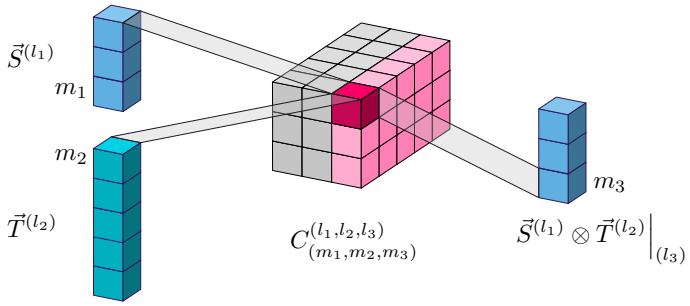


Figure 15: Illustration of the Clebsch-Gordan coefficients drawn as a three-dimensional list of numbers. Note that these are *not* directly a Cartesian or spherical tensor, but merely arranged as a multidimensional list of numbers for visual convenience. In the illustration, $l_1 = 1, l_2 = 2, l_3 = 1$. Therefore, $C^{(l_1,l_2,l_3)}$ has $(2l_1 + 1)(2l_2 + 1)(2l_3 + 1) = 45$ components. The element $C^{(l_1,l_2,l_3)}_{(m_1,m_2,m_3)}$ is highlighted in red for clarity. It encodes how to weight the product of m_1 and m_2 for m_3 such that the output transforms as an l_3 tensor. Bear in mind that this term is only one of the 12 terms (highlighted in pink) that make up m_3 and that for our selection of $l_1 = 1, l_2 = 2$ there would also be an $l_4 = 2$ and $l_5 = 3$ output in the tensor product, which are not shown in the visualisation.

Rotating spherical tensors. For types of higher l, the transformation behaviour under rotations is also known and given by the so-called Wigner D-matrices. The *D* here stands for *Darstellung*, the German word for *representation*. The Wigner D-matrix is how one can represent a rotation in the $2l + 1$ dimensional space of a degree l spherical tensor. Explicit formulas for these matrices exist[34] and are implemented in packages such as `e3nn` [Geiger and Smidt, 2022]. For example, the $(l = 2)$ part above transforms under a rotation $\mathbf{R}$ as:

$$[\lambda_5 \dots \lambda_9]^\top \to \mathcal{D}^{(2)}(\mathbf{R})[\lambda_5 \dots \lambda_9]^\top, \tag{31}$$

where $\mathcal{D}^{(2)}(\mathbf{R})$ is the 5×5 Wigner matrix representing the rotation $\mathbf{R}$ acting on a feature of type $l = 2$.

Tensor products of spherical tensors The tensor product and tensor contraction were the two key operations we encountered that allowed us to move up and down the rank-ladder of Cartesian tensors to produce tensors of higher rank or contract them down to lower ranks. Unfortunately, the tensor product $\vec{S}^{(l_1)} \otimes \vec{T}^{(l_2)}$ of two spherical tensors $\vec{S}^{(l_1)}$ and $\vec{T}^{(l_2)}$ is generally not a spherical tensor anymore. However, as we have learned the spherical tensor types are the atomistic building blocks into which all other types can be decomposed, and so we can decompose the product $\vec{S}^{(l_1)} \otimes \vec{T}^{(l_2)}$ back into spherical tensors.

Luckily for us, a general formula for the decomposition of a tensor product of irreps of $SO(3)$ is known. As a rule, the $(l_1 l_2)$-dimensional tensor product of two spherical tensors of ranks l_1 and l_2 decomposes into:

$$l_1 \otimes l_2 = |l_1 - l_2| \oplus |l_1 - l_2 + 1| \oplus \dots \oplus (l_1 + l_2 - 1) \oplus (l_1 + l_2). \tag{32}$$

This means the $l_1 l_2$-dimensional product decomposes into exactly one spherical tensor for each rank between the absolute difference $|l_1 - l_2|$ and the sum $l_1 + l_2$. As a result, the tensor product of two spherical tensors results in $l_1 + l_2 - |l_1 - l_2| + 1$ new spherical tensors and each valid combination of (l_1, l_2, l_3), where l_3 is the type of the output tensor, is colloquially referred to as a tensor *path*.

As an example, if $l_1 = 1, l_2 = 2$, then the output contains an $l_3 = 1, l_3 = 2$ and $l_3 = 3$ part: $1 \otimes 2 = 1 \oplus 2 \oplus 3$. Each of these components corresponds to one tensor path and Figure 15 shows the $l_3 = 1$ part pictorially.

The coefficients of the decomposition are given by the *Clebsch-Gordan coefficients*, and these are again implemented in packages such as `e3nn`. The Clebsch-Gordan coefficients are clearly basis-dependent, and we will get to the common choice of basis in the machine learning community

[34]The formulas for Wigner D-matrix coefficients are basis-specific and we will get into the choice of basis for the irreps in the next section. In short, the real space spherical harmonics provide the basis of irreps that is typically used in the Machine Learning community.

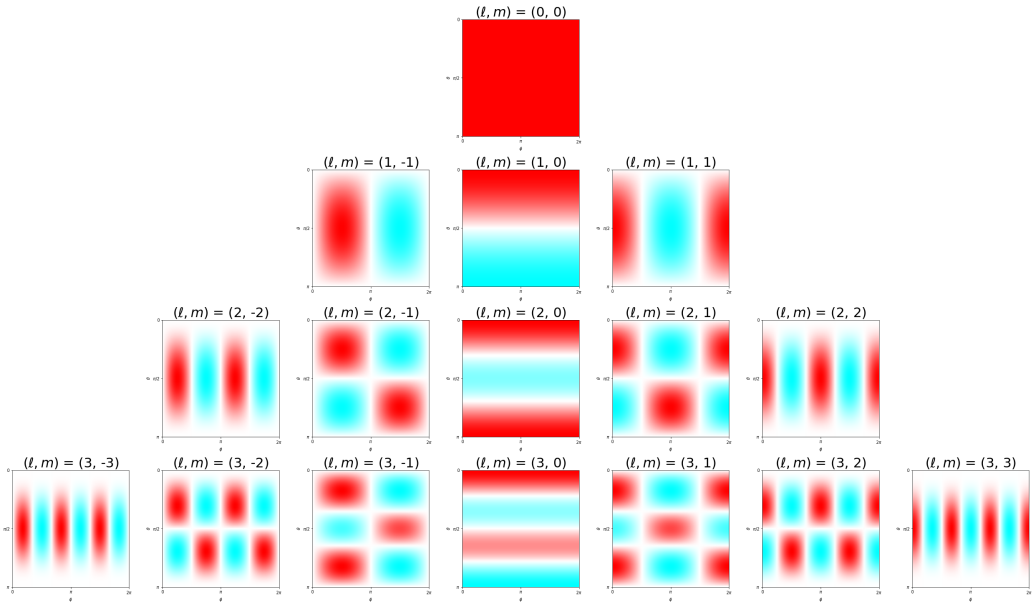


Figure 16: The real spherical harmonics $\vec{Y}_m^{(l)}(\hat{x})$ plotted as a function of $\hat{x} = (\phi, \theta)$ on the unit sphere. The horizontal axis corresponds to the azimuthal angle ϕ, and the vertical axis to the polar angle θ. The saturation of the color at any point represents the magnitude of the spherical harmonic and the hue represents the sign. Figure taken from Wikipedia.

shortly. In component form, they are often denoted by the symbol $C^{(l_1,l_2,l_3)}_{(m_1,m_2,m_3)}$. Specifically, this symbol denotes the weight of the product of component m_1 of the l_1-type factor with component m_2 of the l_2 type factor that goes into making the m_3 component of the outgoing l_3-type spherical tensor. Figure 15 has a visual illustration. $C^{(l_1,l_2,l_3)} \in \mathbb{R}^{(2l_1+1)\times(2l_2+1)\times(2l_3+1)}$ can be seen as a three-dimensional list of numbers.

Spherical harmonics and picking a basis for spherical tensors. Now that we have learned about some of the properties of spherical tensor types and how they behave under rotations and tensor product operations, let us turn to the question of choice of basis. This is crucial, since we always have to present these tensors through a list of numbers in the computer. To obtain spherical tensors, we use a family of functions that can be used to map a point on the unit sphere to a spherical tensor of a given degree l: the real-valued[35] spherical harmonics Y^l_m.

The real spherical harmonic $Y^l_m : S^2 \to \mathbb{R}$ takes a point on the 2-dimensional unit sphere S^2 and maps it to a number. For a given l there are $2l+1$ possible values of m, usually numbered by $-l, \ldots, l-1, l$. If we collect all components for a given l into $\vec{Y}^{(l)} = (Y^l_{-l}, \ldots, Y^l_{l-1}, Y^l_l) : S^2 \to \mathbb{R}^{2l+1}$, then the resulting $2l+1$ dimensional object will transform equivariantly as

$$\vec{Y}^{(l)}(\mathbf{R}\hat{v}) = \mathcal{D}^{(l)}(\mathbf{R})\vec{Y}^{(l)}(\hat{v}). \tag{33}$$

This is precisely the transformation behaviour of spherical tensors of degree l we saw in Equation (31), making $\vec{Y}^{(l)}(\hat{v})$ a spherical tensor. In other words, the functions Y^l_m for a given l provide a basis set of functions for the order l irrep of $SO(3)$.

Upon choosing a basis for each irrep l, the spherical harmonics are unique up to a sign and normalisation constant[36]. Let $\hat{v} = (x, y, z)$ be a point on the unit sphere $S^2 \subset \mathbb{R}^3$. Then, in the most common

[35]The physics community, most textbooks and internet resources tends to use the complex-valued spherical harmonics. The machine learning community on the other hand typically uses the real valued version for memory and computational reasons. While this makes reading about spherical harmonics confusing at times, the two formulations are related by a simple change of basis. See Geiger and Smidt [2022] for more details on this.

[36]See Geiger and Smidt [2022] for a proof. If we assume a usual Cartesian coordinate system, then we can pick a basis for each irrep l by choosing an arbitrary rotation axis and requiring the infinitesimal generator for

basis convention, the real spherical harmonics up to $l = 2$ are given by

$$Y_0^{(0)}(\hat{v}) = c_0 \tag{34}$$

$$Y_{-1}^{(1)}(\hat{v}) = c_1 y, \;\; Y_0^{(1)}(\hat{v}) = c_1 z, \;\; Y_1^{(1)}(\hat{v}) = c_1 x, \tag{35}$$

$$Y_{-2}^{(2)}(\hat{v}) = c_2 xy, \;\; Y_{-1}^{(2)}(\hat{v}) = c_2 yz, \;\; Y_0^{(2)}(\hat{v}) = \frac{c_2}{2\sqrt{3}}(2z^2 - x^2 - y^2), \tag{36}$$

$$Y_1^{(2)}(\hat{v}) = c_2 xz, \;\; Y_2^{(2)}(\hat{v}) = \frac{c_2}{2}(x^2 - y^2),$$

where c_0, c_1 and c_2 are constants that depend upon the choice of normalisation[37].

At this point you might become somewhat uneasy, because the spherical harmonics were only defined on the unit sphere S^2, so how can we use them to go from an arbitrary vector in $\mathbb{R}^3$ to a spherical tensor? It turns out that the spherical harmonics can be extended beyond the unit sphere[38] to the rest of $\mathbb{R}^3$. While this works, it can lead to numerical instabilities when working with vectors of non-unit length and requires careful normalisation. As an alternative, we could split the vector into a radial part $\|\vec{v}\|$ and a directional part $\hat{v} = \vec{v}/\|\vec{v}\|$ and treat them separately. This is the path most architectures based on spherical tensors choose.

Besides the nice fact that the spherical harmonics form a basis of the irreps of $SO(3)$ they also have many other nice properties. Because most are less directly relevant for building equivariant neural networks out of them, we only mention some that help gain an intuition about those functions in passing. The spherical harmonics are solutions to the Laplace equation $\Delta f(\hat{x}) = 0$ on the sphere and they form a complete and orthogonal basis of all smooth functions on the sphere. Therefore any smooth function on the sphere can be decomposed in a (possibly infinite) sum of spherical harmonics, weighted by different coefficients. Decomposing smooth functions into their spherical harmonics components is analogous to decomposing functions on $\mathbb{R}$ into their Fourier components, and the degree l of the spherical harmonics determines the frequency of the function. Low l functions change slowly as one walks along the sphere, while higher l functions change ever more quickly, as can also be seen in Figure 16.

Using spherical tensors to build equivariant architectures. Now that we have learned about spherical tensors, what if instead of Cartesian tensors, we used tensors based on these irreps, as our basis for message passing and accounting? Then we know that these in principle capture all possible types of transformations under rotational symmetries (if we go to high enough l), answering question (1) that we posed at the outset of this subsection. We could further make use of many nice theoretical properties of these irreps, that are known from the representation theory of $SO(3)$. This idea is in fact precisely what many of the models in the equivariant Geometric GNN literature follow, and it leads us to equivariant GNNs with spherical tensors.

rotations around that axis be diagonal. In most conventions people choose the z-axis, and this results in the equations for the spherical harmonics that we give below. In `e3nn` the y-axis is chosen as a default, such that the components of $\vec{Y}^{(1)}$ are proportional to (x, y, z).

[37] See this table on Wikipedia for a more extensive list including a common choice of normalisation constants.

[38] Indeed, we gave the equations in Equation (34) already in the generalised form. In the generalised form, the spherical harmonics Y_m^l can be chosen to be polynomials in the x, y, z coordinates. In fact, the generalised spherical harmonics of degree l are homogeneous harmonic polynomials of degree l, i.e. polynomials such that $\Delta p(\vec{x}) = 0$ and that each term contains a product of l variables. The spherical harmonics can then be interpreted as the restriction of homogeneous harmonic polynomials of degree l onto the sphere, giving them an algebraic rather than a representation theoretic definition as we have done above.

5.3 Equivariant GNNs with spherical tensors – Irreducible representations

Key idea

Spherical EGNNs not only restrict the set of learnable functions to equivariant ones, they also use *spherical* tensor components, which correspond to the irreducible representations of $SO(3)$, as their feature types. This choice comes naturally because of the intimate relationship of spherical tensors with the rotation group $SO(3)$, which gives spherical tensors many convenient properties.

In this section we focus on equivariant GNNs that operate with spherical tensors, the irreducible represntations and therefore the *natural* types of the rotation group $SO(3)$. As with Cartesian tensors, the crux to building equivariant networks is to diligently keep track of the types of different features and how they transform. The architectures in this category leverage the tools from representation theory that we have introduced in the previous section. In summary:

- The *spherical harmonics* are useful as a basis of irreps of degree l and to project vectors onto their spherical tensor components[39].
- The *Clebsch-Gordan coefficients* allow us to decompose a tensor products of spherical tensors into its spherical tensor components.
- The *Wigner D-matrices* represent rotations $\mathbf{R}$ for spherical tensors of degree l and enable us to rotate these tensors.

As we venture into equations for an example spherical EGNN, the formulas will inevitable become more complex and decorated with indices. When this happens, its important to remind yourself that the simple idea from Section 5 that underlies all this: We need to perform diligent accounting of tensor types, while adding learnable parameters where we can. The three tools above and all the indices are simply there to help us operate on and keep track of our accounting with spherical tensors.

Example spherical EGNNs in the literature. Before we dive into details, let us name a few recent examples of spherical EGNNs. The spherical EGNN family contains methods like Clebsch-Gordan Net [Kondor et al., 2018], TFN [Thomas et al., 2018], NeuquIP [Batzner et al., 2022], SEGNN [Brandstetter et al., 2021], MACE [Batatia et al., 2022b], Equiformer [Liao and Smidt, 2023] and many others, including networks using the concepts of steerability and equivariance introduced by Cohen and Welling [2016]. Most of these models build on the e3nn library [Geiger and Smidt, 2022], which makes it easy to work with spherical tensors by implementing the real spherical harmonics, Wigner D-matrices and tools to easily compute, decompose and parameterise tensor products for network layers.

5.3.1 An example spherical EGNN

Having heard of some of the spherical EGNNs in the literature, let us now construct a convolutional, spherical EGNN that will be similar to TFN [Thomas et al., 2018] to showcase the main ideas.

Node features and spherical tensor lists. Spherical EGNNs use tensors for node and sometimes also for edge features. We consider only node features, but the concepts extend straightforwardly to edge feature tensors. Let us denote the (hidden) features at node i by a list of geometric tensors of various l, from 0 to l_{max}[40]:

$$\vec{\mathbf{H}}_i^{(0:l_{\text{max}})} := \bigoplus_{l=0}^{l_{\text{max}}} \vec{\mathbf{H}}_i^{(l)} = \begin{bmatrix} \vec{\mathbf{H}}_i^{(0)} \\ \vdots \\ \vec{\mathbf{H}}_i^{(l_{\text{max}})} \end{bmatrix} \tag{37}$$

[39] In a functional or Fourier theory picture, when evaluating $Y_m^l(\hat{x}')$ we are projecting a delta-function $\delta(\vec{x} - \vec{x}')$, or a sum thereof in the case of a point cloud, onto its spherical degree l component. To better understand the functional perspective of these models, [Uhrin, 2021] and [Blum-Smith and Villar, 2022] are excellent references and written mathematically more precisely than our more loose and introductory discussion.

[40] For current architectures this is often $l_{\text{max}} = 1$ or 2. Also, if you have never encountered the symbol $:=$ before, it means "the left-hand-side is defined as ...".

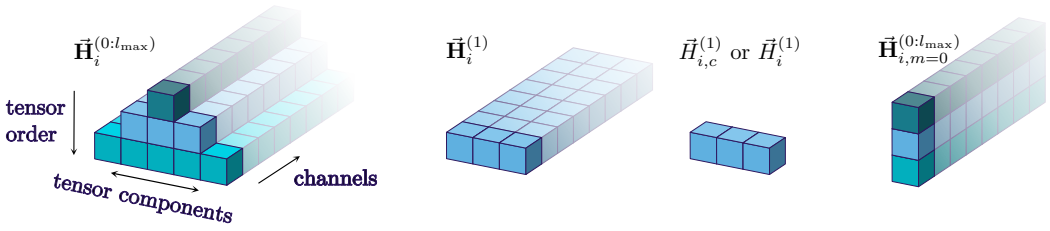


Figure 17: Visualisation of the spherical tensor list at node i and various of its slices. The 'lists' are drawn as pyramids to separate out the three axes.

It is important to pause and realise that the condensed notation $\vec{\mathbf{H}}_i^{(0:l_{\text{max}})}$ has 3 axes: It has a *channel-axis* (indicated by the boldface), because we might have more than a single type l tensor at that node. For simpler notation, we assume the same number of channels for all types l. $\vec{\mathbf{H}}_i^{(0:l_{\text{max}})}$ also has a *tensor-axis* (indicated by the $(0:l_{\text{max}})$ superscript), which reminds us that we are dealing with a concatenation of spherical tensors of various types. Finally, each tensor type l also a *tensor-component-axis*, which has $2l+1$ elements. Figure 17 illustrates this visually. When dealing with slices along these axes, we use the letters c, l and m for channel, tensor and tensor-component axes respectively. Also remember from Section 5.2 that $\vec{\mathbf{H}}^{(0)}$ are simply scalars, and $\vec{\mathbf{H}}^{(1)}$ are vectors[41].

Splitting the radial and angular parts. Next, let us briefly touch upon how to obtain spherical tensors from vectors, as most atomistic problems start from a point-cloud of various atom types in 3D space. How do we get spherical tensors from such a system? Spherical EGNNs split the vectors such as the displacement vector $\vec{x}_{ij}$ between node i and j into a radial $x_{ij} = \|\vec{x}_{ij}\|$ and a directional part $\hat{x}_{ij}$. The directional part is projected to spherical tensors with the spherical harmonics $\vec{Y}^{(l)}(\hat{x}_{ij})$. The radial part is a scalar, but because of its range $x_{ij} \in [0, \infty)$ it can quickly lead to diverging or vanishing scales when multiplied several times with itself, which happens as we stack multiple equivariant layers. To remedy this, we can apply various non-linear functions to x_{ij} to map the range $[0, \infty) \to [-1, 1]^{n_b}$, just as we did for distance-based invariant GNNs in Section 4. Here the number of basis[42] functions n_b as well as the choice of basis functions ψ_b are hyperparameters. A common choice for ψ_b are the radial basis functions centered around various length-scales of interest[43]. We would then carry one channel, or one set of channels, per basis function.

Equivariance and parameterising the tensor product. To allow learning to take place, we need to assign learnable parameters in our architecture that can be updated during training. We know how to do parameterise operations between scalars, as they behave just like normal numbers in standard neural networks. But, how can we parameterize the tensor product?

In Section 5.2 we learned that the tensor product of two spherical tensors $\vec{S}^{(l_1)} \otimes \vec{T}^{(l_2)}$ can be decomposed into $2\,\min(l_1, l_2) + 1$ spherical tensors of various degrees. The Clebsch-Gordan coefficients $C^{l_1,l_2,l_3}_{m_1,m_2,m_3}$ dictate how the components of the tensors need to be combined to retain equivariance, so we cannot add learnable weights to simple products of tensor components without breaking equivariance. Instead we can assign a learnable weight to each output spherical tensor in the decomposition, indexed by l_3, as these tensors are independently equivariant:

$$\vec{S}^{(l_1)} \underline{\otimes} \vec{T}^{(l_2)} = \bigoplus_{l_3=|l_1-l_2|}^{l_1+l_2} \underline{w}_{l_1,l_2,l_3}\, \vec{S}^{(l_1)} \otimes \vec{T}^{(l_2)}\big|_{(l_3)}. \tag{38}$$

[41] Possibly with axes interchanged and a different normalisation, depending on the definiton of the spherical harmonics that is used.

[42] This is not a basis in the mathematical sense, as the functions do not span the entire space. Usually, the 'basis' here is a subset of mathematical basis the space of (reasonable, e.g. square-integrable) functions over $[0, \infty)$, because there are infinitely many basis functions.

[43] Another choice are the Bessel functions for example.

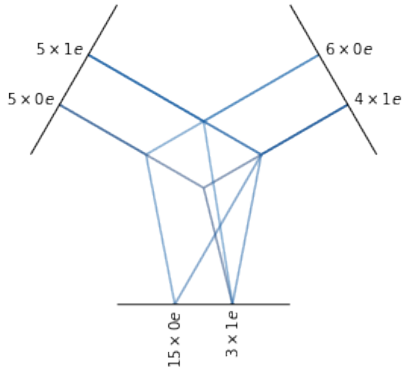


Figure 18: Fully connected tensor product two a spherical tensor lists. One with 5 $(l = 0)$ and $(l = 1)$ tensors and one with 6 $(l = 0)$ and 4 $(l = 1)$ tensors. At the output, we ask for 15 $(l = 0)$ and 3 $(l = 1)$ channels. How many paths are there? The answer is discussed in the main text. We ignore the parity indices `o` (odd) and `e` (even), as we do not consider reflections for simplicity. Figure taken from the `e3nn` documentation by [Geiger and Smidt, 2022].

In Equation (38) the $\underline{w}_{l_1,l_2,l_3}$ are learnable parameters, as indicated by the underline. There is one learnable parameter for each path $(l_1, l_2 \to l_3)$ in the Clebsch-Gordan decomposition and $\vec{S}^{(l_1)} \otimes \vec{T}^{(l_2)}\big|_{(l_3)}$ means extracting only the l_3 component of the tensor product. We can generalise this to the case of two lists of spherical tensors. Assuming a single channel each for the start, we get

$$\vec{S}^{(0:l_{\text{max}})} \underline{\otimes} \vec{T}^{(0:l_{\text{max}})} = \bigoplus_{l_3=0}^{l_{\text{max}}} \left(\sum_{\text{Paths}(l_1,l_2\to l_3)} \underline{w}_{l_1,l_2,l_3}\, \vec{S}^{(l_1)} \otimes \vec{T}^{(l_2)}\big|_{(l_3)} \right). \tag{39}$$

Here $\text{Paths}(l_1, l_2 \to l_3)$ refers to all valid paths from an l_1 and l_2 to and l_3 part. We will sometimes just abbreviate this by (l_1, l_2, l_3).

Let us consider a specific example to get a feel for this equation. Consider the case where we want to take the parameterised tensor product of a feature $\vec{H}^{(0:1)}$ with $\vec{Y}^{(0:1)}(\hat{x}_{ij})$, which may represent a spherical tensor list coming from an initial feature embedding or from a convolutional filter as we will see in a moment. Suppressing the dependence of $\vec{Y}^{(l)}(\hat{x}_{ij})$ on $\hat{x}_{ij}$ for a moment for readability

$$\begin{aligned} &\begin{bmatrix} \vec{Y}^{(0)} \\ \vec{Y}^{(1)} \end{bmatrix} \underline{\otimes} \begin{bmatrix} \vec{H}^{(0)} \\ \vec{H}^{(1)} \end{bmatrix} \\ &\quad = \begin{bmatrix} \underline{w}_{0,0,0}(\vec{Y}^{(0)} \otimes \vec{H}^{(0)})\big|_{(0)} + \underline{w}_{1,1,0}(\vec{Y}^{(1)} \otimes \vec{H}^{(1)})\big|_{(0)} \\ \underline{w}_{0,1,1}(\vec{Y}^{(0)} \otimes \vec{H}^{(1)})\big|_{(1)} + \underline{w}_{1,0,1}(\vec{Y}^{(1)} \otimes \vec{H}^{(0)})\big|_{(1)} + \underline{w}_{1,1,1}(\vec{Y}^{(1)} \otimes \vec{H}^{(1)})\big|_{(1)} \end{bmatrix}. \end{aligned} \tag{40}$$

If this equation looks somewhat intimidating, remember that a tensor product with an $l = 0$ term is just a verbose way of writing scalar multiplication, and that the tensor product of two $l = 1$ terms essentially amounts to the dot product or the cross-product if the output $l = 0$ or $l = 1$ respectively.

We can also make the weights in Equation (40) dependent on a continuous parameter, such as the distance x_{ij} between nodes i and j from which the tensors might originate. If we have chosen a radial basis functions, this would make simply the weights channel dependent. In such cases, we write $\underline{\otimes}_{(x_{ij})}$ and $\underline{w}_{l_1,l_2,l_3}(x_{ij})$.

The expression above was for a single channel only. When $\vec{\mathbf{S}}^{(0:l_{\text{max}})}$ and $\vec{\mathbf{T}}^{(0:l_{\text{max}})}$ are lists of multiple spherical tensors in different channels, we can perform the tensor product between each valid degree-channel pair and collect the results in different channels through different weighted combinations.

$$\vec{\mathbf{U}}^{(0:l_{\text{max}})} = \vec{\mathbf{S}}^{(0:l_{\text{max}})} \underline{\otimes} \vec{\mathbf{T}}^{(0:l_{\text{max}})}. \tag{41}$$

Note that we did not collect all possible degrees in the output, but restricted $\vec{U}$ to tensors of maximally degree l_{max}. All higher order spherical tensors that would be created by the tensor product are simply

dropped. In this multi-channel, multi-degree case, the same idea as before holds, but many more paths become available. In `e3nn`, such operations are adequately called *fully-connected tensor products* and [Geiger and Smidt, 2022] have a nice way of illustrating what is going on, which is illustrated in Figure 18. The illustration shows the fully connected tensor product two a spherical tensor lists: One with 5 $(l=0)$ and $(l=1)$ tensors and one with 6 $(l=0)$ and 4 $(l=1)$ tensors. At the output, we ask for 15 $(l=0)$ and 3 $(l=1)$ channels. How many paths are there? There are $5 \cdot 6$ paths from $(0,0 \to 0)$ and $5 \cdot 4$ from $(1,1 \to 0)$. We ask for 15 $(l=0)$ features in the output, so there are 750 paths ending in $(l=0)$. For $l=1$, the analogous computation over all valid paths $(0,1 \to 1)$, $(1,0 \to 1)$ and $(1,1 \to 1)$ gives 210 paths ending in $(l=1)$, so in total we can assign 960 learnable parameters to this fully-connected tensor product.

Building graph-convolutional filter from fully-connected tensor products We can now use the fully-connected tensor product, together with the projections of displacement vectors to spherical tensors from the previous two sections to define a filter function which we can use in the graph convolution to construct messages. Here is an example of a filter function, that is essentially[44] the filter function used in TFN,

$$\underline{\text{FILTER}}\left(\vec{x}_{ij}, \vec{\mathbf{H}}_j^{(0:l_{\text{max}})}\right) = \vec{Y}^{(0:l_{\text{max}})}(\hat{x}_{ij}) \underline{\otimes}_{(x_{ij})} \vec{\mathbf{H}}_j^{(0:l_{\text{max}})}. \tag{42}$$

We can then use this with a permutation invariant aggregation function of our choice to define the message step. For sum-aggregation,

$$\vec{\mathbf{M}}_i^{(0:l_{\text{max}})} = \sum_{j \in \mathcal{N}(i)} \underline{\text{FILTER}}\left(\vec{x}_{ij}, \vec{\mathbf{H}}_j^{(0:l_{\text{max}})}\right) \tag{43}$$

is a possible, equivariant message and aggregation definition.

Channel-mixing. The fully-connected tensor product gave us a way to parameterise multiplications of spherical tensor lists. Another simple way to add learnable parameters for a single spherical tensor list is to perform type-wise channel mixing. The type-wise channel mixing operation linearly combines the channels of the same type within a spherical tensor list:

$$\underline{\text{MIX}}\left(\vec{\mathbf{M}}_i^{(0:l_{\text{max}})}\right) = \bigoplus_{l=0}^{l_{\text{max}}} \underline{w}_{c'c}^{(l)} \vec{M}_{i,c}^{(l)} \tag{44}$$

Again, for simplicity we set the same number c of channels for all types l, but in practice we could, of course, have a different number of channels for each type.

Non-linearities and gating. Finally, we can apply component-wise non-linearities to our features or messages. For scalars, we can apply component-wise non-linearities just as normal. For higher degrees of l in a spherical tensor list $\vec{\mathbf{M}}_i^{(0:l_{\text{max}})}$, applying component-wise non-linearities looses equivariance. Instead, a common trick is to perform non-linear *gating*. This means that we apply a nonlinearity to a, possibly learnable, combination of scalars[45] in $\vec{\mathbf{M}}_i^{(0:l_{\text{max}})}$ and then use this as a weight to scale the tensors of a given degree l. In equations,

$$\underline{\text{GATED-NL}}\left(\vec{\mathbf{M}}_i^{(0:l_{\text{max}})}\right) = \eta_0(\vec{\mathbf{M}}_i^{(0)}) \oplus \left(\bigoplus_{l=1}^{l_{\text{max}}} \eta_l\left(\underline{\text{GATE}}\left(\vec{\mathbf{M}}_i^{(0)}\right)\right)\ \vec{\mathbf{M}}_i^{(l)}\right) \tag{45}$$

Here, η_l are non-linear functions and the above equation is understood to apply channel-wise and $\underline{\text{GATE}} : \mathbb{R} \to \mathbb{R}$ is an arbitrary function with parameters, such as an MLP, that is applied channel-wise.

Putting the pieces together. Finally let us put the pieces together and write out the equations for a full layer of our spherical EGNN.

[44] The subtle and not very important difference is that in the fully connected tensor product we allow different weights for all paths, while the original TFN ties together all weights with the same filter and input l (i.e. l_1 and l_2).

[45] This can be a simple MLP of the scalars $\vec{\mathbf{M}}_i^{(0)}$ for instance.

(a) PaiNN (b) TFN

Figure 19: **Equivariant message passing.** $\mathcal{G}$-equivariant layers such as PaiNN and TFN propagated geometric quantities such as vectors, relative positions, or tensors.

$$\underline{\text{UPD}}\left(\vec{\mathbf{H}}_i^{(0:l_{\text{max}})}, \vec{\mathbf{M}}_i^{(0:l_{\text{max}})}\right) = \vec{\mathbf{H}}_i^{(0:l_{\text{max}})} + \underline{\text{GATED-NL}}\left(\underline{\text{MIX}}\left(\vec{\mathbf{M}}_i^{(0:l_{\text{max}})}\right)\right) \tag{46}$$

The update function performs a residual update of the spherical tensor features at each node i by first performing the graph convolution with the learnable filter (Equation (43)) to construct and aggregate messages. Next, we mix the channels within a message and apply gated nonlinearities.

The updated node feature then becomes

$$\vec{\mathbf{H}}_i^{(0:l_{\text{max}})} \leftarrow \underline{\text{UPD}}\left(\vec{\mathbf{H}}_i^{(0:l_{\text{max}})}, \vec{\mathbf{M}}_i^{(0:l_{\text{max}})}\right). \tag{47}$$

This completes a single layer of our example spherical EGNN. As with the scalar-vector EGNN in **??**, we can then apply pooling and read out only the scalar channel at the output, in case we are interested in an invariant prediction. As an added bonus, we can now also read out higher equivariant types, in case we wanted to predict equivariant quantities such as an atomic systems dipole moment (an $(l = 1)$ quantity) or quadrupole moment (an $(l = 2)$ quantity) for instance.

Relations to Tensor Field Networks. Tensor Field Network [Thomas et al., 2018] is one of the earliest approaches proposing this type of $\mathcal{G}$-equivariant GNN with spherical tensors, and is very similar to the exemplary spherical EGNN we just built. It has the benefits of processing higher-order tensors directly and in an equivariant fashion, resulting in SO(3)-equivariant model predictions. Despite its well devised theoretical framework, TFN suffers from a few shortcomings. For instance, retaining all the non-zero tensor products $\underline{\otimes}$ up to degree l_{max} becomes computationally unfeasible as we scale l_{max} since it requires $O(l_{\text{max}}^3 C)$ multiplications. Therefore most e3nn networks are limited to the computation of $l_{\text{max}} = 1$ or 2 and $l_f \leq 2$ or 3 for the filter functions l. Notably, when restricting the tensor product to only scalars (up to $l = 0$), we obtain updates of the form similar to Equation (9). Similarly, when using only scalars and vectors (up to $l = 1$), we obtain updates of the form similar to Equation (16), Equation (17) and Equation (18).

5.3.2 Optimisations and improvements.

Several issues highlighted in TFNs have been tackled by subsequent approaches. Below we give a brief selection of a few papers that use interesting ideas for optimisations and improvements. We only give the key ideas, and refer to the cited papers for details.

Fuchs et al. [2020] proposed SE(3)-Transformers, which adapt the TFN framework to use an equivariant variant of the self-attention operation during aggregation and operate on large point clouds while preserving equivariance. SEGNN [Brandstetter et al., 2022] proposes equivariant non-linear convolutions by introducing steerable MLPs that transform equivariant representations. SEGNN follows a standard MPNN framework and uses steerable MLPs for the message as well as update steps, essentially providing a recipe for for building equivariant MPNNs with Spherical tensor features. The Equiformer model [Liao and Smidt, 2023] successfully combines these ideas by interleaving equivariant self-attention aggregation with equivariant non-linear updates into Transformer-style blocks.

MACE [Batatia et al., 2022b] attempts to incorporate many-body interaction terms[46] by relying on a clever factorization of higher-order terms into products of two-body representation, which builds

[46] Many-body effects refer to the collective behaviour of a large number of interacting constituents. They are needed for an accurate description of both the structure and dynamics of large chemical systems.

on the popular Atomic Cluster Expansion (ACE) formalism [Drautz, 2019]. The key idea here is to exchange summation and multiplication[47] to reduce the number of multiplication operations: As a simple example without tensors, think of the expression $(a+b)^2$. This contains the terms a^2, ab, ba, b^2 but required us to perform only one multiplication[48]. MACE essentially introduces an efficient algorithm to compute a parameterised tensor-product of expressions of the form $\underline{\bigotimes}_{b=1}^{B}(\sum_{i=1}^{n_i}\vec{A}_i^{(l)})$. The same idea is also used in Allegro [Musaelian et al., 2022], but without message-passing to allow easier parallelisation of large material simulation systems across multiple GPUs.

eSCN [Passaro and Zitnick, 2023] tackles the computational scalability issue by reducing the SO(3) equivariant convolutions to mathematically equivalent convolutions in $SO(2)$, making the tensor product easier to compute. This is achieved by aligning the node embeddings' primary axis with the edge vector[49], which reduces the rotational symmetry to rotations around that axis, making the problem effectively 2 dimensional. While coming at the cost of two extra Wigner D-matrix rotations to align and unalign the spherical tensors to the given axis, this trick effectively sparsifies the Clebsch-Gordan coefficients and thereby leads to computational speedups for $l > 1$.

[47] This is referred to as *density-trick* in the physics-ML literature and comes from [Drautz, 2019].

[48] The cost of this is that the coefficients of these terms are coupled.

[49] This trick sometimes called as *point-and-shoot* in computational electromagnetics.

6 Unconstrained Geometric GNNs

Key idea

Unlike other methods, architecturally unconstrained GNNs do not 'bake' symmetries into their architecture, leading to greater flexibility in model design and more diverse optimization paths. Instead, they let the model learn approximate symmetries, encourage approximate symmetries through loss terms or data augmentation, or enforce symmetries through alternate strategies such as (global or local) canonicalization.

Overview. We have seen that previous Geometric GNN families, by design, confine the set of learnable functions to equivariant ones, aligning with the goal of accurately modeling equivariance. However, this constraint raises concern about potential impediments to the neural network optimization process. The idea is that a more unconstrained model, i.e. not bound to equivariance, may traverse more diverse optimization paths (ultimately converging to equivariant functions). As motivated by Duval et al. [2023], this increased flexibility could empower the model to capture the intricacies of the data more effectively. In contrast, the strict adherence to equivariant constraints may limit the optimization paths available, or change the optimization landscape in a way that hinders our algorithms, for example, by altering the conditioning or prevalence of local minima[50]. This raises the question of whether the benefits of enforcing Euclidean equivariance as an inductive bias truly offset a potential reduction in optimization diversity within constrained learning spaces.

A useful parallel may be Deep Learning regularization, where equivariant models, akin to a regularization paradigm, enforce specific constraints on the functional learning space. While these constraints are intended to promote desirable properties, such as provable or guaranteed equivariance, unconstrained GNNs are concerned by the potential downside — the risk that such constraints may overly regularize the model, hindering its capacity to fully express the intricacies of the data. A few approaches [Wang et al., 2022b, Hu et al., 2021, Zitnick et al., 2022, Duval et al., 2023] make this argument, proposing a data-driven view of symmetries as opposed to the usual model-based view.

Data augmentation and soft constraints. To explore the potential of this idea, let us consider image classification. While most modern Convolutional Neural Networks and Visual Transformers [Dosovitskiy et al., 2021] are not scale- or rotation-equivariant [Weiler et al., 2018], they can still learn approximate equivariance through rotation and scale diversity in the training data. This adaptability extends to invariant prediction tasks on 3D images, where Gerken et al. [2022] showed that data augmentation methods match invariant networks in accuracy with significantly lower computational cost at inference time. While the results are less successful for equivariant tasks, their work suggest that leveraging unconstrained GNNs holds promise as an effective approach.

ForceNet [Hu et al., 2021] was one of the first unconstrained GNN architectures to explore data augmentation as a soft symmetry constraint. This adaptation of Sanchez-Gonzalez et al. [2020] to 3D atomic systems proposes to implicitly learn symmetries via data augmentation procedures such as adding diverse rotations of the same geometric graph to the training data. Despite showing promising results, ForceNet's accuracy vs. scalability gains were not significant enough to constitute a true Pareto optimal improvement.

Interestingly, architectures from the parallel field of 3D point cloud processing for natural scenes and shapes [Guo et al., 2020] also rely heavily on data augmentation. Models are generally trained with random rotations, translations, and crops of the point cloud, along with corruptions to its categorical values [Qi et al., 2017]. Additional loss terms can also be added as regularisation or soft constraints that encourage the model to preserve symmetries without strictly enforcing it.

Another Geometric GNN with soft constraints is the SCN model [Zitnick et al., 2022], an equivariant architecture that initially utilizes spherical harmonics to represent channel embeddings with explicit orientation information. However, it later relaxes the equivariant constraint to enable more expressive non-linear transformations. Specifically, it projects spherical channels onto a grid and conducts pointwise convolutions followed by a non-linearity, facilitating intricate mixing of various degrees of

[50]The two arguments can converge if the impairment corresponds to a spectrum distribution of the eigenvalues of the Hessian around the fixed points, significantly limiting escape possibilities (e.g., negative eigenvalues) around these fixed points.

spherical harmonics at the expense of strict equivariance. When released, it reached state-of-the-art performance on the Open Catalyst dataset OC20 [Chanussot et al., 2021].

Globally and locally canonicalized GNNs. Iterating on the data augmentation and soft constraint perspective, FAENet [Duval et al., 2023] proposes a model architecture completely free of all design constraints, which directly processes atom relative positions $\vec{x}_{ij}$ using non-linear functions (MLPs):

$$\boldsymbol{s}_i^{(t+1)} = \underline{f_1}\left(\boldsymbol{s}_i^{(t)}, \sum_{j \in \mathcal{N}_i} \boldsymbol{s}_j^{(t)} \odot \underline{f_2}\big(\vec{x}_{ij}, \boldsymbol{s}_i^{(t)}, \boldsymbol{s}_j^{(t)}\big)\right), \tag{48}$$

where $\underline{f_2}(\cdot)$ is the convolution filter encoding the 3D geometric information (MLPs with swish activation) and $\underline{f_1}(\cdot)$ the message passing update function. This inner working is "possible" because FAENet outsources equivariance to the data representation, allowing the model to process geometric information with complete freedom. Building upon Frame Averaging [Puny et al., 2022], FAENet projects data points into a canonical space via Principal Components Analysis (PCA), offering a unique representation of all Euclidean transformations. This canonicalization, illustrated in Figure 20, enables to rigorously (or empirically) preserve symmetries while retaining maximal expressiveness.

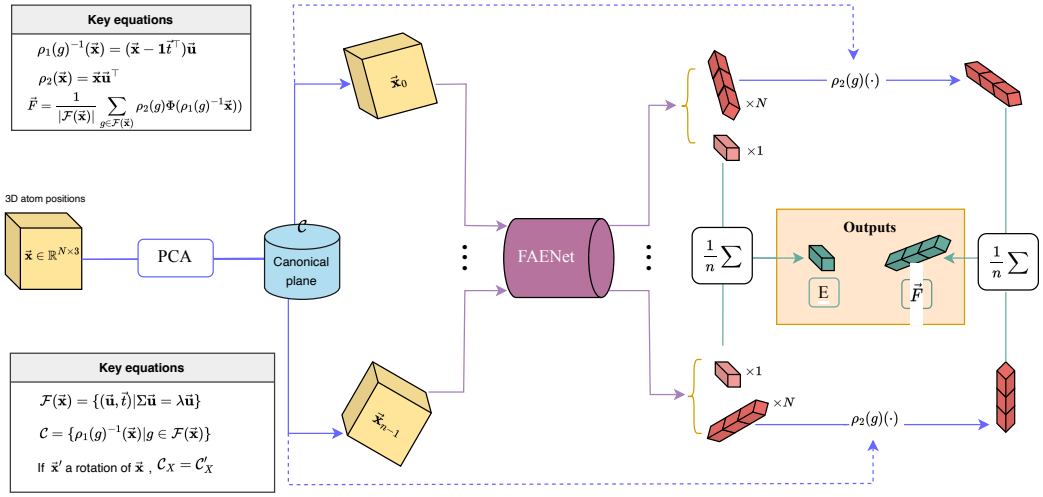


Figure 20: (Stochastic) Frame Averaging pipeline utilized by FAENet to enforce explicit (or approximate) equivariance. Input data is mapped to a global reference frame, offering a unique representation of all Euclidean transformations. It is then passed to the Geometric GNN model called FAENet before being aggregated to yield invariant or equivariant predictions.

Let us briefly highlight the mechanisms of FAENet. We first need to compute the centroid of an atomic graph with n nodes $\vec{t} = \frac{1}{n}\vec{\mathbf{x}}^\top \mathbf{1} \in \mathbb{R}^3$ and the centred covariance matrix $\Sigma = (\vec{\mathbf{x}} - \mathbf{1}\vec{t})^\top(\vec{\mathbf{x}} - \mathbf{1}\vec{t})$. Solving for $(\vec{u}, \lambda) : \Sigma\vec{u} = \lambda\vec{u}$ and assuming distinct eigenvalues $\lambda_1 > \lambda_2 > \lambda_3$ we can then create the *frame* $\mathcal{F}(\vec{\mathbf{x}}) = \big\{(\vec{\mathbf{u}}, \vec{t}) \mid \vec{\mathbf{u}} = [\pm\vec{u}_1, \pm\vec{u}_2, \pm\vec{u}_3]\big\}$, For any given function (including GNNs) $\underline{f}$, $\langle \underline{f} \rangle_{\mathcal{F}}(\vec{\mathbf{x}})$ is $\mathcal{G}$-equivariant or invariant depending on ρ_2:

$$\langle \underline{f} \rangle_{\mathcal{F}}(\vec{\mathbf{x}}) := \frac{1}{|\mathcal{F}(\vec{\mathbf{x}})|} \sum_{g \in \mathcal{F}(\vec{\mathbf{x}})} \rho_2(g)\underline{f}(\rho_1^{-1}(g)\vec{\mathbf{x}}) \tag{49}$$

$$\rho_1(g)\vec{\mathbf{x}} := \vec{\mathbf{x}}\vec{\mathbf{u}}^\top + \mathbf{1}\vec{t}^\top \tag{50}$$

$$\rho_2(g)\vec{\mathbf{y}} := \begin{cases} \vec{\mathbf{y}}\vec{\mathbf{u}}^\top & \text{for equivariant predictions } \vec{\mathbf{y}} \in \mathbb{R}^{n\times 3} \\ \vec{\mathbf{y}} & \text{for invariant predictions} \end{cases} \tag{51}$$

The intuition here is that one can choose appropriate representatives of a group, act on the data (ρ_1), infer with a function ($\underline{f}$), act back to the original space (ρ_2) and finally average those predictions to produce invariant or equivariant outputs. One downside of this approach is that it requires $|\mathcal{F}(\vec{\mathbf{x}})|$

inferences (8 for the group $\mathrm{E}(3)$, 4 for $\mathrm{SE}(3)$) through $\underline{f}$, which can be an expensive (graph-)neural network. To mitigate this effect, FAENet uses Stochastic Frame-Averaging (SFA) where members of a frame are *sampled* instead of being exhaustively averaged over. In different terms, we apply the function $\underline{f}$ on a single frame element, selected randomly at inference and at each training epoch, instead of all members. This approach does not *guarantee* invariance/equivariance but allows the model to *learn* it. In this perspective, it is akin to a geometry-informed data augmentation procedure over a restricted set of $|\mathcal{F}(\vec{\mathbf{x}})|$ frames. It is interesting to note how the frame averaging paradigm relates to existing families of invariant and equivariant GNNs, defined in Section 4 and Section 5. FAENet is similar to invariant GNNs as it treats geometric information as scalar quantities and uses a standard message passing scheme (i.e. unconstrained) to update node representations non-linearly. But unlike them, it can leverage relative atom positions directly instead of a pre-defined scalarisation of geometric information. Besides, by avoiding the application of symmetry-preserving equivariant operations on internal representations, FAENet 'breaks' the expressivity limits of invariant and equivariant GNNs and trivially distinguishes between all known counterexamples proposed by Pozdnyakov et al. [2020] and Joshi et al. [2023a]. Since it aggregates relative atom positions using any non-linear function, it can be considered as a *many-body* approach that iteratively incorporates information coming from higher-order neighborhoods.

Similar to FAENet, Pozdnyakov and Ceriotti [2023] proposed an unconstrained Geometric Transformer architecture based on an alternative to the frame averaging protocol, termed Equivariant Coordinate System Ensemble, which defines local coordinate systems at each atom and averages over the predictions of a non-equivariant network for each coordinate system. Similarly, Kaba et al. [2023] proposed to learn the canonicalization function using a neural network instead of designing it by hand like in Frame Averaging.

Summary. Geometric GNNs not explicitly enforcing symmetries into the model architecture are an emerging and under-explored line of work. Relaxing symmetry constraints in the model design allows for the implementation of more expressive architectures allowing for an interesting accuracy vs. scalability trade-off at inference time. However, the partial or approximate enforcement of symmetries may come at the cost of precision and physical violations in the model's prediction. For instance, a recent benchmark by Fu et al. [2023] found ForceNet-based molecular simulators to be unstable over long timesteps and for large systems. Thus, understanding when to impose strict equivariance in the model as opposed to letting the model learn flexibly is an open question. Similarly, looking at new ways to implicitly learn exact symmetries from data [Dehmamy et al., 2021] could lead to significant improvements in the field. The discussion is continued in Section 8.

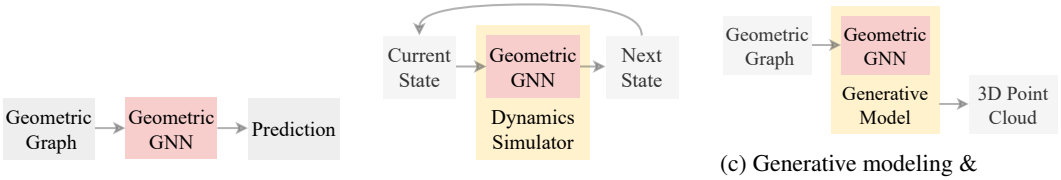
Opinion

Unconstrained GNNs do not state or show that building symmetries into neural networks is not worthwhile. They simply tackle the interplay between the space of functions that a model can learn and the ease of optimisation of ML algorithms. Whether we should rigorously enforce symmetries or not is an open question that probably depends on the application and scale of data available. Here, SE(3) is a small group in terms of the degrees of freedom (a rigid transformation is uniquely determined by 6 parameters), so not rigorously enforcing equivariance may be tolerable and, perhaps, desirable from the optimisation perspective. However, in other cases, incorporating such inductive bias into the model will reduce sample complexity and improve generalization. Finally, note that unconstrained GNNs still enforce node permutation symmetry via permutation equivariant message passing.

7 Applications

Overview. Geometric GNNs have shown promising results across a range of application areas[51], spanning structural biology, biochemistry, and materials science [Zhang et al., 2023, Wang et al., 2023a]. These applications can be broadly categorised in Figure 21 as (1) property prediction, (2) molecular dynamics simulations, (3) generative modeling, and (4) structure prediction. This section concisely describes the utility of Geometric GNNs for each task, followed by a detailed discussion of relevant datasets.

7.1 Tasks



(a) Property prediction (b) Dynamics simulation (c) Generative modeling & structure prediction

Figure 21: **Geometric GNN applications.** Representations learnt by Geometric GNNs are used as part of task-specific pipelines in property prediction, molecular dynamics simulation, generative modeling, and structure prediction.

7.1.1 Property Prediction

The most common application of Geometric GNNs is to predict functional and physical properties of geometric graphs, ranging from quantum mechanical properties of molecules and materials [Gasteiger et al., 2021] to the outcomes of experimental assays in drug discovery [Stokes et al., 2020].

Conventional approaches to determining properties of 3D atomic systems are known to be resource-intensive, both for simulation and experimental approaches. For instance, quantum mechanical properties are computed using simulation techniques such as Density Functional Theory (DFT)[❖] [Kohn et al., 1996] which often require supercomputers. Similarly, obtaining experimental data usually requires specialized equipment tailored to a particular application, with different requirements for biological systems, chemical systems and complex materials. Geometric GNNs have emerged as a fast alternative by learning to predict properties of new systems from large annotated datasets generated by historical simulations or experiments. This expedites the screening of large libraries of molecules and materials in order to discover new candidates with specific desired properties. Notable use cases of Geometric GNNs for property predictions include:

1. Drug discovery – where Geometric GNNs help identify promising drug candidates by modeling relevant properties of potential drugs, including proteins and small molecules interacting with the human body [Huang et al., 2022]. This type of accelerated screening can help practitioners understand a drug's potential efficacy and safety profiles, and prioritize compounds for further experimental testing.
2. Material discovery – where, by accurately predicting properties such as energy, stability, bandgap and conductivity, Geometric GNNs can help researchers optimize material compositions and structures for specific applications. Similar to drug discovery, this can help researchers discover new materials for a diversity of applications [Chanussot et al., 2021, Lee et al., 2023b].
3. Environmental impact assessment – where they help understand the behavior of substances in the environment, evaluating their toxicity, persistence, and bioaccumulation potential. This enables to make informed decisions regarding their use and disposal [Feinstein et al., 2021, Epa, 2020].

[51] In this survey, we purposefully focus on applications of Geometric GNNs to atomic systems. We refer the reader to Xiao et al. [2020], Cao et al. [2020], Gerken et al. [2023] for other perspectives and applications of Geometric Deep Learning, including computer vision.

4. Process optimization – where, by accurately predicting reactivity, selectivity, and solubility, Geometric GNNs can help researchers optimize reaction conditions, design efficient separation processes, and minimize waste generation leading to more sustainable and cost-effective manufacturing processes [Kearnes et al., 2021, Mercado et al., 2023].

Formally, the property prediction task is formulated as a regression or classification problem, most often targeted to the full graph (e.g. energy, band gaps, thermal conductivity, stability) although it can also apply at a node level (e.g. atomic charges). A Geometric GNN model is trained to minimize the loss function $\sum_i \mathcal{L}(\texttt{GNN}\,(\,\mathcal{G}_i)\,,\,y_i\,)$ over a dataset of n 3D geometric graphs $\mathcal{G}_i$, paired with their corresponding property values y_i. The objective is to optimize the model's ability to predict a specific property by minimizing the difference between the predicted value and the ground truth property value. Commonly used loss functions include mean absolute error (MAE) for regression tasks or cross-entropy for classification tasks. Geometric GNNs are very effective for predicting materials and molecules' properties due to their ability to leverage atomic and geometric attributes while respecting data symmetries.

7.1.2 Interatomic Potentials for Molecular Dynamics Simulation

Molecular Dynamics (MD) simulations, also known as atomistic simulations, predict how every atom in a 3D system will move over time based on a general model of the physics governing interatomic interactions [Karplus and McCammon, 2002]. MD simulations are used to model a diversity of materials, including periodic crystal structures, molecules and large-scale protein structures, providing useful insights into various properties and behaviours. MD simulations also represent atomistic systems as a set of N atoms with position vectors $\vec{\mathbf{x}}$ and atomic types $\mathbf{S}$ and aim to solve Newton's equation of motion for all atoms in the system. As such, the system behaviour is governed by the potential energy U, which depends on the interaction of the various atoms in the system. Concretely, U is the summation of of one-body $U(\vec{x}_i)$, two-body $U(\vec{x}_i, \vec{x}_j)$, three-body $U(\vec{x}_i, \vec{x}_j, \vec{x}_k)$, up to N-body interaction terms:

$$U = \sum_{i=1}^{N} U(\vec{x}_i) + \sum_{\substack{i,j=1;\\ i\neq j}}^{N} U(\vec{x}_i, \vec{x}_j) + \sum_{\substack{i,j,k=1;\\ i\neq j\neq k}}^{N} U(\vec{x}_i, \vec{x}_j, \vec{x}_k) + \cdots \tag{52}$$

MD simulations also have the capability to include desired thermodynamic conditions, such as temperature and pressure, by including relevant thermostat and barostat settings found in common MD simulation packages [Thompson et al., 2022, Van Der Spoel et al., 2005, Larsen et al., 2017]. Once the potential energy U of the system is known, one can obtain the forces on each atom using differentiation leading to forces on each atom: $\vec{F}_i = -\partial U/\partial \vec{x}_i$. The acceleration of each atom is then calculated by dividing the forces by the atomic mass m_i. Next, the updated atomic positions are computed by numerically integrating the equations of motion enabling one to study the dynamics of atomic systems.

Geometric GNNs can be infused into MD simulations by using GNN-based interatomic potentials, meaning that $U = \Phi^1(\vec{\mathbf{x}}, \mathbf{S})$ with the forces being determined by differentiation of the GNN-based potential [Batzner et al., 2022, Batatia et al., 2022b,a]. Alternatively, one can perform direct force prediction for each atom with another GNN (head): $\vec{\mathbf{F}} = \Phi^2(\vec{\mathbf{x}}, \mathbf{S})$ [Gasteiger et al., 2021]. Note that, in both cases, we minimize a loss function that measures the discrepancy (e.g. MAE) between the predicted and the ground truth values. A thorough analysis of the advantages and shortcomings of MD potentials is available in Bihani et al. [2023] and Fu et al. [2023]. These studies suggest that MAE-based regression training methods for GNN-based MD simulations are not sufficient to guarantee MD simulation stability likely to due distribution shift. One promising alternative may be training on the dynamics of the trajectory data itself as explored by Bhattoo et al. [2023]. Nonetheless, work by Schaarschmidt et al. [2022] suggests that Geometric GNNs can avoid local minima that classical quantum mechanical simulators often get trapped in, showcasing that further research is needed to better understand the capabilities and limitations of using Geometric GNNs in MD simulations.

7.1.3 Generative Modeling and Design

Geometric GNNs are emerging as a powerful data encoder tool to facilitate the synthesis of new complex geometric structures including molecules, crystals, and 3D objects. Their utilization in the generation pipeline is quite recent and does not seem to have reached its full potential yet: most existing generative approaches still use traditional GNN models [De Cao and Kipf, 2018, Ragoza et al., 2020]. To be more specific about their utilisation within the generative pipeline, let's first distinguish two main categories of generative methods where they are (or could be) used:

Full-graph methods generate all attributes at once (i.e. scalar features, geometric features and adjacency matrix), building on top of Variational Auto Encoders (VAE) [Ren et al., 2022, Pakornchote et al., 2023], Generative Adversarial Networks (GANs) [Nouira et al., 2018, Long et al., 2021], Normalizing Flows [Satorras et al., 2021a, Ahmad and Cai, 2022] and Diffusion models [Zheng et al., 2023, Xu et al., 2022]. In general, these methods are trained to reconstruct the training data from a latent distribution by maximizing its likelihood, or by minimizing the discrepancy between generated samples and the real data distribution. Once trained, they can generate new instances by sampling from the learned latent space, transforming latent representations into valid geometric graphs. But what about Geometric GNNs? They are natural data encoders. Since they preserve data symmetries, they are useful to preserve the likelihood of generated samples when the 3D atomic system is rotated, reflected or translated. For VAE, they are used at train time to create latent representations of the input geometric graph before using an MLP or a GCN to (re-)produce a 3D graph [Xie et al., 2022]. Regarding diffusion models, which is a very active area of research, Hoogeboom et al. [2022] employs an equivariant network jointly operating on continuous (atom coordinates) and categorical features (atom types) in the denoising phase. Finally, GAN based methods could use a Geometric GNN discriminator to predict if the input geometric graph comes from the generator or from the training dataset.

Iterative methods sequentially generate geometric graphs, selecting actions at each step. G-Schnet and G-SphereNet [Gebauer et al., 2019, Luo and Ji, 2022], for example, are auto-regressive models that generate 3D molecules by performing atom-by-atom completion using invariant Geometric GNNs [Schütt et al., 2017, Liu et al., 2021] on the current structure. Alternatively, the generation of 3D atomic systems can be decomposed into compositional objects, constructed step by step using Reinforcement Learning (RL). In this case, Geometric GNNs are used to steer the action choices of the RL algorithm, acting as a reward function that optimises the generation of materials/molecules with targeted desirable properties. AI4Science et al. [2023] employs a GFlowNet [Bengio et al., 2021] to sequentially sample 3D crystals through the selection of the composition, space group and lattice parameters, with any property prediction model as an objective function. This domain-inspired approach enables the flexible incorporation of physical and geometrical constraints and allows to search efficiently through the entire material space.

Overall, applying Geometric GNNs to generative modeling has significant implications for accelerating progress in fields like protein-conditioned molecule generation [Corso et al., 2023, Schneuing et al., 2022], de novo protein design [Ingraham et al., 2019, Dauparas et al., 2022], and electrocatalyst discovery [Zitnick et al., 2020, Kolluru et al., 2022].

7.1.4 Structure Prediction

Protein Structure Prediction: For biomolecules, the structure prediction task entails predicting or generating a plausible set of 3D coordinates of a biomolecule given its 1D sequence representation [AlQuraishi, 2021] (for proteins: the sequence of amino acid residues, for nucleic acids: the sequence of nucleotides). Geometric GNNs play a central role in modern structure prediction systems like AlphaFold [Jumper et al., 2021] and RosettaFold [Baek et al., 2021, 2022]. At a high level, structure prediction systems consist of two modules applied sequentially:

1. The sequence module which constructs and updates latent representations of each residue, generally via a standard Transformer [Lin et al., 2023] or a specialised variant for processing multiple sequence alignments [Rao et al., 2021, Jumper et al., 2021] to capture evolutionary relationships among homologous sequences.

2. The structure module which initialises a fully connected graph with nodes representing the 3D positions of residues. Node representations are then updated via a Geometric GNN with

invariant [Jumper et al., 2021] or equivariant message passing [Baek et al., 2021, Lee et al., 2023a] among all the residues to iteratively refine the predicted 3D structure.

Models are trained via minimizing a loss function that quantifies the difference between the predicted structure and the ground truth 3D structure. The training data usually consists of experimentally determined 3D structures from the Protein Data Bank, so models essentially aim to find the most energetically favorable configuration among multiple possible conformational states [Lane, 2023].

Note that, although closely related to generative modeling, structure prediction involves predicting only the 3D positions of a given input sequence. On the other hand, generative modeling involves learning the distribution of a dataset of molecules and materials, and sampling new systems from the underlying distribution. Interestingly, protein structure prediction models can be repurposed as generative models for protein design [Watson et al., 2023]. This line of research has historical roots in physics-based approaches which attempted to build energy functions grounded in a biophysical understanding of protein folding [Alford et al., 2017].

Molecular Conformer Prediction: In the case of molecular conformer prediction one aims to predict the 3D atomic positions from a molecular formula, usually given a molecular string representation [Weininger, 1988, Krenn et al., 2020, Cheng et al., 2023b]. Some of the most successful methods rely on predicting torsion angles from a graph representation constructed from the molecular string representation where Geometric GNNs serve as an encoder for learning effective representations [Xu et al., 2022].

Crystal Structure Prediction: Similar to molecular structure prediction, crystal structure prediction involves the prediction of 3D crystal structures from two-dimensional crystal compositions. Early work by Chen and Ong [2022] and Choudhary and DeCost [2021] applied Geometric GNNs specifically designed for predicting 3D crystal structures. Using a different approach, Jiao et al. [2023] applied an equivariant diffusion to predict 3D crystal structures from atomic compositions, taking into account the periodic structure of crystals as well as relevant symmetries. A detailed review of various ML methods, including Geometric GNNs, for crystal structure prediction, is given in Riebesell et al. [2023].

Structure Prediction from Experimental Data: An emerging application of geometric GNNs is in hybrid experimental and computational pipelines for molecular structure determination. Notable examples include CryoEM protein structure determination [Jamali et al., 2023] and NMR chemical shift prediction [Guan et al., 2021, Yang et al., 2021]. Along similar lines, Cheng et al. [2023a] explored the use of equivariant diffusion models to predict 3D molecular structures based on incomplete information from real-world characterization instruments.

Opinion

It is interesting to note the contrast in how the input geometric graph is defined across the four tasks. Property prediction and dynamics simulation tend to use **local radial cutoff** graphs. This is presumably due to locality being a strong inductive bias, e.g. quantum mechanical properties or forces of a system are unlikely to be influenced by long-range interactions. On the other hand, generative modeling and structure prediction require models to develop globally coherent representations and generally necessitate operating on **fully connected** graphs. An interesting exception to this observation is the random long-range graph construction scheme in Chroma [Ingraham et al., 2022], a generative model for protein design which was validated in the wet lab.

7.2 Datasets

In the realm of Geometric GNNs, the backbone of success lies in the availability and quality of the data. With numerous datasets proposed to date, it can be challenging to navigate through the landscape of options. Although it is not the primary focus on the paper, we provide a selected list of existing datasets for Geometric GNNs (see Table 1). A more exhaustive list is accessible on a dedicated GitHub repository, which we hope the community will keep up to date.

Our objective in this section goes beyond mere enumeration of available datasets. Since data construction is essential to improve model performance and to allow for true progress in subsequent years, we also discuss promising data directions for the field. By addressing these considerations, we aspire to foster the growth of the Geometric GNN community and empower researchers to build more powerful and robust models. Overall, we encourage practitioners to utilize (and construct) benchmark repositories containing several task-specific datasets, similarly to the Open Catalyst Project [Chanussot et al., 2021, Tran et al., 2023], Matbench [Dunn et al., 2020] and the Open MatSci ML Toolkit [Lee et al., 2023b, Miret et al., 2023] for materials research as well as to the Therapeutic Data Commons [Huang et al., 2022] for drug discovery; all of which provide valuable ways to understand GNN model performance. Here are a few reasons why;

1. **Splits and Leaderboards**. These benchmarks are endowed with thorough and transparent evaluation protocols, including carefully defined train/val/test splits, a visible leaderboard with fixed evaluation metrics and some open-source guides describing how to use each dataset (with some baseline methods implemented). This enables easy utilisation as well as fair evaluation and comparison across methods, which is essential. Besides, the validation datasets contain both In-Domain (ID) and Out-of-Domain (OOD) split, allowing the assessment of model generalisation.
2. **Domain experts**. These datasets are constructed as the fruit of a collaboration between domain experts and the machine learning community, bridging the knowledge gap between scientific domains (e.g. physics, chemistry, materials science, biology) and machine learning communities. This ensures that ML tasks are consistent with underlying practical applications and that the available datasets account for the subtleties of the application domain. As the datasets continue to mature and the underlying problems become more and more complex, the range of domain experts should concurrently expand to include representation from government and corporations in addition to academic researchers.
3. **Continual dataset updates**. These datasets have shown regular updates and expansion through the collection or generation of new data, such as when enough new samples are obtained using DFT or experimental methods. The continual dataset expansions allow ML models to solve more complex and relevant challenges motivated by the underlying application. Ultimately the goal is to create more robust ML models, including both highly specialized models for a given application as well as general ML models that can tackle a wide range of modeling problems.
4. **Diversity of tasks for generalized learning**. In well-maintained benchmarks, it is often possible to pre-train a model on a specific task and fine-tune it on another. As datasets grow, a greater diversity of tasks enables the community to build towards generalist foundation models for scientific applications in 3D atomic systems. Given the success of foundation models in the natural language and vision domains, these developments have the potential to unlock tremendous future research opportunities.

Software and libraries. On a slightly different note, one crucial factor for the applicability and development of Geometric GNNs to any dataset is the existence of handy code bases. We provide a list of useful repositories on github, and encourage the community to maintain it up-to-date.

Opinion

Navigating datasets of 3D atomic systems is currently difficult due to major differences at every level: ground truth simulations, subsample selection, splits, benchmarks vs single dataset, etc. We advocate for additional work providing structure in this domain.

Task	Dataset	Benchmark	# Samples	# Tasks	Domain	Split	Metric	Date	Source
PP	Open MatSci ML Toolkit	Open MatSci ML Toolkit	1.5M	Varied	Crystal Structures	Stratified/Random	MAE	2023*	[Paper][github]
	QM7-b	MoleculeNet	7k	14	Molecule	Random	MAE	2014	[GDB-13][info][info]
	QM9	MoleculeNet	130-134k	12-19	Molecule	Random	MAE	2012	[GDB-17][info][PyG]
	PDBbind	MoleculeNet	5k-13k	1	Protein-ligand	Time	RMSE	2004*	[PDB][info]
	Alchemy	Alchemy Contest	120k	12	Molecule	Stratified/Size†	MAE	2019	[GDB MedChem] [link][HF]
	Matbench	Mathbench	1k-100k	13	Crystal Structures	StratifiedKFold†	MAE, ROC-AUC	2019	[Materials Project][data]
	Atom3D	Atom3D	Varied	8	Mol., RNA, Prot.	Varied	Various	2021	[Varied][github]
	Jarvis	Nist-Jarvis	1k-800k	Varied	Molecules	Random†	MAE	2020*	[Documentation][Paper]
	Therapeutic Data Commons	TDC	Varied	Varied	Molecules & Proteins	Stratified†	Varied	2022	[Documentation][github]
	TorchProtein	TorchProtein	Varied	Varied	Proteins	Stratified†	Varied	2022	[Paper][github]
	TorchDrug	TorchDrug	Varied	Varied	Molecules	Stratified†	Varied	2022	[Paper][github]
PP and MD	OC20	OpenCatalyst Project	560k-133M	3	Catalyst	Extrapolation†	MAE, EwT	2020	[Paper][github]
	OC22	OpenCatalyst Project	50k-10M	3	Catalyst	Extrapolation†	MAE, EwT	2022	[Paper][github]
	ODAC23	OpenCatalyst Project	$\leq$40M	3	Catalyst & MOF	Extrapolation†	MAE, EwT	2023*	[Paper][github]
MD	MD17	–	50k-1M	10	Molecule	Extrapolation	MAE	2017*	[Paper] [HF] [PyG]
	ISO17	–	645K	1	Molecule	Extrapolation	MAE	2016	[info] [Paper]
Gen.Mod	GEOM	GEOM	37M	1	Molecule	Random	MAE, RMSD	2021	[Paper][github]
	Open MatSci ML Toolkit	Open MatSci ML Toolkit	25k	1	Crystal Structures	Random	MAE	2023*	[Paper][github]
	TorchDrug	TorchDrug	Varied	Varied	Molecules	Stratified†	MAE	2022*	[Paper][github]
Struct.Pred	TorchProtein	TorchProtein	Varied	3	Proteins	Stratified†	Varied	2022*	[Paper][github]
	SPICE	–	1.1 M	6	Molecules	Random	MAE	2023	[Paper][github]
	MatBench Discovery	MatBench Discovery	150k-250k	2	Crystal Structures	Varied†	F1, Accuracy, MAE	2023	[Paper] [github]
	ProteinNet	–	35k-105k	7	Proteins	Stratified	Varied	2019	[Paper][github]
	Molecule3D	–	3.9M	4	Molecules	Random	MAE, RMSE, validity	2021	[Paper][link]

Table 1: Summary of benchmark datasets for Geometric GNNs. We categorize each dataset with respect to the application task detailed in Section 7.1. We display various properties (from left to right): the number of samples per dataset, the number of properties which can be predicted, the input data domain, the desired symmetry property, the dataset split method, the metric used to measure performance and the source. * signifies that the dataset has been updated recently. † means that there is an active leaderboard to submit test predictions and compare one's results.

8 Conclusion and Future Research Directions

This opinionated survey aims to provide both newcomers and experienced researchers with a pedagogical understanding of Geometric Graph Neural Networks for 3D atomic systems. Throughout its course, we delve into the foundations, motivations, and distinctive features that set Geometric GNNs apart from traditional GNNs. We hope that our taxonomy of Geometric GNN architectures – invariant, equivariant with Cartesian basis, equivariant with spherical basis, and unconstrained GNNs – establishes clear links between different models, granting the reader with a deeper comprehension of the available methods. In addition, we have summarised a wide range of applications and highlighted the high potential for impact of Geometric GNNs. We hope to inspire further developments in Geometric GNN modeling towards socially beneficial applications such as the discovery of novel medicine [Stokes et al., 2020], energy-efficient materials [Miret et al., 2023], and green chemistry [Anastas and Eghbali, 2010].

Our ultimate aspiration is to contribute to the organization and advancement of this emerging field while igniting the curiosity of newcomers to embark on their own journey into the captivating world of Geometric GNNs. By combining scholarly rigour with a comprehensive overview, we aspire to make this survey the natural go-to paper for the Geometric GNN community, facilitating knowledge dissemination and fostering future advancements.

As we conclude our survey, the following section reflects on the promising research directions that lie ahead and that we believe are worthy of interest.

8.1 To what extent should physics and symmetry be 'baked in' to Geometric GNNs?

Enforcing symmetries: The choice between invariant GNNs, equivariant GNNs, or unconstrained GNNs is an important consideration. Exploring the trade-off between (1) an unconstrained local message passing approach that (approximately) preserves global equivariance (e.g. FAENet [Duval et al., 2023]), and (2) a strictly equivariant method that passes local equivariant messages between atoms (e.g. MACE [Batatia et al., 2022b]), is an interesting open question. Discussing the implications of these design choices and rigorous empirical benchmarking would contribute to a deeper understanding of the trade-offs involved. For instance, rigorously enforcing symmetries can provide greater data efficiency and generalization abilities for model architectures, which is particularly interesting when getting more high-quality data is costly, or when stronger generalisation guarantees are needed. On the other hand, relaxing these constraints may be desirable if enough data is available, enabling greater expressivity and efficiency. Section 6 contains additional arguments.

Energy conservation: On a related note, for molecular dynamics applications, the debate between predicting forces using the gradient of energy versus predicting forces directly from node representations is crucial. While the former improves stability when using Geometric GNNs to run dynamics simulations [Fu et al., 2023], the latter offers memory, runtime, and sometimes performance gains [Gasteiger et al., 2021]. Considering the scalability vs. simulation stability trade-off can help practitioners decide on the appropriate approach for their datasets and tasks. Additionally, developing better metrics to quantify the importance of energy conservation would be valuable with initial proposals emerging in the literature [Bihani et al., 2023]. Given the ambitions to scale molecular dynamics simulations to trillions of atoms at longer and longer timescales, research on both improving stability and compute efficiency at scale are needed.

Opinion

We postulate that **strict equivariance** may be critical for tasks related to precise geometric prediction, such as structure prediction or molecular simulation, where small errors may compound and lead to unstable or unphysical geometries if we use approximately symmetric models [Fu et al., 2023, Bihani et al., 2023]. On the other hand, tasks akin to property prediction may benefit from approximately symmetric models which are optimised for a particular dataset, without the hard constraints that come with enforcing physical symmetries [Duval et al., 2023]. Similarly, these unconstrained models constitute a promising direction for applications where a unique canonical ordering and orientation of data is possible, such as for antibody proteins [Martinkus et al., 2023].

Deeper theoretical characterisation: A first step towards theoretical understanding of Geometric GNNs was based on their ability to solve the geometric graph isomorphism problem [Joshi et al., 2023a] , i.e. mapping unique geometric graphs to unique representations. A deeper understanding of the expressive power, optimisation behaviour, and generalisation capacity of Geometric GNNs will complement a growing landscape of empirical work, while abstracting away domain-specific implementation details. For instance, developing a provably universal, equivariant GNN on sparse graphs with finite tensor and body order remains an open question [Batatia et al., 2022b]. The emergence of new equivariant architectures based on Clifford algebras also presents opportunities for theoretical advancement and unifications [Brehmer et al., 2023, Ruhe et al., 2023]. These efforts have the potential to further understand the generalization ability of equivariant models, which can help practitioners choose what models may be appropriate for their desired use case.

Opinion

While equivariant GNNs operating on **higher-order tensors** have favorable theoretical expressivity, their practical utility remains unclear. Notably, prominent models such as RosettaFold [Baek et al., 2021] and DiffDock [Corso et al., 2023], which use the `e3nn` framework, *do not* use higher-order tensors (they are restricted to tensor order $= 1$). In practice, we believe that the theoretical advantages of higher-order tensors are circumvented by the (currently) high GPU memory requirements and slow speed of tensor product operations. Thus, theoretical studies must be supplemented by empirical benchmarks which fairly compare architectures under compute and time budgets [Jamasb et al., 2023] as well as domain-specific evaluation beyond empirical accuracy [Harris et al., 2023, Buttenschoen et al., 2023].

8.2 How to construct geometric graphs?

Graph creation and coarse-graining. There are various approaches to constructing a geometric graph as input to a Geometric GNN, including radial cutoffs, k-nearest neighbours, and long range connections. While optimal graph construction heuristics are often highly domain-specific, exploring how this construction could be modified to alleviate common structural bottlenecks for GNNs, such as the over-squashing problem [Di Giovanni et al., 2023, Giraldo et al., 2023], may hold great promise. Further studies are also needed to better understand the importance of the local neighborhood in an atomic system compared to long-range interactions that could be effectively modeled with targeted graph constuction.

Additionally, the choice of entities included in graph construction needs careful consideration. An obvious example is that all current Geometric GNN applications assume a coarse-grained implicit solvent system and do not explicitly include entities such as water molecules and ions which play an important role in molecular structure and function [Bellissent-Funel et al., 2016].

When working with coarse-grained representations of atomic systems, it becomes critical to analyse the need for atomic-level precision and completeness of the representation (whether there is a one-to-one mapping back to the all-atom scale) [Badaczewska-Dawid et al., 2020]. Imagine analyzing a molecular system where each atom's position is recorded with ultra-high accuracy. This level of detail might be unnecessary for certain research objectives, and it may even introduce undue computational complexity or overfitting to artefacts from the structure determination process [Dauparas et al., 2022]. In such cases, it raises inquiries about the deliberate introduction of controlled noise to coordinate data. The key question becomes: should this noise be intentionally motivated by physical principles, and if so, what is the optimal amount of noise to inject?

Opinion

We suspect that optimizing the way **information flows** from the perspective of graph machine learning may be promising for boosting performance beyond the physics-based perspective of radial cutoffs and interaction thresholds.

Temporal dynamics and conformational flexibility: Ideal computational representations of molecules and materials should account for both geometric structures as well as temporal dynamics. Looking beyond learning from static structures, dynamics and conformational flexibility is key to the functionality of several classes of proteins important for drug discovery, such as antibodies

and membrane proteins [Carugo and Djinović-Carugo, 2023, Lane, 2023], as well as the dynamic behavior of materials in diverse applications [Bihani et al., 2023, Fu et al., 2023]. Learning from molecular conformational ensembles will be the next frontier for advancing geometric representation learning of small molecules [Axelrod and Gomez-Bombarelli, 2023], crystals [Lee et al., 2023b, Riebesell et al., 2023], proteins [Noé et al., 2019, Janson et al., 2023], and RNA [Joshi et al., 2023b].

8.3 How to scale up Geometric GNNs?

Foundation models: Drawing inspiration from the success of self-supervised learning for large pre-trained foundation models [Bommasani et al., 2021], the question of why we do not yet have Geometric GNNs which can generalise across the range of atomic systems is worth contemplating. Exploring new self-supervised learning tasks to understand the dynamics of general-purpose atomic interactions (so-called *universal potentials*) could significantly enhance downstream performance across a wide range of application domains. On the other hand, while it is pedagogical to group application domains together under the umbrella of 'atomic systems', it may be the case that interactions governing small molecules are semantically different than those in proteins or materials. While not taking a definitive stance, this question presents an interesting research direction for the field. What architectures are expressive and scalable, and how can we efficiently train them?

Recently, Krishna et al. [2023] and DeepMind-Isomorphic [2023] generalised protein structure prediction models to full biological assemblies including proteins, small molecules, nucleic acids, and other ligands. These works show promising results, even outperforming specialised models for certain tasks, and represent an exciting first step towards foundation models for structural biology. Geometric GNNs specialised for biomolecular complexes and supramolecular systems [Steed and Atwood, 2022, Gallego et al., 2022] may require deeper consideration of higher-order symmetries present in self-assembling biological systems. For instance, naturally occurring DNA, perhaps the best-known self-assembling structure, exists in a double helical form.

Large-scale datasets and software infrastructure: Training foundation models necessitates large datasets and associated software infrastructure. The availability of large-scale collections of predicted protein structures derived from both AlphaFold2 [Jumper et al., 2021] and ESMFold [Lin et al., 2023] is an extremely promising data source towards this end. The AlphaFold Database [Varadi et al., 2021] and ESM Atlas (MGnify 2023 release [Richardson et al., 2022]) contain over 200M and 772M predicted structures, respectively. Bespoke software tools for predicted protein structures such as FoldSeek [van Kempen et al., 2023] for efficient clustering and FoldComp [Kim et al., 2023] for compression have started being used as part of pipelines to scale up Geometric GNNs for protein structure annotation [Jamasb et al., 2023].

Similarly to the Open Catalyst Project [Zitnick et al., 2020, Tran et al., 2023] and Open MatSci ML Toolkit [Miret et al., 2023, Lee et al., 2023b] for materials discovery, the release of large public datasets and benchmarks targeted to specific use cases and specially curated for advancing deep learning architecture development are much needed. Such efforts may necessitate rethinking the data generation process to be tailored towards training large deep learning models, as motivated by recent initiatives for small molecules [Mathiasen et al., 2023, Beaini et al., 2023]. We hope to see more community-driven and public initiatives in this direction, as motivated in Section 7.2.

References

K. Adams, L. Pattanaik, and C. W. Coley. Learning 3d representations of molecular chirality with invariance to bond rotations. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. (Cited on page 73)

R. Ahmad and W. Cai. Free energy calculation of crystalline solids using normalizing flows. *Modelling and Simulation in Materials Science and Engineering*, 30(6):065007, 2022. (Cited on page 42)

M. AI4Science, A. Hernandez-Garcia, A. Duval, A. Volokhova, Y. Bengio, D. Sharma, P. L. Carrier, M. Koziarski, and V. Schmidt. Crystal-gfn: sampling crystals with desirable properties and constraints. *arXiv preprint arXiv:2310.04925*, 2023. (Cited on page 42)

R. F. Alford, A. Leaver-Fay, J. R. Jeliazkov, M. J. O'Meara, F. P. DiMaio, H. Park, M. V. Shapovalov, P. D. Renfrew, V. K. Mulligan, K. Kappel, et al. The rosetta all-atom energy function for macromolecular modeling and design. *Journal of chemical theory and computation*, 13(6):3031–3048, 2017. (Cited on page 43)

M. AlQuraishi. Machine learning in protein structure prediction. *Current opinion in chemical biology*, 65:1–8, 2021. (Cited on page 42)

P. Anastas and N. Eghbali. Green chemistry: principles and practice. *Chemical Society Reviews*, 39 (1):301–312, 2010. (Cited on page 46)

B. Anderson, T. S. Hy, and R. Kondor. Cormorant: Covariant molecular neural networks. *Advances in neural information processing systems*, 32, 2019. (Cited on page 73)

K. Atz, F. Grisoni, and G. Schneider. Geometric deep learning on molecular representations. *Nature Machine Intelligence*, 3(12):1023–1032, 2021. (Cited on page 64, 65, 74)

S. Axelrod and R. Gomez-Bombarelli. Molecular machine learning with conformer ensembles. *Machine Learning: Science and Technology*, 4(3):035025, 2023. (Cited on page 48)

A. E. Badaczewska-Dawid, A. Kolinski, and S. Kmiecik. Computational reconstruction of atomistic protein structures from coarse-grained models. *Computational and structural biotechnology journal*, 18:162–176, 2020. (Cited on page 47)

M. Baek, F. DiMaio, I. Anishchenko, J. Dauparas, et al. Accurate prediction of protein structures and interactions using a three-track neural network. *Science*, 2021. (Cited on page 42, 43, 47)

M. Baek, R. McHugh, I. Anishchenko, D. Baker, and F. DiMaio. Accurate prediction of nucleic acid and protein-nucleic acid complexes using rosettafoldna. *bioRxiv*, pages 2022–09, 2022. (Cited on page 42)

V. Bapst, T. Keck, A. Grabska-Barwińska, C. Donner, E. D. Cubuk, S. Schoenholz, A. Obika, A. Nelson, T. Back, D. Hassabis, et al. Unveiling the predictive power of static structure in glassy systems. *Nature Physics*, 16(4):448–454, 2020. (Cited on page 69)

A. P. Bartók, R. Kondor, and G. Csányi. On representing chemical environments. *Physical Review B*, 2013. (Cited on page 17)

I. Batatia, S. Batzner, D. P. Kovács, A. Musaelian, G. N. Simm, R. Drautz, C. Ortner, B. Kozinsky, and G. Csányi. The design space of e (3)-equivariant atom-centered interatomic potentials. *arXiv preprint*, 2022a. (Cited on page 15, 41)

I. Batatia, D. P. Kovács, G. N. C. Simm, C. Ortner, and G. Csányi. MACE: higher order equivariant message passing neural networks for fast and accurate force fields. In *NeurIPS*, 2022b. (Cited on page 31, 35, 41, 46, 47, 63, 73)

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint*, 2018. (Cited on page 8, 73)

S. Batzner, A. Musaelian, L. Sun, M. Geiger, J. P. Mailoa, M. Kornbluth, N. Molinari, T. E. Smidt, and B. Kozinsky. E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature communications*, 13(1):1–11, 2022. (Cited on page 7, 9, 31, 41, 73)

D. Beaini, S. Huang, J. A. Cunha, G. Moisescu-Pareja, O. Dymov, S. Maddrell-Mander, C. McLean, F. Wenkel, L. Müller, J. H. Mohamud, et al. Towards foundational models for molecular learning on large-scale multi-task datasets. *arXiv preprint arXiv:2310.04292*, 2023. (Cited on page 48)

J. Behler. Perspective: Machine learning potentials for atomistic simulations. *The Journal of chemical physics*, 145(17):170901, 2016. (Cited on page 73)

M.-C. Bellissent-Funel, A. Hassanali, M. Havenith, R. Henchman, P. Pohl, F. Sterpone, D. Van Der Spoel, Y. Xu, and A. E. Garcia. Water determines the structure and dynamics of proteins. *Chemical reviews*, 116(13):7673–7697, 2016. (Cited on page 47)

A. Benamira, B. Devillers, E. Lesot, A. K. Ray, M. Saadi, and F. D. Malliaros. Semi-supervised learning and graph neural networks for fake news detection. In *ASONAM*, 2020. (Cited on page 9)

E. Bengio, M. Jain, M. Korablyov, D. Precup, and Y. Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021. (Cited on page 42)

R. Bhattoo, S. Ranu, and N. A. Krishnan. Learning the dynamics of particle-based systems with lagrangian graph neural networks. *Machine Learning: Science and Technology*, 4(1):015003, 2023. (Cited on page 41)

V. Bihani, U. Pratiush, S. Mannan, T. Du, Z. Chen, S. Miret, M. Micoulaut, M. M. Smedskjaer, S. Ranu, and N. Krishnan. Egraffbench: Evaluation of equivariant graph neural network force fields for atomistic simulations. *arXiv preprint arXiv:2310.02428*, 2023. (Cited on page 41, 46, 48)

C. M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7: 108–116, 1995. doi: 10.1162/neco.1995.7.1.108. (Cited on page 72)

B. Blum-Smith and S. Villar. Machine learning and invariant theory. *arXiv preprint arXiv:2209.14991*, 2022. (Cited on page 31)

C. Bodnar, F. Frasca, N. Otter, Y. Wang, P. Lio, G. F. Montufar, and M. Bronstein. Weisfeiler and lehman go cellular: Cw networks. *NeurIPS*, 2021. (Cited on page 71)

A. Bogatskiy, S. Ganguly, T. Kipf, R. Kondor, D. W. Miller, D. Murnane, J. T. Offermann, M. Pettee, P. Shanahan, C. Shimmin, et al. Symmetry group equivariant architectures for physics. *arXiv preprint*, 2022. (Cited on page 6)

R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021. (Cited on page 48)

J. Brandstetter, R. Hesselink, E. van der Pol, E. Bekkers, and M. Welling. Geometric and physical quantities improve E(3) equivariant message passing. *International Conference on Learning Representations*, 2021. (Cited on page 31, 73)

J. Brandstetter, R. Hesselink, E. van der Pol, E. J. Bekkers, and M. Welling. Geometric and physical quantities improve e(3) equivariant message passing. In *ICLR*, 2022. (Cited on page 35)

J. Brehmer, P. De Haan, S. Behrends, and T. Cohen. Geometric algebra transformers. *arXiv preprint arXiv:2305.18415*, 2023. (Cited on page 12, 47)

M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Process. Mag.*, 34(4):18–42, 2017. doi: 10.1109/MSP.2017.2693418. (Cited on page 72)

M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint*, 2021. (Cited on page 6, 10)

M. Buttenschoen, G. M. Morris, and C. M. Deane. Posebusters: Ai-based docking methods fail to generate physically valid poses or generalise to novel sequences. *arXiv preprint arXiv:2308.05777*, 2023. (Cited on page 47)

W. Cao, Z. Yan, Z. He, and Z. He. A comprehensive survey on geometric deep learning. *IEEE Access*, 8:35929–35949, 2020. doi: 10.1109/ACCESS.2020.2975067. URL https://ieeexplore.ieee.org/document/9003285. (Cited on page 40)

O. Carugo and K. Djinović-Carugo. Structural biology: A golden era. *PLoS Biology*, 21(6):e3002187, 2023. (Cited on page 48)

L. Chanussot, A. Das, S. Goyal, T. Lavril, M. Shuaibi, M. Riviere, K. Tran, J. Heras-Domingo, C. Ho, W. Hu, et al. Open catalyst 2020 (oc20) dataset and community challenges. *ACS Catalysis*, 11(10): 6059–6072, 2021. (Cited on page 15, 38, 40, 44, 68)

C. Chen and S. P. Ong. A universal graph deep learning interatomic potential for the periodic table. *Nature Computational Science*, 2(11):718–728, 2022. (Cited on page 43)

A. Cheng, A. Lo, S. Miret, B. Pate, and A. Aspuru-Guzik. Reflection-equivariant diffusion for 3d structure determination from isotopologue rotational spectra in natural abundance, 2023a. (Cited on page 43)

A. H. Cheng, A. Cai, S. Miret, G. Malkomes, M. Phielipp, and A. Aspuru-Guzik. Group selfies: a robust fragment-based molecular string representation. *Digital Discovery*, 2023b. (Cited on page 43)

S. Chmiela, A. Tkatchenko, H. E. Sauceda, I. Poltavsky, K. T. Schütt, and K.-R. Müller. Machine learning of accurate energy-conserving molecular force fields. *Science advances*, 3(5):e1603015, 2017. (Cited on page 15)

S. Chmiela, H. E. Sauceda, K.-R. Müller, and A. Tkatchenko. Towards exact molecular dynamics simulations with machine-learned force fields. *Nature communications*, 9(1):1–10, 2018. (Cited on page 73)

K. Choudhary and B. DeCost. Atomistic line graph neural network for improved materials property predictions. *npj Computational Materials*, 7(1):185, 2021. (Cited on page 43)

T. Cohen and M. Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016. (Cited on page 31, 73)

G. Corso, H. Stärk, B. Jing, R. Barzilay, and T. S. Jaakkola. Diffdock: Diffusion steps, twists, and turns for molecular docking. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023. (Cited on page 42, 47)

J. Dauparas, I. Anishchenko, N. Bennett, H. Bai, R. J. Ragotte, L. F. Milles, B. I. Wicky, A. Courbet, R. J. de Haas, N. Bethel, et al. Robust deep learning based protein sequence design using proteinmpnn. *Science*, 2022. (Cited on page 7, 9, 13, 42, 47)

N. De Cao and T. Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018. (Cited on page 42)

DeepMind-Isomorphic. Performance and structural coverage of the latest, in-development alphafold model. 2023. (Cited on page 48)

N. Dehmamy, R. Walters, Y. Liu, D. Wang, and R. Yu. Automatic symmetry discovery with lie algebra convolutional network. *Advances in Neural Information Processing Systems*, 34:2503–2515, 2021. (Cited on page 39)

A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire, et al. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3767–3776, 2021. (Cited on page 9)

F. Di Giovanni, T. K. Rusch, M. M. Bronstein, A. Deac, M. Lackenby, S. Mishra, and P. Velicković. How does over-squashing affect the power of gnns? *arXiv preprint arXiv:2306.03589*, 2023. (Cited on page 47)

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. (Cited on page 37)

R. Drautz. Atomic cluster expansion for accurate and transferable interatomic potentials. *Physical Review B*, 99(1):014104, 2019. (Cited on page 36)

W. Du, H. Zhang, Y. Du, Q. Meng, W. Chen, N. Zheng, B. Shao, and T.-Y. Liu. Se (3) equivariant graph neural networks with complete local frames. In *International Conference on Machine Learning*, pages 5583–5608. PMLR, 2022. (Cited on page 21)

C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia. Incorporating second-order functional knowledge for better option pricing. *Advances in neural information processing systems*, pages 472–478, 2001. (Cited on page 69)

A. Dunn, Q. Wang, A. Ganose, D. Dopp, and A. Jain. Benchmarking materials property prediction methods: the matbench test set and automatminer reference algorithm. npj comput mater 6, 1 (september 2020), 1-10. *DOI: https://doi. org/10.1038/s41524-020-00406-3*, 2020. (Cited on page 44)

A. Duval, V. Schmidt, S. Miret, Y. Bengio, A. Hernández-García, and D. Rolnick. PhAST: Physics-aware, scalable, and task-specific GNNs for accelerated catalyst design. *Preprint arXiv:2211.12020*, 2022. (Cited on page 14, 15, 68)

A. Duval, V. Schmidt, A. Hernández-García, S. Miret, F. D. Malliaros, Y. Bengio, and D. Rolnick. Faenet: Frame averaging equivariant GNN for materials modeling. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 9013–9033. PMLR, 2023. (Cited on page 14, 37, 38, 46, 70, 74)

D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015. (Cited on page 12)

V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. *JMLR*, 2023. (Cited on page 71)

N. Dym and H. Maron. On the universality of rotation equivariant point cloud networks. In *ICLR*, 2020. (Cited on page 18)

U. Epa. User's guide for test (version 5.1)(toxicity estimation software tool): a program to estimate toxicity from molecular structure. *Chemical Characterization and Exposure Division Cincinnati O, ed*, 2020. (Cited on page 40)

J. Feinstein, G. Sivaraman, K. Picel, B. Peters, Á. Vázquez-Mayagoitia, A. Ramanathan, M. MacDonell, I. Foster, and E. Yan. Uncertainty-informed deep transfer learning of perfluoroalkyl and polyfluoroalkyl substance toxicity. *Journal of chemical information and modeling*, 61(12): 5793–5803, 2021. (Cited on page 40)

T. Frank, O. T. Unke, and K. R. Muller. So3krates: Equivariant attention for interactions on arbitrary length-scales in molecular systems. In *Advances in Neural Information Processing Systems*, 2022. (Cited on page 73)

X. Fu, Z. Wu, W. Wang, T. Xie, S. Keten, R. Gomez-Bombarelli, and T. S. Jaakkola. Forces are not enough: Benchmark and critical evaluation for machine learning force fields with molecular simulations. *Transactions on Machine Learning Research*, 2023. (Cited on page 15, 39, 41, 46, 48)

F. Fuchs, D. Worrall, V. Fischer, and M. Welling. SE(3)-transformers: 3D roto-translation equivariant attention networks. *Advances in Neural Information Processing Systems*, 33:1970–1981, 2020. (Cited on page 35, 73)

L. Gallego, J. F. Woods, and M. Rickhaus. Recent concepts for supramolecular 2d materials. *Organic Materials*, 4(03):137–145, 2022. (Cited on page 48)

J. Gasteiger, J. Groß, and S. Günnemann. Directional message passing for molecular graphs. In *ICLR*, 2020. (Cited on page 17)

J. Gasteiger, F. Becker, and S. Günnemann. Gemnet: Universal directional graph neural networks for molecules. *Advances in Neural Information Processing Systems*, 34:6790–6802, 2021. (Cited on page 12, 17, 40, 41, 46, 68, 70, 71, 73)

J. Gasteiger, M. Shuaibi, A. Sriram, S. Günnemann, Z. W. Ulissi, C. L. Zitnick, and A. Das. Gemnet-oc: Developing graph neural networks for large and diverse molecular simulation datasets. *Trans. Mach. Learn. Res.*, 2022, 2022. (Cited on page 18, 72, 73)

N. Gebauer, M. Gastegger, and K. Schütt. Symmetry-adapted generation of 3d point sets for the targeted discovery of molecules. *Advances in neural information processing systems*, 32, 2019. (Cited on page 42)

M. Geiger and T. Smidt. e3nn: Euclidean neural networks. *arXiv preprint*, 2022. (Cited on page 28, 29, 31, 33, 34)

J. Gerken, O. Carlsson, H. Linander, F. Ohlsson, C. Petersson, and D. Persson. Equivariance versus augmentation for spherical images. In *International Conference on Machine Learning*, pages 7404–7421. PMLR, 2022. (Cited on page 37)

J. E. Gerken, J. Aronsson, O. Carlsson, H. Linander, F. Ohlsson, C. Petersson, and D. Persson. Geometric deep learning and equivariant neural networks. *Artificial Intelligence Review*, 56: 14605–14662, 2023. doi: 10.1007/s10462-023-10502-7. URL https://link.springer.com/article/10.1007/s10462-023-10502-7/fulltext.html. (Cited on page 40)

J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017. (Cited on page 6, 8)

J. H. Giraldo, K. Skianis, T. Bouwmans, and F. D. Malliaros. On the trade-off between over-smoothing and over-squashing in deep graph neural networks. In *CIKM*, 2023. (Cited on page 47, 68)

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *aistats*, pages 315–323, 2011. (Cited on page 69)

J. Godwin, M. Schaarschmidt, A. Gaunt, A. Sanchez-Gonzalez, Y. Rubanova, P. Veličković, J. Kirkpatrick, and P. Battaglia. Simple gnn regularisation for 3d molecular property prediction & beyond. *arXiv preprint arXiv:2106.07971*, 2021. (Cited on page 13)

C. Goller and A. Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 347–352. IEEE, 1996. (Cited on page 8)

M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005. (Cited on page 8, 73)

Y. Guan, S. S. Sowndarya, L. C. Gallegos, P. C. S. John, and R. S. Paton. Real-time prediction of 1 h and 13 c chemical shifts with dft accuracy using a 3d graph neural network. *Chemical Science*, 12 (36):12012–12026, 2021. (Cited on page 43)

Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(12):4338–4364, 2020. (Cited on page 37)

T. Halgren. Merck molecular force field. i. basis, form, scope, parameterization, and performance of mmff94. *Comput. Chem.*, 17:490–519, 1996. (Cited on page 73)

W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017. (Cited on page 9)

J. Han, Y. Rong, T. Xu, and W. Huang. Geometrically equivariant graph neural networks: A survey. *arXiv preprint arXiv:2202.07230*, 2022. (Cited on page 74)

C. Harris, K. Didi, A. Jamasb, C. Joshi, S. Mathis, P. Lio, and T. Blundell. Posecheck: Generative models for 3d structure-based drug design produce unrealistic poses. In *NeurIPS 2023 Generative AI and Biology (GenBio) Workshop*, 2023. (Cited on page 47)

K. He, X. Zhang, S. Ren, and J. Sun. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. doi: 10.1109/CVPR.2016.90. (Cited on page 73)

J. Hoja, L. Medrano Sandonas, B. G. Ernst, A. Vazquez-Mayagoitia, R. A. DiStasio Jr, and A. Tkatchenko. Qm7-x, a comprehensive dataset of quantum-mechanical properties spanning the chemical space of small organic molecules. *Scientific data*, 8(1):43, 2021. (Cited on page 73, 77)

E. Hoogeboom, V. G. Satorras, C. Vignac, and M. Welling. Equivariant diffusion for molecule generation in 3d. In *International conference on machine learning*, pages 8867–8887. PMLR, 2022. (Cited on page 42)

W. Hu, M. Shuaibi, A. Das, S. Goyal, A. Sriram, J. Leskovec, D. Parikh, and C. L. Zitnick. Forcenet: A graph neural network for large-scale quantum calculations. *Preprint arXiv:2103.01436*, 2021. (Cited on page 13, 14, 37, 74)

K. Huang, T. Fu, W. Gao, Y. Zhao, Y. Roohani, J. Leskovec, C. W. Coley, C. Xiao, J. Sun, and M. Zitnik. Artificial intelligence foundation for therapeutic science. *Nature chemical biology*, 18 (10):1033–1036, 2022. (Cited on page 40, 44)

J. Ingraham, V. Garg, R. Barzilay, and T. Jaakkola. Generative models for graph-based protein design. *NeurIPS*, 2019. (Cited on page 42)

J. Ingraham, M. Baranov, Z. Costello, V. Frappier, A. Ismail, S. Tie, W. Wang, V. Xue, F. Obermeyer, A. Beam, et al. Illuminating protein space with a programmable generative model. *BioRxiv*, 2022. (Cited on page 13, 43)

K. Jamali, L. Käll, R. Zhang, A. Brown, D. Kimanius, and S. H. Scheres. Automated model building and protein identification in cryo-em maps. *bioRxiv*, 2023. (Cited on page 43)

A. R. Jamasb, A. Morehead, Z. Zuobai, C. K. Joshi, K. Didi, S. V. Mathis, C. Harris, J. Tang, J. Cheng, P. Liò, and T. L. Blundell. Evaluating representation learning on the protein structure universe. *NeurIPS 2023 Workshop on Machine Learning for Structural Biology*, 2023. (Cited on page 14, 47, 48)

G. Janson, G. Valdes-Garcia, L. Heo, and M. Feig. Direct generation of protein conformational ensembles via machine learning. *Nature Communications*, 2023. (Cited on page 48)

S. Jegelka. Theory of graph neural networks: Representation and learning. *arXiv preprint arXiv:2204.07697*, 2022. (Cited on page 71)

R. Jiao, W. Huang, P. Lin, J. Han, P. Chen, Y. Lu, and Y. Liu. Crystal structure prediction by joint equivariant diffusion. *arXiv preprint arXiv:2309.04475*, 2023. (Cited on page 43)

B. Jing, S. Eismann, P. Suriana, R. J. L. Townshend, and R. Dror. Learning from protein structure with geometric vector perceptrons. In *ICLR*, 2020. (Cited on page 15, 21)

C. Joshi. Transformers are graph neural networks. *The Gradient*, 2020. (Cited on page 6)

C. K. Joshi, C. Bodnar, S. V. Mathis, T. Cohen, and P. Liò. On the expressive power of geometric graph neural networks. In *International Conference on Machine Learning*, 2023a. (Cited on page 17, 18, 39, 47, 71)

C. K. Joshi, A. R. Jamasb, R. Viñas, C. Harris, S. Mathis, and P. Liò. Multi-state rna design with geometric multi-graph neural networks. *arXiv preprint*, 2023b. (Cited on page 48)

J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. **Z**ídek, A. Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 2021. (Cited on page 7, 9, 42, 43, 48)

S.-O. Kaba and S. Ravanbakhsh. Equivariant networks for crystal structures. In *Advances in Neural Information Processing Systems*, 2022. (Cited on page 13)

S.-O. Kaba, A. K. Mondal, Y. Zhang, Y. Bengio, and S. Ravanbakhsh. Equivariance with learned canonicalization functions. In *International Conference on Machine Learning*, pages 15546–15566. PMLR, 2023. (Cited on page 39)

M. Karplus and J. A. McCammon. Molecular dynamics simulations of biomolecules. *Nature structural biology*, 9(9):646–652, 2002. (Cited on page 41)

S. M. Kearnes, M. R. Maser, M. Wleklinski, A. Kast, A. G. Doyle, S. D. Dreher, J. M. Hawkins, K. F. Jensen, and C. W. Coley. The open reaction database. *Journal of the American Chemical Society*, 143(45):18820–18826, 2021. (Cited on page 41)

H. Kim, M. Mirdita, and M. Steinegger. Foldcomp: a library and format for compressing and indexing large protein structure sets. *Bioinformatics*, 39(4), Mar. 2023. doi: 10.1093/bioinformatics/btad153. (Cited on page 48)

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. (Cited on page 6, 8)

J. Klicpera, J. Groß, and S. Günnemann. Directional message passing for molecular graphs. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020. (Cited on page 73)

W. Kohn, A. D. Becke, and R. G. Parr. Density functional theory of electronic structure. *The Journal of Physical Chemistry*, 100(31):12974–12980, 1996. (Cited on page 40)

A. Kolluru, M. Shuaibi, A. Palizhati, N. Shoghi, A. Das, B. Wood, C. L. Zitnick, J. Kitchin, and Z. W. Ulissi. Open challenges in developing generalizable large scale machine learning models for catalyst discovery. *ACS Catalysis*, 2022. doi: 10.1021/acscatal.2c02291. (Cited on page 15, 42)

R. Kondor, Z. Lin, and S. Trivedi. Clebsch-gordan nets: a fully fourier space spherical convolutional neural network. *Advances in Neural Information Processing Systems*, 31, 2018. (Cited on page 31, 73)

A. Kosmala, J. Gasteiger, N. Gao, and S. Günnemann. Ewald-based long-range message passing for molecular graphs. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 17544–17563. PMLR, 2023. (Cited on page 12)

M. Krenn, F. Häse, A. Nigam, P. Friederich, and A. Aspuru-Guzik. Self-referencing embedded strings (selfies): A 100% robust molecular string representation. *Machine Learning: Science and Technology*, 1(4):045024, 2020. (Cited on page 43)

R. Krishna, J. Wang, W. Ahern, P. Sturmfels, P. Venkatesh, I. Kalvet, G. R. Lee, F. S. Morey-Burrows, I. Anishchenko, I. R. Humphreys, et al. Generalized biomolecular modeling and design with rosettafold all-atom. *bioRxiv*, pages 2023–10, 2023. (Cited on page 48)

R. Lam, A. Sanchez-Gonzalez, M. Willson, P. Wirnsberger, M. Fortunato, F. Alet, S. Ravuri, T. Ewalds, Z. Eaton-Rosen, W. Hu, et al. Learning skillful medium-range global weather forecasting. *Science*, page eadi2336, 2023. (Cited on page 9)

T. J. Lane. Protein structure prediction has reached the single-structure frontier. *Nature Methods*, 20 (2):170–173, 2023. (Cited on page 43, 48)

A. H. Larsen, J. J. Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Du$\underline{a}$k, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. B. Jensen, J. Kermode, J. R. Kitchin, E. L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, and K. W. Jacobsen. The atomic simulation environment—a python library for working with atoms. *Journal of Physics: Condensed Matter*, 29(27):273002, 2017. (Cited on page 41)

T. Le, F. Noe, and D.-A. Clevert. Representation learning on biomolecular structures using equivariant graph attention. In *Learning on Graphs Conference*, pages 30–1. PMLR, 2022. (Cited on page 21)

J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019. (Cited on page 72)

J. H. Lee, P. Yadollahpour, A. Watkins, N. C. Frey, A. Leaver-Fay, S. Ra, K. Cho, V. Gligorijević, A. Regev, and R. Bonneau. Equifold: Protein structure prediction with a novel coarse-grained structure representation. *bioRxiv*, 2023a. doi: 10.1101/2022.10.07.511322. (Cited on page 43)

K. L. K. Lee, C. Gonzales, M. Nassar, M. Spellings, M. Galkin, and S. Miret. Matsciml: A broad, multi-task benchmark for solid-state materials modeling. *arXiv preprint arXiv:2309.05934*, 2023b. (Cited on page 40, 44, 48, 73)

Z. Li, X. Wang, Y. Huang, and M. Zhang. Is distance matrix enough for geometric deep learning? *arXiv preprint arXiv: 2302.05743*, 2023. (Cited on page 18)

Y. Liao and T. E. Smidt. Equiformer: Equivariant graph attention transformer for 3d atomistic graphs. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023. (Cited on page 31, 35, 73)

Z. Lin, H. Akin, R. Rao, B. Hie, Z. Zhu, W. Lu, N. Smetanin, R. Verkuil, O. Kabeli, Y. Shmueli, et al. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023. (Cited on page 42, 48)

S. Liu, W. Du, Y. Li, Z. Li, Z. Zheng, C. Duan, Z. Ma, O. Yaghi, A. Anandkumar, C. Borgs, et al. Symmetry-informed geometric representation for molecules, proteins, and crystalline materials. *arXiv preprint arXiv:2306.09375*, 2023. (Cited on page 74)

Y. Liu, L. Wang, M. Liu, X. Zhang, B. Oztekin, and S. Ji. Spherical message passing for 3D graph networks. *Preprint arXiv:2102.05013*, 2021. (Cited on page 42, 73)

Y. Liu, L. Wang, M. Liu, Y. Lin, X. Zhang, B. Oztekin, and S. Ji. Spherical message passing for 3d molecular graphs. In *ICLR*, 2022. (Cited on page 18, 73)

T. Long, N. M. Fortunato, I. Opahle, Y. Zhang, I. Samathrakis, C. Shen, O. Gutfleisch, and H. Zhang. Constrained crystals deep convolutional generative adversarial network for the inverse design of crystal structures. *npj Computational Materials*, 7(1):66, 2021. (Cited on page 42)

Y. Luo and S. Ji. An autoregressive flow model for 3d molecular geometry generation from scratch. In *International Conference on Learning Representations (ICLR)*, 2022. (Cited on page 42)

A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013. (Cited on page 69)

T. M. MacRobert. Spherical harmonics: an elementary treatise on harmonic functions with applications. 1947. (Cited on page 70)

K. Martinkus, J. Ludwiczak, W.-C. Liang, J. Lafrance-Vanasse, I. Hotzel, A. Rajpal, Y. Wu, K. Cho, R. Bonneau, V. Gligorijevic, et al. Abdiffuser: full-atom generation of in-vitro functioning antibodies. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. (Cited on page 46)

A. Mathiasen, H. Helal, P. Balanca, K. Klaeser, J. Dean, C. Luschi, D. Beaini, A. W. Fitzgibbon, and D. Masters. Repurposing density functional theory to suit deep learning. In *1st Workshop on the Synergy of Scientific and Machine Learning Modeling @ ICML2023*, 2023. (Cited on page 48)

R. Mercado, S. M. Kearnes, and C. W. Coley. Data sharing in chemistry: Lessons learned and a case for mandating structured reaction data. *Journal of Chemical Information and Modeling*, 63(14): 4253–4265, 2023. (Cited on page 41)

S. Miret, K. L. K. Lee, C. Gonzales, M. Nassar, and M. Spellings. The open matsci ML toolkit: A flexible framework for machine learning in materials science. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. (Cited on page 44, 46, 48, 73)

F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein. Fake news detection on social media using geometric deep learning. *arXiv preprint arXiv:1902.06673*, 2019. (Cited on page 9)

A. Morehead and J. Cheng. Geometry-complete perceptron networks for 3d molecular graphs. *arXiv preprint arXiv:2211.02504*, 2022. (Cited on page 21)

C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, 2019. (Cited on page 71)

A. Musaelian, S. L. Batzner, A. Johansson, L. Sun, C. J. Owen, M. Kornbluth, and B. Kozinsky. Learning local equivariant representations for large-scale atomistic dynamics. *Nature Communications*, 2022. doi: 10.1038/s41467-023-36329-y. (Cited on page 36, 73)

F. Noé, S. Olsson, J. Köhler, and H. Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457):eaaw1147, 2019. (Cited on page 48)

A. Nouira, J. Crivello, and N. Sokolovska. Crystalgan: Learning to discover crystallographic structures with generative adversarial networks. *AAAI Spring Symposium Combining Machine Learning with Knowledge Engineering*, 2018. (Cited on page 42)

I. Nusrat and S.-B. Jang. A comparison of regularization techniques in deep neural networks. *Symmetry*, 10:648, 2018. (Cited on page 72)

T. Pakornchote, N. Choomphon-anomakhun, S. Arrerut, C. Atthapak, S. Khamkaeo, T. Chotibut, and T. Bovornratanaraks. Diffusion probabilistic models enhance variational autoencoder for crystal structure generative modeling. *arXiv preprint arXiv:2308.02165*, 2023. (Cited on page 42)

S. Passaro and C. L. Zitnick. Reducing so (3) convolutions to so (2) for efficient equivariant gnns. *arXiv preprint arXiv:2302.03655*, 2023. (Cited on page 36)

S. N. Pozdnyakov and M. Ceriotti. Incompleteness of graph convolutional neural networks for points clouds in three dimensions. *arXiv preprint*, 2022. (Cited on page 17, 72)

S. N. Pozdnyakov and M. Ceriotti. Smooth, exact rotational symmetrization for deep learning on point clouds. *arXiv preprint arXiv:2305.19302*, 2023. (Cited on page 39)

S. N. Pozdnyakov, M. J. Willatt, A. P. Bartók, C. Ortner, G. Csányi, and M. Ceriotti. Incompleteness of atomic structure representations. *Physical Review Letters*, 2020. (Cited on page 17, 18, 39)

O. Puny, M. Atzmon, E. J. Smith, I. Misra, A. Grover, H. Ben-Hamu, and Y. Lipman. Frame averaging for invariant and equivariant network design. In *International Conference on Learning Representations*, 2022. (Cited on page 38)

C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. (Cited on page 37)

M. Ragoza, T. Masuda, and D. R. Koes. Learning a continuous representation of 3d molecular structures with deep generative models. *arXiv preprint arXiv:2010.08687*, 2020. (Cited on page 42)

P. Ramachandran, B. Zoph, and Q. V. Le. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*, 7, 2017. (Cited on page 69)

R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014. (Cited on page 12, 73)

R. M. Rao, J. Liu, R. Verkuil, J. Meier, J. Canny, P. Abbeel, T. Sercu, and A. Rives. Msa transformer. In *International Conference on Machine Learning*, pages 8844–8856. PMLR, 2021. (Cited on page 42)

Z. Ren, S. I. P. Tian, J. Noh, F. Oviedo, G. Xing, J. Li, Q. Liang, R. Zhu, A. G. Aberle, S. Sun, et al. An invertible crystallographic representation for general inverse design of inorganic crystals with targeted properties. *Matter*, 5(1):314–335, 2022. (Cited on page 42)

L. Richardson, B. Allen, G. Baldi, M. Beracochea, M. L. Bileschi, T. Burdett, J. Burgin, J. Caballero-Pérez, G. Cochrane, L. J. Colwell, T. Curtis, A. Escobar-Zepeda, T. A. Gurbich, V. Kale, A. Korobeynikov, S. Raj, A. B. Rogers, E. Sakharova, S. Sanchez, D. J. Wilkinson, and R. D. Finn. MGnify: the microbiome sequence data analysis resource in 2023. *Nucleic Acids Research*, 51 (D1):D753–D759, Dec. 2022. doi: 10.1093/nar/gkac1080. (Cited on page 48)

J. Riebesell, R. E. Goodall, A. Jain, P. Benner, K. A. Persson, and A. A. Lee. Matbench discovery-an evaluation framework for machine learning crystal stability prediction. *arXiv preprint arXiv:2308.14920*, 2023. (Cited on page 43, 48)

H. A. Rowley, S. Baluja, and T. Kanade. Rotation invariant neural network-based face detection. In *Proceedings. 1998 IEEE computer society conference on computer vision and pattern recognition (Cat. No. 98CB36231)*, pages 38–44. IEEE, 1998. (Cited on page 72)

D. Ruhe, J. Brandstetter, and P. Forré. Clifford group equivariant neural networks. *arXiv preprint arXiv:2305.11141*, 2023. (Cited on page 47)

A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR, 2020. (Cited on page 37, 69)

V. G. Satorras, E. Hoogeboom, F. Fuchs, I. Posner, and M. Welling. E(n) equivariant normalizing flows. *Neural Information Processing Systems*, 2021a. (Cited on page 14, 42)

V. G. Satorras, E. Hoogeboom, and M. Welling. E(n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021b. (Cited on page 21, 73)

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008. (Cited on page 6, 8)

M. Schaarschmidt, M. Rivière, A. Ganose, J. Spencer, A. Gaunt, J. Kirkpatrick, S. Axelrod, P. Battaglia, and J. Godwin. Learned force fields are ready for ground state catalyst discovery. *ARXIV.ORG*, 2022. doi: 10.48550/arXiv.2209.12466. (Cited on page 41)

A. Schneuing, Y. Du, C. Harris, A. Jamasb, I. Igashov, W. Du, T. Blundell, P. Lió, C. Gomes, M. Welling, et al. Structure-based drug design with equivariant diffusion models. *arXiv preprint*, 2022. (Cited on page 42)

K. Schütt, P.-J. Kindermans, H. E. Sauceda Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *Advances in neural information processing systems*, 30, 2017. (Cited on page 12, 14, 15, 42, 70, 73)

K. Schütt, O. Unke, and M. Gastegger. Equivariant message passing for the prediction of tensorial properties and molecular spectra. In *International Conference on Machine Learning*, pages 9377–9388. PMLR, 2021a. (Cited on page 15, 21, 73)

K. Schütt, O. Unke, and M. Gastegger. Equivariant message passing for the prediction of tensorial properties and molecular spectra. In *ICML*, 2021b. (Cited on page 21, 71)

K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko, and K.-R. Müller. Schnet-a deep learning architecture for molecules and materials. *The Journal of Chemical Physics*, 148(24): 241722, 2018. (Cited on page 16, 69)

A. Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020. (Cited on page 73)

M. Shuaibi, A. Kolluru, A. Das, A. Grover, A. Sriram, Z. Ulissi, and C. L. Zitnick. Rotation invariant graph neural networks using spin convolutions. *Preprint arXiv:2106.09575*, 2021. (Cited on page 17, 68, 73)

A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997. (Cited on page 8)

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56): 1929–1958, 2014. (Cited on page 72)

J. W. Steed and J. L. Atwood. *Supramolecular chemistry*. John Wiley & Sons, 2022. (Cited on page 48)

J. M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackermann, et al. A deep learning approach to antibiotic discovery. *Cell*, 2020. (Cited on page 40, 46)

P. Thölke and G. De Fabritiis. Torchmd-net: Equivariant transformers for neural network based molecular potentials. *Preprint arXiv:2202.02541*, 2022. (Cited on page 21, 73)

N. Thomas, T. Smidt, S. Kearnes, L. Yang, L. Li, K. Kohlhoff, and P. Riley. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *Preprint arXiv:1802.08219*, 2018. (Cited on page 12, 14, 31, 35, 73)

A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, et al. Lammps-a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications*, 271:108171, 2022. (Cited on page 41)

R. Tran, J. Lan, M. Shuaibi, B. M. Wood, S. Goyal, A. Das, J. Heras-Domingo, A. Kolluru, A. Rizvi, N. Shoghi, et al. The open catalyst 2022 (oc22) dataset and challenges for oxide electrocatalysts. *ACS Catalysis*, 13(5):3066–3084, 2023. (Cited on page 15, 44, 48)

M. Uhrin. Through the eyes of a descriptor: Constructing complete, invertible descriptions of atomic environments. *Physical Review B*, 104(14):144110, 2021. (Cited on page 31)

O. T. Unke and M. Meuwly. PhysNet: A neural network for predicting energies, forces, dipole moments, and partial charges. *Journal of chemical theory and computation*, 15(6):3678–3693, 2019. (Cited on page 12, 17, 73)

D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, and H. J. Berendsen. Gromacs: fast, flexible, and free. *Journal of computational chemistry*, 26(16):1701–1718, 2005. (Cited on page 41)

M. van Kempen, S. S. Kim, C. Tumescheit, M. Mirdita, J. Lee, C. L. M. Gilchrist, J. Söding, and M. Steinegger. Fast and accurate protein structure search with foldseek. *Nature Biotechnology*, May 2023. doi: 10.1038/s41587-023-01773-0. (Cited on page 48)

M. Varadi, S. Anyango, M. Deshpande, S. Nair, C. Natassia, G. Yordanova, D. Yuan, O. Stroe, G. Wood, A. Laydon, A. Žídek, T. Green, K. Tunyasuvunakool, S. Petersen, J. Jumper, E. Clancy, R. Green, A. Vora, M. Lutfi, M. Figurnov, A. Cowie, N. Hobbs, P. Kohli, G. Kleywegt, E. Birney, D. Hassabis, and S. Velankar. AlphaFold protein structure database: massively expanding the structural coverage of protein-sequence space with high-accuracy models. *Nucleic Acids Research*, 50(D1):D439–D444, Nov. 2021. doi: 10.1093/nar/gkab1061. (Cited on page 48)

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. (Cited on page 6)

P. Velicković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. *ICLR*, 2018. (Cited on page 6)

P. Velicković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *ICLR*, 2018. (Cited on page 8)

H. Wang, T. Fu, Y. Du, W. Gao, K. Huang, Z. Liu, P. Chandak, S. Liu, P. Van Katwyk, A. Deac, et al. Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972):47–60, 2023a. (Cited on page 9, 40)

L. Wang, Y. Liu, Y.-C. Lin, H. Liu, and S. Ji. ComENet: Towards complete and efficient message passing for 3D molecular graphs. *Neural Information Processing Systems*, 2022a. doi: 10.48550/arXiv.2206.08515. (Cited on page 17, 18, 72)

L. Wang, Y. Zhou, Y. Wang, X. Zheng, X. Huang, and H. Zhou. Regularized molecular conformation fields. *Advances in Neural Information Processing Systems*, 35:18929–18941, 2022b. (Cited on page 37)

L. Wang, H. Liu, Y. Liu, J. Kurtin, and S. Ji. Learning hierarchical protein representations via complete 3d graph networks. In *ICLR*, 2023b. (Cited on page 17)

X. Wang and M. Zhang. Graph neural network with local frame for molecular potential energy surface. In B. Rieck and R. Pascanu, editors, *Learning on Graphs Conference, LoG 2022, 9-12 December 2022, Virtual Event*, volume 198 of *Proceedings of Machine Learning Research*, page 19. PMLR, 2022. (Cited on page 17)

Y. Wang, Z. Li, and A. B. Farimani. Graph neural networks for molecules. *arXiv preprint arXiv:2209.05582*, 2022c. (Cited on page 74)

Y. Wang, K. Yi, X. Liu, Y. G. Wang, and S. Jin. ACMP: Allen-cahn message passing with attractive and repulsive forces for graph neural networks. In *The Eleventh International Conference on Learning Representations*, 2023c. URL https://openreview.net/forum?id=4fZc_79Lrqs. (Cited on page 73)

J. L. Watson, D. Juergens, N. R. Bennett, B. L. Trippe, J. Yim, H. E. Eisenach, W. Ahern, A. J. Borst, R. J. Ragotte, L. F. Milles, et al. De novo design of protein structure and function with rfdiffusion. *Nature*, 620(7976):1089–1100, 2023. (Cited on page 43)

M. Weiler, M. Geiger, M. Welling, W. Boomsma, and T. S. Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. *Advances in Neural Information Processing Systems*, 31, 2018. (Cited on page 21, 37)

D. Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988. (Cited on page 43, 65)

Y.-P. Xiao, Y.-K. Lai, F.-L. Zhang, C. Li, and L. Gao. A survey on deep geometry learning: From a representation perspective. *Computational Visual Media*, 6:113–133, 2020. doi: 10.1007/s41095-020-0174-8. URL https://link.springer.com/article/10.1007/s41095-020-0174-8/fulltext.html. (Cited on page 40)

T. Xie and J. C. Grossman. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Phys. Rev. Lett.*, 2018. (Cited on page 17)

T. Xie, X. Fu, O. Ganea, R. Barzilay, and T. S. Jaakkola. Crystal diffusion variational autoencoder for periodic material generation. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. (Cited on page 42)

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *ICLR*, 2019. (Cited on page 71)

M. Xu, L. Yu, Y. Song, C. Shi, S. Ermon, and J. Tang. Geodiff: A geometric diffusion model for molecular conformation generation. In *International Conference on Learning Representations*, 2022. (Cited on page 42, 43)

K. Yan, Y. Liu, Y. Lin, and S. Ji. Periodic graph transformers for crystal material property prediction. In *NeurIPS*, 2022. (Cited on page 13)

Z. Yang, M. Chakraborty, and A. D. White. Predicting chemical shifts with graph neural networks. *Chemical science*, 12(32):10802–10809, 2021. (Cited on page 43)

C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34: 28877–28888, 2021. (Cited on page 73)

F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. (Cited on page 73)

A. Zee. *Group theory in a nutshell for physicists*. Princeton University Press, 2016. (Cited on page 19, 26)

X. Zhang, L. Wang, J. Helwig, Y. Luo, C. Fu, Y. Xie, M. Liu, Y. Lin, Z. Xu, K. Yan, et al. Artificial intelligence for science in quantum, atomistic, and continuum systems. *arXiv preprint arXiv:2307.08423*, 2023. (Cited on page 9, 40)

S. Zheng, J. He, C. Liu, Y. Shi, Z. Lu, W. Feng, F. Ju, J. Wang, J. Zhu, Y. Min, et al. Towards predicting equilibrium distributions for molecular systems with deep learning. *arXiv preprint arXiv:2306.05445*, 2023. (Cited on page 42)

C. L. Zitnick, L. Chanussot, A. Das, S. Goyal, J. Heras-Domingo, C. Ho, W. Hu, T. Lavril, A. Palizhati, M. Riviere, et al. An introduction to electrocatalyst design using machine learning for renewable energy storage. *Preprint arXiv:2010.09435*, 2020. (Cited on page 7, 9, 42, 48, 73)

L. Zitnick, A. Das, A. Kolluru, J. Lan, M. Shuaibi, A. Sriram, Z. Ulissi, and B. Wood. Spherical channels for modeling atomic interactions. *Advances in Neural Information Processing Systems*, 35:8054–8067, 2022. (Cited on page 37, 74)

A Lexicon

In this section, we provide background material on a variety of essential concepts to Geometric GNNs for 3D atomic systems.

A.1 Geometric vocabulary

1. *Scalars*: are quantities that do not have any direction or orientation and consequently remain unchanged under rotations or reflections. For example, the temperature or the energy of a system.
2. *(Geometric) vectors*: here we use the word vector to refer to an object that transforms with 'simple' rotation matrices $\mathbf{R}$, i.e. in the standard representation of $SO(3)$. In contrast to the general machine learning literature, where vectors are simply one-dimensional lists of values, vectors in our usage of the word are objects that have both magnitude and direction. In the context of 3D atomic systems, a geometric vector can represent geometric attributes such as position, velocity, or forces acting on atoms. They are often expressed as *Cartesian vectors*, that is, geometric vectors that are described within a Cartesian coordinate system (i.e. they have components along the coordinate axes (x, y, z) in 3D space). They have the same dimension as the space they exist in.
3. *Geometric Tensors*: are mathematical objects that generalize geometric vectors to higher-dimensional spaces. In contrast to general machine learning, in which tensor just means a multi-dimensional list of numbers, we use the word tensor to describe objects that transform in a consistent way under under group actions (rotations in $SO(3)$ for us). Scalars and vectors are two special subclasses of tensors that are so common that we gave them their extra name. Objects that still transform consistently under rotations, but not invariantly (i.e. with the identity) or through simple rotation matrices are tensors. An example of a tensor would be the moment of inertia tensor in physics. Tensors are essential in many areas of physics, for example in differential geometry and general relativity, where they describe quantities in curved spacetimes. In the context of 3D atomic systems, geometric tensors can be used to represent higher-order geometric attributes or relationships. For instance, the stress and strain[52] tensors in a material are second-order geometric tensors, capturing how the material deforms under external forces.
    - *Cartesian Tensors*: are geometric tensors whose components aligned with the coordinate directions of a Cartesian coordinate system. These tensors are used to represent quantities that are directly related to the physical axes of the coordinate system.
    - *Spherical Tensors*: are used to describe quantities that are invariant under rotations and have components aligned with spherical harmonics. They are particularly useful in problems involving spherical symmetry and rotational invariance.
4. The *tensor order* (of a feature): refers to the way a Cartesian tensor transforms under rotation. It indicates the level of complexity or the number of components needed to fully describe the feature. In the context of Geometric GNNs, features are often represented as tensors, where each dimension corresponds to a specific aspect or property of the feature. The tensor order determines the number of indices needed to access and manipulate the feature's values.
    - Scalar features are zero-order tensors because they do not have any additional dimensions or indices.
    - Geometric vectors are first-order tensors because they have one index to represent their components along different axes.
    - Geometric tensors have tensor order equal or greater than two. The strain tensor, for instance, has a tensor order of two because it requires two indices to access its entries. Higher-order tensors (e.g. spherical tensors) can have a tensor order greater than 2 as they involve more complex structures with multiple indices. These tensors can represent more intricate features, such as higher-order interactions or relationships between elements in a system.

[52]The strain tensor in a material describes how a material's shape deforms when subjected to mechanical loads. In a 3D Cartesian coordinate system, the strain tensor has components corresponding to the changes in lengths and angles in different directions.

5. The *type* of a tensor: is often used interchangeably with the *tensor order*, but they can have slightly different meaning depending on the context. In particular, the type of a feature tells us how it changes under symmetry transformation, i.e. how the tensor components change when the coordinate system or basis vectors are transformed.
6. The *body order* of a message, feature or model layer: refers to the number of atoms involved in the related computations. It is often used to describe the level of complexity (i.e. number of atoms considered) in a specific interaction model. For example, a two-body message involves pairwise interactions between two atoms (e.g. distances) while a three-body message involves interactions among three atoms (e.g. bond angles). We often use *many-body* interactions to describe interactions with more than 3-4 atoms [Batatia et al., 2022b].
7. The *frame of reference*, in the context of 3D atomic systems, refers to the specific spatial arrangement and atom ordering used to represent the system. Essentially, it is the viewpoint from which the researcher reads the atomic system.

Going further

→ An Introduction to Tensors for Students of Physics and Engineering (Kolecki, 2002) article technical

→ The Tensor Product, Demystified (math3ma, 2018) blog visual

→ How to Conquer Tensorphobia (Kun, 2014) blog technical

A.2 Group theory

Group theory is a branch of mathematics that studies the symmetries and transformations of objects. In the context of 3D Geometric GNN models, group theory is particularly relevant because it helps us capture and exploit the symmetries present in atomic systems.

Indeed, atoms in a molecule or in a material exhibit specific spatial arrangements and undergo transformations such as rotations, reflections, and translations. These transformations preserve the overall structure and properties of the system. Group theory allows us to formally describe and analyze these transformations, enabling us to uncover hidden relationships and patterns.

Formally, a **group** $(\mathcal{G}, \star)$ is a set of elements $\mathcal{G}$ together with a binary operation $\star : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ satisfying the following three conditions:

1. *Associativity*: $\forall \mathfrak{g}_1, \mathfrak{g}_2, \mathfrak{g}_3 \in \mathcal{G}$, we have $(\mathfrak{g}_1 \star \mathfrak{g}_2) \star \mathfrak{g}_3 = \mathfrak{g}_1 \star (\mathfrak{g}_2 \star \mathfrak{g}_3)$.
2. *Identity*: there exists an identity element $\mathfrak{e} \in \mathcal{G}$ such that $\forall \mathfrak{g} \in \mathcal{G}$, we have $\mathfrak{e} \star \mathfrak{g} = \mathfrak{g} \star \mathfrak{e} = \mathfrak{g}$
3. *Inverse*: each element has an inverse - that is, $\forall \mathfrak{g} \in \mathcal{G}$, $\exists \mathfrak{h} \in \mathcal{G}$ such that $\mathfrak{g} \star \mathfrak{h} = \mathfrak{h} \star \mathfrak{g} = \mathfrak{e}$.

We denote by $|\mathcal{G}|$ the size of a group $\mathcal{G}$, and call this the **order** of $\mathcal{G}$. If $(\mathcal{G}, \star)$ is a group and $\mathcal{H} \in \mathcal{G}$ is a subset such that $(\mathcal{H}, \star)$ satisfies the above group axioms, then we call $\mathcal{H}$ a **subgroup** of $\mathcal{G}$, which we write as $\mathcal{H} \leq \mathcal{G}$.

Now that we have seen what a group is, let's see the **different symmetry groups of interest** for 3D atomic systems, illustrated in Figure 22.

1. $\mathrm{E}(d)$: the Euclidean group includes translations, rotations, and reflections. It represents all possible transformations in d-dimensional Euclidean space.
2. $\mathrm{O}(d)$: the orthogonal group represents rotations and reflections. $\mathrm{O}(d) \leq \mathrm{E}(d)$.
3. $\mathrm{SO}(d)$: the Special Orthogonal group consists of rotations without reflections, in the d-dimensional space. $\mathrm{SO}(d) \leq \mathrm{O}(d)$.
4. $\mathrm{SE}(d)$: the Special Euclidean group combines translations and rotations. $\mathrm{SE}(d) \leq \mathrm{E}(d)$.
5. $\mathrm{T}(d)$: the Translation group represents the symmetry transformations of pure spatial translations in d-dimensions, without any rotation or reflection. $\mathrm{T}(d) \leq \mathrm{SE}(d)$.

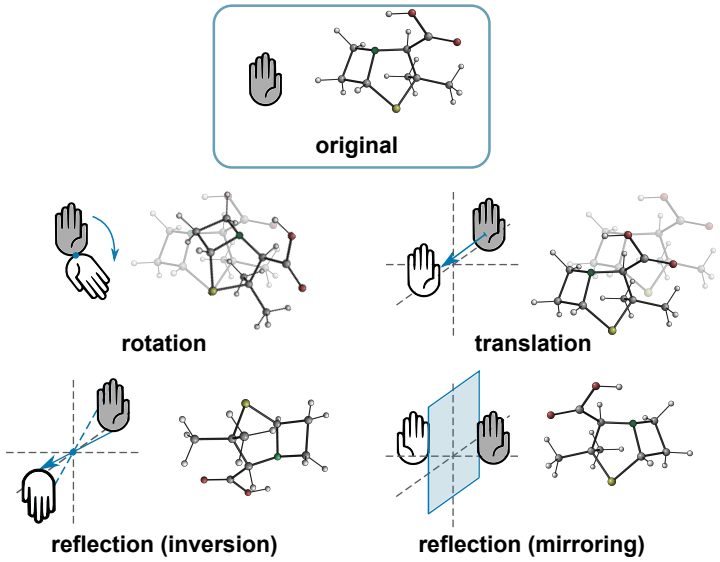


Figure 22: Illustration of the different euclidean symmetries for 3D atomic systems. Source: [Atz et al., 2021].

6. S_n: the symmetric group is the group of all permutations of the set of n atoms.

For example, for a rotation angle θ, a rotation matrix around the z-axis around would be written:

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \in \mathrm{SO}(3)$$

What it means for Geometric GNNs?

Typically, we would like Geometric GNNs to exhibit $\mathrm{E}(3)$ or $\mathrm{SE}(3)$-equivariance. While rotations and translations do not pose a concern, reflections do. In isolation, a molecule and its mirror image share the same internal features and properties, regardless of its chirality. Since ML datasets often showcase molecules in isolation, $\mathrm{E}(3)$-equivariance is desirable. However, molecular functionality is most often conferred by intermolecular interactions with surrounding components, meaning that a molecule's properties may differ from those of its mirror image. In such cases, we no longer require equivariance to reflections, making $\mathrm{SE}(3)$-equivariance desirable.

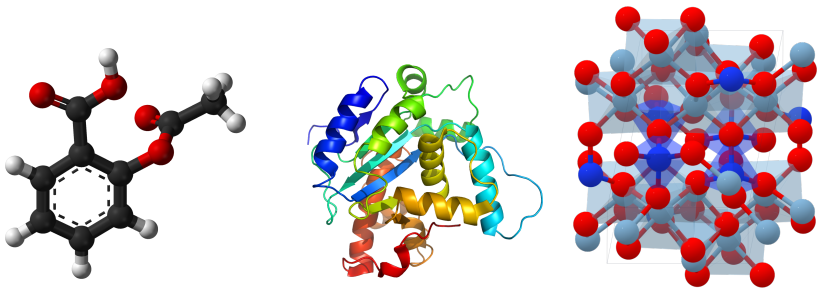
A.3 Data structures

All data structures defined below: molecules, proteins and material, are encapsulated under the term "atomic systems". Alternatively, we refer to them as "molecules and materials".

A.3.1 (Small) Molecules

A molecule is a group of atoms bonded together in a non-periodic manner, forming the basic unit of a substance. Molecules can exist in different forms like gases, liquids, or solids. Atoms, which are the building blocks of molecules, combine in specific ways to create molecules with unique structures and properties. Scientists analyze molecular structures to understand how they interact, participate in chemical reactions, and contribute to the properties of substances. This knowledge leads to advancements in various domains including medicine,, technology, environmental science, etc.

In the field of machine learning and computational chemistry, molecules are typically described using different representations (see Figure 24). Traditional approaches have focused on 1D descriptions



(a) Small molecule (b) Protein (c) Material

Figure 23: Examples of different data structures.

such as molecular fingerprints or SMILES strings, and 2D topology graphs, where atoms and bonds are represented by nodes and edges, respectively. To construct meaningful representations that capture the molecule's topology, scientists have opted for Graph Neural Networks on 2D graphs due to their ability to efficiently account for atomic interactions through message passing mechanisms.

Going beyond 2D graphs, there has been an increasing recognition of the importance of considering the 3D geometric conformations[53] of molecules for property prediction tasks. This highlights the necessity of leveraging 3D structures to enhance the understanding and prediction of molecular properties.

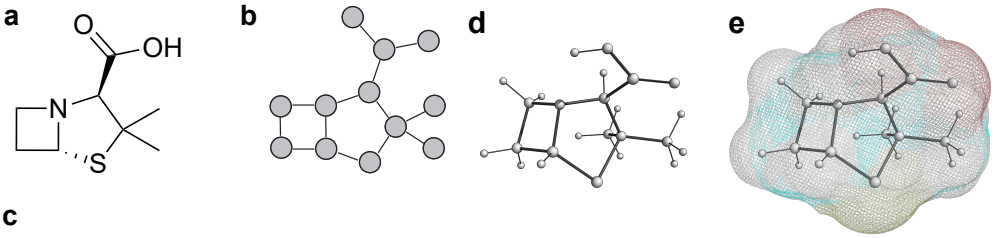


Figure 24: Exemplary molecular representations for a targeted molecule (i.e., the penam substructure of penicillin). **a.** 2D Kekulé structure. **b.** 2D molecular graph. **c.** SMILES string Weininger [1988], in which atom type, bond type and connectivity are specified by alphanumerical characters. **d.** 3D graph with atom positions. **e.** Molecular surface represented as a mesh colored according to the respective atom types. Source: [Atz et al., 2021].

Going further

→ Molecular representations for machine learning applications in chemistry (Raghunathan, 2021) article visual technical

A.3.2 Proteins

A protein is a large molecule that plays a crucial role in various biological processes in living organisms. Protein sequences are made up of smaller units called amino acids, which are connected together in a specific order to form a long chain called a polypeptide.

Proteins have a unique 3D structure that is essential for their function. This structure can be divided into different levels. The primary structure refers to the linear sequence of amino acids in the polypeptide chain. The secondary structure describes the local folding patterns that arise from interactions between neighboring amino acids. Examples of secondary structures include helices and sheets. The tertiary structure represents the overall 3D arrangement of the protein, including its

[53] conformations represent the 3D structure of an atomic system: the specific arrangement of atoms, their bond lengths, bond angles, and torsion angles. Different conformations often impact molecular properties.

folds and twists. Finally, in some cases, multiple protein chains can come together to form a complex known as the quaternary structure.

Scientists can use ML to modelize and analyse the 3D structure of proteins. Doing so, they can gain insights on how it interacts with other molecules and performs specific tasks in the body. This understanding is crucial for various fields including medicine, buichemistry and drug discovery.

Going further

→ Protein–protein interaction prediction with deep learning: A comprehensive review (Soleymani, 2022) article

→ Paper list github.com/lirongwu/awesome-protein-representation-learning

A.3.3 Solid-State Materials

Solid-state materials are characterised by their composition and structure leading to unique properties making them amendable to a diverse set of applications. Solid-state materials lose their fixed structure when they transition to liquid or gas phases, which have less restrictions on what spaces atoms can occupy. The solid, or condensed matter, state lends itself to a a fixed structure and definite volume. Solids exhibiting a very regular, periodic structure are referred to as crystals while solids with no such positional large scale order are called amorphous.

Crystals, also known as crystalline materials, are solid substances characterized by a periodic arrangement of atoms and molecules in all spatial directions. A crystal structure includes atom types and positions as well as the translational axes along which the structure repeats. Since crystal structures extend infinitely, scientists define the smallest part needed to fully define the crystal as the *unit cell*. This unit cell describes the crystal structure and composition, as well as how the crystal is repeated in space. This repeated ordered structure gives crystals unique shapes and properties, e.g. transparency or high melting points, making it crucial for the development of new materials with customised properties. As a result, ML methods often attempt to harness this periodicity to be as accurate as possible in their predictions.

Going further

→ Self-supervised learning of materials concepts from crystal structures via deep neural networks (Suzuki, 2022) article technical

→ Equivariant Networks for Crystal Structures (Kaba, 2023) article

A.4 Periodic boundary conditions

Periodic boundary conditions (pbc) are commonly used to represent the infinite repeating nature of crystals and other periodic systems. Instead of considering just a single 3D unit cell where all atoms would lie, one imagine an infinite array of identical unit cells extending in all directions.

To incorporate this periodicity into simulations, one imposes pbc on the simulation cell. This means that if an atom or molecule exits the cell on one side, it re-enters on the opposite side as if it were moving through a periodic lattice. By doing this, scientists can simulate the behavior of an infinite crystal using a finite computational domain.

In practice, we implement pbc by defining a simulation cell $\vec{\mathbf{c}}$ and a set of cell offsets $\vec{\mathbf{o}}_{ij} = [a, b, c]$ where $a, b, c \in \{0, 1, -1\}$ are associated with each edge (i, j) of the graph to encode edges across neighboring unit cells. These cell offsets thus help determine distances between pairs of atoms while considering the periodic repetition of the crystal lattice. Distances under pbc are written $d_{ij} = ||(\mathbf{x}_i - \mathbf{x}_j) + \vec{o}_{ij} \cdot \vec{\mathbf{c}}||$. Figure 8 provides a visual illustration.

In different terms, when two atoms are adjacent across a cell boundary, we calculate their distance by accounting for the periodicity of the lattice. For instance, if the cells are on top of each other (wrt z-axis), the cell offset vector would be $[0, 0, -1]$ or $[0, 0, 1]$. By considering these pbc, we can more accurately model the behavior of crystals and other periodic systems.

Note that, in the context of PBCs, the Euclidean group actions must be extended to account for rotations not only within the primary unit cell but also across its periodic replicas. This means that the Geometric GNN must also exhibit equivariance to group actions that extend across neighboring unit cells, allowing for a robust representation of euclidean symmetries in the presence of periodicity.

Going further

→ Molecular Dynamics: Periodic Boundary Conditions (Mcelfresh, 2020) blog visual

→ Periodic Boundary Conditions (Digital Research Alliance of Canada, 2023) blog visual

A.5 Quantum chemistry background

Quantum chemistry serves as the foundation for understanding the behavior of atoms and molecules at the atomic scale. It provides insights into the interactions of atoms and electron leading to a better understanding of electronic structures, molecular interactions, and the fundamental mechanisms governing chemical processes. This section outlines some key concepts that form the basis for machine learning projects applied to physics and chemistry.

In quantum chemistry, electrons govern how atoms form interatomic (i.e. between atoms) bonds and thereby interact with each other. Their behaviour is modeled using *wave functions*, meaning, by probability distributions expressing the likelihood of finding an electron at a specific location around an atom. Having knowledge about electrons' wave functions is essential because they hold the key to understanding how atoms in a given structure behave, therefore unlocking insights into chemical reactions, material properties and behaviors that are critical to understanding current materials systems and potential new designs.

The *Schrödinger equation* is the fundamental equation in quantum mechanics that governs the wavefunctions' behavior. The equation incorporates a Hamiltonian operator, which represents the total energy of the system. Solving the Schrödinger equation for a given system yields the wavefunctions and corresponding energy levels for its particles. However, it is mathematically intractable beyond the Hydrogen atom.

The *Density Functional Theory* (DFT) is a computational approach which determines the electronic structure of molecules and solids by approximating electron density, i.e. the Schrödinger equation using diverse sets of functionals for different atomic systems. While computationally tractable for many systems, running DFT remains computationally expensive and becomes impractical for intricate systems, we propose the use ML to efficiently approximate DFT calculations. ML models hold the potential to significantly reduce computational costs while maintaining accuracy, making them invaluable tools for researchers.

For instance, ML models that efficiently and accurately approximate DFT-based energy calculations could be used to model the relationship between the potential energy of a molecule and its nuclear coordinates, called the *potential energy surface* (PES). The PES provides valuable insights into molecular stability, chemical reactions, and the geometry of molecules. Analyzing it helps us understand how molecules interact, react, and transition between different energy states.

Opinion

While a deep understanding of quantum chemistry is not "required" to use and develop Geometric GNNs, being familiar with basic quantum chemistry principles may enhance your comprehension of the field as well as your ability to design more meaningful models. Besides, running DFT (or any other quantum chemistry numerical method) remains essential to construct bigger and more versatile databases to train machine learning models on.

Going further

→ How do we model atoms? video

A.6 Energy conservation

The potential energy of a system represents the energy stored within it. It arises from the interplay of attractive and repulsive forces between atoms, which are determined by factors such as atom types and atom positions. The potential energy represents the energy that can be released when these components undergo positional changes, determining the stability and behaviour of the system. When the force field is conservative, the forces experienced by the atoms can be derived from the potential energy by taking the derivative with respect to their positions,

$$E = -\int F(\vec{x})d\vec{x}, \quad \text{and} \quad F = -\nabla E(\vec{x}) \tag{53}$$

By definition, these forces will always tend to minimize the potential energy, driving the system towards a relaxed state, analogous to gradient descent optimization.

In ML models for chemical systems, a single neural network is used to predict the total energy of the system by summing the contributions from individual atoms. To maintain energy conservation, the model calculates the forces on each atom by computing the derivative of the final predicted energy with respect to the atom's position. This allows for accurate predictions of both energy and forces, enabling efficient exploration and analysis of chemical systems.

In practice, this energy conservation requirement is also incorporated into the model's loss function, which includes a new term calculating the difference between predicted forces (obtained through backpropagation of the predicted energy) and ground truth forces (energy conserving). By considering the forces, the model adheres to the fundamental principle of energy conservation and constrains the space of functions explored by the neural network.

However, it is worth noting that this energy conserving requirement may hamper model performance. In certain cases, breaking free from this constraint may be beneficial. For example, in the Open Catalyst Project [Chanussot et al., 2021], non-energy conserving models[54] outperformed energy-conserving ones [Gasteiger et al., 2021, Shuaibi et al., 2021, Duval et al., 2022]. Whether to strictly enforce the energy conservation principle remains an active area of research. Furthermore, energy conservation itself does not necessarily yield accurate approximation of forces given that modeling function does not guarantee effectively modeling its gradient. As such further research is needed to explore the effects of energy and force conservation in atomic systems.

A.7 GNN architectural details

A.7.1 Message Passing

Message passing is often used to describe the functioning of the family of Graph Neural Networks (GNN) models. Why? Because updating the representation of each node i can be seen as passing a message from neighbouring nodes ($j \in \mathcal{N}_i$) to the node of interest. In its simplest form, the message $\mathbf{m}_{ij}$ is computed via a learnable message function $\underline{f_1}$ of the neighbour's representation $\boldsymbol{s}_j$ and $\boldsymbol{s}_i$. The updated representation at layer (t+1) is obtained by applying a learnable update function $\underline{f_2}$ on the aggregated messages $\oplus_{j\in\mathcal{N}_i}\mathbf{m}_{ij}$ coming from neighbouring nodes and the existing representation $\mathbf{s}_i$. $\oplus$ is not learnable and often denotes the sum or mean operator.

$$\begin{aligned}\mathbf{m}_{ij} &= \underline{f_1}(\mathbf{s}_i^{(t)}, \mathbf{s}_j^{(t)}) \\ \mathbf{s}_i^{(t+1)} &= \underline{f_2}(\mathbf{s}_i^{(t)}, \oplus_{j\in\mathcal{N}_i}\mathbf{m}_{ij})\end{aligned}$$

Message Passing has proven very successful so far but it also comes with its set of own limitations. Among them we find *over-smoothing*, where node features become too similar after multiple message passing layers, loosing discriminative power due to excessive aggregation, and *over-squashing*, which denotes an excessive information compression through bottleneck edges, likely leading to information loss [Giraldo et al., 2023].

[54] These models predict forces from final atom representations using a separate neural network and include a loss term about force predictions.

Going further

→ A Gentle Introduction to Graph Neural Networks (Distil, 2021) blog visual

→ Graph Representation Learning Book (Hamilton, 2020) book

A.7.2 Activation function

As for any deep learning model, the choice of non-linear activation function plays a central role in modeling complex non-linearities of atomic interactions. The Rectified Linear Unit (**ReLU**) activation [Glorot et al., 2011] is widely used in many deep learning models [Sanchez-Gonzalez et al., 2020, Bapst et al., 2020]. However, given the inherent nature of forces, ReLU may not be ideal to model atomic forces because its output is modeled as piece-wise linear hyperplanes with sharp boundaries. As a result, a wide array of alternatives have been explored: Tanh, Leaky-ReLU [Maas et al., 2013], SoftPlus [Dugas et al., 2001], Shifted SoftPlus [Schütt et al., 2018], and Swish [Ramachandran et al., 2017].

While the choice of activation function ultimately depends on the specific problem domain and dataset characteristics, Geometric GNNs most commonly use the **Swish** activation $\text{swish}(x) = x \cdot \text{sigmoid}(x)$ as it offers several advantages. It provides a smoother output landscape and has non-zero activation for negative inputs. This smoothness is essential when dealing with atomic systems, ensuring that small perturbations in the input space result in gradual changes in the output space, which promotes stability and avoids abrupt changes in predictions. Additionally, Swish has a smooth gradient, making backpropagation more stable and efficient, mitigating issues like vanishing or exploding gradients in deep networks. Finally, its non-linear behavior allows Geometric GNNs to model complex molecular interactions and spatial relationships effectively.

Opinion

The swish activation works well but the community should not stop looking for better alternatives, endowed with similar desirable properties.

A.7.3 Basis functions

In Geometric GNNs, the choice of basis function $\psi : \mathbb{R}^d \to \mathbb{R}^a$ is essential to transform geometric information, i.e. atom relative positions $\vec{x}_{ij}$, into discriminative representations. In different words, basis functions encode the spatial relationship between atoms, enabling Geometric GNNs to effectively model the structural properties of the system. The dimension a of the encoded geometric information is a hyperparameter depending on the choice of the basis function. It is usually chosen to be significantly larger than the default dimensionality to capture fine-grained distinctions in atom positions. Below, we describe the most widespread basis functions:

1. **Identity:** $\psi_{\text{id}}(\vec{x}_{ij}) = \mathbf{x}_{ij}$. The standard basis function uses the "scalarised" geometric information for each edge (i, j), basically keeping the same values but loosing its geometric aspect ($\vec{x} \in \mathbb{R}^d$ to $\mathbf{x} \in \mathbb{R}^d$).
2. **Radial Basis Function** (RBF): $\psi_{\text{rbf}}(\vec{x}_{ij}) = [\psi_1, \dots, \psi_a]$, where ψ_k is the output of the k-th basis function $\psi_k(\vec{x}_{ij}) = \text{e}^{(\|\vec{x}_{ij}\| - \mu_k)^2/(2\cdot\sigma^2)}$ representing the distance between atoms (i, j), encoded using Gaussian functions that decay with distance. By considering pairwise distances, the RBF basis function can encode the varying degrees of influence that atoms exert on each other based on their spatial proximity. The Gaussian means are evenly distributed on $[0, 1]$, i.e., $\mu_k = k/(a-1)$ and the standard deviation is $\sigma = 1/(a-1)$. Values of $\vec{x}_{ij}$ are often normalized to lie in $[0, 1]$. ψ_{rbf} results in a a-dimensional vector.
3. **Sine:** $\psi_{\text{sin}}(\hat{x}_{ij}) = [\psi_1, \dots, \psi_b]$, where ψ_k is the output of the k-th basis function $\psi_k(\hat{x}_{ij}) = \sin(1.1^k \hat{x}_{ij})$. This design is based on function approximation using the Fourier series. ψ_{sin} is applied to each dimension of the unit vector $\hat{x}_{ij}$, resulting in an $a = b \times d$ vector.
4. **Spherical Harmonics:** $\psi_{\text{sph}}(\hat{x}_{ij}) = [\vec{\mathbf{Y}}^{(0)}(\theta, \phi), \dots, \vec{\mathbf{Y}}^{(L)}(\theta, \phi)]$, where the polar and azimuthal angles θ and ϕ are directly computed from the unit directional vector $\hat{x}_{ij} \in$

$\mathbb{R}^3, \|\vec{x}_{ij}\| = 1$. The vector-valued function $\vec{\mathbf{Y}}^{(l)} : S^2 \to \mathbb{R}^{(2l+1)}$ is the list of Laplace's spherical harmonics $\vec{Y}_m^{(l)} : S^2 \to \mathbb{R}$ with order $l \geq 0$ and degree $m \in \{-l, \dots, l\}$ [MacRobert, 1947]. The spherical harmonics are special function which encode angular information from the surface of the sphere, forming an orthonormal basis for Fourier transformations of functions on the sphere, like sine waves on $\mathbb{R}$. They are equivariant in $\mathrm{SO}(3)$, transforming in a predictable manner when the input is rotated by $\mathbf{R} \in \mathrm{SO}(3)$:

$$Y_m^{(l)}(\mathbf{R} \cdot \hat{x}_{ij}) = \sum_{m'} \vec{D}_{mm'}^{(l)}(\mathbf{R}) \vec{Y}_{m'}^{(l)}(\hat{x}_{ij})$$

where $\vec{D}_{mm'}^{(l)}$ are the entries of the Wigner-D matrix $\vec{\mathbf{D}}^{(l)} \in \mathbb{R}^{(2l+1)\times(2l+1)}$. For Geometric GNNs, spherical harmonics are often used as convolutional filter (or simple input feature) as they enable to capture the spatial orientation and angular relationships between atoms. The list of spherical harmonics is displayed here and an implementation is available here.

5. **MLP:** $\psi_{\text{MLP}}(\vec{x}_{ij}) = \sigma(\underline{\boldsymbol{W}} \cdot \vec{x}_{ij} + \underline{\boldsymbol{b}})$, where $\sigma(\cdot)$ is a non-linear activation function, and $\underline{\boldsymbol{W}}$ and $\underline{\boldsymbol{b}}$ are learnable parameters. This method is flexible because it can be applied to any geometric quantity (atom relative position, concatenation of scalar distance and angles, [Duval et al., 2023]); and powerful because two-layers MLP with enough hidden layers are universal approximators. However, the MLP basis function cannot preserve equivariance due to the presence of non-linearities.

Going further

→ Radial Basis Functions, RBF Kernels, & RBF Networks Explained Simply (Ye, 2020) blog visual

→ Achieving Rotational Invariance with Bessel-Convolutional Neural Networks (Delchevalerie, 2021) article video

→ Spherical Harmonic blog

A.7.4 Examples of architecture

In this subsection, we display the architecture of **GemNet** [Gasteiger et al., 2021] to demonstrate that best-performing invariant methods often come at the cost of a complex functioning. We also describe below the well known SchNet [Schütt et al., 2017] architecture, having as main objective to depict the widely used *continuous convolution*.

SchNet introduced the use of continuous filters to handle unevenly spaced data, e.g. atoms at arbitrary positions. Given 3D atom input positions $\vec{\mathbf{x}} \in \mathbb{R}^{n\times 3}$, the continuous-filter convolutional layer t requires a filter-generating function

$$\underline{\psi}^t : \mathbb{R}^3 \to \mathbb{R}^a,$$

that maps positions to the corresponding discrete convolution filter values. This learnable filter generating function is modeled with a neural network where the (invariant scalar) output $\mathbf{s}_i^{t+1}$ of the continuous convolutional layer at position $\vec{x}_i$ is given by

$$\mathbf{s}_i^{t+1} = \sum_{j\in\mathcal{N}_i} \mathbf{s}_j^t \odot \underline{\psi}^t(\vec{x}_{ij}) = \sum_{j\in\mathcal{N}_i} \mathbf{s}_j^t \odot \underline{f}^t(\psi(d_{ij})) \tag{54}$$

where "$\odot$" represents the element-wise multiplication, $\underline{f}$ represents a two-layer MLP with softplus activation function and ψ a radial basis function[❖] . These feature-wise convolutions are applied for computational efficiency. The interactions between feature maps are handled by separate object-wise or, specifically, atom-wise layers in SchNet.

Going further

→ Graph ML in 2023: The State of Affairs, (Galkin, 2022) blog visual

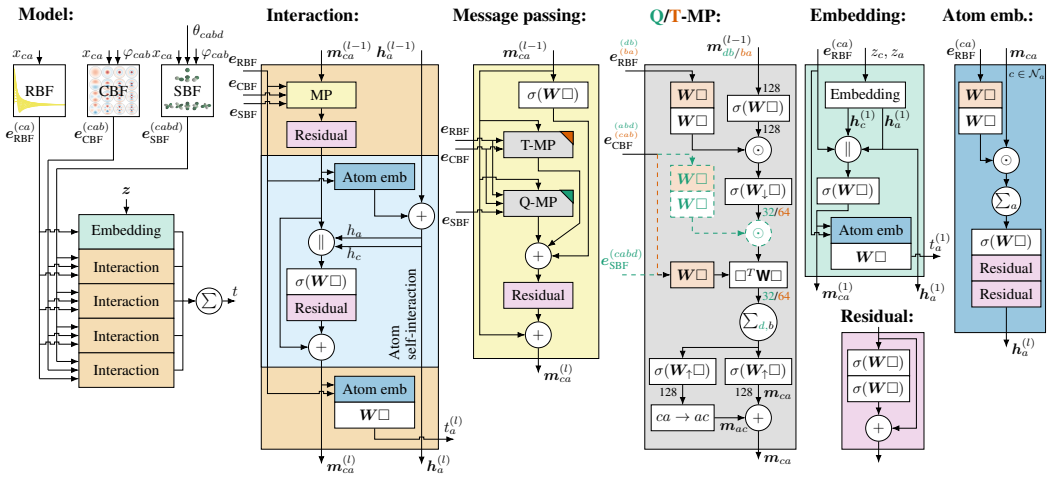


Figure 25: The full GemNet architecture. □ denotes the layer's input, $\|$ concatenation, σ the SiLU non-linearity, and orange a layer with weights shared across interaction blocks. Differences between two-hop message passing (Q-MP) and one-hop message passing (T-MP) are denoted by dashed lines. Numbers next to connecting lines denote embedding sizes. Taken from the original paper [Gasteiger et al., 2021].

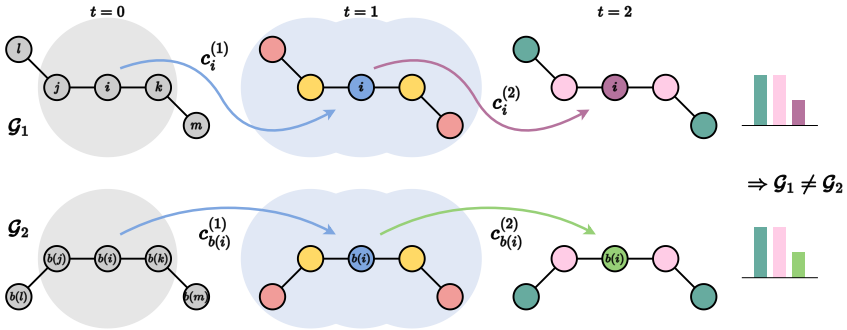


Figure 26: The Geometric Weisfeiler-Leman Test, an upper bound on the expressive power of equivariant GNNs [Joshi et al., 2023a]. GWL distinguishes non-isomorphic geometric graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ by injectively assigning colours to distinct neighbourhood patterns, up to global symmetries (here $O(d)$). Each iteration expands the neighbourhood from which geometric information can be gathered (shaded for node i). Example inspired by Schütt et al. [2021b].

A.8 Expressive power of Geometric GNN

The graph isomorphism problem and the Weisfeiler-Leman (WL) test for distinguishing non-isomorphic graphs have become a powerful tool for analysing the expressive power of traditional GNNs [Jegelka, 2022]. It was shown by Xu et al. [2019], Morris et al. [2019] that message passing GNNs are at most as powerful as WL at distinguishing non-isomorphic graphs and suffer from the same failure modes as WL. The WL framework has since become a major driver of progress in designing more expressive GNNs [Dwivedi et al., 2023, Bodnar et al., 2021].

However, WL does not directly apply to geometric graphs as they exhibit a stronger notion of geometric isomorphism that must accounts for spatial symmetries. Two geometric graphs can only be *geometrically isomorphic* if the underlying graphs are isomorphic *and* the geometric attributes are equivalent, up to global group actions like rotations and reflections.

Joshi et al. [2023a] recently proposed the Geometric WL (GWL) framework for characterising the expressivity of Geometric GNNs by their ability to solve geometric graph isomorphism, i.e. to provide distinct representations for any two different geometric graphs, up to group actions. In addition to their theoretical contributions, they proposed several synthetic experiments to test new Geometric

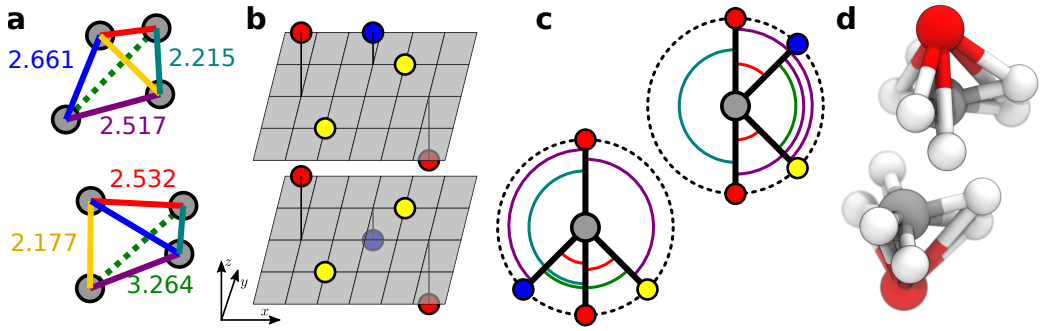


Figure 27: Counterexamples from [Pozdnyakov and Ceriotti, 2022] where pairs are distinct but cannot be discriminated by the unordered list of distances or distances and angles between atoms. They were created to demonstrate that some Geometric GNNs cannot distinguish between these pairs of structures using k-body scalarisation. (a) The two tetrahedra share the same list of pairwise distances, as per color coding. (b) The two structure share the same list of pairwise distances, and in addition the list of distances of each atom relative to its neighbors. (c) The two environments share the same list of distances and angles relative to the central (gray) atom. (d) The two environments share the same list of distances, angles and tetrahedra around the central (gray) atom.

GNNs' expressivity in a practical manner. For instance, one task asks models to distinguish between several counterexamples from Pozdnyakov and Ceriotti [2022]. These edge cases consists of pairs of configurations that are indistinguishable when comparing their set of k-body scalars, illustrated in Figure 27. In other terms, any model for atom-centred properties that uses 2, 3 or 4-body order features will incorrectly give identical results for the different configurations. Hence, Geometric GNNs that do not process enough information to uniquely distinguish two atomic systems, such as SchNet or DimeNet, automatically fails this experiment.

Opinion

Under the GWL framework, Geometric GNNs require injective aggregation, update and readout functions to be maximally expressive or universal. While some current architectures have claimed to be universal or *complete* under specific conditions [Gasteiger et al., 2022, Wang et al., 2022a], we believe that a provably universal, equivariant GNN on sparse graphs with finite tensor and body order remains an open question.

Going further

→ On the Expressive Power of Geometric Graph Neural Networks. article

→ Geometric GNN Dojo: Reference implementations and synthetic experiments to evaluate Geometric GNN expressivity in practice. visual

A.9 On inductive biases

A major research area in deep learning is to find ways to express preferences over the kinds of functions we would like to solve our problems. For instance with Maximum A Posteriori (MAP) estimation one could use a prior distribution over a model's parameters to express a preference, a prior belief, over the possible values of the function's parameters. Those preferences are called *inductive biases* because they are baked-in the algorithms in ways that voluntarily restrict the accessible function space during learning, or favour specific regions. Those restrictions are most often designed to improve data efficiency (the number of data points required to get to some level of performance) and/or generalization (the ability to keep performing well on new data). Typical inductive biases include: parameter regularization (as in the MAP example) [Bishop, 1995, Srivastava et al., 2014, Nusrat and Jang, 2018], enforcing invariance or equivariance to certain input transformations [Lee et al., 2019, Rowley et al., 1998, Bronstein et al., 2017] (as is the case for Geometric GNNs) or

into the very architecture of the neural network [He et al., Yu and Koltun, 2016, Battaglia et al., 2018, Sherstinsky, 2020, Wang et al., 2023c]. Discovering, leveraging and evaluating inductive biases is therefore of paramount importance when developing learning approaches to solve real-world problems, especially when data (or compute) is scarce.

Going further

→ Inductive Biases for Deep Learning of Higher-Level Cognition (Goyal & Bengio, 2020) article technical

→ Useful Inductive Biases for Deep Learning in Molecular Science (Welling, 2022) video

B An opinionated history of methods

Similarly to other field, the first prediction models for molecular energy and forces relied on hand-crafted representations [Behler, 2016, Halgren, 1996, Chmiela et al., 2018] built on physical properties. This was the case until recently, where research moved to end-to-end machine learning models based on Graph Neural Networks [Gori et al., 2005].

Multiple work came expanding the application of ML techniques to a broad set of materials modeling tasks ranging from solid-state [Zitnick et al., 2020, Miret et al., 2023, Lee et al., 2023b] to molecular [Hoja et al., 2021, Ramakrishnan et al., 2014] structures. Most existing GNN architectures have tried to incorporate physics-informed 3D symmetries, either directly in the model architecture, making model predictions explicitly invariant or equivariant to the desired transformations, or via the data. There are Geometric GNNs.

A first line of Geometric GNNs were constructed to be **E(3)-invariant** by extracting scalar representations from atomic relative positions [Unke and Meuwly, 2019, Klicpera et al., 2020, Liu et al., 2021, Shuaibi et al., 2021, Ying et al., 2021, Adams et al., 2022]. The evolution was steady as we went from SchNet [Schütt et al., 2017] using pair-wise distances, to DimeNet [Klicpera et al., 2020] adding bond angles, to SphereNet [Liu et al., 2022] and GemNet [Gasteiger et al., 2021, 2022] additionally incorporating torsion angles (i.e. quadruplet of atoms). The scalarisation of geometric information enables to apply traditional message passing schemes (with any non-linearity) and the use of additional information enables models to distinguish a larger set of atomic systems. Nevertheless, this comes at a greater computational cost since GemNet must look at 3-hop neighbourhoods to compute torsion information for each update step, at both training and inference time. Besides, these models extract a set of pre-defined scalars representing geometric information and cannot represent equivariant properties directly, due to their invariant nature.

In parallel, **Equivariant GNNs with Spherical Coordinates** [Thomas et al., 2018, Anderson et al., 2019, Fuchs et al., 2020, Brandstetter et al., 2021, Batatia et al., 2022b, Frank et al., 2022], also called Spherical EGNN, focused on enforcing equivariance using irreducible representations of the SO(3) group. Such models build upon the concepts of steerability and equivariance introduced by Cohen and Welling [Cohen and Welling, 2016]. They use spherical tensors as node embeddings and ensure equivariance to SO(3) by placing constraints on the operations that can be performed [Kondor et al., 2018]. Specifically, they compute linear operations with a generalized tensor product between the atom embeddings and edges' directions. Tensor Field Networks [Thomas et al., 2018], NequIP [Batzner et al., 2022], SEGNN [Brandstetter et al., 2021], MACE [Batatia et al., 2022b], Allegro [Musaelian et al., 2022], Equiformer [Liao and Smidt, 2023], and various others lie in this category. They are commonly referred to as *e3nn networks*. While these methods are expressive and generalize well, they can be hard to implement, constraint a lot the functional space and optimization landscape, and very computationally expensive at training and inference (due to the Clebsch-Gordan tensor product with spherical representations).

For this reason, **Equivariant GNNs with Cartesian Coordinates** [Schütt et al., 2021a, Satorras et al., 2021b, Thölke and De Fabritiis, 2022], also called Cartesian EGNNs, started to model equivariant interactions in the Cartesian space, updating both scalar and vector representations. These GNNs achieve good performance while being relatively fast by avoiding the expensive equivariant operations of Spherical EGNNs. The authors manually design two separate sets of functions to deal with each type of representation and often use complex operations to mix their information, restricting the

set of possible operations (e.g. vector element-wise dot product) to preserve equivariance.. As downsides, in addition to rendering the global architecture hard to understand, the decomposition of Cartesian tensors into spherical tensors offers many nice properties that Cartesian EGNNs lack. For instance, they lack structured, hierarchical and compact representation of geometric information, which improves model efficiency and memory utilization. They also don't capture information across different angular directions, which makes the model more sensitive to angular variations and dependencies present in atomic interactions.

Overall, we have witnessed in recent years incredible progress in terms of model architectures across invariant and equivariant GNNs. However, all above approaches restrict the model learning space, reducing the number of possible operations inside the model architecture. Although this is theoretically desirable, in practice, it may hamper the learning capacity of the model. For this reason, we have seen the appearance of **unconstrained Geometric GNNs**. Unconstrained GNNs do not enforce symmetries via the model architecture. Instead, they attempt to implicitly learn them through simple data augmentation [Hu et al., 2021]; they relax them [Zitnick et al., 2022] or they enforce them by mapping input data to a unique canonical space of all euclidean representations [Duval et al., 2023].

In this work, we attempt to **bridge an existing gap in the literature** by providing a holistic and opinionated overview of the field of Geometric GNNs, encompassing all important aspects. While we are the first work of such kind, we acknowledge the presence of some great recent works also attempting to bridge this gap [Wang et al., 2022c, Han et al., 2022, Liu et al., 2023, Atz et al., 2021]. Wang et al. [2022c] focuses on the the molecular applications of GNNs; Liu et al. [2023] proposes a benchmarking platform for various GNNs and molecular datasets; Han et al. [2022], Atz et al. [2021] offer a short summary of the field. Overall, they all provide a descriptive and relatively general overview of the field, not always specific to Geometric GNNs. Our distinguishing contributions can be listed as follows: (1) a thorough description of all steps and variations in the Geometric GNNs modeling pipeline; (2) a novel taxonomy of methods containing a clear mathematical relation between them; (3) a concise description of how Geometric GNNs power different applications, with associated datasets; (4) a detailed list of promising future research directions, plus opinions on many ongoing discussions that divide the field; (5) the inclusion of almost all contextual material needed to understand the field; (6) a new notation scheme; (7) an exhaustive list of approaches and datasets that need to be updated by the community. We hope that our work will be useful to the whole community of researchers, helping experienced ones to efficiently navigate the field and guiding newcomers to integrate it.

Going further

→ Towards Geometric Deep Learning (Bronstein, 2023) blog

C Data

C.1 Data splits

1. **Random split**: ensures training, validation, and test data are sampled from the same underlying probability distribution.
   - *Pros*: Simple to implement.
   - *Cons*: May not preserve the underlying distribution of data. May result in variability in performance due to random variations in the data split. May not capture specific challenges or biases in the data.
   - *When to use?* They can provide a good baseline evaluation, especially when the dataset is well-balanced and representative.
2. **Stratified split**: The dataset is divided while maintaining a similar class distribution across sets (e.g., balanced representation of active and inactive compounds).It ensures each of the training, validation, and test sets to cover the full range of provided labels.
   - *Pros*: Helps ensure that each split contains a representative distribution of different classes or properties, reducing the risk of biased evaluations.

- *Cons*: Requires class or property information for stratification, which may not always be available or applicable. Small or imbalanced datasets may still pose challenges. Not optimal to measure generalisation ability.
- *When to use ?* When there is a class or property imbalance in the dataset.

3. **Extrapolation split**: The dataset is divided such that some targeted molecules are placed in the test set without being included in the training set. This is often referred to as out-of-distribution (OOD).
   - *Pros*: Aims to assess a model's ability to generalize to unseen chemical environments. It challenges the model to predict properties based on general chemical principles rather than memorizing specific training instances, yielding more robust models.
   - *Cons*: Such split requires careful curation of OOD examples that maintain chemical relevance while representing unseen combinations. The design of OOD examples may inadvertently introduce biases or artifacts (e.g. low density region), impacting the fairness of model evaluations.
   - *When to use ?* When we care about ood generalisation of the model. For e.g., in materials or drug discovery applications.
4. **Time Split**: The dataset is divided based on a chronological order, such as using earlier time points for training and later time points for validation or testing.
   - *Pros*: Reflects real-world scenarios where models are trained on historical data and tested on future data. Allows evaluation of model performance under temporal variations.
   - *Cons*: Assumes that data collected at different time points are representative of the same underlying distribution. May not be suitable for all datasets or tasks.
   - *When to use?* In presence of meaningful temporal data.
5. **Group split**: The dataset is divided based on specific groups or categories present in the data (e.g., different targets, protein families, chemical series).
   - *Pros*: Enables evaluation of model performance on specific subsets of the data that may have distinct characteristics or challenges. Can provide insights into target or group-specific performance.
   - *Cons*: Requires prior knowledge or information about the groups or categories. May introduce biases if the groups are not representative or if data in different groups have varying characteristics.
   - *When to use?* When the data can be grouped based on specific characteristics or challenges.

C.2 Examples of predicted properties

- *band gap*: determines a material's electronic behavior and is relevant in areas such as semiconductor design and solar cell applications.
- *dielectric constant*: measures the ability of a substance or material to store electrical energy. It is an expression of the extent to which a material holds or concentrates electric flux. Dieletric constant has important applications in energy storage devices and electrical substation equipments.
- *Refractive index* is a measure of how light propagates through a material. It is an important property in optics and photonics applications, as it determines the material's ability to manipulate light.
- *glass*: is a classification property indicating if a material is a glass former or not. Glass-forming materials are important in the field of materials science, as their amorphous structure offers unique properties and applications in fields like optics, electronics, and energy storage.
- *jdft2d*: Exfoliation energy represents the energy required to separate or exfoliate a layered material into individual layers. This property is relevant in the field of 2D materials, where exfoliation plays a crucial role in obtaining thin layers with desired properties.
- *Formation energy* of a material provides insights into its stability and the energy involved in its formation. It is useful for understanding the feasibility of synthesizing new materials and their thermodynamic properties

- *Phonon energy*: the frequency of the highest frequency optical phonon mode peak provides information about the lattice vibrations in a material. It is relevant for understanding thermal conductivity, phonon transport, and thermal properties of materials.
- *Forces*: the forces acting on each atom, commonly used to perform molecular dynamics simulations.
- *Relaxed energy*: refers to the minimum energy state of a molecule or system, obtained through computational methods. It provides insights into the energetics of the molecule and aids in understanding its structural properties and interactions.
- *Dipole Moment*: is a measure of the separation of positive and negative charges within a molecule. It quantifies the molecule's polarity and can be important for predicting its behavior in certain chemical reactions or interactions.
- *HOMO* (Highest Occupied Molecular Orbital): refers to the highest energy level in a molecule that is occupied by electrons. It is an electronic property used to describe the reactivity and stability of molecules.
- *Polarizability*: is a property that describes the ability of a molecule to be deformed by an external electric field. It reflects the molecule's response to changes in the electric field and is important in studying intermolecular forces and interactions.
- *Heat Capacity*: measures the amount of heat energy required to raise the temperature of a substance. It can be used to understand and predict how a molecule will respond to changes in temperature, providing insights into its thermodynamic properties.
- *Toxicity*: refers to the degree to which a substance can cause harm or adverse effects to living organisms. Machine learning models trained to predict toxicity can help identify potentially toxic compounds and aid in drug development and safety assessment.
- *Solubility*: is the property of a substance to dissolve in a solvent to form a homogeneous solution. Solubility prediction involves determining the likelihood of a compound to dissolve in a particular solvent or under specific conditions. Accurate solubility predictions are valuable in environmental studies, drug discovery and material science.

C.3 Atom-type rescaling

In this data pre-processing approach, the target values (e.g., molecule energy) are shifted or corrected based on a learned or calculated energy contribution from individual atom types. The idea is to factor out the contributions from different atom types and create a more interpretable target variable that represents the interaction energy between atoms. This can help the model focus on learning the residual energy variations after accounting for the atomic contributions.

To calculate the shift values (e.g. energy contributions) of each atom type based on the training dataset, one must first construct a matrix $\boldsymbol{A}$ of shape $(n, z_{max}+1)$, where n is the number of dataset samples and z_{max} represents the largest atomic number. This matrix counts the occurrences of each atomic type for each structure.

Next, one must solve a linear equation $\boldsymbol{Ax} = \boldsymbol{q}$ to calculate the per atom shifts, where $\boldsymbol{x}$ is a vector of shape $(z_{max}+1)$ representing the unknown shifts for each atom type, and $\boldsymbol{q}$ is the target quantity of shape (n), which can be the ground truth energy. The solution $\boldsymbol{x}$ provides the shifts for each atom type.

Finally, to obtain the shifted energies y', the sum of the shifts corresponding to every atom in the atomic system is subtracted from the ground truth energy y. This ensures that the energy values are adjusted according to the atom-type specific shifts, effectively balancing the contributions of different atom types to the overall potential energy.

$$y' = y - \sum_{i \in \mathcal{G}_j} x_i$$

Target value shifting is particularly common in the field of computational chemistry. It's used to improve the prediction accuracy of ML models by factoring out known atomic contributions from the target values, allowing to focus on the finer details of atomic interactions. In different terms, the objective is to obtain rescaled target values exhibiting a desirable distribution and reduced variance,

which enhances the quality of the training data and improves the stability/convergence of the GNN model during training. While it requires accurate calculation of energy contributions, this approach can help capture subtleties that might be challenging to learn directly from raw data. For instance, it is commonly applied on the QM7-X dataset [Hoja et al., 2021], as described in this work.

Finally, note that although slightly less common on materials and molecules, atom type shift can be applied to input features instead of target variables. This is the case when ML models use atomic properties having very different scales, ultimately making it easier for the model to learn patterns across different atom types.

Opinion

If utilised, this pre-processing step must be reported clearly in the paper for reproducibility, with corresponding shift values.