Appendix

A. Contributions

- Charline and Yee Whye conceived the project and Yee Whye initially came up with an equivariant form of self-attention.
- Through discussions between Michael, Charline, Hyunjik and Yee Whye, this was modified to the current `LieSelfAttention` layer, and Michael derived the equivariance of the `LieSelfAttention` layer.
- Michael, Sheheryar and Hyunjik simplified the proof of equivariance and further developed the methodology for the `LieTransformer` in its current state, and created links between `LieTransformer` and other related work.
- Michael wrote the initial framework of the `LieTransformer` codebase. Charline and Sheheryar wrote the code for the shape counting experiments, Michael wrote the code for the QM9 experiments, Sheheryar wrote the code for the Hamiltonian dynamics experiments, after helpful discussions with Emilien.
- Charline carried out the experiments for Table 1, Michael carried out most of the experiments for Table 2 with some help from Hyunjik, Sheheryar carried out the experiments for Figure 3b and all the Hamiltonian dynamics experiments.
- Hyunjik wrote all sections of the paper except the experiment sections: the shape counting section was written by Charline, the QM9 section by Michael and the Hamiltonian dynamics section was written by Emilien and Sheheryar.

B. Formal definitions for Groups and Representation Theory

Definition 2. *A **group** G is a set endowed with a single operator $\cdot : G \times G \mapsto G$ such that*

1. *Associativity:* $\forall g, g', g'' \in G, (g \cdot g') \cdot g'' = g \cdot (g' \cdot g'')$
2. *Identity:* $\exists e \in G$, $\forall g \in G$ $g \cdot e = e \cdot g = g$
3. *Invertibility:* $\forall g \in G, \exists g^{-1} \in G, g \cdot g^{-1} = g^{-1} \cdot g = e$

Definition 3. *A **Lie group** is a finite-dimensional real smooth manifold, in which group multiplication and inversion are both smooth maps.*

The general linear group $GL(n, \mathbb{R})$ of invertible $n \times n$ matrices is an example of a Lie group.

Definition 4. *Let S be a set, and let $\mathrm{Sym}(S)$ denote the set of invertible functions from S to itself. We say that a group G **acts** on S via an action $\rho : G \to \mathrm{Sym}(S)$ when ρ is a group **homomorphism**: $\rho(g_1 g_2)(s) = (\rho(g_1) \circ \rho(g_2))(s)$ $\forall s \in S$.*

If S is a vector space V and this action is, in addition, a linear *function, i.e. $\rho : G \to GL(V)$, where $GL(V)$ is the set of linear invertible functions from V to itself, then we say that ρ is a **representation** of G.*

C. Proofs

Lemma 1. *The function composition $f \circ f_K \circ ... \circ f_1$ of several equivariant functions f_k, $k \in \{1, 2, ..., K\}$ followed by an invariant function f, is an invariant function.*

Proof. Consider group representations $\pi_1, \dots, \pi_K$ that act on $f_1, \dots f_K$ respectively, and representation π_0 that acts on the input space of f_1. If each f_k is equivariant with respect to π_k, π_{k-1} such that $f_k \circ \pi_{k-1} = \pi_k \circ f_k$, and f is invariant such that $f \circ \pi_k = f$, then we have

$$\begin{aligned} f \circ f_k \circ ... \circ f_1 \circ \pi_0 &= f \circ f_k \circ ... \circ \pi_1 \circ f_1 \\ &\vdots \\ &= f \circ \pi_k \circ f_k \circ ... \circ f_1 \\ &= f \circ f_k \circ ... \circ f_1, \end{aligned}$$

hence $f \circ f_K \circ ... \circ f_1$ is invariant. □

Lemma 2. *The group equivariant convolution $\Psi : \mathcal{I}_U \to \mathcal{I}_U$ defined as: $[\Psi f](g) \triangleq \int_G \psi(g'^{-1}g) f(g') dg'$ is equivariant with resepct to the regular representation π of G acting on $\mathcal{I}_U$ as $[\pi(u)f](g) \triangleq f(u^{-1}g)$.*

Proof.

$$\begin{aligned}
\Psi[\pi(u)f](g) &= \int_G \psi(g'^{-1}g)[\pi(u)f](g')dg' \\
&= \int_{uG} \psi(g'^{-1}g) f(u^{-1}g')dg' \\
&= \int_G \psi(g'^{-1}u^{-1}g) f(g')dg' \\
&= [\Psi f](u^{-1}g \\
&= [\pi(u)[\Psi f]](g).
\end{aligned}$$

The second equality holds by invariance of the left Haar measure. □

Proposition 1. *The lifting layer $\mathcal{L}$ is equivariant with respect to the representation π.*

Proof. Note $\mathcal{L}[\pi(u)f_{\mathcal{X}}](g) = \mathtt{f}_i$ for $g \in s(ux_i)H$ and $[\pi(u)\mathcal{L}[f_{\mathcal{X}}]](g) = \mathcal{L}[f_{\mathcal{X}}](u^{-1}g) = \mathtt{f}_i$ for $g \in us(x_i)H$. Hence $\mathcal{L}[\pi(u)f_{\mathcal{X}}] = \pi(u)\mathcal{L}[f_{\mathcal{X}}]$ because the two cosets are equal: $s(ux_i)H = us(x_i)H \ \forall u \in G$. Note that this holds because:

- If $g \in s(x_i)H = \{g \in G | gx_0 = x_i\}$, then g maps x_0 to x_i, hence ug maps x_0 to ux_i. So if $g \in s(x_i)H$ then $ug \in s(ux_i)H = \{g \in G | gx_0 = ux_i\}$, the set of all g that map x_0 to ux_i. In summary, $us(x_i)H \subset s(ux_i)H$
- Conversely, if $g \in s(ux_i)H$, then we know that $u^{-1}g$ maps x_0 to x_i, so $u^{-1}g \in s(x_i)H$, hence $g \in us(x_i)H$. In summary, $s(ux_i)H \subset us(x_i)H$
- We have shown $us(x_i)H \subset s(ux_i)H$ and $s(ux_i)H \subset us(x_i)H$, thus $s(ux_i)H = us(x_i)H$.

□

Proposition 2. *LieSelfAttention is equivariant with respect to the regular representation π.*

Proof. Let $\mathcal{I}_U = \mathcal{L}(G, \mathbb{R}^D)$ be the space of unconstrained functions $f : G \to \mathbb{R}^D$. We can define the regular representation π of G acting on $\mathcal{I}_U$ as follows:

$$[\pi(u)f](g) = f(u^{-1}g) \tag{7}$$

f is defined on the set $G_f = \cup_{i=1}^n s(x_i)H$ (i.e. union of cosets corresponding to each x_i). Note $G_{\pi(u)f} = uG_f$, and G_f does not depend on the choice of section s.

Note that for all provided choices of k_c and k_l, we have:

$$k_c([\pi(u)f](g), [\pi(u)f](g')) = k_c(f(u^{-1}g), f(u^{-1}g')) \tag{8}$$

$$k_l(g^{-1}g') = k_l((u^{-1}g)^{-1}(u^{-1}g')) \tag{9}$$

Hence for all choices of F, we have that

$$\begin{aligned}
\alpha_{\pi(u)f}(g, g') &= F(k_c([\pi(u)f](g), [\pi(u)f](g')), k_l(g^{-1}g')) \\
&= F(k_c(f(u^{-1}g), f(u^{-1}g')), k_l((u^{-1}g)^{-1}u^{-1}g')) \\
&= \alpha_f(u^{-1}g, u^{-1}g')
\end{aligned} \tag{10}$$

We thus prove equivariance for the below choice of `LieSelfAttention` $\Phi : \mathcal{I}_U \to \mathcal{I}_U$ that uses softmax normalisation, but a similar proof holds for constant normalisation. Let $A_f(g, g') \triangleq \exp(\alpha_f(g, g'))$, hence Equation (10) also holds for

A_f.

$$[\Phi f](g) = \int_{G_f} w_f(g, g') f(g') dg' \tag{11}$$

$$= \int_{G_f} \frac{A_f(g, g')}{\int_{G_f} A_f(g, g'') dg''} f(g') dg' \tag{12}$$

Hence:

$$\begin{aligned} w_{\pi(u)f}(g, g') &= \frac{A_{\pi(u)f}(g, g')}{\int_{G_{\pi(u)f}} A_{\pi(u)f}(g, g'') dg''} \\ &= \frac{A_f(u^{-1}g, u^{-1}g')}{\int_{uG_f} A_f(u^{-1}g, u^{-1}g'') dg''} \\ &= \frac{A_f(u^{-1}g, u^{-1}g')}{\int_{G_f} A_f(u^{-1}g, g'') dg''} \\ &= w_f(u^{-1}g, u^{-1}g') \end{aligned} \tag{13}$$

Then we can show that Φ is equivariant with respect to the representation π as follows:

$$\begin{aligned} \Phi[\pi(u)f](g) &= \int_{G_{\pi(u)f}} w_{\pi(u)f}(g, g')[\pi(u)f](g') dg' \\ &= \int_{uG_f} w_f(u^{-1}g, u^{-1}g') f(u^{-1}g') dg' \\ &= \int_{G_f} w_f(u^{-1}g, g') f(g') dg' \\ &= [\Phi f](u^{-1}g) \\ &= [\pi(u)[\Phi f]](g) \end{aligned} \tag{14}$$

□

Equivariance holds for any α_f that satisfies Equation (10). Multiplying α_f by an indicator function $\mathbb{1}\{d(g, g') < \lambda\}$ where $d(g, g')$ is some function of $g^{-1}g'$, we can show that *local* self-attention that restricts attention to points in a neighbourhood also satisfies equivariance. When approximating the integral with Monte Carlo samples (equivalent to replacing G_f with $\hat{G}_f$) we obtain a self-attention layer that is equivariant in expectation for constant normalisation of attention weights (i.e. $\mathbb{E}[\hat{\Phi}[\pi(u)f](g)] = \Phi[\pi(u)f](g) = [\pi(u)[\Phi f]](g)$ where $\hat{\Phi}$ is the same as Φ but with $\hat{G}_f$ instead of G_f). However for softmax normalisation we obtain a biased estimate due to the nested MC estimate in the denominator's normalising constant.

D. Introduction to Self-Attention

Self-attention (Vaswani et al., 2017) is a mapping from an input set of N vectors $\{x_1, \ldots, x_N\}$, where $x_i \in \mathbb{R}^D$, to an output set of N vectors in $\mathbb{R}^D$. Let us represent the inputs as a matrix $X \in \mathbb{R}^{N \times D}$ such that the ith row $X_{i:}$ is x_i. **Multihead self-attention** (MSA) consists of M **heads** where M is chosen to divide D. The output of each head is a set of N vectors of dimension D/M, where each vector is obtained by taking a weighted average of the input vectors $\{x_1, \ldots, x_N\}$ with weights given by a weight matrix W, followed by a linear map $W^V \in \mathbb{R}^{D \times D/M}$. Using m to index the head ($m = 1, \ldots, M$), the output of the mth head can be written as:

$$f^m(X) \triangleq WXW^{V,m} \in \mathbb{R}^{N \times D/M}$$
$$\text{where} \quad W \triangleq \texttt{softmax}(XW^{Q,m}(XW^{K,m})^\top) \in \mathbb{R}^{N \times N}$$

where $W^{Q,m}, W^{K,m}, W^{V,m} \in \mathbb{R}^{D \times D/M}$ are learnable parameters, and the softmax normalisation is performed on each row of the matrix $XW^{Q,m}(XW^{K,m})^\top \in \mathbb{R}^{N \times N}$. Finally, the outputs of all heads are concatenated into a $N \times D$ matrix

and then right multiplied by $W^O \in \mathbb{R}^{D\times D}$. Hence MSA is defined by:

$$MSA(X) \triangleq [f^1(X), \dots, f^M(X)]W^O \in \mathbb{R}^{N\times D}. \tag{15}$$

Note $XW^Q(XW^K)^\top$ is the Gram matrix for the dot-product kernel, and softmax normalisation is a particular choice of normalisation. Hence MSA can be generalised to other choices of kernels and normalisation that are equally valid (Wang et al., 2018; Tsai et al., 2019).

E. LieSelfAttention: Details

We explore the following non-exhaustive list of choices for content-based attention, location-based attention, combining content and location attention and normalisation of weights:

Content-based attention $k_c(f(g), f(g'))$:

1. Dot-product: $\frac{1}{\sqrt{d_v}}\left(W^Q f(g)\right)^\top W^K f(g') \in \mathbb{R}$ for $W^Q, W^K \in \mathbb{R}^{d_v\times d_v}$
2. Concat: $\texttt{Concat}[W^Q f(g), W^K f(g')] \in \mathbb{R}^{2d_v}$
3. Linear-Concat-linear: $W\texttt{Concat}[W^Q f(g), W^K f(g')] \in \mathbb{R}^{d_s}$ for $W \in \mathbb{R}^{d_s\times 2d_v}$.

Location-based attention $k_l(g^{-1}g')$ for Lie groups G:

1. Plain: $\nu[\log(g^{-1}g')]$
2. MLP: $\texttt{MLP}(\nu[\log(g^{-1}g')])$

where $\log : G \to \mathfrak{g}$ is the log map from G to its Lie algebra $\mathfrak{g}$, and $\nu : \mathfrak{g} \to \mathbb{R}^d$ is the isomorphism that extracts the free parameters from the output of the log map (Finzi et al., 2020). We can use the same log map for discrete subgroups of Lie groups (e.g. $C_n \leq SO(2), D_n \leq O(2)$). See Appendix F for an introduction to the Lie algebra and the exact form of $\nu \circ \log(g)$ for common Lie groups.

Combining content and location attention $\alpha_f(g, g')$:

1. Additive: $k_c(f(g), f(g')) + k_l(g^{-1}g')$
2. MLP: $\texttt{MLP}[\texttt{Concat}[k_c(f(g), f(g')), k_l(g^{-1}g')]]$
3. Multiplicative: $k_c(f(g), f(g')) \cdot k_l(g^{-1}g')$

Note that the MLP case is a strict generalisation of the additive combination, and for this option k_c and k_l need not be scalars.

Normalisation of weights $\{w_f(g, g')\}_{g'\in G_f}$:

1. Softmax: $\text{softmax}\left(\{\alpha_f(g, g')\}_{g'\in G_f}\right)$
2. Constant: $\{\frac{1}{|G_f|}\alpha_f(g, g')\}_{g'\in G_f}$

We also outline how to extend the single-head `LieSelfAttention` described in Algorithm 1 extends to **Multihead equivariant self-attention**. Let M be the number of heads, assuming it divides d_v, with m indexing the head. Then the output of each head is:

$$V^m(g) = \int_{G_f} w_f(g, g')W^{V,m} f(g')dg' \in \mathbb{R}^{d_v/M} \tag{16}$$

The only difference is that $W^{Q,m}, W^{K,m}, W^{V,m} \in \mathbb{R}^{d_v/M \times d_v}$. The multihead self-attention combines the heads using $W^O \in \mathbb{R}^{d_v \times d_v}$, to output:

$$f_{out}(g) = W^O \begin{bmatrix} V^1(g) \\ \vdots \\ V^M(g) \end{bmatrix} \tag{17}$$

F. Lie Algebras and Log maps

In this section we briefly introduce Lie algebras and log maps, mainly summarising relevant sections of Hall (2015). See the reference for a formal and thorough treatment of Lie groups and Lie algebras.

Given a Lie group G, a smooth manifold, its **Lie algebra** $\mathfrak{g}$ is a vector space defined to be the tangent space at the identity element $e \in G$ (together with a bilinear operation called the Lie bracket $[x, y]$, whose details we omit as it is not necessary for understanding log maps). We most commonly deal with **matrix Lie groups**, namely subgroups of the general linear group $GL(n; \mathbb{C})$, the group of all $n \times n$ invertible matrices with entries in $\mathbb{C}$. This includes rotation/reflection groups $SO(n)$ and $O(n)$, as well as the group of translations $T(n)$ and roto-translations $SE(n)$, that are isomorphic to subgroups of $GL(n+1; \mathbb{C})$. For example, $SE(n)$ is isomorphic to the group of matrices of the form:

$$\begin{bmatrix} & & & | \\ & R & & x \\ & & & | \\ — & 0 & — & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}$$

where $R \in SO(n)$ and $x \in \mathbb{R}^n$. For such matrix Lie Groups G, the Lie algebra is precisely the set of all matrices X such that $\exp(tX) \in G$ for all $t \in \mathbb{R}$, where $\exp$ is the matrix exponential ($\exp(A) = I + A + A^2/2! + \ldots$). Hence the Lie algebra $\mathfrak{g}$ can be thought of as a set of matrices, and the matrix exponential $\exp$ can be thought of as a map from the Lie algebra $\mathfrak{g}$ to G. This map turns out to be surjective for all the groups mentioned below, and hence we may define the log map $\log : G \to \mathfrak{g}$ in the other direction. Since the effective dimension of Lie algebra, say d, is smaller than the number of entries of the $n \times n$ (or $(n+1) \times (n+1)$ in the case of $SE(n)$ and $T(n)$) matrix element of the Lie algebra, we use a map $\nu : \mathfrak{g} \to \mathbb{R}^d$ that extracts the free parameters from the Lie algebra element, to obtain a form that is suitable as an input to a neural network. See below for concrete examples.

- $G = T(n), t \in \mathbb{R}^n, \nu[\log(t)] = t$
- $G = SO(2), R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \in \mathbb{R}^{2\times 2}$

$$\nu[\log(R)] = \theta = \arctan(R_{10}/R_{01}) \tag{18}$$

- $G = SE(2), R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \in \mathbb{R}^{2\times 2}, t \in \mathbb{R}^2$

$$\nu[\log(tR)] = \begin{bmatrix} t' \\ \theta \end{bmatrix} \tag{19}$$

  where $t' = V^{-1}t, V = \begin{bmatrix} a & -b \\ b & a \end{bmatrix}, a \triangleq \frac{\sin\theta}{\theta}, b \triangleq \frac{1-\cos\theta}{\theta}$

- $G = SO(3), R \in \mathbb{R}^{3\times 3}, t \in \mathbb{R}^3$:

$$\nu[\log(R)] = \nu\left[\frac{\theta}{2\sin\theta}(R - R^\top)\right] = \frac{\theta}{2\sin\theta}\begin{bmatrix} R_{21} - R_{12} \\ R_{02} - R_{20} \\ R_{10} - R_{01} \end{bmatrix} \tag{20}$$

where $\cos\theta = \frac{\mathrm{Tr}(R)-1}{2}$. Note that the Taylor expansion of $\theta/\sin\theta$ should be used when θ is small.

- $G = SE(3), R \in \mathbb{R}^{3\times3}, t \in \mathbb{R}^3$:

$$\nu[\log(tR)] = \begin{bmatrix} t' \\ r' \end{bmatrix} \tag{21}$$

where $t' = V^{-1}t, r' = \nu[\log(R)], V = I + \frac{1-\cos\theta}{\theta^2}(R - R^\top) + \frac{\theta-\sin\theta}{\theta^3}(R - R^\top)^2$.

Canonical lifting without Log map. Recall that location-based attention only requires a function $k_l(g^{-1}g')$ which we are free to parameterise in any way. Since various groups G can naturally be expressed in terms of real matrices (see above), $g^{-1}g' \in G$ can be expressed as a (flattened) real vector. For example, any $g \in SO(2)$ can simply be expressed as a vector $[t, \theta]^\intercal$ where $t \in \mathbb{R}^2$ and $\theta \in [0, 2\pi)$. Therefore, we can bypass the log map $\nu \circ \log$ and directly use this vector, which we found to be more numerically stable and sometimes resulted in better performance of `LieTransformer`. In particular, for `LieConv-SE2` and `LieTransformer-SE2` on the Hamiltonian spring dynamics task, we did not use the log map and instead opted for this "canonical" lift. We plan to also try this for `LieConv-SE3` and `LieTransformer-SE3` for the QM9 task.

G. Memory and Time Complexity Comparison with LieConv

G.1. LieConv

- Inputs: $\{g, f(g)\}_{g\in G_f}$ where
  - $f(g) \in \mathbb{R}^{d_v}$
  - G_f defined as in Section 3.1.
- Outputs: $\{g, \frac{1}{|\mathtt{nbhd}(g)|}\sum_{g'\in\mathtt{nbhd}(g)} k_L(g^{-1}g')f(g')\}_{g\in G_f}$ where
  - $\mathtt{nbhd}(g) = \{g' \in G_f : \nu[\log(g)] < r\}$. Let us assume that $|\mathtt{nbhd}(g)| \approx n\ \forall g$.
  - $k_L(g^{-1}g') = \mathtt{MLP}_\theta(\nu[\log(g^{-1}g')]) \in \mathbb{R}^{d_{out}\times d_v}$.

There are (at least) two ways of computing `LieConv`: 1. Naive and 2. PointConv Trick.

1. **Naive**
   - **Memory**: Store $k_L(g^{-1}g') \in \mathbb{R}^{d_{out}\times d_v}\ \forall g \in G_f, g' \in \mathtt{nbhd}(g)$. This requires $O(|G_f|nd_{out}d_v)$ memory.
   - **Time**: Compute $k_L(g^{-1}g')f(g')\forall g \in G_f, g' \in \mathtt{nbhd}(g)$. This requires $O(|G_f|nd_{out}d_v)$ flops.
2. **PointConv Trick**

   One-line summary: instead of applying a shared linear map then summing across `nbhd`, first sum across `nbhd` then apply the linear map.

   Details: $k_L(g^{-1}g') = \mathtt{MLP}_\theta(\nu[\log(g^{-1}g')]) = \mathtt{reshape}(HM(g^{-1}g'), [d_{out}, d_v])$ where
   - $M(g^{-1}g') \in \mathbb{R}^{d_{mid}}$ are the final layer activations of $\mathtt{MLP}_\theta$.
   - $H \in \mathbb{R}^{d_{out}d_v\times d_{mid}}$ is the final linear layer of $\mathtt{MLP}_\theta$.

   The trick assumes $d_{mid} \ll d_{out}d_v$, and reorders the computation as:

$$\begin{aligned}&\sum_{g'\in\mathtt{nbhd}(g)} \mathtt{reshape}(HM(g^{-1}g'), [d_{out}, d_v])f(g') \\ &= \mathtt{reshape}(H, [d_{out}, d_v d_{mid}]) \sum_{g'\in\mathtt{nbhd}(g)} M(g^{-1}g') \otimes f(g')\end{aligned}$$

   where $\otimes$ is the Kronecker product: $x\otimes y = [x_1y_1, \dots x_1y_{d_y}, \dots, x_{d_x}y_1, \dots x_{d_x}y_{d_y}] \in \mathbb{R}^{d_xd_y}$. So $M(g^{-1}g')\otimes f(g') \in \mathbb{R}^{d_v d_{mid}}$.

- **Memory**: Store $M(g^{-1}g')\ \forall g \in G_f, g' \in \texttt{nbhd}(g)$, and store H. This requires $O(|G_f|nd_{mid} + d_{out}d_v d_{mid})$ memory.
- **Time**: Compute $\sum_{g' \in \texttt{nbhd}(g)} M(g^{-1}g') \otimes f(g')$ via matrix multiplication: $\begin{bmatrix} | & & | \\ M(g^{-1}g'_1) & \dots & M(g^{-1}g'_n) \\ | & & | \end{bmatrix} \begin{bmatrix} — & f(g'_1) & — \\ & \vdots & \\ — & f(g'_n) & — \end{bmatrix}$. This requires $O(d_v n d_{mid})$ flops.
  Then multiply by H, requiring $O(d_v d_{out} d_{mid})$ flops.
  This is done for each $g \in G_f$, so the total number of flops is $O(|G_f|d_v d_{mid}(n + d_{out}))$.

G.2. Equivariant Self-Attention

- Inputs: $\{g, f(g)\}_{g \in G_f}$ where
  - $f(g) \in \mathbb{R}^{d_v}$
  - G_f defined as in Section 3.1.
- Outputs: $\{g, f(g) + \sum_{g' \in \texttt{nbhd}(g)} w_f(g, g') W^V f(g')\}_{g \in G_f}$ where
  - $\texttt{nbhd}(g) = \{g' \in G_f : \nu[\log(g)] < r\}$. Let us assume that $|\texttt{nbhd}(g)| \approx n\ \forall g$.
  - $\{w_f(g, g')\}_{g' \in G_f} = \text{softmax}\left(\{\alpha_f(g, g')\}_{g' \in G_f}\right)$
  - $\alpha_f(g, g') = k_f(f(g), f(g')) + k_x(g^{-1}g')$
  - $k_f(f(g), f(g')) = \left(W^Q f(g)\right)^\top W^K f(g') \in \mathbb{R}$
  - $k_x(g) = \texttt{MLP}_\phi(\nu[\log(g)]) \in \mathbb{R}$
  - $W^Q, W^K, W^V \in \mathbb{R}^{d_v \times d_v}$.
- **Memory**: Store $\alpha_f(g, g')$ and $W^V f(g')\ \forall g \in G_f, g' \in \texttt{nbhd}(g)$. This requires $O(|G_f|nd_v)$ memory.
- **Time**: Compute $k_f(f(g), f(g'))$ and $w_f(g, g')\ \forall g \in G_f, g' \in \texttt{nbhd}(g)$. This requires $O(|G_f|nd_v^2)$ flops.

With multihead self-attention (M heads), the output is:

$$f(g) + W^O \begin{bmatrix} V^1 \\ \vdots \\ V^M \end{bmatrix}$$

where $W^O \in \mathbb{R}^{d_v \times d_v}$, $V^m = \sum_{g' \in \texttt{nbhd}(g)} w_f(g, g') W^{V,m} f(g')$ for $W^{K,m}, W^{Q,m}, W^{V,m} \in \mathbb{R}^{d_v/M \times d_v}$.

- **Memory**: Store $\alpha_f^m(g, g')$ and $W^{V,m} f(g')\ \forall g \in G_f, g' \in \texttt{nbhd}(g), m \in \{1, \dots, M\}$. This requires $O(M|G_f|n + M|G_f|nd_v/M) = O(|G_f|n(M + d_v))$ memory.
- **Time**: Compute $k_f^m(f(g), f(g'))$ and $w_f^m(g, g')\ \forall g \in G_f, g' \in \texttt{nbhd}(g), m \in \{1, \dots, M\}$. This requires $O(M|G_f|nd_v d_v/M) = O(|G_f|nd_v^2)$ flops.

H. Other Equivariant/Invariant building blocks

G-Pooling is simply averaging over the features across the group: Inputs: $\{f(g)\}_{g \in G_f}$ Output: $\bar{f}(g) \triangleq \frac{1}{|G_f|} \sum_{g \in G_f} f(g)$ Note that G-pooling is invariant with respect to the regular representation.

Pointwise MLPs are MLPs applied independently to each $f(g)$ for $g \in G_f$. It is easy to show that any such pointwise operations are equivariant with respect to the regular representation.

LayerNorm (Ba et al., 2016) is defined as follows:

Inputs: $\{g, f(g)\}_{g \in G_f}$ where

- $f(g) \in \mathbb{R}^{d_v}$
- G_f defined as in Section 3.1.

Outputs: $\{g, \beta \odot \frac{f(g)-m(g)}{\sqrt{v(g)+\epsilon}} + \gamma\}_{g \in G_f}$ where

- Division in fraction above is *scalar* division i.e. $\sqrt{v(g)+\epsilon} \in \mathbb{R}$.
- $m(g) = \text{Mean}_c f_c(g') \in \mathbb{R}$.
- $v(g) = \text{Var}_c f_c(g') \in \mathbb{R}$.
- $\beta, \gamma \in \mathbb{R}^D$ are learnable parameters.

BatchNorm We also describe BatchNorm (Ioffe & Szegedy, 2015) that is used in (Finzi et al., 2020) for completeness:

Inputs: $\{g, f^b(g)\}_{g \in G_f, b \in \mathcal{B}}$ where

- $f(g) \in \mathbb{R}^{d_v}$
- G_f defined as in Section 3.1, $\mathcal{B}$ is the batch of examples.

Outputs: $\{g, \beta \odot \frac{f^b(g)-\mathbf{m}(g)}{\sqrt{\mathbf{v}(g)+\epsilon}} + \gamma\}_{g \in G_f, b \in \mathcal{B}}$ where

- Division in fraction above denotes *pointwise* division i.e. $\sqrt{\mathbf{v}(g)+\epsilon} \in \mathbb{R}^D$.
- $\mathbf{m}(g) = \text{Mean}_{g' \in \texttt{nbhd}(g), b \in \mathcal{B}} f^b(g') \in \mathbb{R}^D$ - Mean is taken for every channel.
- $\mathbf{v}(g) = \text{Var}_{g' \in \texttt{nbhd}(g), b \in \mathcal{B}} f^b(g') \in \mathbb{R}^D$ - Var is taken for every channel.
- $\texttt{nbhd}(g) = \{g' \in G_f : \nu[\log(g)] < r\}$.
- $\beta, \gamma \in \mathbb{R}^D$ are learnable parameters.

A moving average of $\mathbf{m}(g)$ and $\mathbf{v}(g)$ are tracked during training time for use at test time. It is easy to check that both BatchNorm and LayerNorm are equivariant wrt the action of the regular representation π on f (for BatchNorm, note $g' \in \texttt{nbhd}(g)$ iff $u^{-1}g' \in \texttt{nbhd}(u^{-1}g)$).

I. Experimental details

I.1. Counting shapes in 2D point clouds

Each training / test example consists of up to two instances of each of the following shapes: triangles, squares, pentagons and the "L" shape. The x_i are the 2D coordinates of each point and $\mathtt{f}_i = 1$ for all points.

We performed an architecture search on the `LieTransformer` first and then set the architecture of the `SetTransformer` such that the models have a similar number of parameters (1075k for the `SetTransformer` and 1048k for both `LieTransformer-T2` and `LieTransformer-SE2`) and depth.

Model architecture. The architecture used for the `SetTransformer` (Lee et al., 2019) consists of 8 layers in the encoder, 8 layers in the decoder and 4 attention heads. No inducing points were used.

The architecture used for both `LieTransformer-T2` and `LieTransformer-SE2` is made of 10 layers, 8 heads and feature dimension $d_v = 128$. The dimension of the kernel used is 12. One lift sample was used for each point.

Training procedure. We use Adam (Kingma & Ba, 2015) with parameters $\beta_1 = 0.5$ and $\beta_2 = 0.9$ and a learning rate of $1e-4$. Models are trained with mini-batches of size 32 until convergence.

I.2. QM9

For the QM9 experiment setup we follow the approach of Anderson et al. (2019) for parameterising the inputs and for the train/validation/test split. The $\mathtt{f}_i$ is a learnable linear embeddings of the vector $[1, c_i, c_i^2]$ for charge c_i, with different linear maps for each atom type. We split the available data as follows: 100k samples for training, 10% for a test set and the rest used for validation.

In applying our model to this task, we ignore the bonding structure of the molecule. As noted in (Klicpera et al., 2019) this should not be needed to learn the task, although it may be helpful as auxiliary information. Given most methods compared against do not use such information, we follows this for a fair comparison (an exception is the $SE(3)$-Transformer (Fuchs et al., 2020) that uses the bonding information). It would be possible to utilise the bonding structure both in the neighbourhood selection step and as model features by treating only atoms that are connected via a bond to another atom as in the neighbourhood of that atom.

We performed architecture and hyperparameter optimisation on the ϵ_{HOMO} task and then trained with the resulting hyperparameters on the other 11 tasks. `LieTransformer-T3` uses 13 layers of attention blocks (performance saturated at 13 layers), using 8 heads (M) in each layer and feature dimension $d_v = 848$. The attention kernel uses the $linear - concat - linear$ feature embedding, identity embedding of the Lie algebra elements, and an MLP to combine these embeddings into the final attention coefficients. The final part of the model used had minor differences to the one in diagram 1. Instead of a global pooling layer followed by a 3 layer MLP, a single linear layer followed by global pooling was used. A single lift sample was used (since $H = \{e\}$ for $T(3)$-invariant models), with the radius of the neighbourhood chosen such that $|\mathtt{nbhd}_\eta(g)| = 50 \; \forall g \in G$ and we uniformly sample 25 points from this neighbourhood. `LieTransformer-SE3` used a similar hyperparameter setting, except using 30 layers (performance saturated at 30 layers) and 2 lift samples were used for each input point ($|\hat{H}| = 2$), with the radius of the neighbourhood η chosen such that the $|\mathtt{nbhd}_\eta(g)| = 25 \; \forall g \in G$ and we uniformly sample 20 points from this neighbourhood. All models were trained using Adam, with a learning rate of $3e-4$ and a batch size of 75 for 500 epochs. For the `LieConv` models we used the hyperparameter setting that was used in Finzi et al. (2020).

Training these models with $T(3)$ and $SE(3)$ equivariance took approximately 3 and 6 days respectively on a single Nvidia Tesla-V100.

I.3. Hamiltonian dynamics

Spring dynamics simulation. We exactly follow the setup described in Appendix C.4 of Finzi et al. (2020) for generating the trajectories used in the train and test data.

Model architecture. In all results shown except Figures 7 and 8, we used a `LieTransformer-T2` with 5 layers, 8 heads and feature dimension $d_v = 160$. The attention kernel uses dot-product for the content component, a 3-layer MLP with hidden layer width 16 for the location component and addition to combine the content and location attention components. (Also, see end of Appendix F for a relevant discussion about the use of the log map in location-attention k_l for this task.) We use constant normalisation of the weights. We observed a significant drop in performance when, instead of constant normalisation, we used softmax normalisation (which caused small gradients at initialization leading to optimization difficulties). The architecture had 842k parameters. Our small models (with 139k parameters) in Figures 7 and 8 use 3 layers and feature dimension $d_v = 80$, keeping all else fixed. `LieTransformer-SE(2)` and `LieConv-SE(2)` in Figures 7 and 8 used 2 lift samples, which were deterministically chosen to be $\hat{H} = C_2 < SO(2)$, where C_2 is the group of $180°$ rotations. In fact, this yields *exact* equivariance to $T(2) \rtimes C_2$. Note that the true Hamiltonian $H(\mathbf{q}, \mathbf{p})$ for the spring system separates as $H(\mathbf{q}, \mathbf{p}) = K(\mathbf{p}) + V(\mathbf{q})$ where K and V are the kinetic and potential energies of the system respectively. Following Finzi et al. (2020), our model parameterises the potential term V. In particular, x_i is particle i's position and $\mathtt{f}_i = (m_i, k_i)$ where m_i is its mass and k_i is used to define the spring constants: $k_i k_j$ is spring constant for the spring connecting particles i and j (see Appendix C.4 of Finzi et al. (2020) for details).

Training details. To train the `LieTransformer`, we used Adam with a learning rate of 0.001 with cosine annealing and a batch size of 100. For a training dataset of size n, we trained the model for $400\sqrt{3000/n}$ epochs (although we found model training usually converged with fewer epochs). When $n \leq 100$, we used the full dataset in each batch. For training the `LieConv` baseline, we used their default architecture (with 895k parameters) and hyperparameter settings for this task,

except for the number of epochs which was $400\sqrt{3000/n}$ to match the setting used for training `LieTransformer`.[1] The small `LieConv` models (with 173k parameters) in Figures 7 and 8 use 3 layers and 192 channels (instead of the default 4 layers and 384 channels). Lastly, only for the data efficiency results in Figure 4, we used early stopping by validation loss and generated *nested* training datasets as the training size varies, keeping the test dataset fixed.

Loss computation. One small difference between our setup and that of Finzi et al. (2020) is in the way we compute the test loss. Since we compare models' losses not only over 5-step roll-outs but also longer 100-step roll-outs, we average the individual time step losses using a geometric mean rather than an arithmetic mean as in Finzi et al. (2020). Since the losses for later time steps are typically orders of magnitude higher than for earlier time steps (see e.g. Figure 5), a geometric mean prevents the losses for later time steps from dominating over the losses for the earlier time steps. During training, we use an arithmetic mean across time steps to compute the loss for optimization, exactly as in Finzi et al. (2020). This applies for both `LieTransformer` and `LieConv`.

J. Additional experimental results

J.1. Hamiltonian dynamics

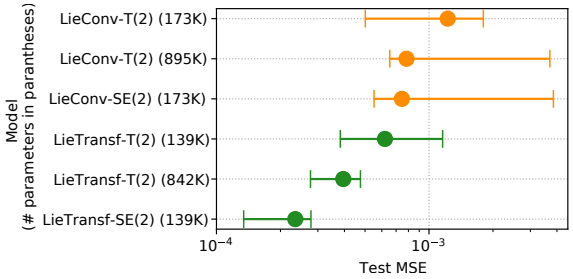


Figure 8. Comparison of models by group and number of parameters over 100-step trajectory roll-outs. Similar to the results of Figure 7, LieTransformer models outperform their LieConv counterparts when fixing the group and using approximately equal number of parameters. Moreover, models (approximately) equivariant to SE(2) outperform their T(2) counterparts, with LieTransformer-SE(2) again outperforming all other models despite having the smallest number of parameters. Plot shows the median across at least 5 random seeds with interquartile range.

[1] This yields better results for `LieConv` compared to those reported by Finzi et al. (2020), where they use fewer total epochs.

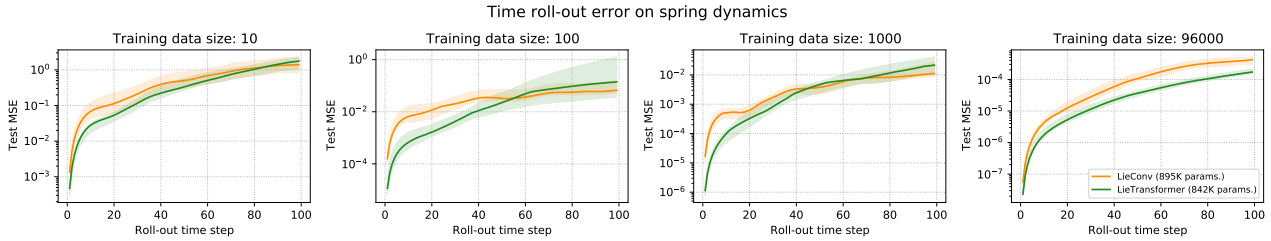


Figure 9. Plots of model error as a function of time step for various data sizes. As can be seen, the LieTransformer generally outperforms LieConv across various training data sizes.

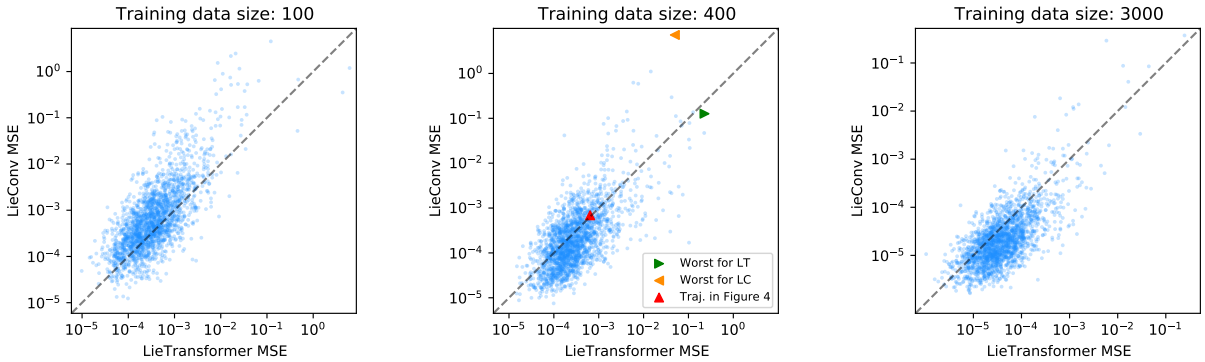
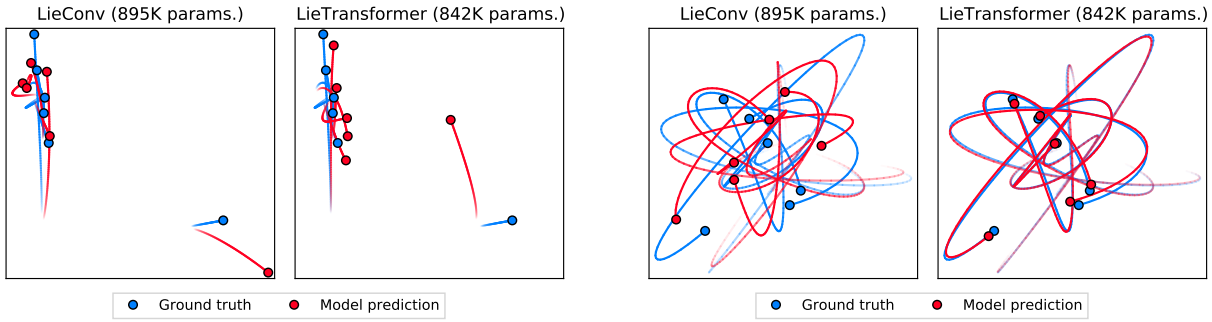


Figure 10. Scatter plots comparing the MSE of the LieTransformer against the MSE of LieConv for various training dataset sizes. Each point in a scatter plot corresponds to a 100-step test trajectory, indicating the losses achieved by both models on that trajectory. In the middle figure we have highlighted the MSEs corresponding to the trajectories shown in Figures 6 and 11.



(a) Test trajectory where LieTransformer has the highest error.

(b) Test trajectory where LieConv has the highest error.

Figure 11. Additional example trajectories comparing LieTransformer and LieConv. Both models are trained on a dataset of size 400. See Figure 10 for a scatter plot showing these test trajectories and the one in Figure 6 in relation to all other trajectories in the test dataset.