

EECE 576L

Advanced Topics in Deep Learning

Lecture 4: Group Equivariant Deep Learning

Part 5: Implementation and verification (optional)

Renjie Liao

University of British Columbia

2026/27 Winter Session, Term 1 (Fall 2026)

Outline

1. Groups and actions
2. Group convolutions
3. Steerable kernels
4. Typed features and 3D networks
- 5. Implementation and verification**

Outline

1. Groups and actions
2. Group convolutions
3. Steerable kernels
4. Typed features and 3D networks
5. Implementation and verification
 - **Transformation contracts**
 - Quarter-turn implementation
 - Geometric message passing
 - Numerical tests
 - Cost and symmetry choices

Transformation Contracts

The task fixes how the prediction must transform.

Task	Transformed input	Required output
Classification	Move or rotate image	Same class scores
Segmentation	Move or rotate image	Move or rotate mask
Force prediction	Rotate and move points	Rotate force vectors

Composition Preserves Equivariance

Assumption: every operation respects its input and output actions.

$$F_\ell T_g^{\ell-1} = T_g^\ell F_\ell \quad (g \in G)$$



$$F_2 F_1 T_g^0 = F_2 T_g^1 F_1 = T_g^2 F_2 F_1$$

Result: the complete network is equivariant.

An invariant readout then makes the final prediction invariant.

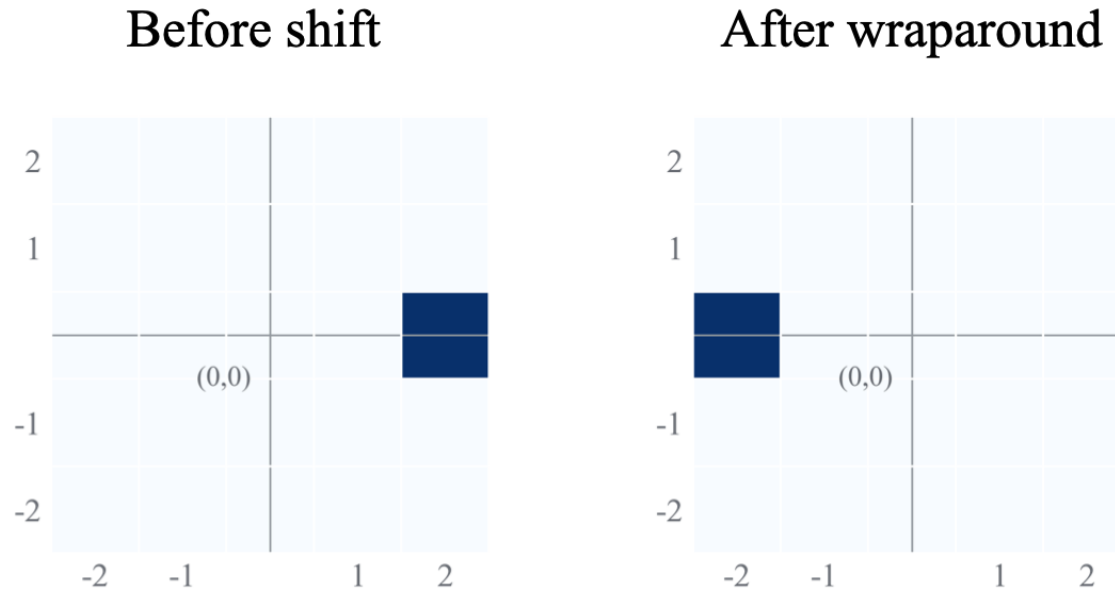
Outline

1. Groups and actions
2. Group convolutions
3. Steerable kernels
4. Typed features and 3D networks
5. Implementation and verification
 - Transformation contracts
 - **Quarter-turn implementation**
 - Geometric message passing
 - Numerical tests
 - Cost and symmetry choices

An Exact Periodic-Grid Example

Square grid, odd kernels, unit stride, circular padding.

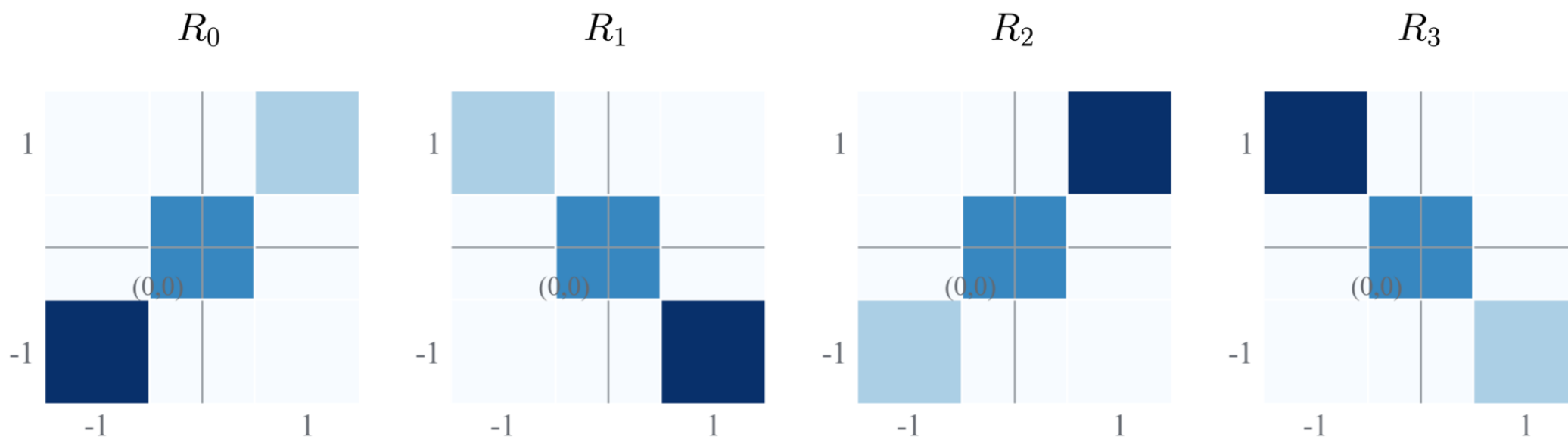
$$p4 = \mathbb{Z}^2 \rtimes C_4$$



This periodic quotient makes integer shifts and quarter-turns exact.

One Kernel, Four Derived Orientations

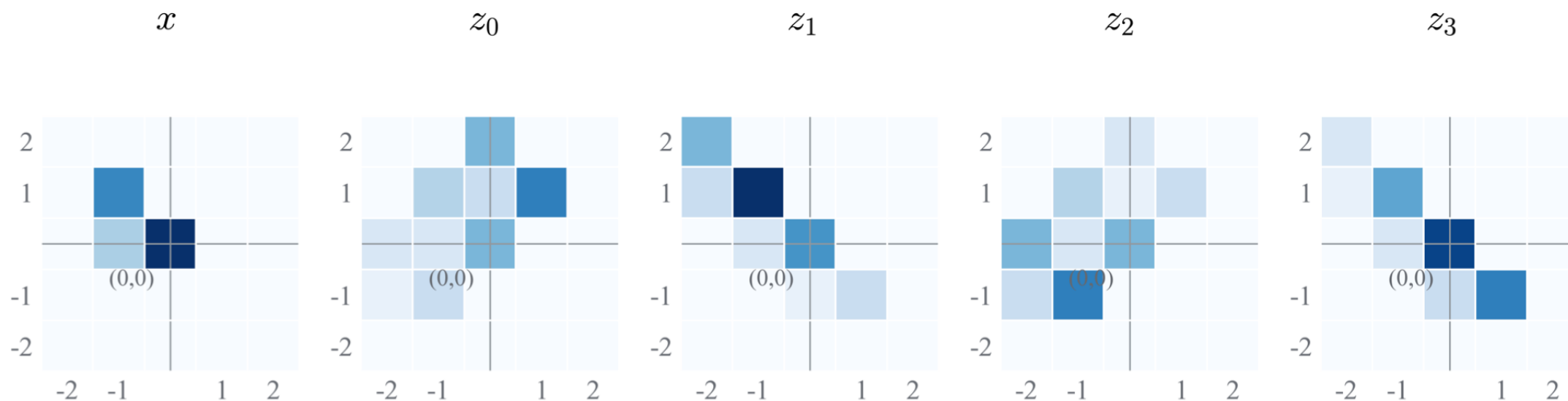
Learning one base kernel determines the complete rotated bank.



These four arrays share parameters throughout training.

Lifting Creates Four Spatial Response Maps

The asymmetric input gives a spatial response for every orientation.



$$z = \text{lift}(x, w) \in \mathbb{R}^{B \times C_{\text{out}} \times 4 \times H \times W}$$

A Minimal Lifting Implementation

Each convolution uses a rotated view of the same base weight.

```
def lift(x, w):  
    return torch.stack([  
        spatial_corr(x, torch.rot90(w, r, (-2,-1)))  
        for r in range(4)], dim=2)
```

Base weight

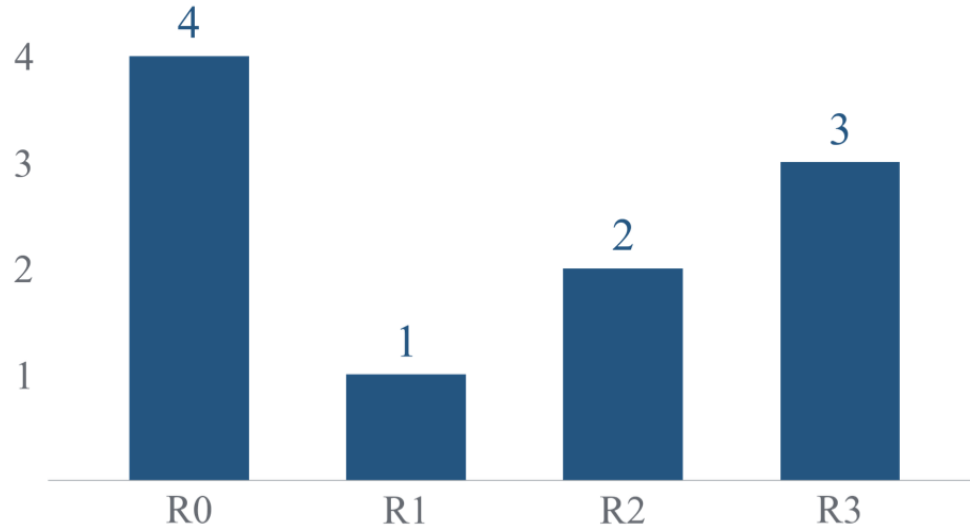
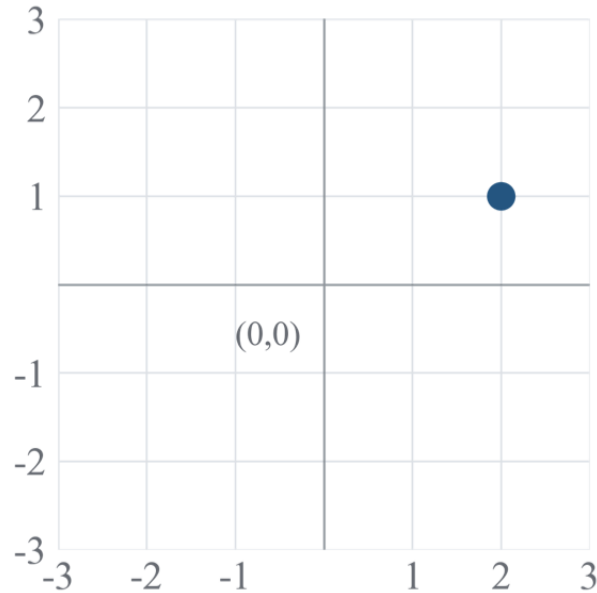
$$C_{\text{out}} \times C_{\text{in}} \times k \times k$$

Lifted feature

$$B \times C_{\text{out}} \times 4 \times H \times W$$

The Hidden Orientation Axis

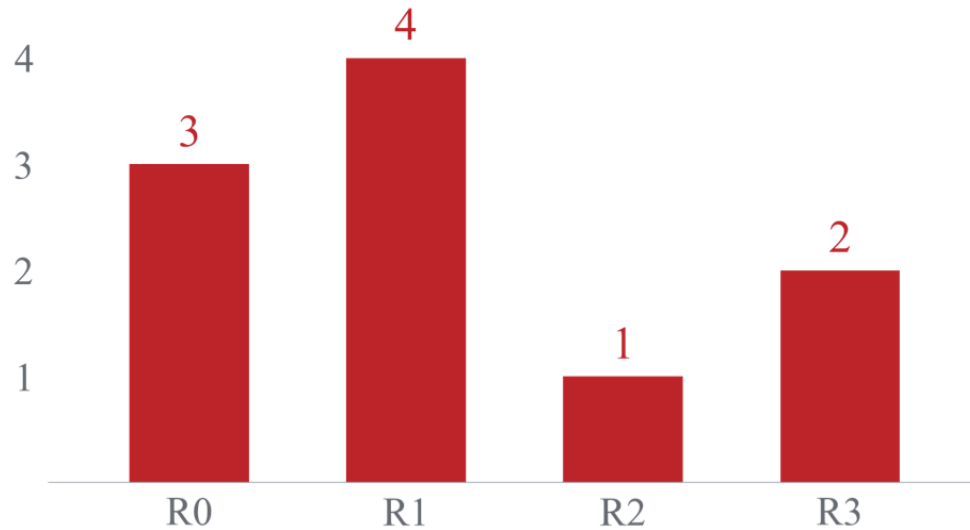
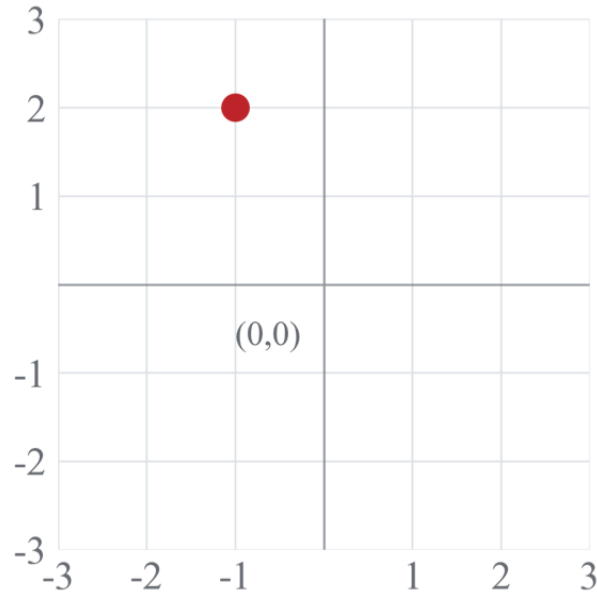
A quarter-turn moves the spatial position and cycles the four entries.



$$t = (2, 1), \quad z(t, :) = (4, 1, 2, 3)$$

The Hidden Orientation Axis After Rotation

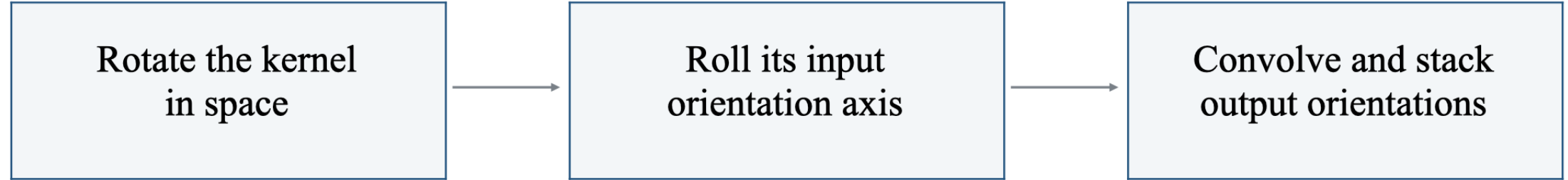
A quarter-turn moves the spatial position and cycles the four entries.



$$R_1 t = (-1, 2), \quad (T_1 z)(R_1 t, :) = (3, 4, 1, 2)$$

The Hidden Group-Convolution Algorithm

For each output orientation, rotate space and shift the input axis.



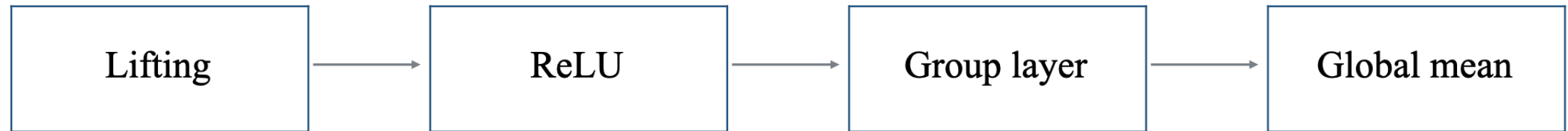
```
wr = torch.rot90(w, r, (-2,-1)).roll(r, dims=2)
```

$$w : C_{\text{out}} \times C_{\text{in}} \times 4 \times k \times k$$

Flatten channels and input orientations in the same order.

A Complete Quarter-Turn Classifier

Hidden features retain the orientation axis until the readout.



$$y_{b,c,r,u,v} \leftarrow y_{b,c,r,u,v} + b_c$$

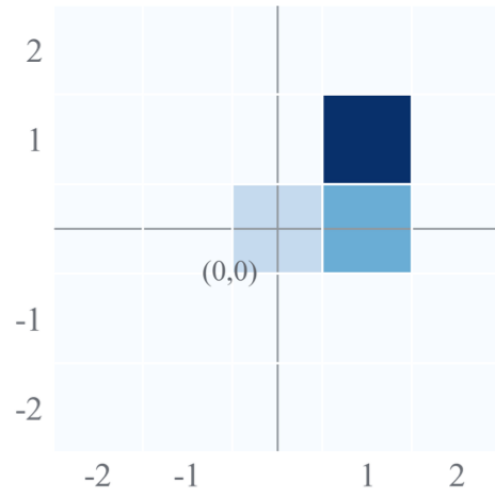
ReLU is valid because this action permutes feature entries.

Share each bias and normalization statistic across orientations.

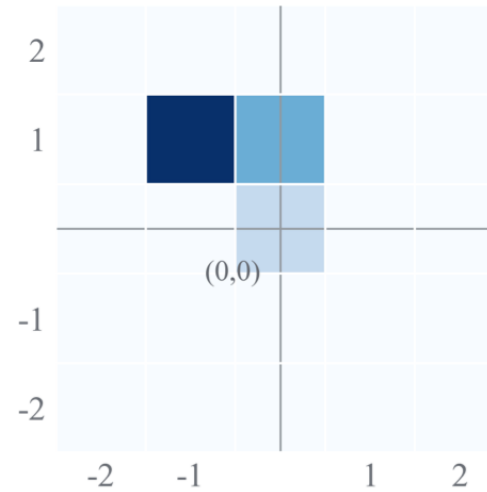
Orientation Pooling Still Leaves a Spatial Map

$$q_c(u, v) = \frac{1}{4} \sum_r y_{c,r,u,v}$$

Orientation-pooled map



After a quarter-turn



$$z_c^{\text{cls}} = \frac{1}{HW} \sum_{u,v} q_c(u, v) = 0.28$$

Training the Base Parameters

The optimizer updates the shared coefficients.



Backpropagation passes through the derived rotations.

Every update stays inside the equivariant layer family.

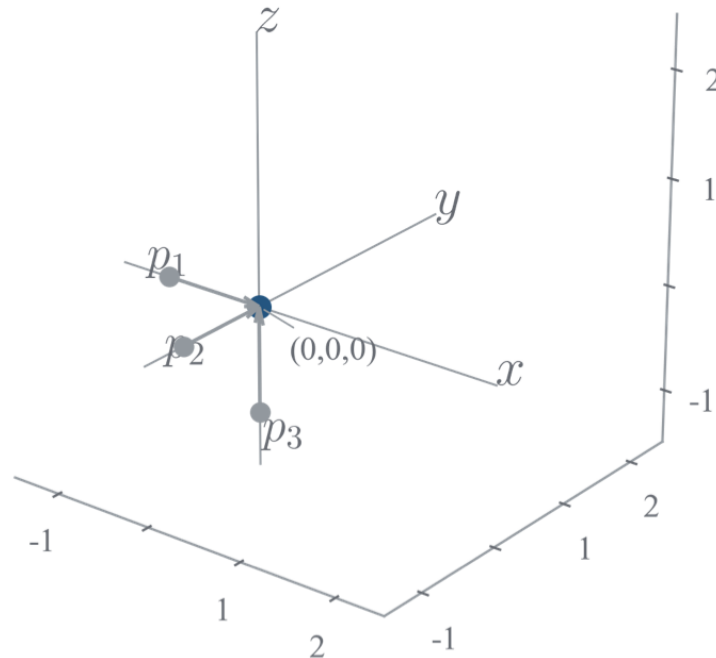
Augmentation can cover variation outside the chosen group.

Outline

1. Groups and actions
2. Group convolutions
3. Steerable kernels
4. Typed features and 3D networks
5. Implementation and verification
 - Transformation contracts
 - Quarter-turn implementation
 - **Geometric message passing**
 - Numerical tests
 - Cost and symmetry choices

Scalar Features Weight Neighbor-to-Node Vectors

A radial coefficient and a scalar feature scale each relative vector.



$$p_i \in \mathbb{R}^3, \quad h_j \in \mathbb{R}$$

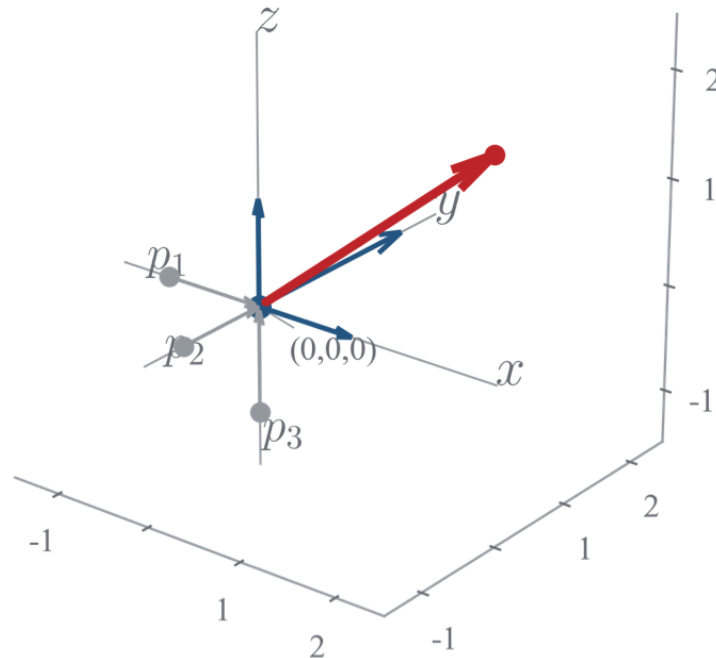
$$r_{ij} = p_i - p_j$$

$$m_{ij} = a(\|r_{ij}\|^2)h_j r_{ij}$$

The edge points from the neighbor toward the receiving node.
Here all three neighbors are one unit away.

Summing the Typed Messages

The scalar weights change magnitudes while preserving vector type.



$$h_1 = 1, \quad h_2 = 2, \quad h_3 = 1, \quad a(1) = 1$$

$$m_{i1} = (1, 0, 0), \quad m_{i2} = (0, 2, 0)$$

$$m_{i3} = (0, 0, 1)$$

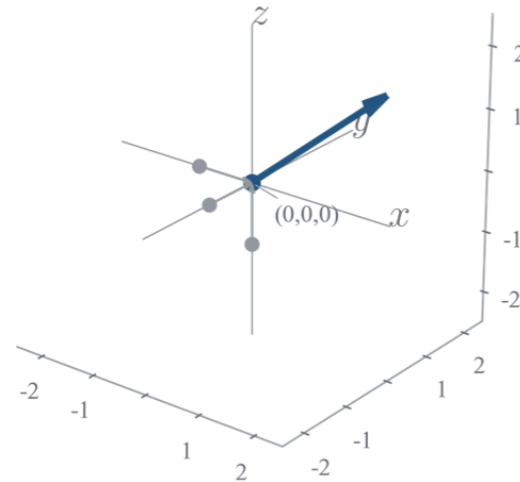
$$v_i = \sum_j m_{ij} = (1, 2, 1)$$

Blue vectors are messages.
The red vector is their sum.

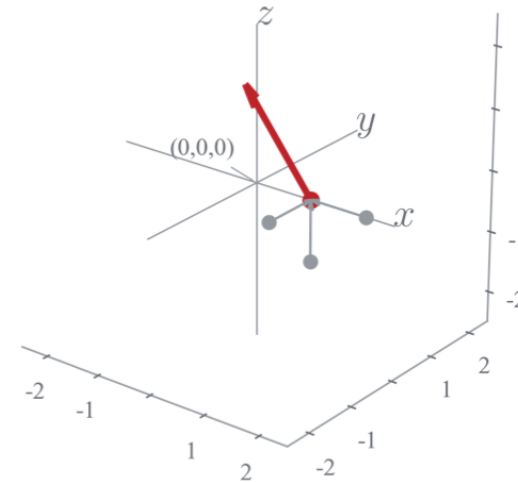
Rigid Motion Preserves the Message Rule

Distances and scalar weights remain fixed. The vector sum rotates.

Before rigid motion



After rigid motion



$$p_i \mapsto Rp_i + t, \quad v_i \mapsto Rv_i = (-2, 1, 1)$$

The Scalar-to-Vector Implementation

The demo replaces the learned radial function by a smooth fixed profile.

```
def scalar_to_vector(pos, h):  
    delta = pos[:, None, :] - pos[None, :, :]  
    d2 = (delta * delta).sum(-1)  
    weight = torch.exp(-d2) * h[None, :]  
    return (weight[..., None] * delta).sum(dim=1)
```

$$O(N^2) \longrightarrow O(|E|)$$

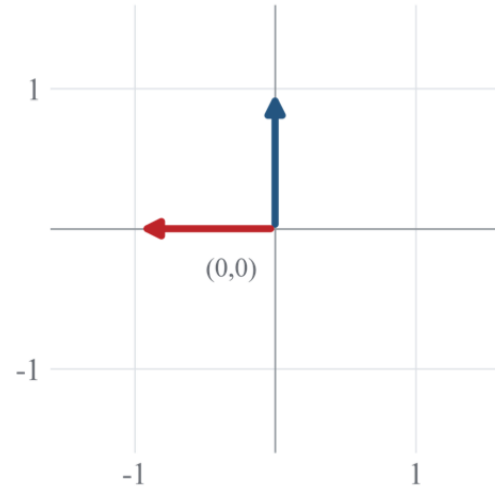
A radius graph reduces pair operations. Neighbor search adds cost.

A Compatible Vector Nonlinearity

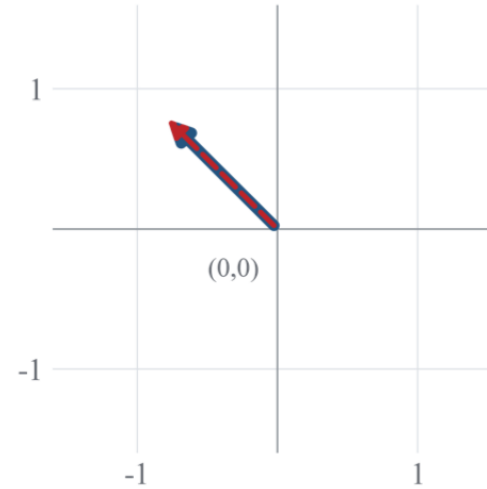
A norm gate rescales a vector without choosing a coordinate direction.

$$v = (1, -1, 0), \quad R = \text{diag}(-1, -1, 1)$$

Componentwise ReLU



Norm gate



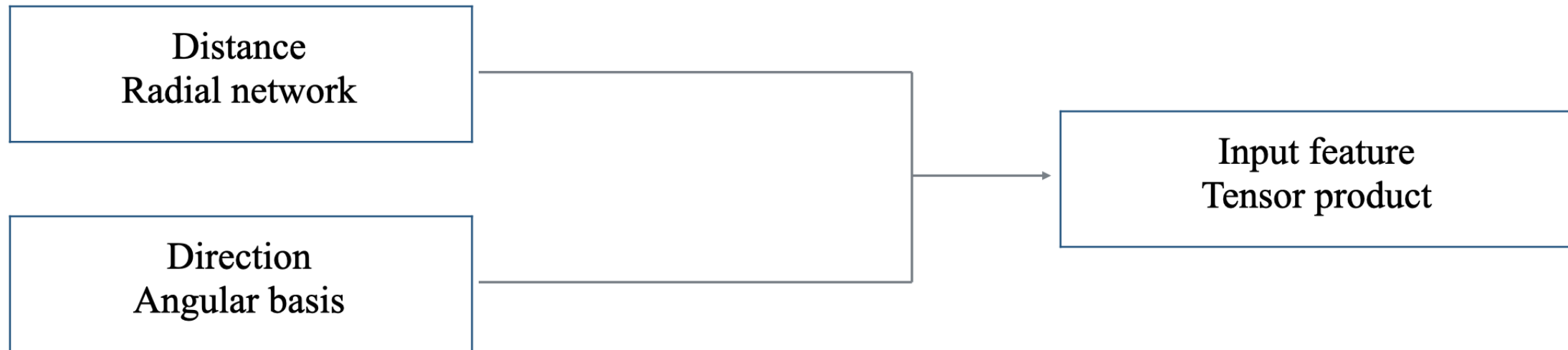
Blue: rotate, then activate

Red: activate, then rotate

$$\text{gate}(v) = \text{sigmoid}(\|v\|)v$$

Typed Message Passing in a Library

Libraries combine learned radial weights with fixed type-compatible bases.



$$m_{ij} = \sum_a w_a(\|r_{ij}\|) \mathcal{C}_a(h_j \otimes Y_{J_a}(\hat{r}_{ij}))$$

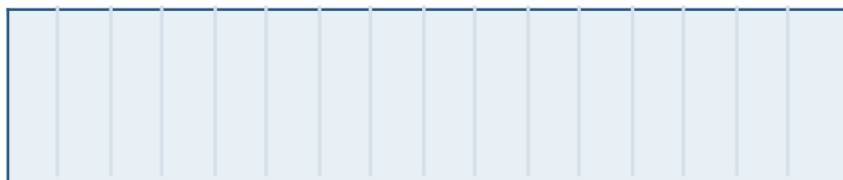
Sum by destination, then apply a compatible activation.

Feature Types Specify a Tensor Layout

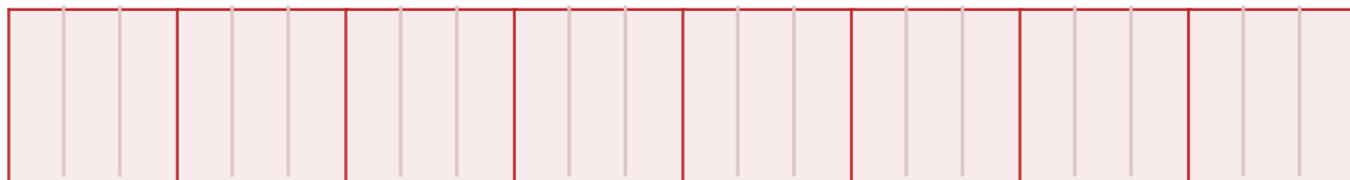
Copies are learned channels. Coordinates within a type transform together.

$16 \times 0e + 8 \times 1o$

16 scalar coordinates



8 vector copies, each with 3 coordinates



$$16 \cdot 1 + 8 \cdot 3 = 40$$

Inversion preserves even scalars and negates polar vectors.

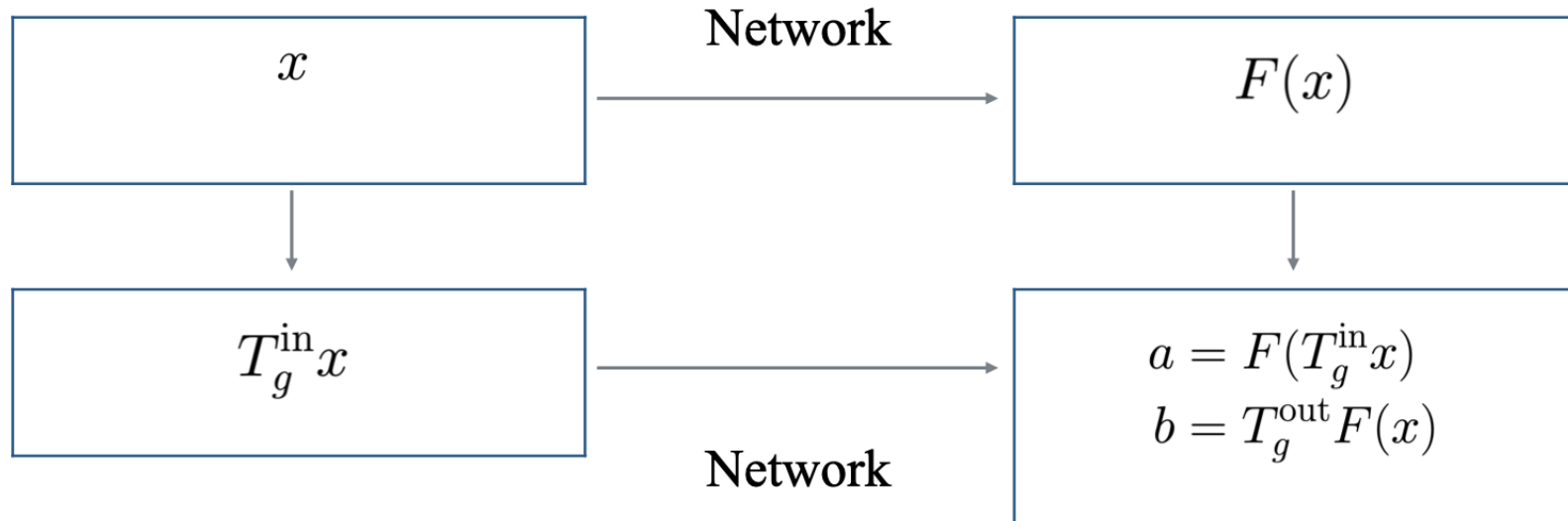
e3nn supports 3D types. escnn supports typed spatial fields.

Outline

1. Groups and actions
2. Group convolutions
3. Steerable kernels
4. Typed features and 3D networks
5. Implementation and verification
 - Transformation contracts
 - Quarter-turn implementation
 - Geometric message passing
 - **Numerical tests**
 - Cost and symmetry choices

Equivariance as a Two-Path Test

The same input and transformation define two comparable outputs.



$$\varepsilon_{\text{rel}} = \frac{\|a - b\|_2}{\max(\|a\|_2, \|b\|_2, \epsilon)}$$

A Quarter-Turn Unit Test

CPU float64, nonsymmetric random inputs, and random base weights.

```
for r in range(4):  
    a = lift(torch.rot90(x, r, (-2,-1)), w)  
    b = rotate_hidden(lift(x, w), r)  
    torch.testing.assert_close(a, b,  
                               atol=1e-10, rtol=1e-10)
```

Hidden action `torch.rot90(z, r, (-2,-1)).roll(r, dims=2)`

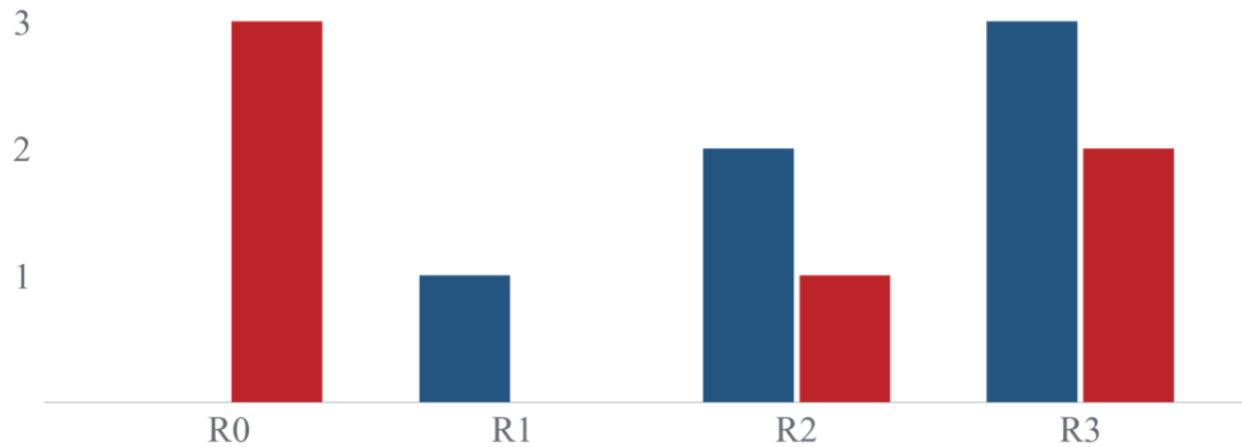
Use square grids, odd kernels, unit stride, and circular padding.

An Orientation-Dependent Bias Fails the Test

For zero input, four unrelated biases already break the rotation rule.

Transform input, then predict

Predict, then transform



$$\max_r |a_r - b_r| = 3$$

Conditions for the Complete Pipeline

Operation	Required condition
Bias and normalization	Parameters and statistics respect the feature type
Residual addition	Both branches use the same feature action
Subsampling	The output lattice supports the chosen action
Pooling	Reduce the intended orientation and spatial orbit
Dropout	Transport masks consistently for samplewise equality

A Useful Test Sequence

Transformation laws	Identity, inverse, and composition
----------------------------	------------------------------------

Components and readouts	Nonconstant inputs and multiple random weights
--------------------------------	--

Geometric inputs	Transform points and all typed blocks. Permute nodes.
-------------------------	---

Training conditions	Device, precision, sampling, stochastic layers, gradients
----------------------------	---

Small numerical errors support an implementation check.

Outline

1. Groups and actions
2. Group convolutions
3. Steerable kernels
4. Typed features and 3D networks
5. Implementation and verification
 - Transformation contracts
 - Quarter-turn implementation
 - Geometric message passing
 - Numerical tests
 - **Cost and symmetry choices**

Parameter Copies and Orientation Channels

Hold the number of learned feature copies per orientation fixed.

 A

Orientations

 C_{in}

Input copies

 C_{out}

Output copies

Hidden-layer input

$$C_{\text{in}}^{\text{total}} = AC_{\text{in}}$$

$$P_{\text{lift}} = C_{\text{out}}C_{\text{in}}k^2$$

$$P_{\text{group}} = AC_{\text{out}}C_{\text{in}}k^2$$

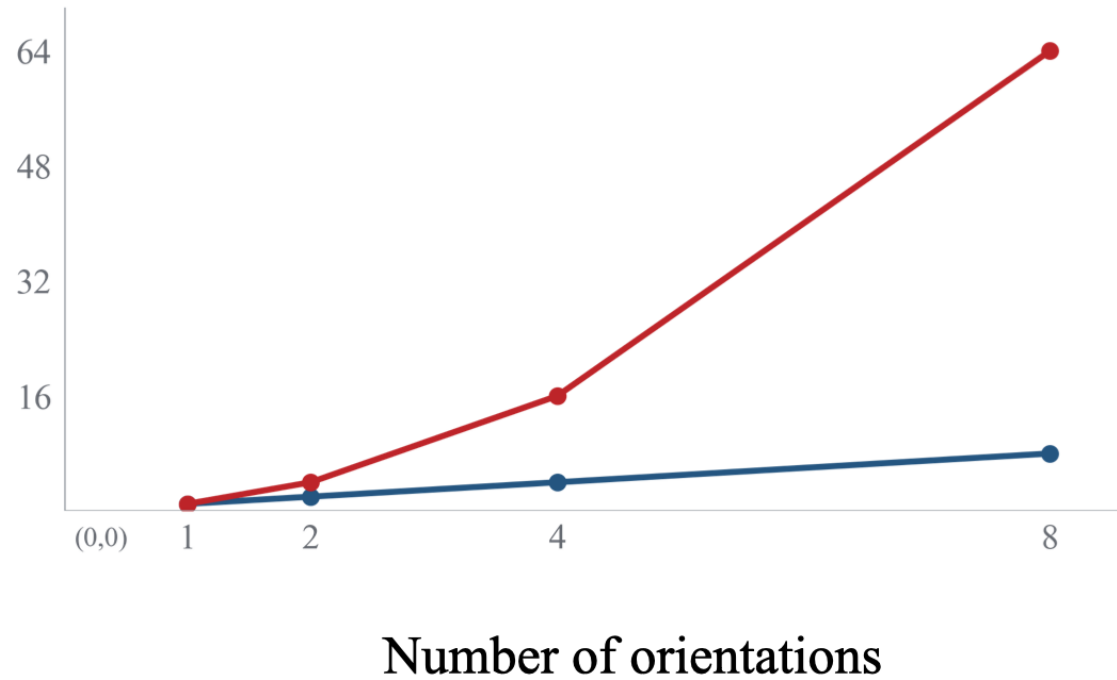
Biases are excluded from these kernel counts.

The Direct Group-Convolution Cost

At fixed copies per orientation, both input and output axes add work.

Parameter factor

Multiply-add factor



$$O(A^2 C_{\text{out}} C_{\text{in}} k^2 H W)$$

Choosing the Symmetry Budget

The transformation assumption must agree with the prediction task.

Invariant class scores

Appropriate when moving or rotating the input preserves its label.

Spatial or vector outputs

Preserve the information needed to locate regions or predict directions.

Reflection symmetry

Include it only when reflecting the input preserves the target rule.

A Fair Experimental Comparison

Use the same data split and a comparable tuning budget.

Ordinary network

Same network
with augmentation

Equivariant network

Compare ordinary and transformed-input accuracy.

Report equivariance error, parameter count, runtime, and data size.

Include inputs for which the symmetry is only approximate.

Questions?

Implementation and verification

Choose a symmetry. Specify its action. Test the implementation.

References

- [1] Bekkers. An Introduction to Group Equivariant Deep Learning. University of Amsterdam.
- [2] PyTorch contributors. Conv2d documentation.
- [3] PyTorch contributors. torch.rot90 documentation.
- [4] Geiger & Smidt (2022). e3nn: Euclidean Neural Networks.

References (continued)

- [5] e3nn contributors. Convolution implementation guide.
- [6] e3nn contributors. Equivariance Testing guide.
- [7] e3nn contributors. Irreps API documentation.
- [8] escnn contributors. Official documentation and typed-field API.