A PROOFS OF PROPOSITION

The concept of equivariant basis expansion involves using a set of equivariant vectors in the output space to represent any equivariant operation. It differs from the notion of basis functions commonly used in linear mappings. In mathematical terms, any vector-valued function can be decomposed into multiple scalar-valued functions using a basis of the output space. For a mapping $f : R^m \to R^n$, we can simply use n orthogonal vectors $\{e_i\}$ to rewrite the function as $f(x) = \sum_i^n [f(x)^T e_i] e_i = \sum_i^n g_i(x) e_i$, which is not interesting at all. However, if f is an equivariant operation and each e_i also transforms equivariantly, then $g_i(x)$ is forced to be *invariant*. These vectors e_i enable us to describe an equivariant function based on how it affects the norms of these vectors. Importantly, the equivariant basis does not need to be orthogonal, and the vectors may exhibit linear dependencies.

A.1 SCALARIZATION AND EQUIVARIANT BASIS

Proposition 1 means that we can use the input type-1 vectors $\{u_i \in R^3\}$ as equivariant basis. When $\{u_i\}$ span the whole output space, it naturally holds. When $\{u_i\}$ do not span the space, let $\{w_1, \cdots, w_r\}$ denote an orthonormal basis of the orthogonal complement to span($\{u_i\}$), then $f(h_{\text{in}}) = \sum_i g_i(h_{\text{in}}) u_i + \sum_t \beta_t(h_{\text{in}}) w_t$, where h_{in} is the collection of all input vectors and scalars. The proposition claims that $\beta_T(h_{\text{in}}) = 0$. To prove this, we can construct a group action $\hat{Q} \in O(3)$ such that $\hat{Q}(u_i) = u_i$ for all $\{u_i\}$ and $\hat{Q}(w_t) = -w_t$ for all $\{w_t\}$. Therefore, $\hat{Q}(f(h_{\text{in}}) = \sum_i g_i(h_{\text{in}}) \hat{Q} u_i + \sum_t \beta_t(h_{\text{in}}) \hat{Q} w_t = \sum_i g_i(h_{\text{in}}) u_i - \sum_t \beta_t(h_{\text{in}}) w_t$. While since the input vectors $\{u_i\}$ are linear independent with w_t, $\hat{Q} h_{\text{in}} = h_{\text{in}}$. Therefore, the equivariant of operation f implies that

$$\sum_i g_i(h_{\text{in}}) u_i + \sum_t \beta_t(h_{\text{in}}) w_t = \sum_i g_i(h_{\text{in}}) u_i - \sum_t \beta_t(h_{\text{in}}) w_t, \tag{11}$$

which means all $\beta_t(h_{\text{in}}) = 0$. For a general discussion about the scalarization method, please refer to Villar et al. (2021).

A.2 TENSOR PRODUCT AND EQUIVARIANT BASIS

For a convolution layer, the equivariant operation has the form $f(x, h) = \kappa(x) h$, which is linear transformation of feature h. We only consider the output has a single type-l. We can use multiple such operations to construct equivariant operation whose output has multiple type-$l_1, l_2, \cdots$ components. Both the input and output are steerable features. So the matrix $\kappa(x)$ splits into blocks $\kappa^{lj}(x) \in R^{(2l+1)\times(2j+1)}$, which transform a type-j vector in h to a type-l vector. These block are steerable under the group action $g \in SO(3)$:

$$\kappa^{lj}(D^1(g)x) = D^l(g) \kappa^{lj}(x) D^j(g)^{-1}. \tag{12}$$

Follow the proof in Weiler et al. (2018); Fuchs et al. (2020), there is a basis

$$\eta_J^{lj}(x) = \sum_{m=-J}^{J} Y_{Jm}(x/||x||) C_{Jm}^{lj}, \tag{13}$$

where C_{Jm}^{lj} is CG coefficient and $Y_{Jm}(\cdot)$ is the m^{th} dimension of the J^{th} spherical harmonic $Y_J : R^3 \to R^{2J+1}$. And kernel $\kappa^{lj}(x)$ can be expanded with some scalar function $\varphi(||x||)$

$$\kappa^{lj}(x) = \sum_{J=|j-l|}^{j+l} \varphi_J^{jl}(||x||) \eta_J^{lj}(x). \tag{14}$$

If we take equivariant basis $e_i(x, h) = \eta_J^{lj}(x) h^j$ and invariant coefficients $g_i(||x||) = \varphi_J^{jl}(||x||)$, where index i fuses the index J and j, then tensor product method $f(x, h) = \kappa(x) h$ can be directly written as

$$f(x, h) = \sum_i g_i(||x||) e_i(x, h). \tag{15}$$

B THE PERFORMANCE OF SINET

B.1 COMPUTATIONAL COMPLEXITY OF SINET

According to Passaro & Zitnick (2023), the computational complexity of tensor product method is $O(n_c \cdot L^6 + n_c^2 \cdot L^3)$, where L is the largest l in the node feature and n_c is the number of channels. In scalar interaction, this complexity is $O(n_c^2 \cdot L^2)$. We decompose scalar interaction into three steps. Firstly, we construct L matrices $U^i \in R^{(2l_i+1)\times(n_c+1)}$, where we follow the same setting in Passaro & Zitnick (2023) to assume the channel of edge feature is 1. Computing the inner product of each vector in U^i accounts for the complexity of $O(n_c^2 \cdot L)$. Therefore the total complexity in step 1 is $O(n_c^2 \cdot L^2)$. Secondly, a neural network is used to compute the n_c^2 coefficients for each l, and the total output dimension of the neural network is $n_c^2 \cdot L$. This step accounts for $O(n_c^2 \cdot L^2)$ operations and depends on the hidden dimension of the neural network. The last step applies $O(L) \times O(n_c)$ linear combinations and each use n_c+1 vectors of dimension $2l_i+1$ to compute a vector of the same dimension, leading to $O(n_c \cdot L)$ operations. Therefore, the last step needs $O(n_c^2 \cdot L^2)$ operations. The overall computational complexity is $O(n_c^2 \cdot L^2)$.

In contrast, the scalar interaction method has a higher number of trainable parameters compared to the tensor product method. This increase in parameters primarily arises from the additional neural network, and it can be adjusted by modifying the number and dimension of the hidden layers. As a consequence, there is also a slight increase in memory usage, which is still within acceptable limits.

B.2 EXPRESSIVENESS

With the assumption that for an equivariant operation $f : H_{\text{in}} \to V^l$, there are enough type-$l$ vectors in the collection of input features fragments to span the output space V^l, the expressiveness of scalar interaction is limited by the invariant function $g_i(h_{\text{in}})$ that acts as coefficients. Scalar interaction first computes the inner product between vectors of the same type-l_i. Let $U^i = [h_1^{l_i}, \cdots, h_k^{l_i}] \in R^{(2l_i+1)\times(n_{l_i})}$ denote all type-l_i fragments of input feature h_{in}, then the scalars are computed as $M = (U^i)^T U^i$. Villar et al. (2021) gives some analysis on this inner-product procedure. With Cholesky decomposition of M, we can reconstruct $h'_1, \cdots, h_k$ modulo the orthogonal group $O(2l+1)$. This tells us that if g_i is an invariant function under group $O(2l+1)$ and takes U^i as input, it can be written as

$$g_i(U^i) = g_i(Cholesky\_decompose((U^i)^T U^i) = \sigma_i(M). \tag{16}$$

Since σ is a mapping that takes $\text{vec}(M)$ as input, we can use a neural network to approximate it.

When $g_i(h_{\text{in}})$ is $SO(3)$ invariant, the reconstruction with Cholesky decomposition can no longer contain all information that should be used. Hence, scalar interaction based on inner product is no longer universal. We regard this as a limitation of scalar interaction. And methods beyond computing inner product could be a solution for that.

C OTHER WORK FOR REDUCING THE COMPUTATIONAL COMPLEXITY OF TENSOR PRODUCT

The modified tensor product method proposed by Passaro & Zitnick (2023) can also reduce the computational cost of tensor product. The motivation of their work is that if the irreps' primary axis is aligned to the edge's direction, the steerable filter in the tensor product will only have a single rotational freedom. Thus, they utilize the equivariance of tensor product and propose to first rotate the irreps with $\hat{g}$ to align the primary axis, then apply the tensor product, and at last rotate the output with $\hat{g}^{-1}$. Their computational complexity is $O(n_c^2 \cdot L^3)$. However, this approach cannot handle more complicated situation. For example, in the equivariant MLP of non-linear message passing of SEGNN and Equiformer, neither of the input features is the spherical harmonics of a 3D vector x_{ij}, which cannot be modeled with their modified tensor product method.

D EXPERIMENT DETAILS

All experiments were conducted on an NVIDIAő A100 Tensor Core GPU with 80GB of memory. The implementation of the scalar interaction method is based on the e3nn PyTorch library. To

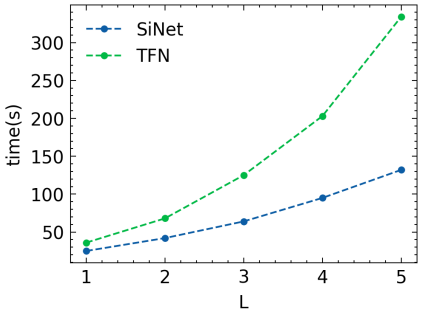

Figure 4: The inference time for one epoch of different models

ensure high performance, we followed the implementation approach of the tensor product in e3nn and utilized the torch.jit module for compilation.

An detailed aspect of the scalar interaction method is how it incorporates radial information, which is commonly utilized in the tensor product method to assign weights to different paths. In scalar interaction, the radial information can be treated as a type-0 vector and concatenated with one of the input features. However, to maintain consistency with the tensor product method, SINet adopts a similar interface where the embedding of the radial information is treated as an additional input. It is then added to the hidden layer of the neural network in the scalar interaction operation. This allows SINet to effectively leverage the radial information while maintaining compatibility with the tensor product approach.

As our equivariant model based on scalar interaction, SINet utilizes non-linear message passing similar to SEGNN Brandstetter et al. (2022b). For SiFormer, we adopt the architecture of Equiformer and replace the tensor product except the first layer with scalar interaction. We also remove most of the non-linearity, such as layer norm and gate layer, in the models since scalar interaction can provide non-linear mapping.

D.1 The Inference Speed of SINet

We also compare the speed of a equivariant message passing model based on scalar interaction with that based on tensor product. We run the model on the QM9 dataset and compare the inference time for one epoch of different models. We set the channels 32 and batch size 128.

D.2 Discussion about the Experiment for REBIF Completeness

Figure 3 illustrates the completeness of REBIF in SINet. It can be observed that in SINet, the input fragment vectors on a node may not span the entire output space for an equivariant operation. Although this imposes a limitation on the universality of using input features as a basis for the output space, it is not a problem when considering the following two perspectives. Firstly, such incompleteness introduces a meaningful bias to the feature direction. Consider a feature that does not contain a complete basis for a space. What direction components does it possess? The answer lies in the relative direction between its neighbors. Therefore, with SINet, the direction between nearby neighbors is more likely to be preserved in the feature, while in the tensor product method, these directions are fused with new directions generated by the tensor product. Secondly, if there is a requirement to make all input features complete, one can easily apply multiple tensor product layers at the beginning of a network and then utilize scalar interaction for the remaining layers.

D.3 Hyper-parameters for QM9 experiments

For SINet, the model is optimized using the Adam optimizer with no weight decay. The initial learning rate is set to 0.001, and it is reduced by a factor of 0.2 at epochs 400, 500, 600, and 700. The SINet model configuration includes $L = 3$ (the maximum l value), $n_c = 32$ (the number of channels), and a neural network with a single hidden layers of dimension 128 in the scalar interaction operation. The model consists of 8 message passing layers.

As for SiFormer, we follow the same training method and hyper-parameters as described in Liao & Smidt (2022).

D.4 HYPER-PARAMETERS FOR N-BODY EXPERIMENTS

The model is optimized using the Adam optimizer without weight decay. The initial learning rate is set to 0.001 and is multiplied by 0.99 every 10 epochs. The configuration for SINet includes $L = 1$ (the maximum l value), $n_c = 32$ (the number of channels), and a neural network with a single hidden layers of dimension 64 in the scalar interaction operation. The model consists of a total of 6 message passing layers.