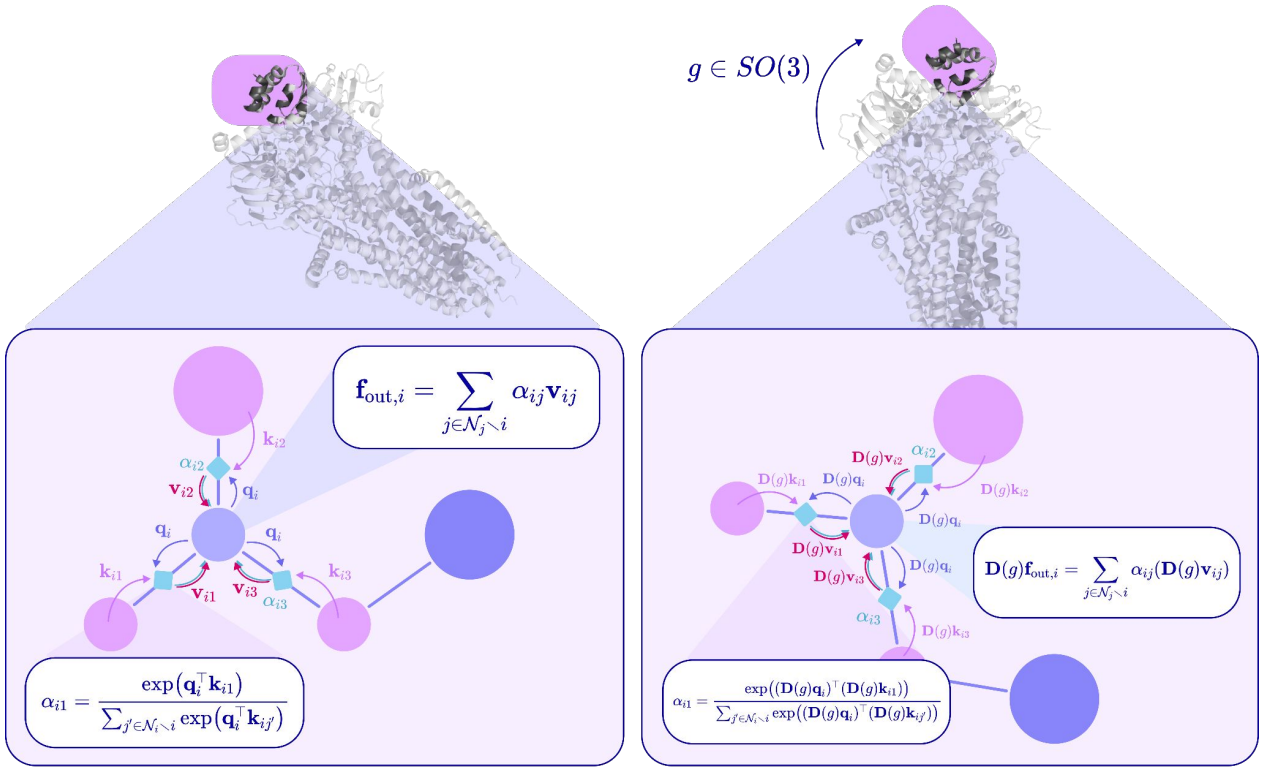
Spherical Equivariant Graph Transformers

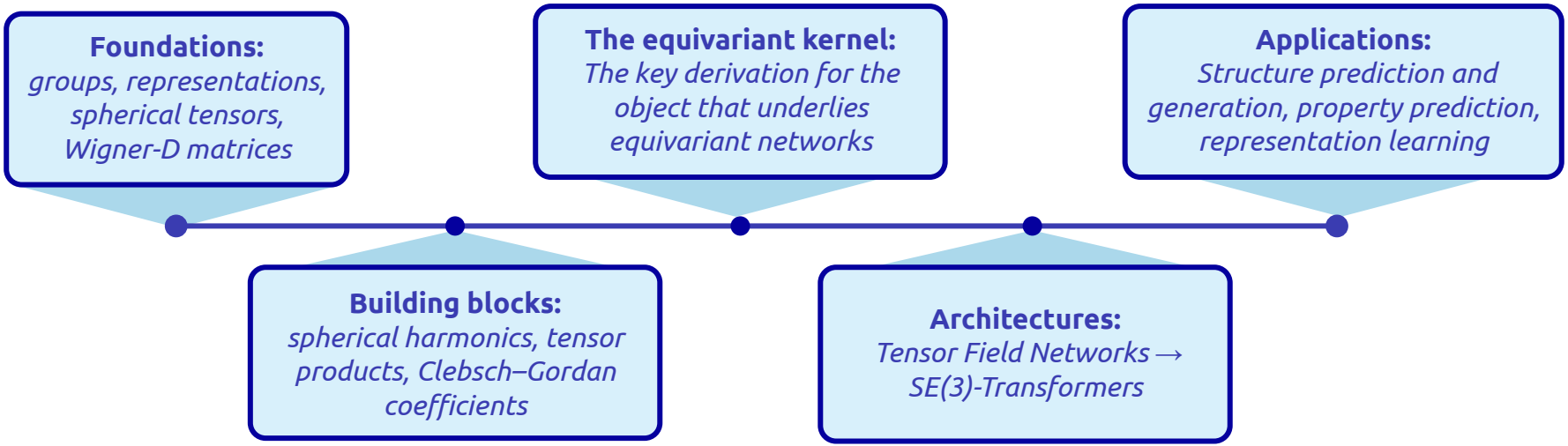
Sophia Tang | ***Cambridge Ellis Unit Summer School on Probabilistic Machine Learning*** | ***July 24, 2026***



Read the 99-page tutorial 👇

Roadmap

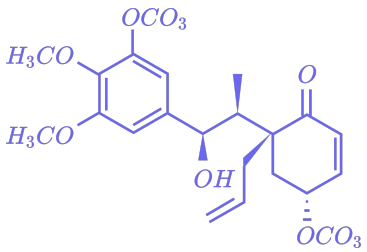
From group theory to a spherical equivariant graph transformers



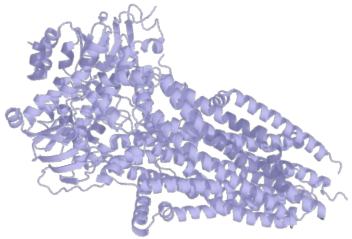
Why Equivariance?

Learning from inherent symmetries in data

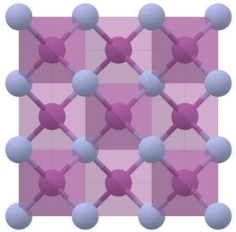
Small Molecules

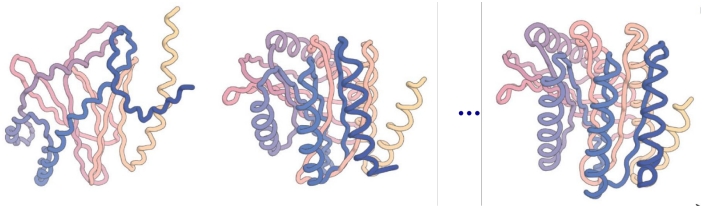


Biomolecules

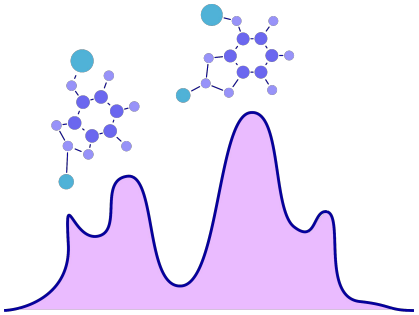


Crystal Structures and Materials

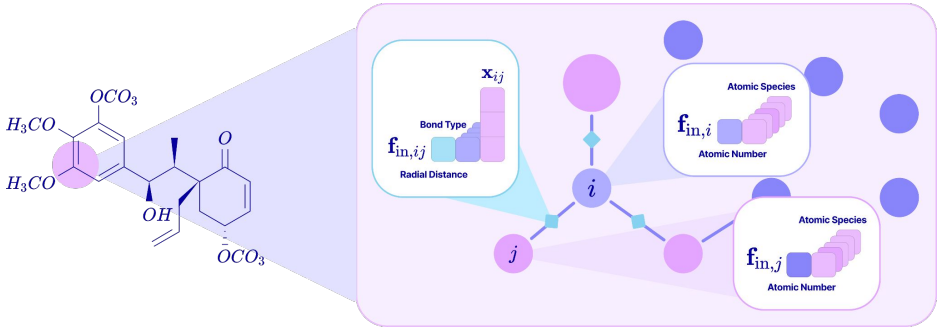




Structure Prediction and Generation



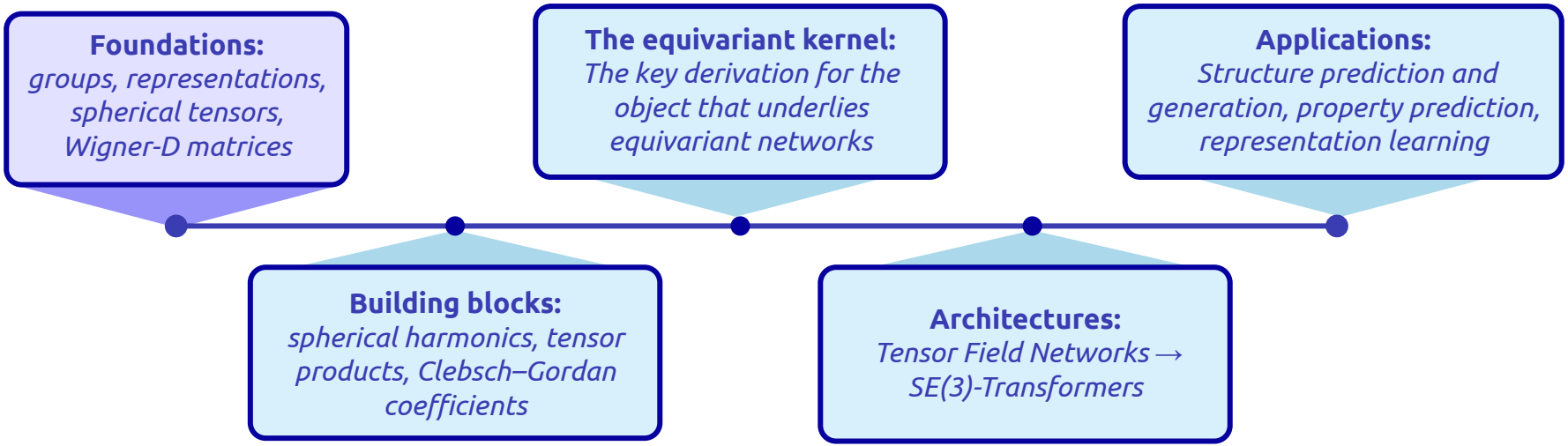
Property Prediction



Representation Learning

Roadmap

From group theory to a spherical equivariant graph transformers

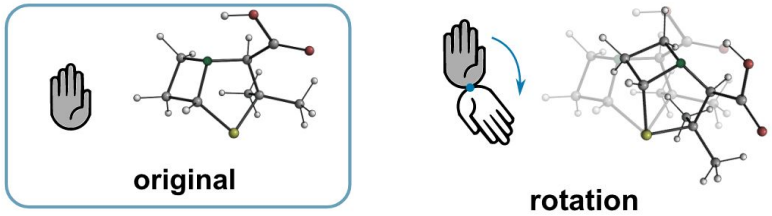


SO(3) and SE(3) Groups

Which symmetry group are we actually enforcing?

SO(3) Group

pure 3D rotations: orthogonal 3×3 matrices, determinant 1 → preserve lengths and angles

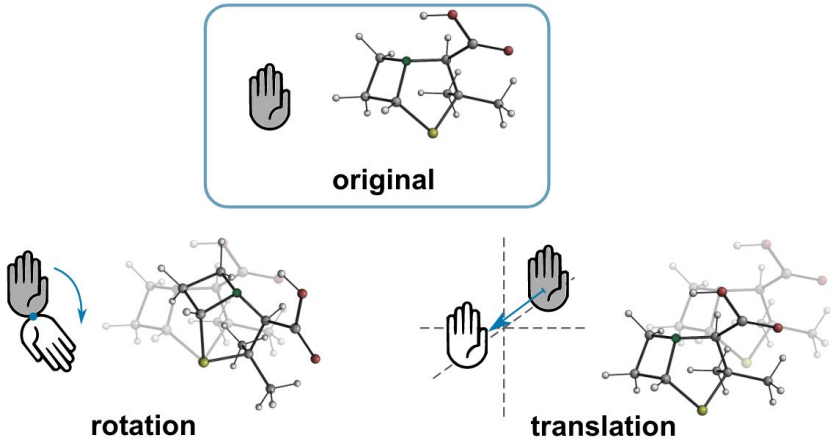


$$g \in \mathrm{SO}(3)$$

g denotes a specific action within the group

SE(3) Group

all rigid 3D motions (rotations + translations)



$$g \in \mathrm{SE}(3)$$

Invariance and Equivariance

How do we preserve symmetries in data?

A **function W** that acts on a **vector f** is **invariant under a group G** if the *output is the same before and after the action g ∈ G* is applied, for all actions in the group

$$\mathbf{W}(\rho_k(g)\mathbf{f}) = \mathbf{W}\mathbf{f}$$

A **function W is equivariant under group G** if the *output of the function also undergoes the same action g when the input is transformed by g*

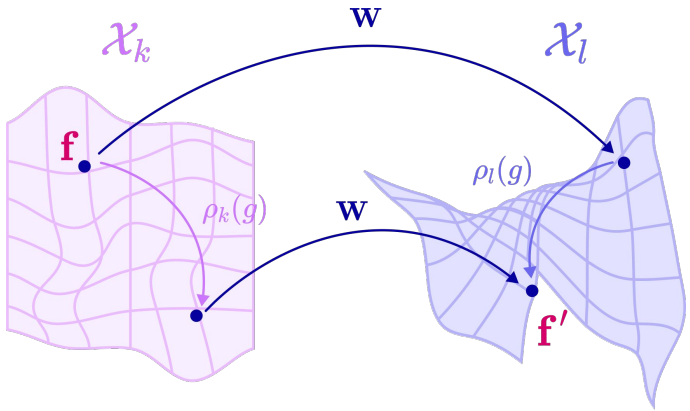
$$\mathbf{W}(\rho_k(g)\mathbf{f}) = \rho_l(g)(\mathbf{W}\mathbf{f})$$

$$\rho_l(g) : \mathcal{X}_l \to \mathcal{X}_l,\ \rho_k(g) : \mathcal{X}_k \to \mathcal{X}_k, \mathbf{W} : \mathcal{X}_k \to \mathcal{X}_l$$

$$\forall \mathbf{f} \in \mathcal{X}_k, g \in G$$

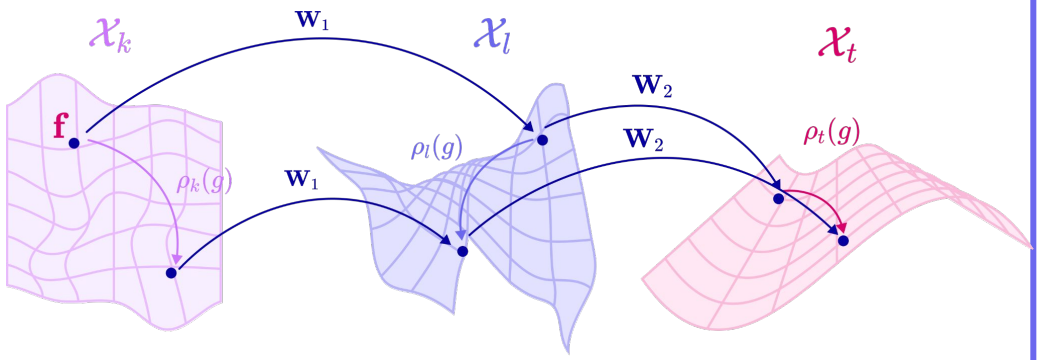
Properties of Equivariant Functions

Commutative Property of Equivariant Functions:
applying the group representation ρ followed by W is the same as applying W followed by ρ in both subspaces



$$\mathbf{W}(\rho_k(g)\mathbf{f}) = \rho_l(g)(\mathbf{W}\mathbf{f})$$

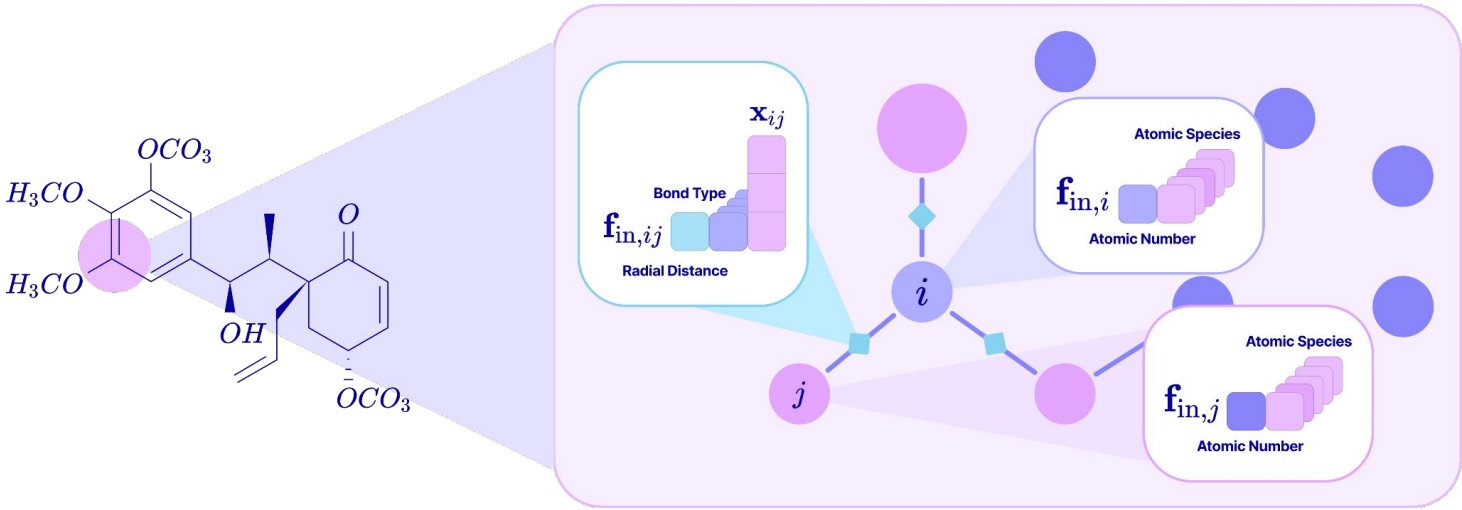
Commutative Property of Composing Multiple Equivariant Functions W1 and W2:
applying the group representation ρ on the input before the two subsequent functions produces the same output as applying the group representation on the output of the composed functions.



$$\mathbf{W}_2(\mathbf{W}_1(\rho_k(g)\mathbf{f})) = \rho_t(g)(\mathbf{W}_2(\mathbf{W}_1\mathbf{f}))$$

From Point Cloud to Geometric Graph

Giving structure to a set of atoms in space



Displacement vector from node *j* to node *i*

$$\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$$

Irreducible Representations (*Irreps*)

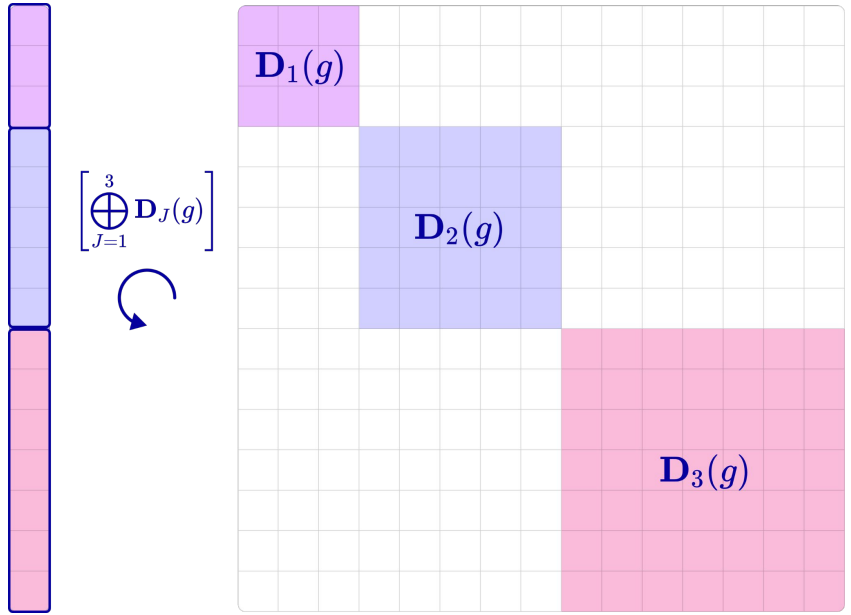
Applying group actions while preserving equivariance

All **group representations** (actions g on a specific subspace) can be decomposed into the **direct sum ⊕ (concatenation of matrices along the diagonal) of irreducible representations via a change in basis**:

Block diagonal matrix can then be used to transform a higher-dimensional tensor

$$\mathbf{D}(g) = \mathbf{Q}^\top \left[\bigoplus_J \mathbf{D}_J(g) \right] \mathbf{Q}$$

Change of basis



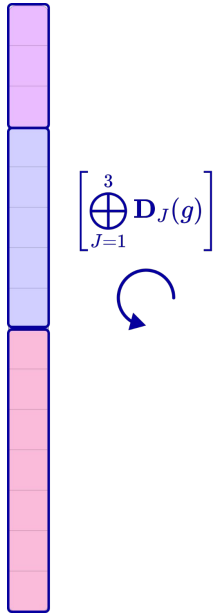
Spherical Tensors: *Irreps* of SO(3)

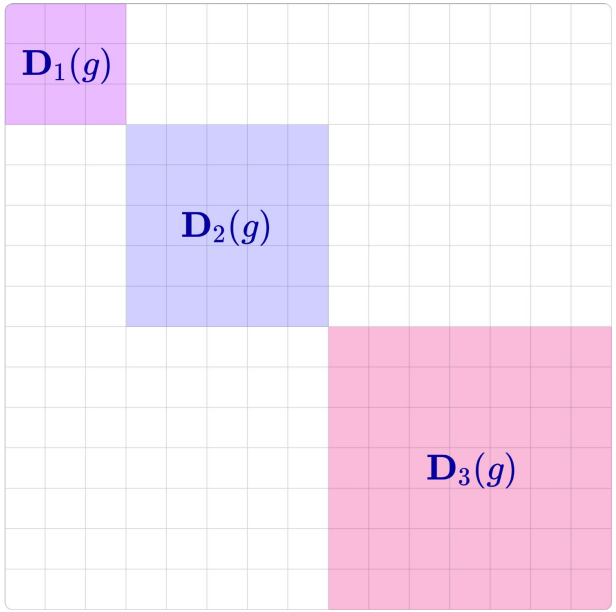
The feature type that rotates directly under SO(3) without change of basis

Block diagonal matrix can then be used to transform a higher-dimensional tensor

$$\mathbf{D}(g) = \mathbf{Q}^\top \left[\bigoplus_J \mathbf{D}_J(g) \right] \mathbf{Q}$$

Change of basis





Spherical tensors are special vectors that transform directly under SO(3) block diagonal marices without change of basis needed (irreducible representations of SO(3))

Wigner-D Matrices

How to transform different types of spherical tensors

Type-0 tensors

Scalars

remain unchanged under 3D rotation

$$\mathbf{D}_0(g) = [1]$$

1 × 1 matrix with a single entry of 1

Type-1 tensors

3D Vectors

transform by the standard 3 × 3 rotation matrices under 3D rotation

$$\mathbf{D}_1(g) = \mathbf{R}_x(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_z(\gamma)$$

*rotate by angle **α** **about the x-axis***

*rotate by angle **β** **about the y-axis***

*rotate by angle **γ** **about the z-axis***

Type-l tensors

(2l+1)-dim tensors

transform by the corresponding *(2l + 1) × (2l + 1)* type-l Wigner-D matrix

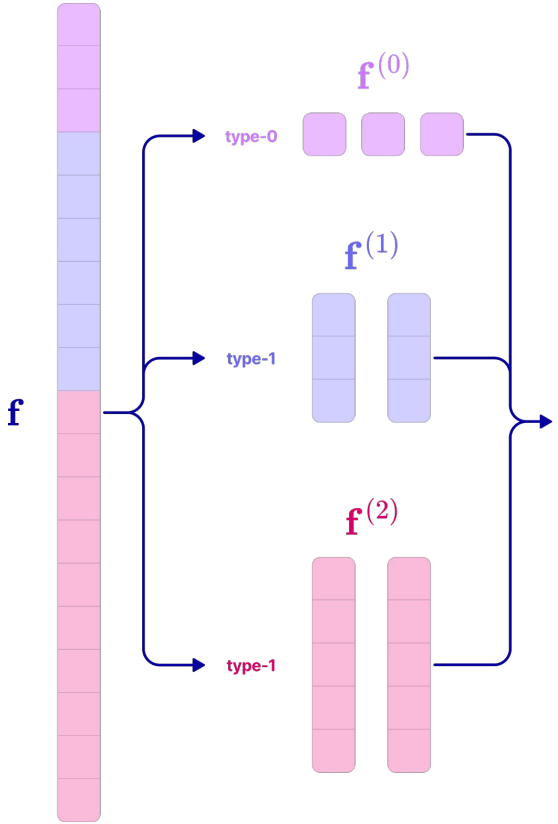
$$\mathbf{D}_l(g) \in \mathbb{R}^{(2l+1)\times(2l+1)}$$

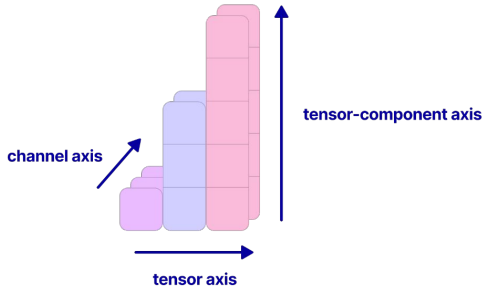
A *type-l irrep* is a set of *(2l + 1)×(2l + 1)* matrices called Wigner-D matrices that rotate a type-l spherical tensor by an element g ∈ SO(3)

Organizing Features as Tensor Lists

The feature vectors of a geometric graph can be organized into types of spherical tensors

Feature vector f corresponding to each node in the point cloud contains data on its properties (e.g., atomic number, charge, hydrophobicity, etc.)





Channel axis: *number of features of each type at the node. If a node contains three type-2 features, that feature has 3 channels.*

Tensor-component axis: *dimensions of each type of spherical tensor. For the type-k spherical tensors at the node, the tensor-component axis has dimension 2k + 1.*

Tensor axis: *number of different types of spherical tensor features at the node. If a node has type-0, type-1, and type-2 features, it has a tensor axis dimension of 3.*

Equivariant Condition with Wigner-D Matrices

How to transform different types of spherical tensors

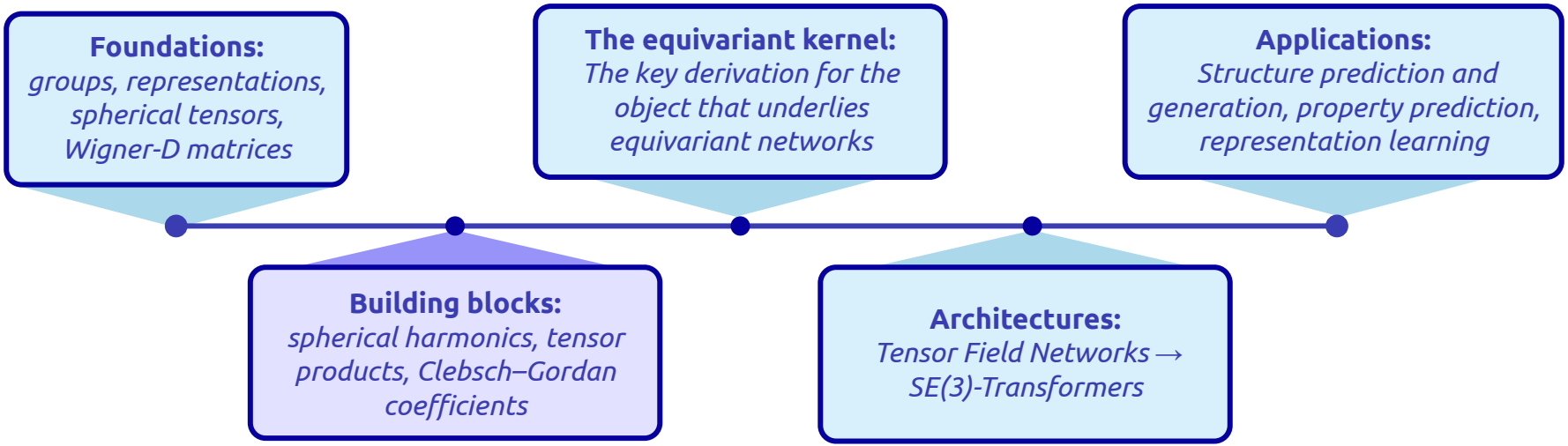
A **function** (or kernel), which we will denote as **W**, that transforms a ***type-k* spherical tensor** to a ***type-l* spherical tensor**, is equivariant if it satisfies the following **equivariance condition**:

$$\mathbf{D}_l(g)(\mathbf{W}\mathbf{f}) = \mathbf{W}(\mathbf{D}_k(g)\mathbf{f})$$

By decomposing a Cartesian tensor into its spherical tensor components, we enable them to transform directly and equivariantly under the Wigner-D matrices.

Roadmap

From group theory to a spherical equivariant graph transformers



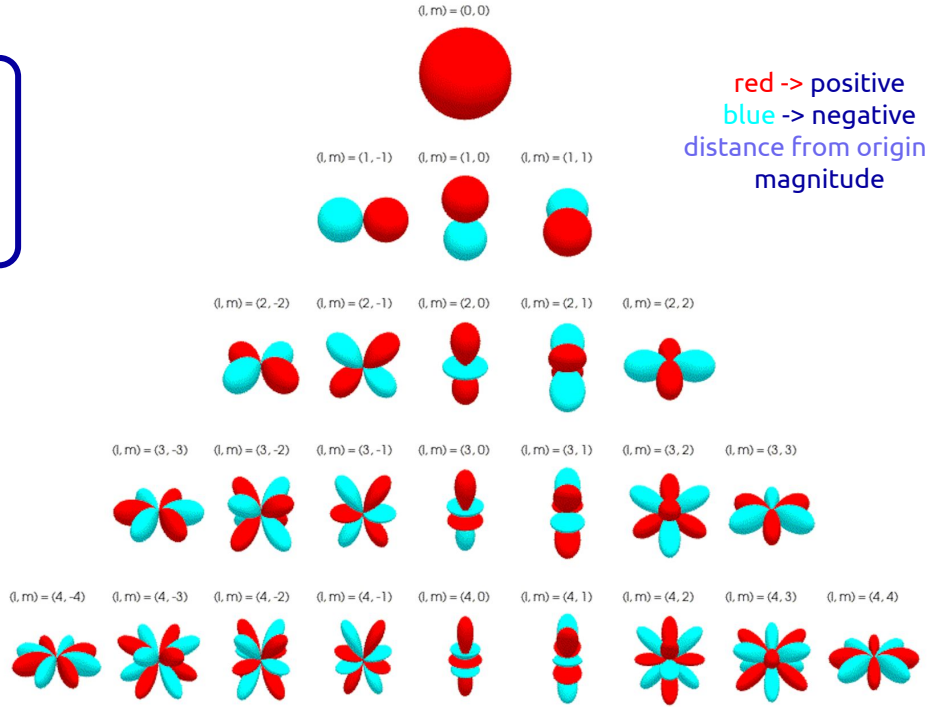
Spherical Harmonics

Transforming 3-dimensional vectors into their spherical tensor components

Spherical harmonics are **functions that project 3-dimensional vectors into spherical tensors** that transform equivariantly and directly under Wigner-D matrices

$$Y_m^{(l)}(\hat{\mathbf{x}}) : S^2 \to \mathbb{R}$$

A spherical harmonic function *(indexed by its degree l and order m)* **takes a unit vector (length of 1) on the unit sphere and returns a real number**



red -> positive
blue -> negative
distance from origin -> magnitude

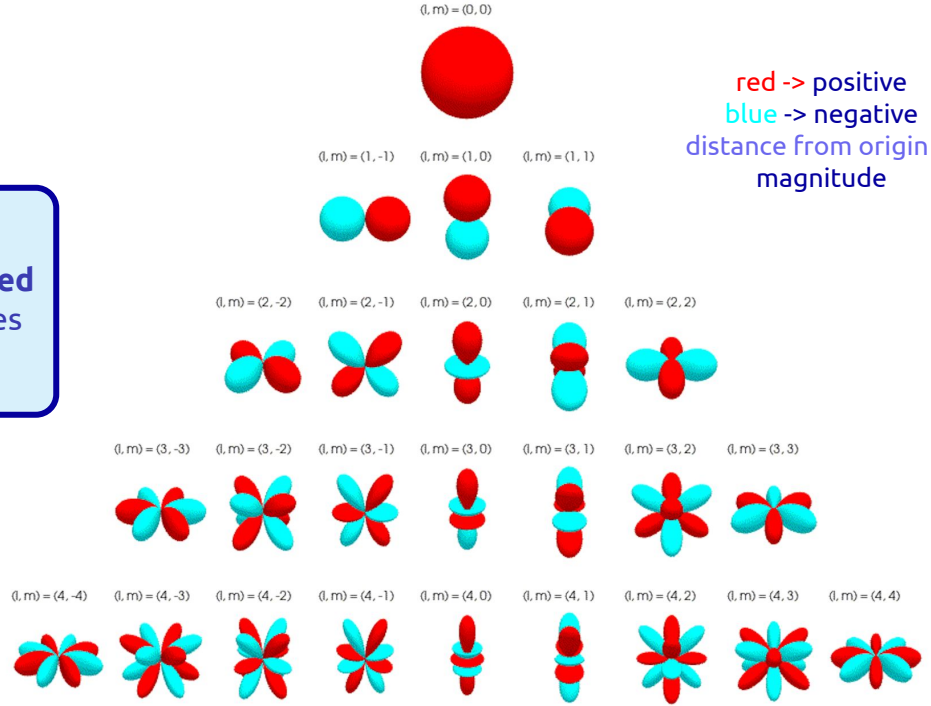
Aside: Spherical Harmonics in Quantum Mechanics

Transforming 3-dimensional vectors into their spherical tensor components

Spherical harmonics are **functions that project 3-dimensional vectors into spherical tensors** that transform equivariantly and directly under Wigner-D matrices

The spherical harmonic corresponding to |l, m⟩ is the **angular component of the wavefunction of an isolated electron**, which describes how the probability densities vary with orientation around the origin.

l → total angular momentum number
m → magnetic quantum number



red -> positive
blue -> negative
distance from origin -> magnitude

Spherical Harmonics

Transforming 3-dimensional vectors into their spherical tensor components

Spherical harmonics are **functions that project 3-dimensional vectors into spherical tensors** that transform equivariantly and directly under Wigner-D matrices

$$\mathbf{Y}^{(l)}(\hat{\mathbf{x}}) = \begin{bmatrix} Y_{-l}^{(l)}(\hat{\mathbf{x}}) \\ \vdots \\ Y_{l-1}^{(l)}(\hat{\mathbf{x}}) \\ Y_{l}^{(l)}(\hat{\mathbf{x}}) \end{bmatrix} : S^2 \to \mathbb{R}^{2l+1}$$

vector of *2l + 1* spherical harmonic functions indexed by *m* that **transform points on the unit sphere into *type-l* spherical tensors** that rotate directly under *type-l* Wigner-D matrices

m = -l, ..., l -> total 2l+1 functions

Spherical Harmonics as Equivariant Functions

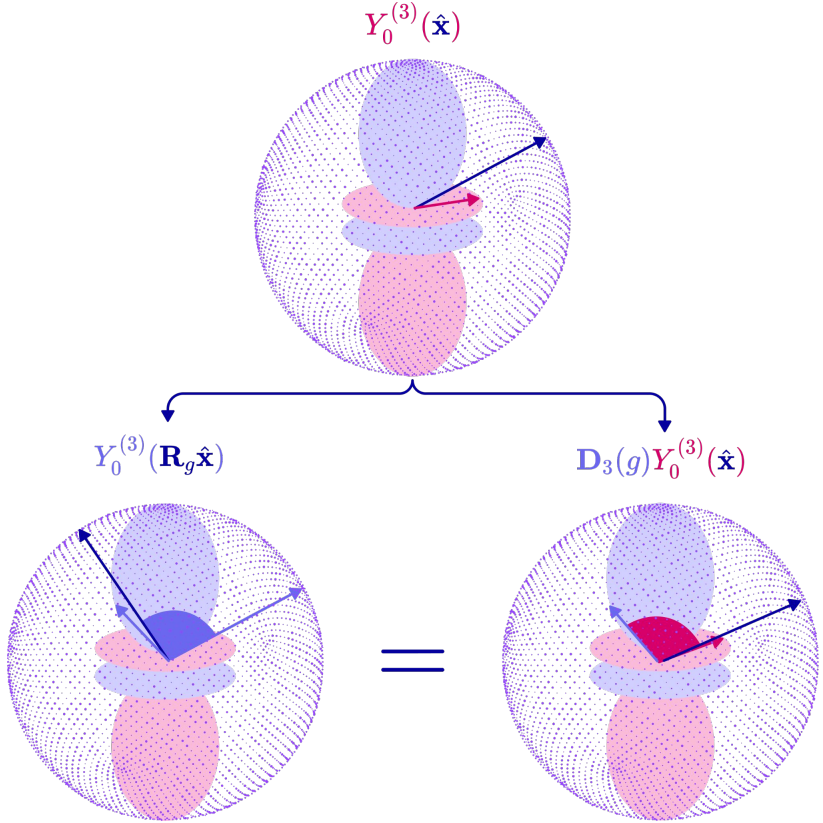
Transforming 3-dimensional vectors into their spherical tensor components

rotation of the input 3-dimensional vector by the 3 × 3 rotation matrix *R* for g ∈ SO(3)

=

rotating the spherical harmonic projection by the *type-l* Wigner-D matrix for g.

$$\mathbf{Y}^{(l)}(\mathbf{R}_g\hat{\mathbf{x}}) = \mathbf{D}_l(g)\mathbf{Y}^{(l)}(\hat{\mathbf{x}})$$



Spherical Harmonics as Equivariant Functions

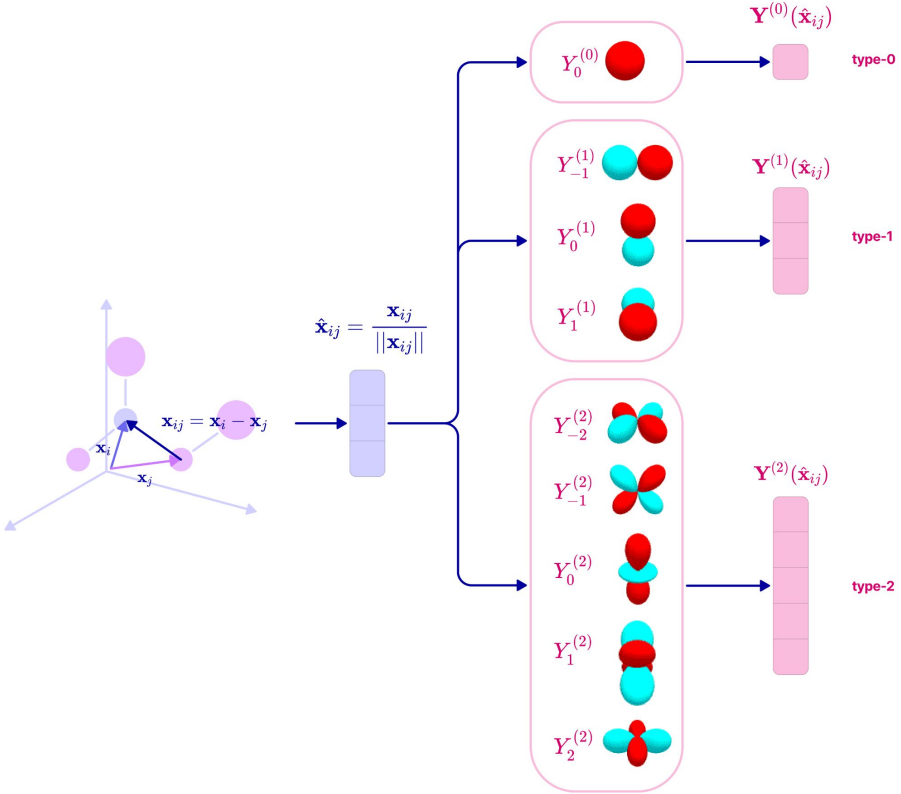
Transforming 3-dimensional vectors into their spherical tensor components

The type-J vectors of spherical harmonic functions with elements ***m = –J to J*** are used to **project angular unit vectors of edges into higher-degree spherical tensors** that can *model different frequencies of rotationally symmetric features*

Higher degree spherical tensors form the **basis set of SO(3)- equivariant kernels** that can combine to capture complex rotationally symmetric chemical properties with high precision

Lower-degree harmonics → model features with broader, smoother variations under rotations

Higher-degree harmonics → model features with sharper, finer variations under rotation



Tensor Product

How do we exchange information between features of different types without breaking equivariance?

tensor product **converts the type-*k* and type-*l* spherical tensors into a *(2l + 1) × (2k +1)* matrix** by calculating the product of every pair of dimensions indexed by *mk* and *ml*

$$\mathbf{s}^{(k)} \otimes \mathbf{t}^{(l)} = \mathbf{s}^{(k)}\mathbf{t}^{(l)\top} = \begin{bmatrix} s^{(k)}_{-m_k} t^{(l)}_{-m_l} & s^{(k)}_{-m_k} t^{(l)}_{-m_l+1} & \cdots & s^{(k)}_{-m_k} t^{(l)}_{m_l} \\ s^{(k)}_{-m_k+1} t^{(l)}_{-m_l} & s^{(k)}_{-m_k+1} t^{(l)}_{-m_l+1} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ s^{(k)}_{m_k} t^{(l)}_{-m_l} & s^{(k)}_{m_k} t^{(l)}_{-m_l+1} & \cdots & s^{(k)}_{m_k} t^{(l)}_{m_l} \end{bmatrix}$$

Tensor Product

How do we exchange information between features of different types without breaking equivariance?

Equivariance Condition

$$\mathbf{D}(g)(\mathbf{s}^{(k)} \otimes \mathbf{t}^{(l)}) = (\mathbf{D}_k(g)\mathbf{s}^{(k)}) \otimes (\mathbf{D}_l(g)\mathbf{t}^{(l)})$$

rotating the tensor product by g = rotating the individual spherical tensors by their respective Wigner-D matrices and then taking the tensor product

?

What is the representation D(g) under which the tensor product rotates equivariantly?

Tensor Product

How do we exchange information between features of different types without breaking equivariance?

vectorize!

$$\mathrm{vec}\Big((\mathbf{D}_k(g)\mathbf{s}^{(k)}) \otimes (\mathbf{D}_l(g)\mathbf{t}^{(l)})\Big) = \mathrm{vec}\Big((\mathbf{D}_k(g)\mathbf{s}^{(k)})(\mathbf{D}_l(g)\mathbf{t}^{(l)})^\top\Big)$$

$$\mathrm{vec}(\mathbf{AXB}) = (\mathbf{B}^\top \otimes \mathbf{A})\mathrm{vec}(\mathbf{X})$$

$$= \mathrm{vec}\Big(\mathbf{D}_k(g)\mathbf{s}^{(k)}\mathbf{t}^{(l)\top}\mathbf{D}_l(g)^\top\Big)$$

$$= (\mathbf{D}_k(g) \otimes \mathbf{D}_l(g))\mathrm{vec}(\mathbf{s}^{(k)} \otimes \mathbf{t}^{(l)})$$

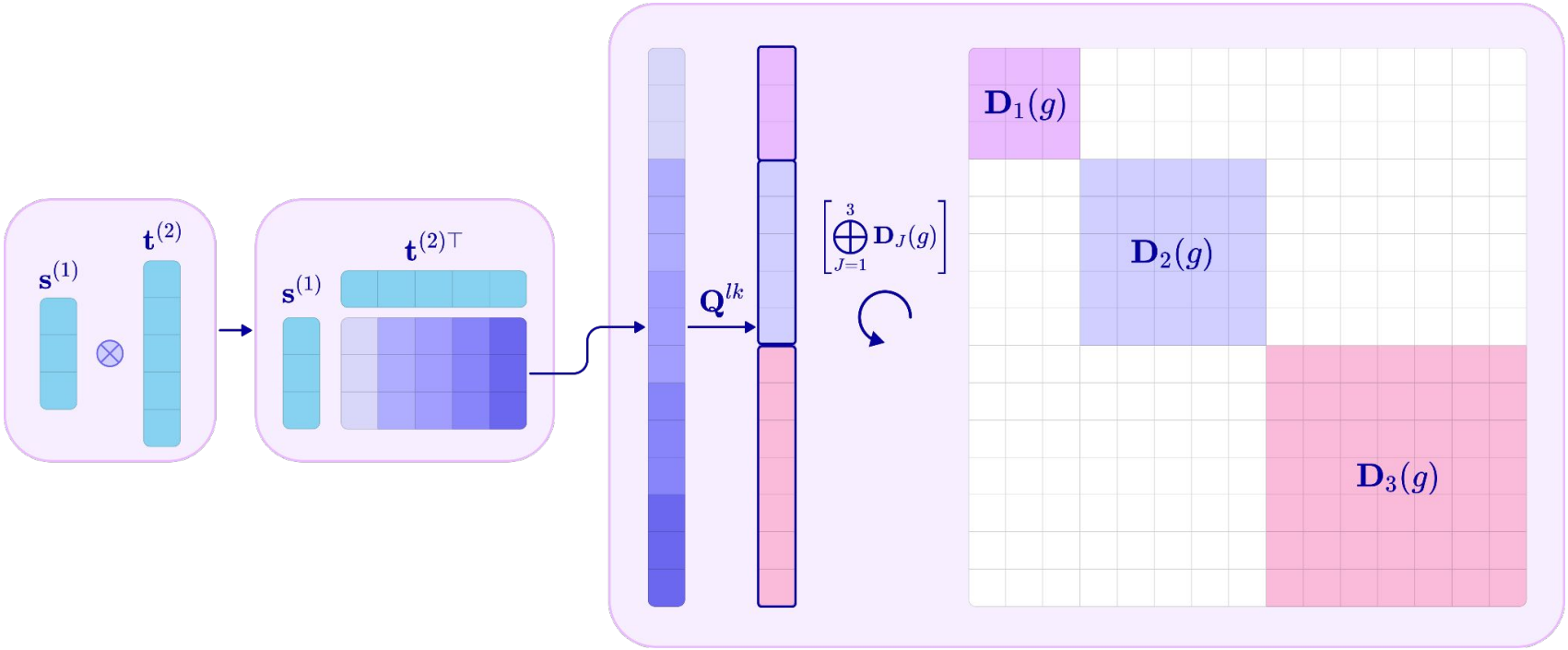
Kronecker Product

$$\mathbf{D}_{k\otimes l}(g) = \mathbf{D}_k(g) \otimes \mathbf{D}_l(g) = \mathbf{Q}^{lk\top}\left[\bigoplus_{J=|k-l|}^{k+l} \mathbf{D}_J(g)\right]\mathbf{Q}^{lk}$$

Matrix that transforms the tensor product of a type-k and type-l spherical tensor equivariantly

Tensor Product

How do we exchange information between features of different types without breaking equivariance?



tensor product of the type-1 spherical tensor **s** *and the type-2 spherical tensor* **t**, *which produces a 15-dimensional tensor* → *tensor is decomposed into the direct sum of exactly one type-1 (|2 − 1|), type-2, and type-3 (2 + 1) spherical tensor*

Clebsch-Gordan Decomposition

How do we decompose any tensor product into its spherical tensor components?

Clebsch-Gordan coefficients (CG coefficients) form the change-of-basis matrices transforming tensors from the combined tensor product space into their spherical tensor components

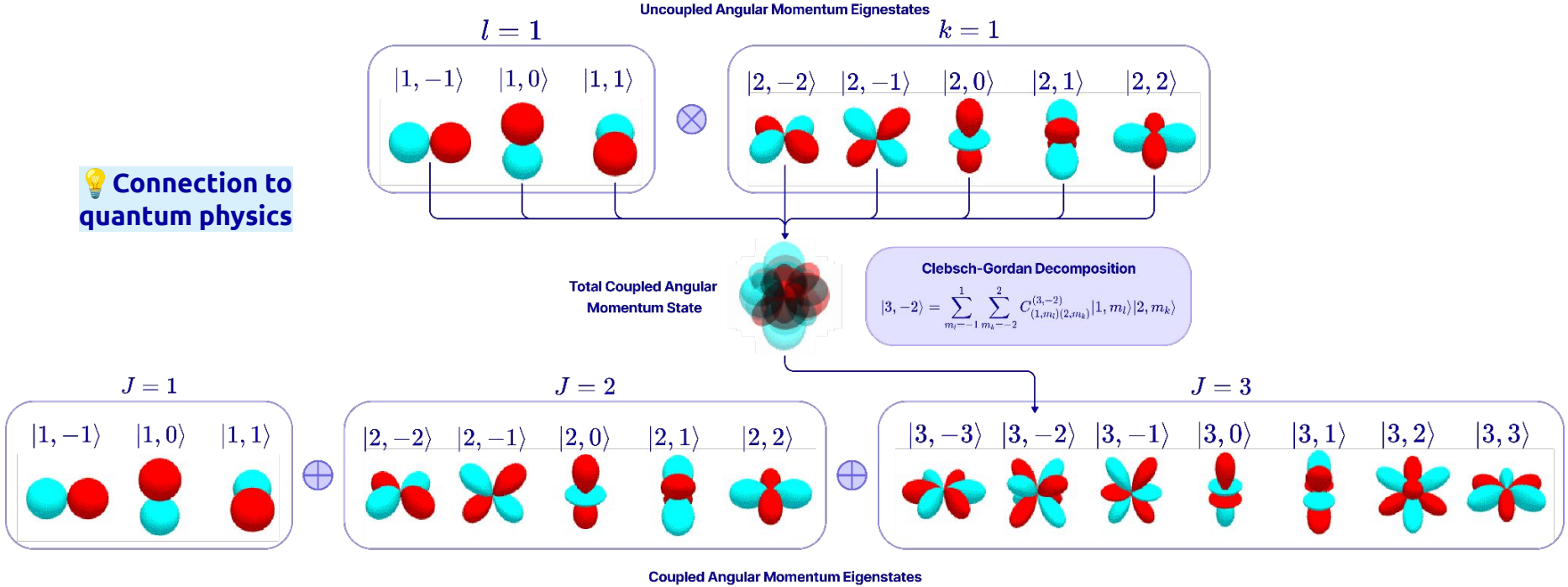
$$(\mathbf{s}^k \otimes \mathbf{t}^l)^{(J)}_m = \sum_{m_l=-l}^{l} \sum_{m_k=-k}^{k} C^{(J,m)}_{(l,m_l)(k,m_k)} s^{(k)}_{m_k} t^{(l)}_{m_l}$$

When decomposing the tensor product of a type-k and type-l tensor, **each CG coefficient scales the product of the *ml dimension of the type-l tensor* and the *mk dimension of the type-k tensor*** to give the ***mth dimension of the type-J spherical tensor component***.

Clebsch-Gordan Decomposition

How do we decompose any tensor product into its spherical tensor components?

💡 Connection to quantum physics



The coupling of two angular momentum eigenstates (equivalent to the tensor product) can be decomposed into a direct sum of coupled angular momentum states with total angular momentum ranging from |k − l| to k + l.

Parameterizing the Tensor Product

How do we introduce learnable parameters to the tensor product without breaking equivariance?

tensor product of a type-k and type-l spherical tensor can be **decomposed into 2 min(l, k) + 1 spherical tensors of types ranging from J = |k –l| to k + l**

$$\mathbf{D}_J(g) w_{(l,k,J)} (\mathbf{s}^k \otimes \mathbf{t}^l)_J = w_{(l,k,J)} \mathbf{D}_J(g) (\mathbf{s}^k \otimes \mathbf{t}^l)_J$$

Learnable scalar weights can be applied separately to each component of the decomposed spherical tensor components, since they are orthogonal and rotate independently under the *irreps* of SO(3)

The **equivariance condition holds** since the weight w is a type-0 scalar that is invariant to rotations.

Parameterizing the Tensor Product

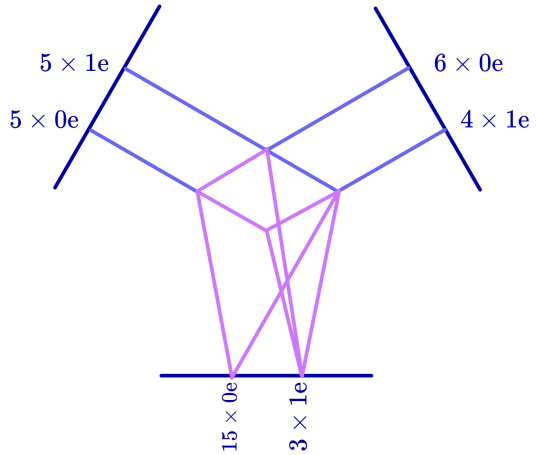
How do we introduce learnable parameters to the tensor product without breaking equivariance?

Generalize to multi-type input tensors:

$$\mathbf{s}^{k=0..K} \otimes \mathbf{t}^{l=0..L} = \bigoplus_{J=0}^{K+L} \left(\sum_{\mathrm{Path}(l,k,J)} w_{(l,k,J)} (\mathbf{s}^k \otimes \mathbf{t}^l)_J \right)$$

Each entry of the type-J spherical tensor component (m = −J, …, J):

$$(\mathbf{s}^k \otimes \mathbf{t}^l)_m^{(J)} = \sum_{m_l=-l}^{l} \sum_{m_k=-k}^{k} C_{(l,m_l)(k,m_k)}^{(J,m)} s_{m_k}^{(k)} t_{m_l}^{(l)}$$



Calculating the Parameterized Tensor Product

How do we introduce learnable parameters to the tensor product without breaking equivariance?

Consider:

$$\vec{\mathbf{f}}^{(0:1)} = \begin{bmatrix} \mathbf{f}^{(0)} \\ \mathbf{f}^{(1)} \end{bmatrix}$$

feature list containing a type-0 and type-1 tensor

$$\vec{\mathbf{Y}}^{(0:1)}(\hat{\mathbf{x}}_{ij}) = \begin{bmatrix} \mathbf{Y}^{(0)} \\ \mathbf{Y}^{(1)} \end{bmatrix}$$

angular unit vector x between nodes i and j projected into type-0 and type-1 tensors

Step 1: determine **every path (l, k, J)** that results in a type-J spherical tensor component for all possible values of J ranging from |0 – 0| = 0 to 1 + 1 = 2

Regrouped by J

$J = 0 : (0,0,0), (1,1,0)$:

(l=0, k=0): range is |0−0|=0 to 0+0=0, so only J=0 → path (0,0,0)

(l=1, k=1): range is |1−1|=0 to 1+1=2, so J=0, 1, 2 → paths (1,1,0), (1,1,1), (1,1,2)

$J = 1 : (0,1,1), (1,0,1), (1,1,1)$:

(l=0, k=1): range is |1−0|=1 to 1+0=1, so only J=1 → path (0,1,1)

(l=1, k=0): range is |0−1|=1 to 0+1=1, so only J=1 → path (1,0,1)

$J = 2 : (1,1,2)$:

Calculating the Parameterized Tensor Product

How do we introduce learnable parameters to the tensor product without breaking equivariance?

Consider:

$$\vec{\mathbf{f}}^{(0:1)} = \begin{bmatrix} \mathbf{f}^{(0)} \\ \mathbf{f}^{(1)} \end{bmatrix}$$

feature list containing a type-0 and type-1 tensor

$$\vec{\mathbf{Y}}^{(0:1)}(\hat{\mathbf{x}}_{ij}) = \begin{bmatrix} \mathbf{Y}^{(0)} \\ \mathbf{Y}^{(1)} \end{bmatrix}$$

angular unit vector x between nodes i and j projected into type-0 and type-1 tensors

Step 2: compute the **tensor products between every pair of tensors in the input lists**, decompose them into their spherical tensor components, and scale each path by a weight

$J = 0 : (0,0,0), (1,1,0)$:

$$w_{(0,0,0)}(\mathbf{f}^{(0)} \otimes \mathbf{Y}^{(0)})_0 + w_{(1,1,0)}(\mathbf{f}^{(1)} \otimes \mathbf{Y}^{(1)})_0$$

$J = 1 : (0,1,1), (1,0,1), (1,1,1)$:

$$w_{(0,1,1)}(\mathbf{f}^{(0)} \otimes \mathbf{Y}^{(1)})_1 + w_{(1,0,1)}(\mathbf{f}^{(1)} \otimes \mathbf{Y}^{(0)})_1 + w_{(1,1,1)}(\mathbf{f}^{(1)} \otimes \mathbf{Y}^{(1)})_1$$

$J = 2 : (1,1,2)$:

$$w_{(1,1,2)}(\mathbf{f}^{(1)} \otimes \mathbf{Y}^{(1)})_2$$

Calculating the Parameterized Tensor Product

How do we introduce learnable parameters to the tensor product without breaking equivariance?

Step 3: concatenate each weighted sum into a single 9-dimensional vector!

$$\begin{bmatrix} \mathbf{f}^{(0)} \\ \mathbf{f}^{(1)} \end{bmatrix} \otimes \begin{bmatrix} \mathbf{Y}^{(0)} \\ \mathbf{Y}^{(1)} \end{bmatrix} = \begin{bmatrix} w_{(0,0,0)}(\mathbf{f}^{(0)} \otimes \mathbf{Y}^{(0)})_0 + w_{(1,1,0)}(\mathbf{f}^{(1)} \otimes \mathbf{Y}^{(1)})_0 \\ w_{(0,1,1)}(\mathbf{f}^{(0)} \otimes \mathbf{Y}^{(1)})_1 + w_{(1,0,1)}(\mathbf{f}^{(1)} \otimes \mathbf{Y}^{(0)})_1 + w_{(1,1,1)}(\mathbf{f}^{(1)} \otimes \mathbf{Y}^{(1)})_1 \\ w_{(1,1,2)}(\mathbf{f}^{(1)} \otimes \mathbf{Y}^{1})_2 \end{bmatrix}$$

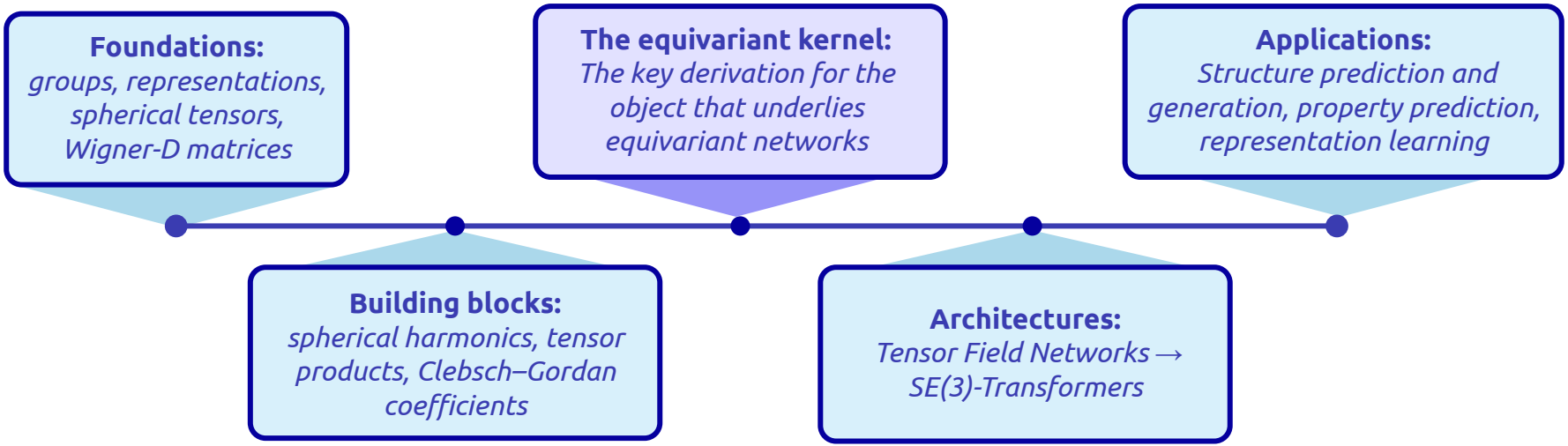
Review of Concepts

Overview of key concepts of SO(3)-equivariance

1. **SO(3) group = 3D rotations.** Its representations are $N \times N$ orthogonal matrices, decomposable into irreps called **Wigner-D matrices.**
2. Wigner-D matrices act *directly* on **spherical tensors** (generated by spherical harmonics) → no change of basis needed. **Spherical harmonics:** a complete, orthonormal basis on the sphere that projects vectors → spherical tensors.
3. Spherical tensors have **types (degrees)** $l = 0, 1, \ldots$, each $(2l+1)$-dimensional. **Higher l = faster change under rotation = higher-frequency harmonics.**
4. The **tensor product** combines two tensors into a higher-degree one. The tensor product is not itself spherical but splits into one tensor of each type $|k-l| \ldots k+l$ via **Clebsch–Gordan coefficients.**
5. This lets us **pass features as messages** (type-k neighbor → type-l center) without breaking equivariance: extract the type-l component of the tensor product with a spherical-harmonic tensor.
6. **Learnable weights** apply only after decomposition → one weight per path (l, k, J).

Roadmap

From group theory to a spherical equivariant graph transformers



Constructing an Equivariant Kernel

How can we facilitate message-passing between spherical tensors of different type?

We define a **kernel W** as a **function that takes the displacement vector as input and outputs a (2l + 1) × (2k + 1) linear transformation matrix**:

$$\mathbf{W}^{lk}(\mathbf{x}_{ij}) : \mathbb{R}^3 \rightarrow \mathbb{R}^{(2l+1)\times(2k+1)}$$

dependent on the **displacement vector from node j to i**, which encodes the distance and relative angular relationship between the two nodes

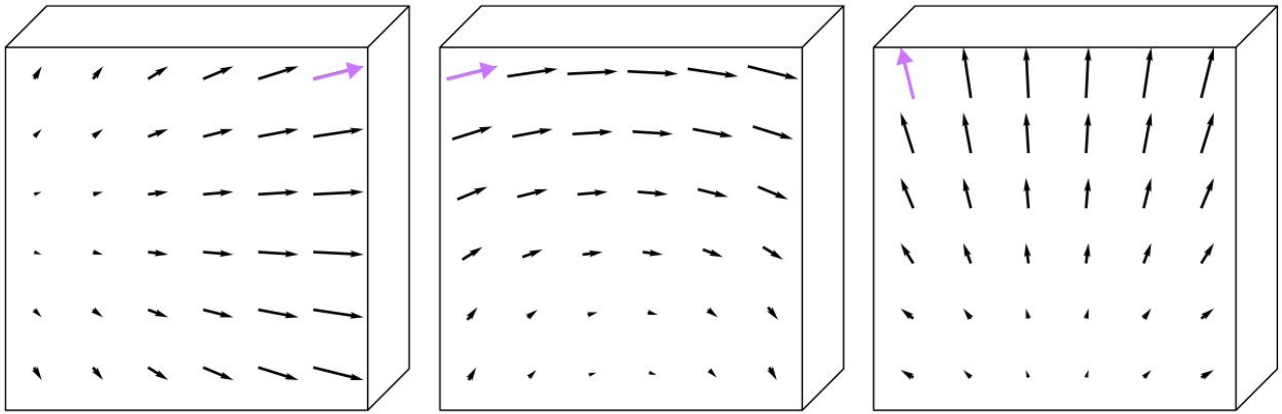
…that *transforms a type-k input feature into a type-l feature*

Deriving the Equivariant Kernel Constraint

How can we facilitate message-passing between spherical tensors of different type?

$$\mathbf{D}_l(g)\mathbf{f}^l_{\text{out},j} \neq \mathbf{D}_l(g)\mathbf{W}^{lk}(\mathbf{x}_{ij})\mathbf{f}^k_{\text{in},j}$$ ❌

💡 **Rotating a tensor field requires *rotating the vector first***



$f(\mathbf{x})$ $f(\mathbf{R}_g^{-1}\mathbf{x})$ $\mathbf{R}_g f(\mathbf{R}_g^{-1}\mathbf{x})$

$$\mathbf{W}^{lk}(\mathbf{x}_{ij})\mathbf{f}^k_{\text{in},j} \xrightarrow{\text{rotate by g}} \mathbf{D}_l(g)\left(\mathbf{W}^{lk}(\mathbf{R}_g^{-1}\mathbf{x}_{ij})\mathbf{f}^k_{\text{in},j}\right)$$ ✅

Deriving the Equivariant Kernel Constraint

How can we facilitate message-passing between spherical tensors of different type?

Equivariance constraint:

$$\mathbf{D}_l(g)\mathbf{f}^l_{\text{out},j} = \mathbf{W}^{lk}(\mathbf{x}_{ij})\mathbf{D}_k(g)\mathbf{f}^k_{\text{in},j}$$

Substitute definition of rotating the *type-l* output of the kernel by g ***in terms of the type-k input:***

$$\mathbf{D}_l(g)\mathbf{W}^{lk}(\mathbf{R}_g^{-1}\mathbf{x}_{ij})\mathbf{f}^k_{\text{in},j} = \mathbf{W}^{lk}(\mathbf{x}_{ij})\mathbf{D}_k(g)\mathbf{f}^k_{\text{in},j}$$

Multiply both sides by the **inverse of the type-k Wigner-D matrix**

Kernel Constraint:

$$\mathbf{W}^{lk}(\mathbf{R}_g\mathbf{x}_{ij}) = \mathbf{D}_l(g)\mathbf{W}^{lk}(\mathbf{x}_{ij})\mathbf{D}_k(g)^{-1}$$

❗ kernel is SO(3)- equivariant iff it is a solution to the constraint

rotating the displacement vector by g = **rotating the type-*k* input by the inverse of g, applying the kernel, and rotating the type-*l* output by g**

Deriving the Equivariant Kernel Constraint

How can we facilitate message-passing between spherical tensors of different type?

From the kernel constraint, we can derive the expression for an equivariant kernel:

$$\mathbf{W}^{lk}(\mathbf{x}_{ij}) = \sum_{J=|k-l|}^{k+l} \varphi_J^{lk}(\|\mathbf{x}_{ij}\|)\mathbf{W}_J^{lk}(\|\mathbf{x}_{ij}\|)$$

Every equivariant kernel can be constructed by taking the **linear combination of the basis kernels!**

$$\mathbf{W}_J^{lk}(\mathbf{x}) = \text{unvec}(\mathbf{Q}_J^{lk\top}\mathbf{Y}^J(\hat{\mathbf{x}}_{ij})) = \sum_{m=-J}^{J} \mathbf{Q}_{Jm}^{lk\top} Y_m^{(J)}(\hat{\mathbf{x}}_{ij})$$

spherical harmonic functions with *degree J* and *order m* evaluated on the angular unit vector

Constructing the Radial Function

Equivariant feed-forward networks with learned radial functions

The **learned radial function** takes the scalar distance as input and returns a weight for each basis kernel for values of ***J = |k – l| to |k + l|***

$$\varphi^{lk}(\|\mathbf{x}_{ij}\|) : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^{(2min(l,k)+1)(\mathrm{mi})(\mathrm{mo})}$$

# of output types (output channels)

# of entries in the equivariant basis kernel

# of input types (input channels)

Radial distance is SO(3)-invariant → **output of the radial function is invariant to rotations of the input graph** for any function definition

$$\varphi^{lk}_{(J,c_l,c_k)}(\|\mathbf{R}_g\mathbf{x}_{ij}\|) = \varphi^{lk}_{(J,c_l,c_k)}(\|\mathbf{x}_{ij}\|)$$

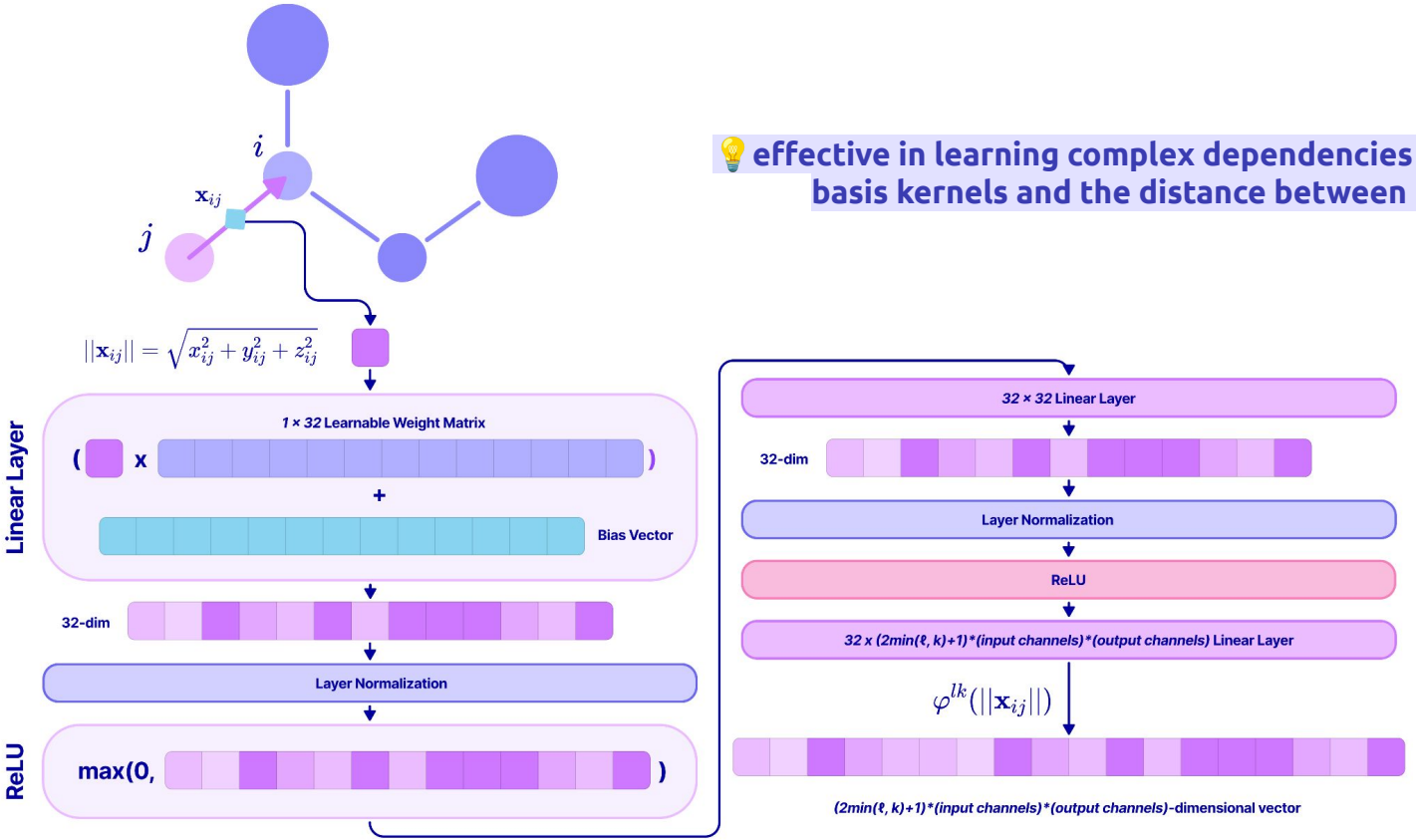
Equivariant Feed-Foward Network!

$$\mathbf{FFN} : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^{(2min(l,k)+1)(\mathrm{mi})(\mathrm{mo})}$$

💡 **effective in learning complex dependencies between the basis kernels and the distance between nodes**

Constructing the Radial Function

Equivariant feed-forward networks with learned radial functions



💡 effective in learning complex dependencies between the basis kernels and the distance between nodes

Mechanism of the Equivariant Kernel

The equivariant kernel performs 3 key steps to capture high-frequency rotationally symmetric patterns

1. **Extract the type-*l* component** of the tensor product between the type-*k* feature and the type-*J* harmonic projection of the displacement, via **Clebsch–Gordan coefficients**:

$$(\mathbf{Y}^{(J)}(\hat{\mathbf{x}}_{ij}) \otimes \mathbf{f}^{k}_{\text{in},j,c_k})^{(l)}_{m_l} = \sum_{m=-J}^{J} \sum_{m_k=-k}^{k} C^{(l,m_l)}_{(J,m)(k,m_k)} Y^{(J)}_{m} f^{(k)}_{m_k}$$

2. **Scale that type-l component by a learnable radial weight:**

$$\varphi^{lk}_{(J,c_l,c_k)}(\|\mathbf{x}_{ij}\|)(\mathbf{Y}^{(J)}(\hat{\mathbf{x}}_{ij}) \otimes \mathbf{f}^{k}_{\text{in},j,c_k})^{(l)}$$

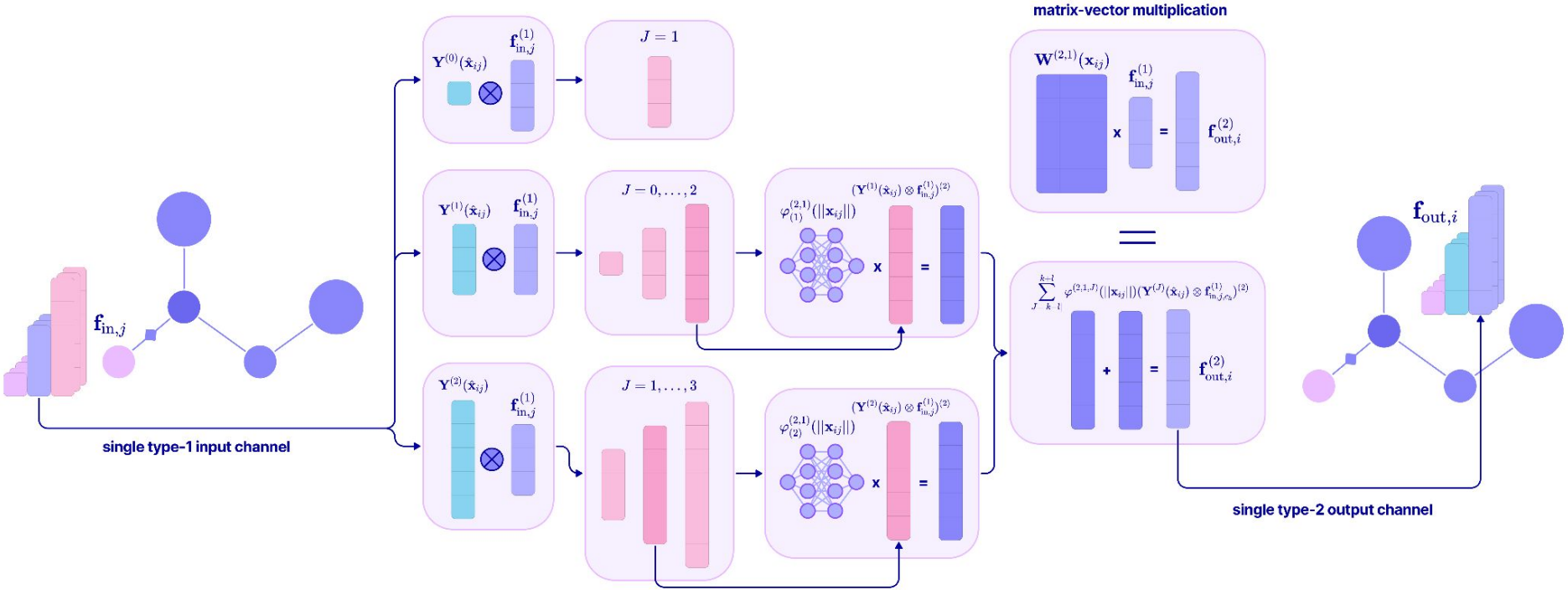
3. **Repeat & sum over all projections *J = |k–l| … k+l***, each with its own weight → the output type-l message:

$$\mathbf{W}^{lk}_{(c_l,c_k)}(\mathbf{x}_{ij})\mathbf{f}^{k}_{\text{in},j,c_k} = \sum_{J=|k-l|}^{k+l} \varphi^{lk}_{(J,c_l,c_k)}(\|\mathbf{x}_{ij}\|)(\mathbf{Y}^{(J)}(\hat{\mathbf{x}}_{ij}) \otimes \mathbf{f}^{k}_{\text{in},j,c_k})^{(l)}$$

Notation for kernel that ***transforms the type-k feature at channel ck in node j*** for message passing ***to the type-l feature at channel cl in node i***

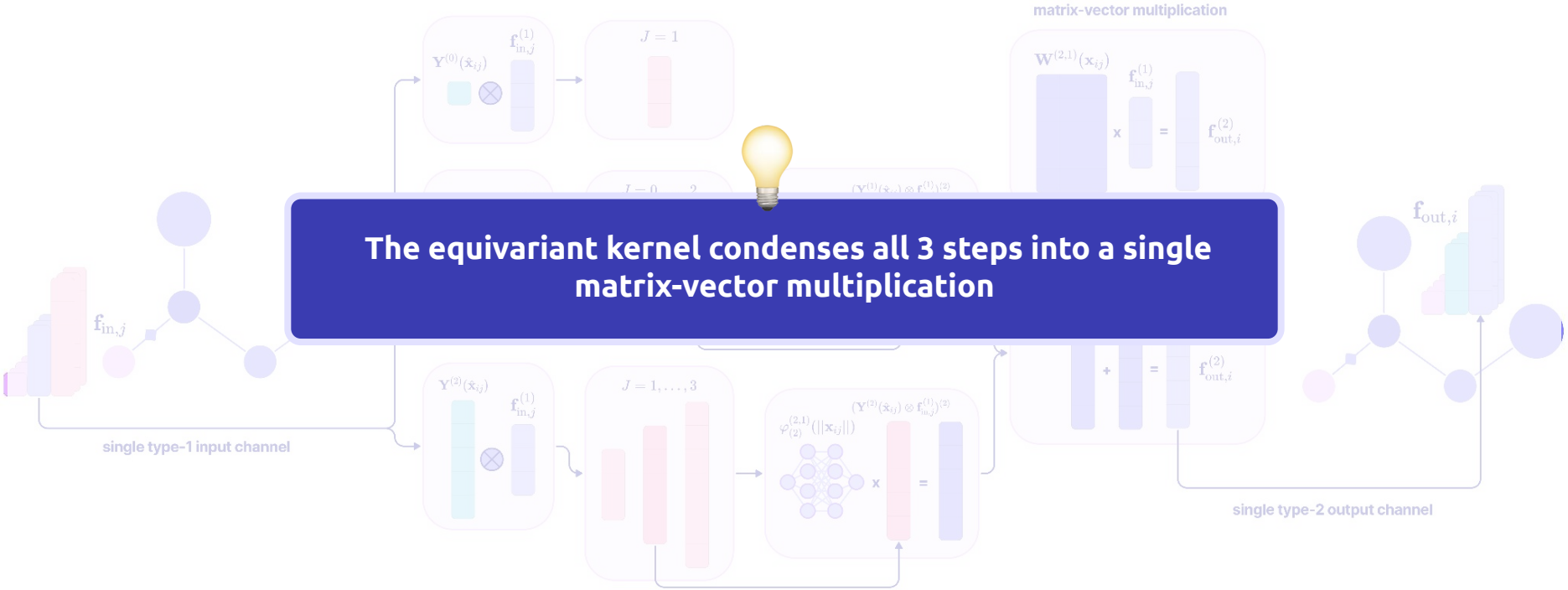
Mechanism of the Equivariant Kernel

The equivariant kernel performs 3 key steps to capture high-frequency rotationally symmetric patterns



Mechanism of the Equivariant Kernel

The equivariant kernel performs 3 key steps to capture high-frequency rotationally symmetric patterns



The equivariant kernel condenses all 3 steps into a single matrix-vector multiplication

Rules of Equivariant Layers

Constructing learnable neural networks that preserve equivariance

1. **Mixing different tensor types → tensor product.** To transform between different tensor types, take the tensor product, decompose it, and extract the target type. This condenses into one matrix-vector multiply with an equivariant kernel (spherical harmonics + Clebsch–Gordan) that satisfies the kernel constraint.

2. **Nonlinearities can only be applied on scalars.** Element-wise nonlinearities break equivariance for l > 0. Instead, use learnable functions that turn tensors into scalar weights, then scale the whole tensor.

3. **Weighted sums are safe as long as the tensors are of the same type.** Wigner-D matrices are linear, so they preserve addition and scaling within a type:

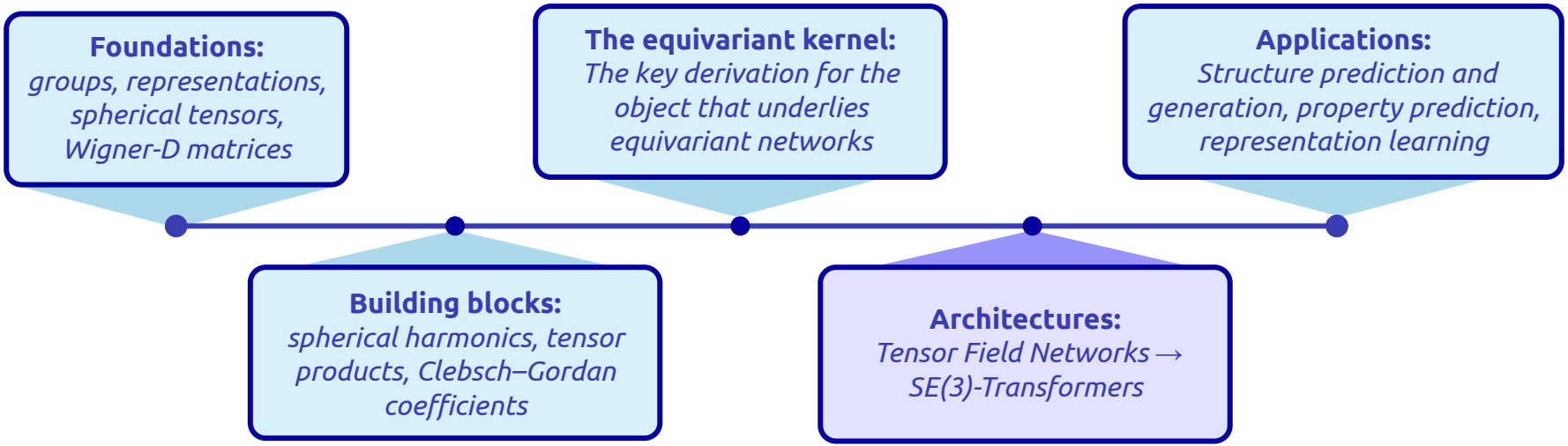
$$\mathbf{D}_l(g)(a\mathbf{s}^l + b\mathbf{t}^l) = a\mathbf{D}_l(g)\mathbf{s}^l + b\mathbf{D}_l(g)\mathbf{t}^l$$

   E.g. rotating the sum of two 3D vectors = summing the rotated vectors.

4. **Aggregation: permutation-invariant + rotation-equivariant.** Nodes are unordered, so message aggregation (sum or average of same-type messages) must not depend on order.

Roadmap

From group theory to a spherical equivariant graph transformers



The Tensor Field Network (TFN)

The first fully SE(3)-equivariant message-passing model

Thomas et al. 2018

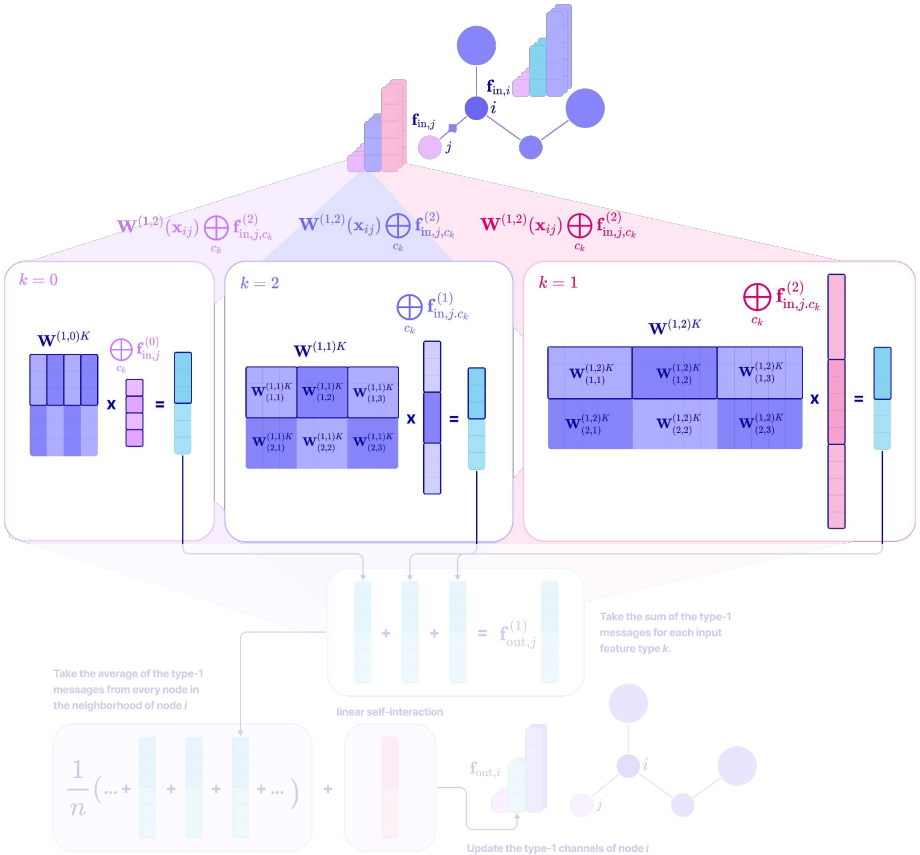
The TFN Message-Passing Equation

SE(3)-equivariant message passing in action

$$\mathbf{f}_{\text{out},i,c_l}^{l} = \underbrace{\sum_{c_l'} w_{(c_l,c_l')}^{ll} \mathbf{f}_{\text{in},i,c_l'}^{l}}_{\text{linear self-interaction}} + \underbrace{\frac{1}{n} \sum_{j \neq i}^{n} \sum_{k \geq 0} \underbrace{\sum_{c_k} \underbrace{\mathbf{W}_{(c_l,c_k)}^{lk}(\mathbf{x}_{ij}) \mathbf{f}_{\text{in},j,c_k}^{k}}_{\text{channels } c_k \to c_l \text{ message}}}_{\text{message from type-}k\text{ input channels}}}_{\text{type-}l\text{ message from node } j}}_{\text{average message over all } n \text{ adjacent nodes } j}$$

The TFN Message-Passing Equation

SE(3)-equivariant message passing in action



For every node, the TFN layer generates an **updated output tensor** equivariantly from its neighbors' features:

1. **Radial network per k → l:** distance → radial weights (one per basis kernel × channel pair).
2. **Build the kernel:** weighted sum of basis kernels

$$\mathbf{W}^{lk}_{(c_l,c_k)}(\|\mathbf{x}_{ij}\|) = \sum_{J=|k-l|}^{k+l} \varphi^{lk}_{(J,c_l,c_k)}(\|\mathbf{x}_{ij}\|)\mathbf{W}^{lk}_{J}(\|\mathbf{x}_{ij}\|)$$

3. **Reshape into a block matrix:** each block maps one input channel → one output channel
4. **Message:** concat type-k inputs → one matmul → all type-l messages. Sum over input types k.
5. **Aggregate:** mean over n neighbors (permutation-invariant) + **self-interaction**

❓ What is this?

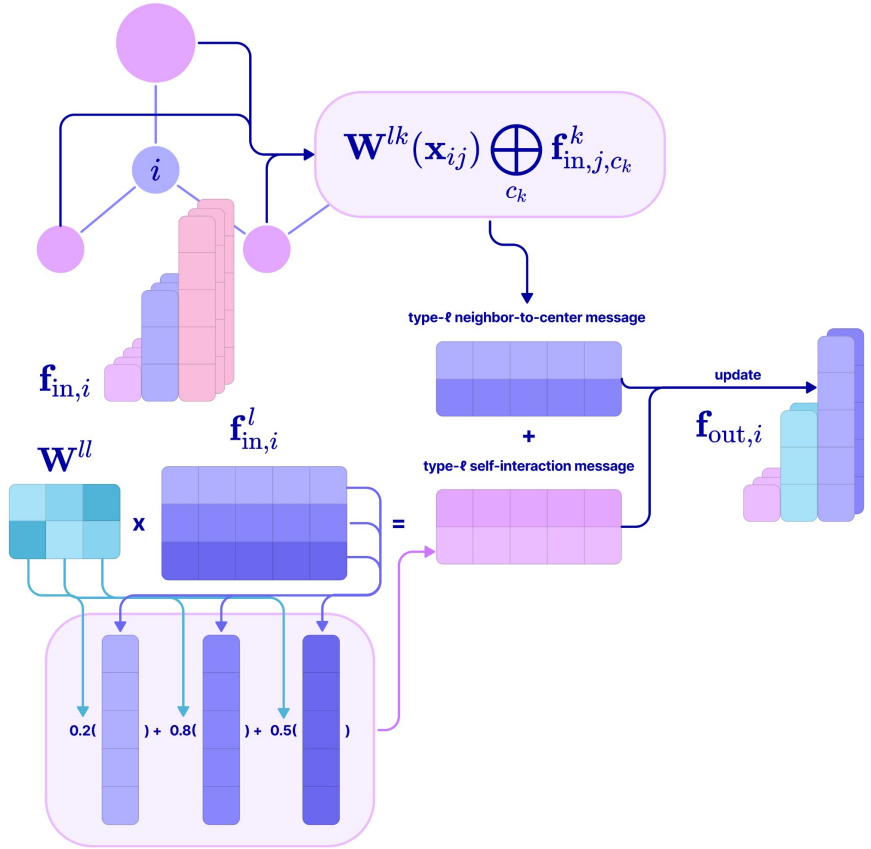
Self-Interaction

Two ways to integrate a nodes' own features into the message passing network

1) Linear Self Interaction

Learnable weighted sum of a node's own same-type channels. Acts as an *equivariant skip connection*.

$$\sum_{c_l'} w^{ll}_{(c_l, c_l')} \mathbf{f}^{l}_{\text{in},i,c_l'}$$



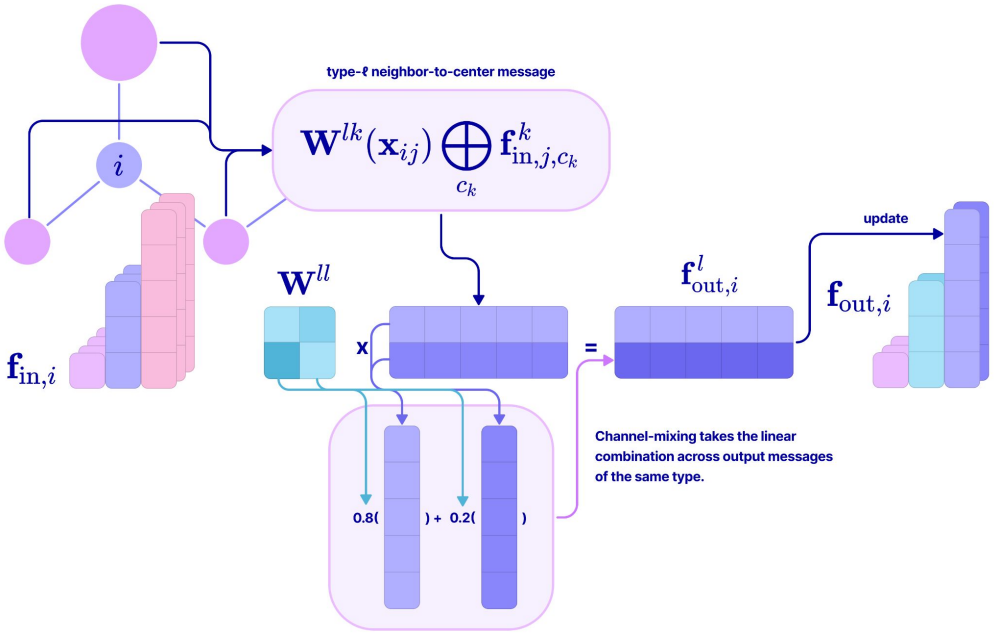
Self-Interaction

Two ways to integrate a nodes' own features into the message passing network

2) Channel Mixing

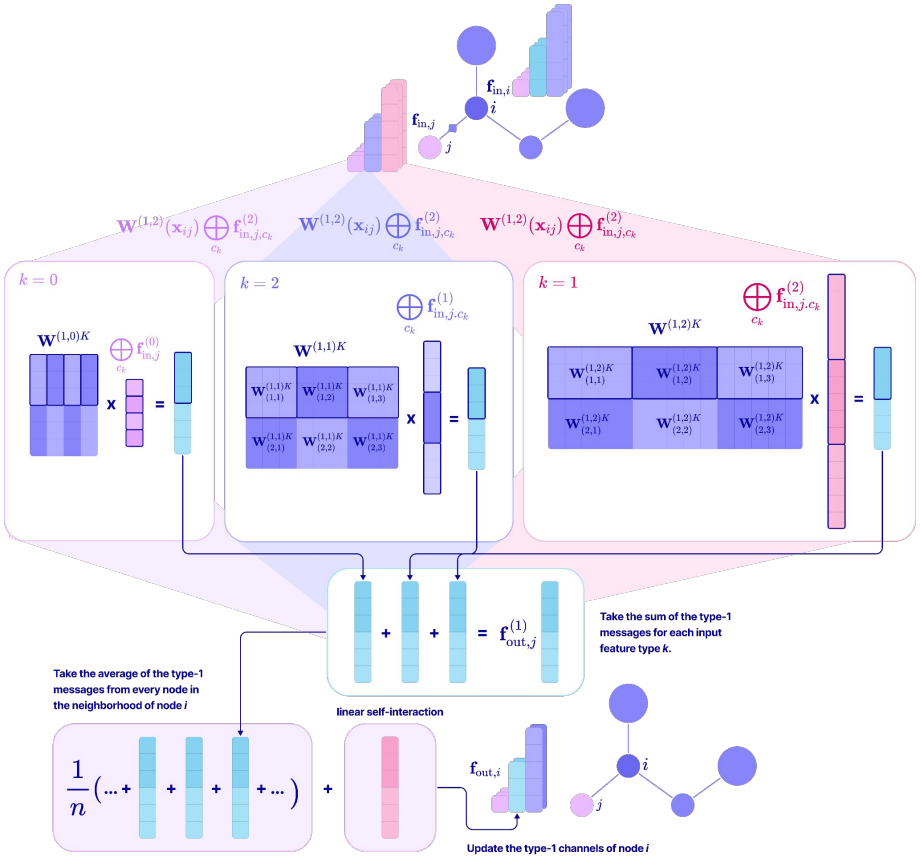
Update each channel with a **weighted mix of all same-type neighbor-message channels.**

$$\mathbf{f}^{l}_{\text{out},i,c_l} = \sum_{c'_l} w^{ll}_{(c_l,c'_l)} \mathbf{f}^{l}_{\text{out},i,c'_l}$$



The TFN Message-Passing Equation

SE(3)-equivariant message passing in action



For every node, the TFN layer generates an **updated output tensor** equivariantly from its neighbors' features:

1. **Radial network per k → l:** distance → radial weights (one per basis kernel × channel pair).
2. **Build the kernel:** weighted sum of basis kernels

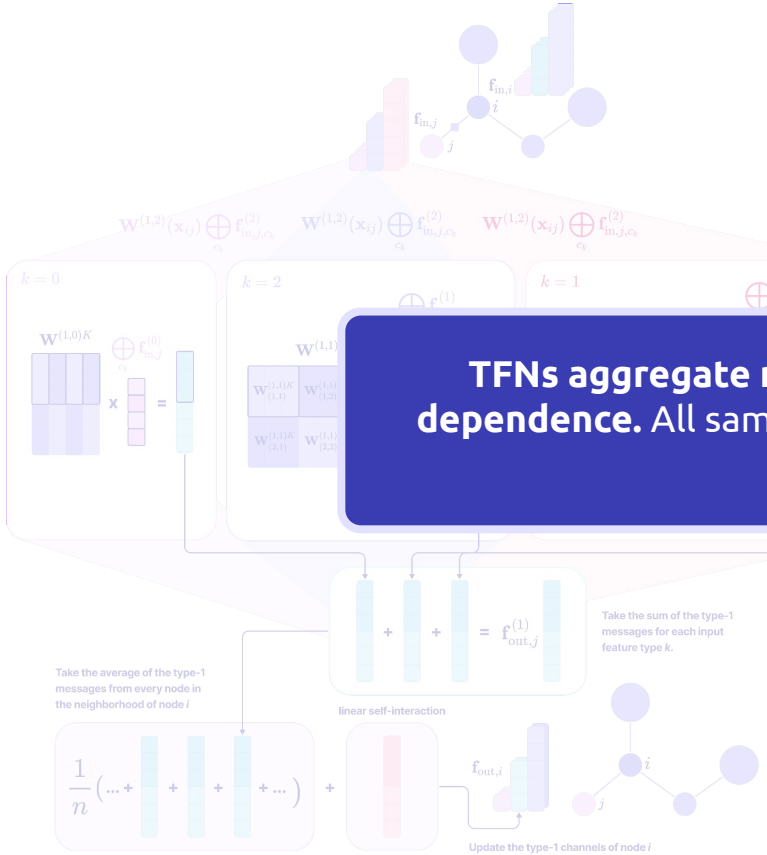
$$\mathbf{W}^{lk}_{(c_l,c_k)}(\|\mathbf{x}_{ij}\|) = \sum_{J=|k-l|}^{k+l} \varphi^{lk}_{(J,c_l,c_k)}(\|\mathbf{x}_{ij}\|)\mathbf{W}^{lk}_{J}(\|\mathbf{x}_{ij}\|)$$

3. **Reshape into a block matrix:** each block maps one input channel → one output channel
4. **Message:** concat type-k inputs → one matmul → all type-l messages. Sum over input types k.
5. **Aggregate:** mean over n neighbors (permutation-invariant) + **self-interaction**

✅**Prevents neighbor messages from drowning out a node's own features**

The TFN Message-Passing Equation

Self-interaction and averaged neighbor messages



TFNs aggregate messages uniformly with no angular dependence. All same-distance, same-type neighbors get the same angular filter.

For every node, the TFN layer generates an **updated output tensor** equivariantly from its neighbors' features:

1. **Radial network per k → l:** distance → radial weights (one per basis kernel × channel pair).
2. **Build the kernel:** weighted sum of basis kernels
$(\|\mathbf{x}_{ij}\|)\mathbf{W}_j^{lk}(\|\mathbf{x}_{ij}\|)$
ach block maps one
annel
4. **Message:** concat type-k inputs → one matmul → all type-l messages. Sum over input types k.
5. **Aggregate:** mean over n neighbors (permutation-invariant) + **self-interaction**

✅**Prevents neighbor messages from drowning out a node's own features**

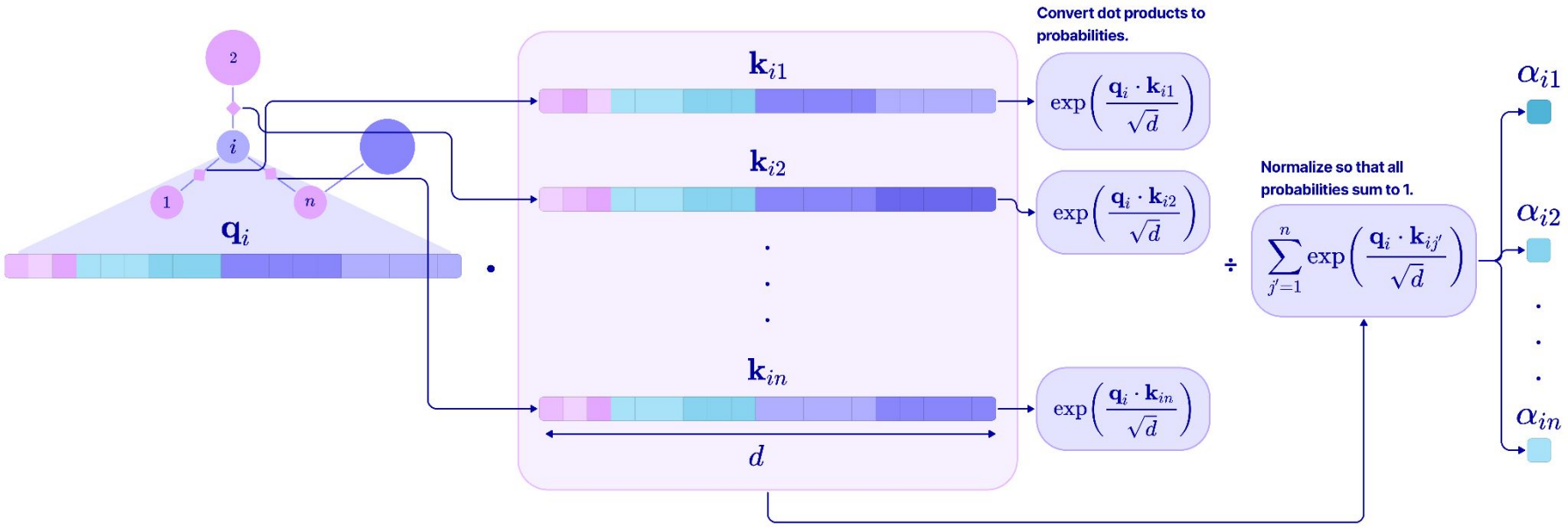
The SE(3)-Transformer

Equipping equivariant networks with attention mechanisms

Fuchs et al. 2020

Primer on Self-Attention for Sequences

Weighting neighbors by learned importance



Rather than aggregating uniformly, attention scores can weight certain messages between nodes with greater importance than others.

Equivariant Self-Attention

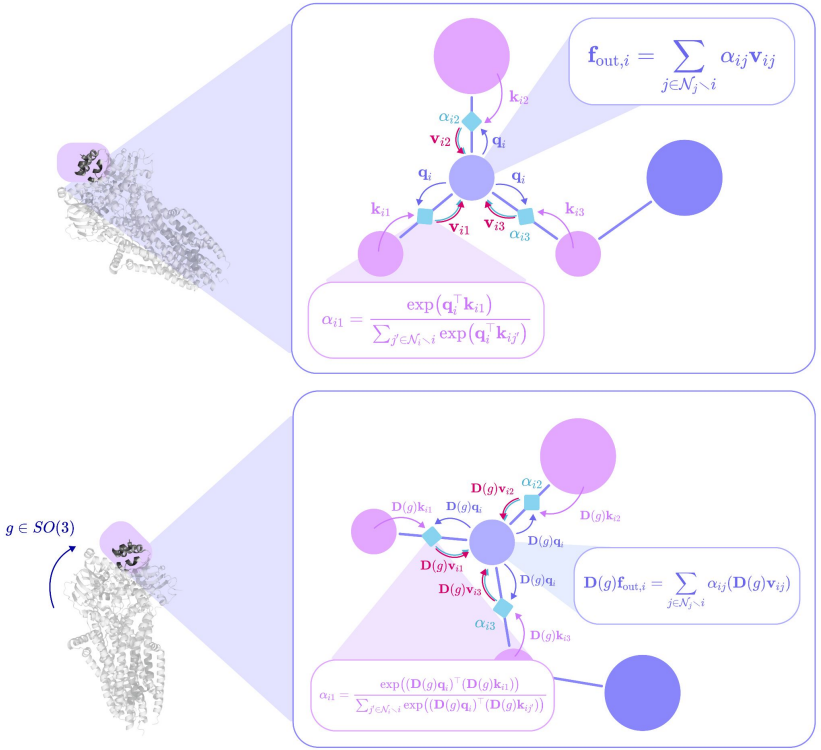
Equipping equivariant networks with attention mechanisms

🔑**The key component that weights messages depending on a learned importance between nodes**

$$\mathbf{f}^{l}_{\text{out},i,c_l} = \underbrace{\sum_{c'_l} w^{ll}_{(i,c_l,c'_l)} \mathbf{f}^{l}_{\text{in},i,c'_l}}_{\text{attentive self-interaction}} + \sum_{j \neq i}^{n} \underbrace{\alpha_{ij}}_{\text{attention score}} \underbrace{\sum_{k \geq 0} \underbrace{\sum_{c_k} \underbrace{\mathbf{W}^{lkV}_{(c_l,c_k)}(\mathbf{x}_{ij}) \mathbf{f}^{k}_{\text{in},j,c_k}}_{\text{channel } c_k \to c_l \text{ value message}}}_{\text{message from type-}k\text{ input channels}}}_{\text{type-}l\text{ message from node }j}}_{\text{weighted sum over all } n \text{ adjacent nodes}}$$

Equivariant Self-Attention

Equipping equivariant networks with attention mechanisms



Generating a message for node *i*:

1. **Query (node-based):** linear self-interaction across same-degree channels → concatenate all channels/degrees into one query embedding for the center node.
2. **Key (edge-based):** generated per edge like a TFN neighbor-to-center message; same dimension as the query.
3. **Raw attention scores:** dot product of the center's query with each incoming edge's key.
4. **Softmax:** normalize scores over edges sharing a destination → probabilities summing to 1.
5. **Aggregate:** scale each edge's value message by its attention score and sum (vs. TFN's plain average).
6. **Attentive self-interaction:** concatenate the center node's own features to the message → project to the output fiber with per-node attention weights → final output.

Equivariant Self-Attention

Equipping equivariant networks with attention mechanisms

Query vector (node-based, via linear self-interaction, concatenated across channels/degrees):

$$\mathbf{q}_i = \bigoplus_{l \geq 0} \underbrace{\underbrace{\bigoplus_{c_l} \sum_{c'_l} w^{ll}_{c'_l, c_l} \mathbf{f}^l_{\text{in},i,c'_l}}_{\text{query emb for all type-}l\text{ channels}}}_{\text{query emb for all input degrees}}$$

Attention score (query · key per incoming edge → softmax over shared-destination edges and sums to 1):

$$\alpha_{ij} = \frac{\exp\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_{ij}}{\sqrt{d}}\right)}{\sum_{j' \neq i}^{n} \exp\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_{ij'}}{\sqrt{d}}\right)}$$

Scale each edge's value by its score and sum (replaces TFN's average)

$$\sum_{j \neq i}^{n} \alpha_{ij} \mathbf{v}^l_{ij}$$

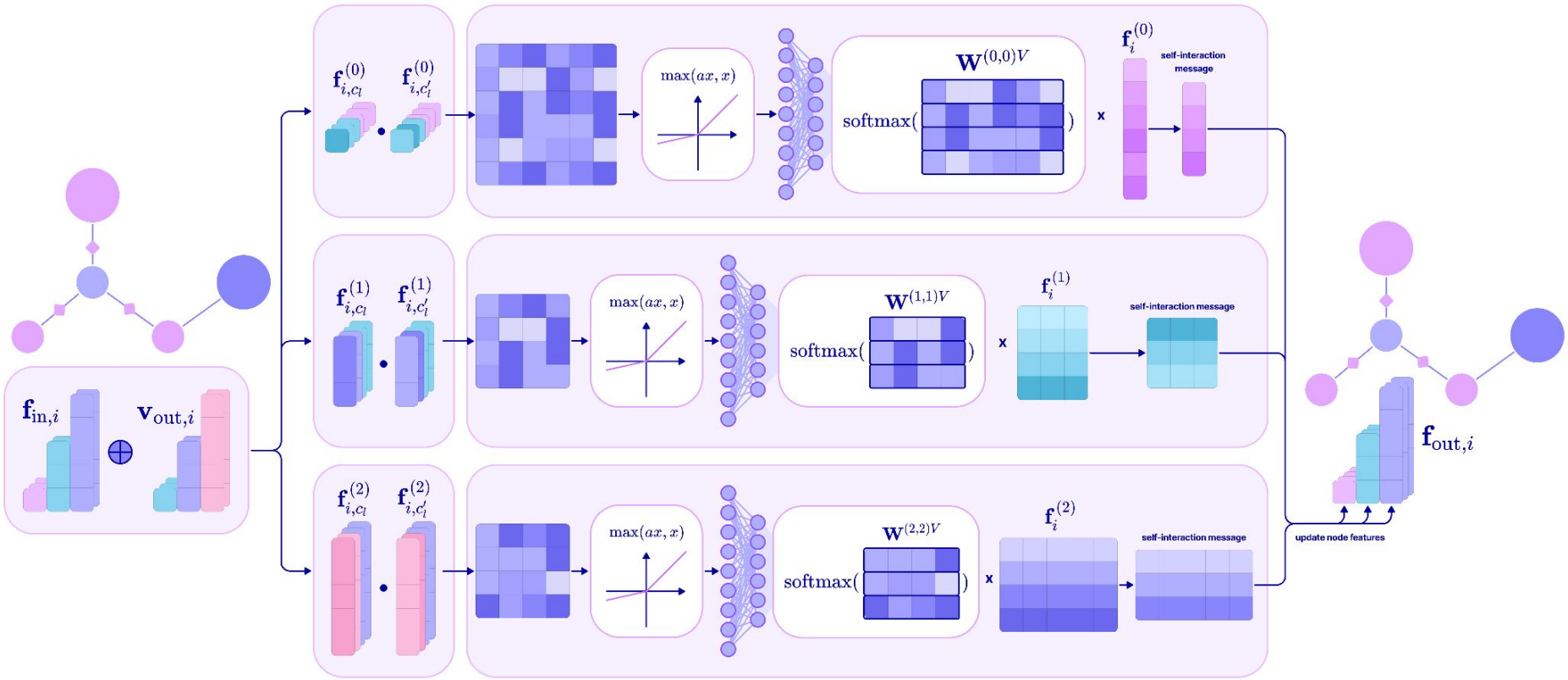
neighbor → center msg

$$\mathbf{k}_{ij} = \bigoplus_{l \geq 0} \underbrace{\underbrace{\sum_{k \geq 0} \mathbf{W}^{lkK}(\mathbf{x}_{ij}) \bigoplus_{c_k} \mathbf{f}^k_{\text{in},j,c_k}}_{\text{key emb for all type-}l\text{ channels}}}_{\text{key emb for all degrees in fiber}}$$

Key (edge-based, built like a TFN message, matches query dimension)

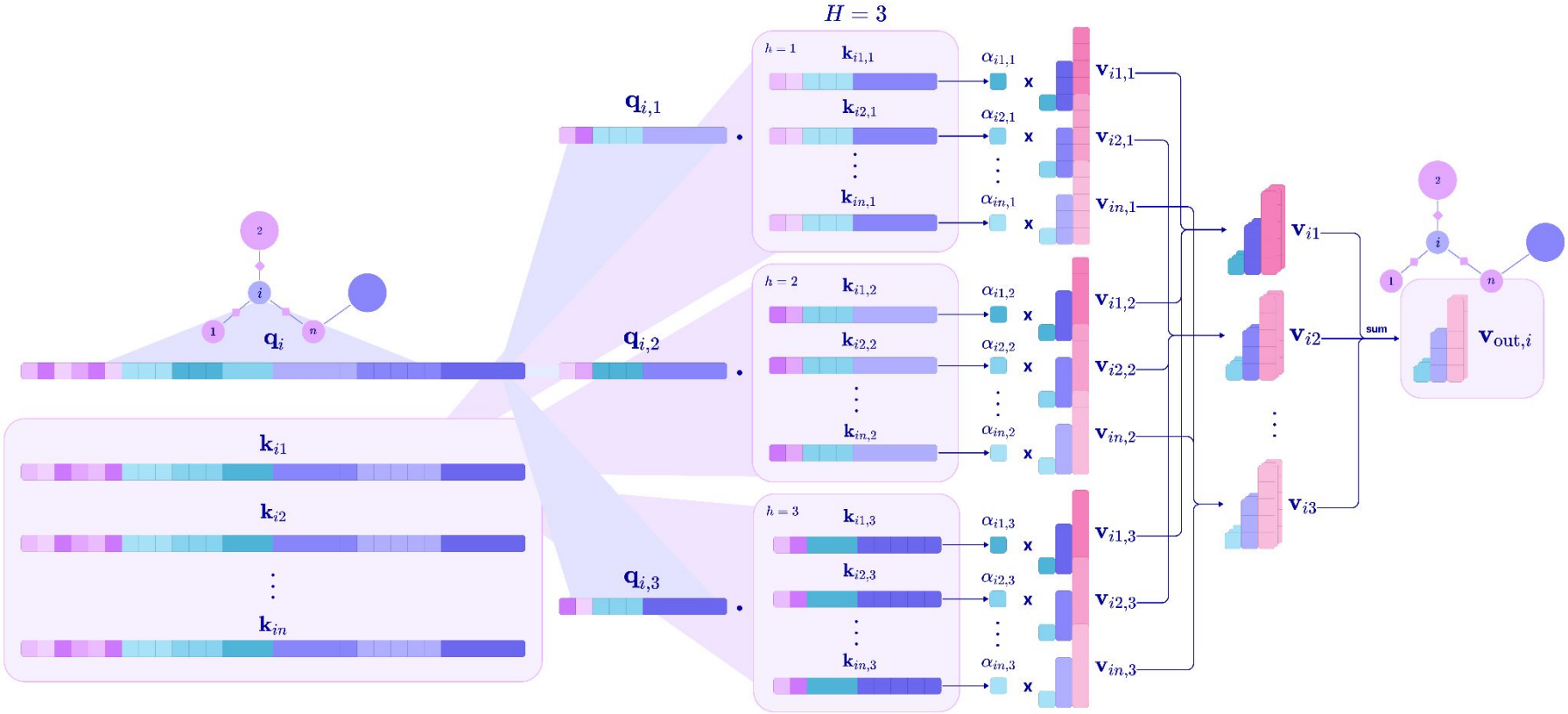
Attentive Self-Interaction

Equipping equivariant networks with attention mechanisms



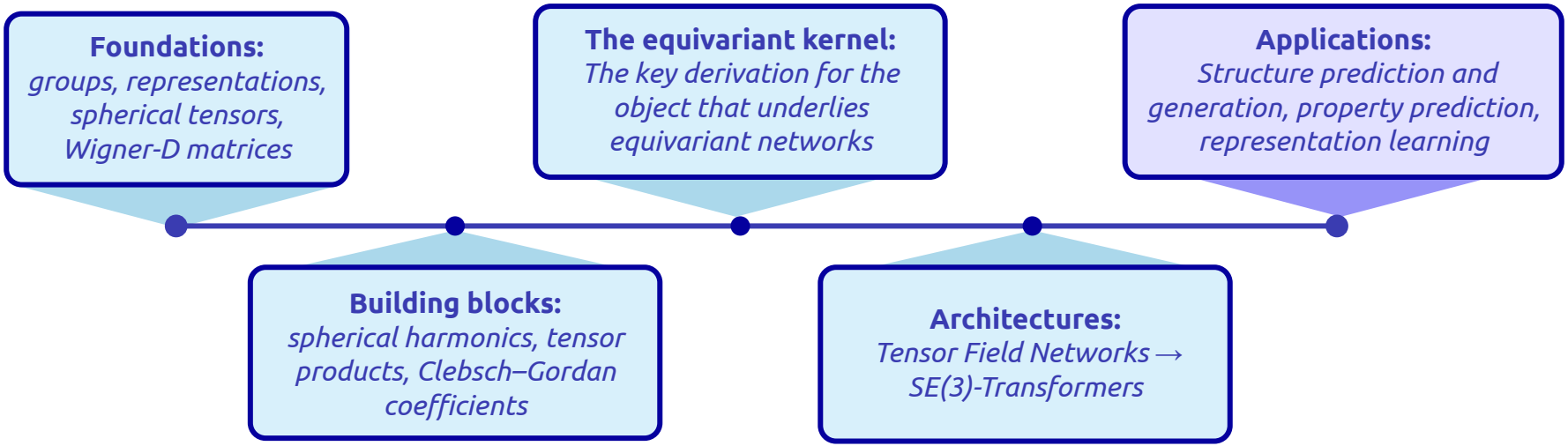
Multi-Head Self-Attention

Equipping equivariant networks with attention mechanisms



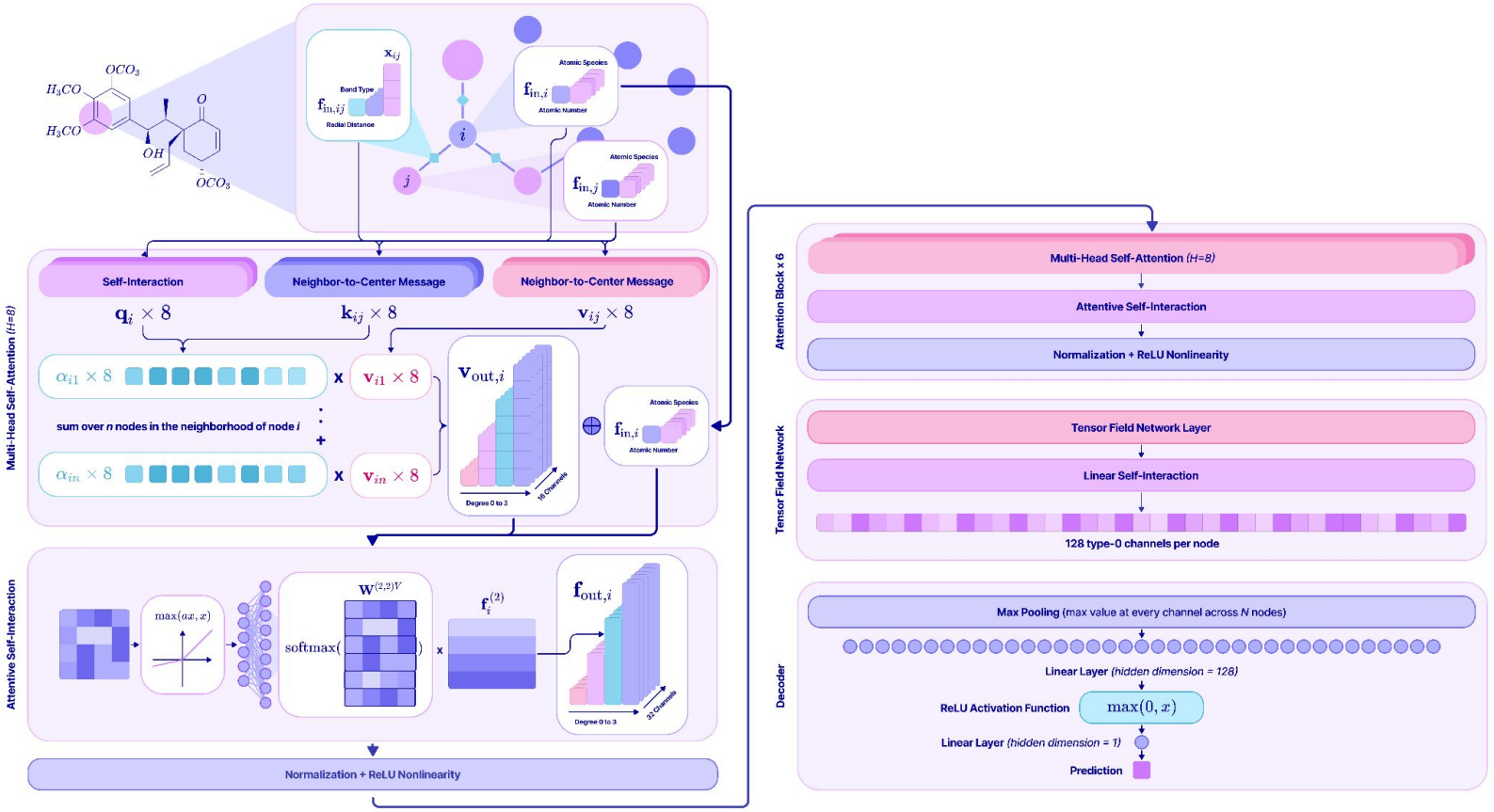
Roadmap

From group theory to a spherical equivariant graph transformers



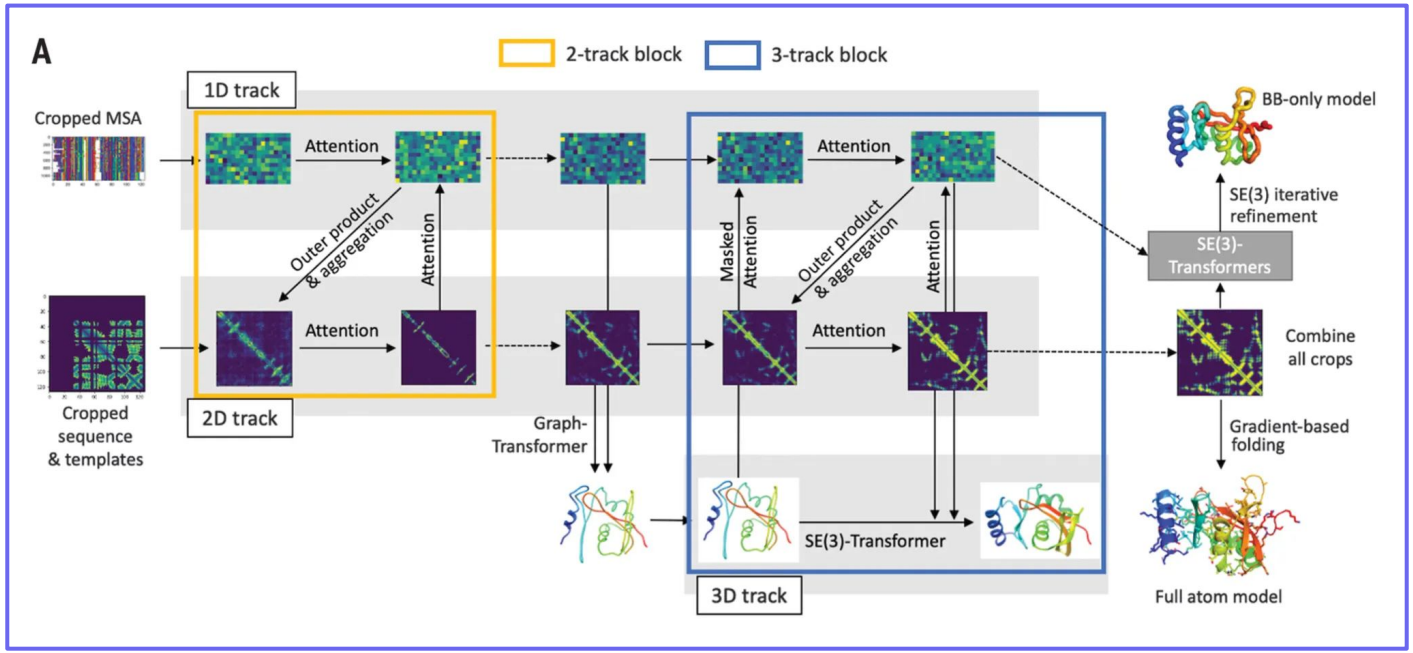
Chemical Property Prediction

Prediction of system-level chemical properties invariant to translations and rotations



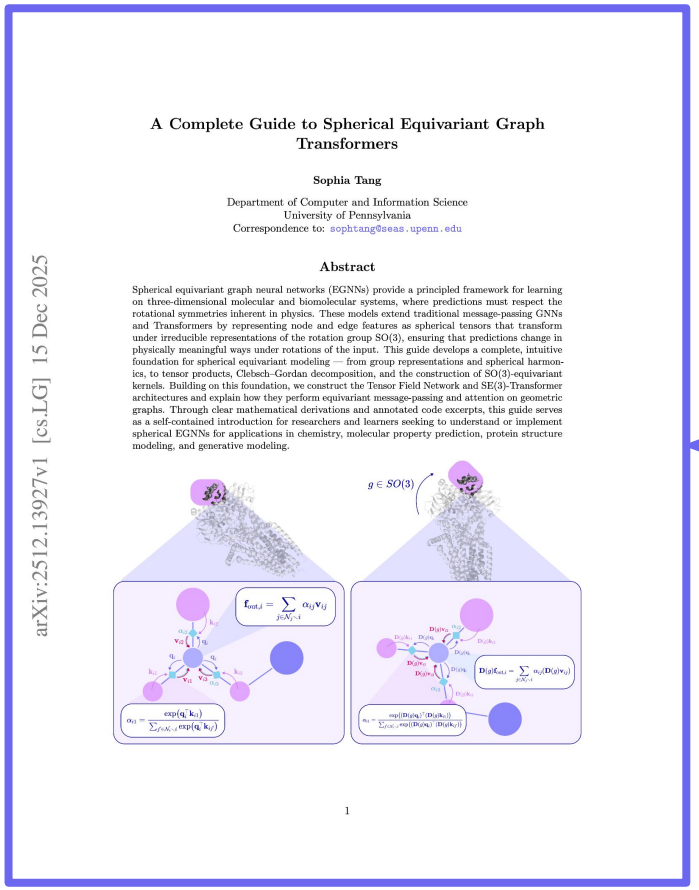
Protein Structure Prediction and Generation

RosettaFold and RFDiffusion are built on spherical equivariant networks



Beak et al. 2021

Additional Resources



A Complete Guide to Spherical Equivariant Graph Transformers

Sophia Tang

Department of Computer and Information Science
University of Pennsylvania
Correspondence to: sophtang@seas.upenn.edu

Abstract

Spherical equivariant graph neural networks (EGNNs) provide a principled framework for learning on three-dimensional molecular and biomolecular systems, where predictions must respect the rotational symmetries inherent in physics. These models extend traditional message-passing GNNs and Transformers by representing node and edge features as spherical tensors that transform under irreducible representations of the rotation group SO(3), ensuring that predictions change in physically meaningful ways under rotations of the input. This guide develops a complete, intuitive foundation for spherical equivariant modeling — from group representations and spherical harmonics, to tensor products, Clebsch–Gordan decomposition, and the construction of SO(3)-equivariant kernels. Building on this foundation, we construct the Tensor Field Network and SE(3)-Transformer architectures and explain how they perform equivariant message-passing and attention on geometric graphs. Through clear mathematical derivations and annotated code excerpts, this guide serves as a self-contained introduction for researchers and learners seeking to understand or implement spherical EGNNs for applications in chemistry, molecular property prediction, protein structure modeling, and generative modeling.

Full derivations and code snippets in my 99-page tutorial 👇

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

Alexandre Duval[*,1,2] **Simon V. Mathis**[*,3] **Chaitanya K. Joshi**[*,3] **Victor Schmidt**[*,1,4]
Santiago Miret[5] **Fragkiskos D. Malliaros**[2] **Taco Cohen**[6]
Pietro Liò[3] **Yoshua Bengio**[1,4] **Michael Bronstein**[7]

[1]Mila [2]Université Paris-Saclay[†] [3]University of Cambridge [4]Université de Montréal
[5]Intel Labs [6]Qualcomm AI Research[‡] [7]University of Oxford

[*]Equal first authors. Correspondence to:
{alexandre.duval, schmidtv}@mila.quebec
{simon.mathis, chaitanya.joshi}@cl.cam.ac.uk

A Complete Guide to Protein Folding Prediction with RoseTTAFold: Part I

A 3-hour breakdown of the three-track RoseTTAFold protein prediction model that leverages multi-modal deep learning architectures to transform single sequences into dynamic 3D structures.

Thanks for Listening!

Feel free to ask any questions or contact me at sophtang@engineering.upenn.edu

Penn