Table of Contents

Typo There is a typo in Figure 5: g_1 is translation applied in t (not x), while g_2 is translation applied in x (not y). Whenever applicable, we use the same strength in both x and y axis.

A PDE Symmetry Groups and Deriving Generators

Symmetry augmentations encourage invariance of the representations to known symmetry groups of the data. The guiding principle is that inputs that can be obtained from one another via transformations of the symmetry group should share a common representation. In images, such symmetries are known *a priori* and correspond to flips, resizing, or rotations of the input. In PDEs, these symmetry groups can be derived as Lie groups, commonly denoted as Lie point symmetries, and have been categorized for many common PDEs [11]. An example of the form of such augmentations is given in Figure 6 for a simple PDE that rotates a point in 2-D space. In this example, the PDE exhibits both rotational symmetry and scaling symmetry of the radius of rotation. For arbitrary PDEs, such symmetries can be derived, as explained in more detail below.

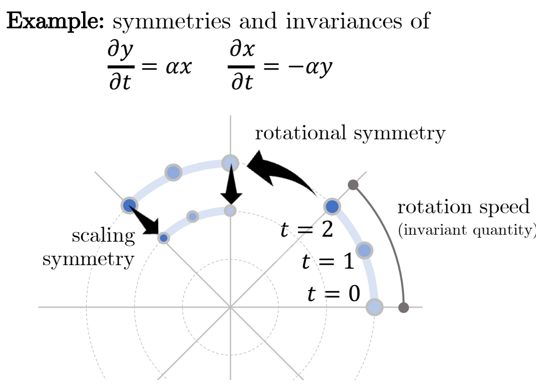

Figure 6: Illustration of the PDE symmetry group and invariances of a simple PDE, which rotates a point in 2-D space. The PDE symmetry group here corresponds to scalings of the radius of the rotation and fixed rotations of all the points over time. A sample invariant quantity is the rate of rotation (related to the parameter α in the PDE), which is fixed for any solution to this PDE.

The Lie point symmetry groups of differential equations form a Lie group structure, where elements of the groups are smooth and differentiable transformations. It is typically easier to derive the symmetries of a system of differential equations via the infinitesimal generators of the symmetries, (*i.e.,* at the level of the derivatives of the one parameter transforms). By using these infinitesimal generators, one can replace *nonlinear* conditions for the invariance of a function under the group transformation, with an equivalent *linear* condition of *infinitesimal* invariance under the respective generator of the group action [11].

In what follows, we give an informal overview to the derivation of Lie point symmetries. Full details and formal rigor can be obtained in Olver [11], Ibragimov [13], among others.

In the setting we consider, a differential equation has a set of p independent variables $\boldsymbol{x} = (x^1, x^2, \ldots, x^p) \in \mathbb{R}^p$ and q dependent variables $\boldsymbol{u} = (u^1, u^2, \ldots, u^q) \in \mathbb{R}^q$. The solutions take the form $\boldsymbol{u} = f(\boldsymbol{x})$, where $u^\alpha = f^\alpha(\boldsymbol{x})$ for $\alpha \in \{1, \ldots, q\}$. Solutions form a graph over a domain $\Omega \subset \mathbb{R}^p$:

$$\Gamma_f = \{(\boldsymbol{x}, f(\boldsymbol{x})) : \boldsymbol{x} \in \Omega\} \subset \mathbb{R}^p \times \mathbb{R}^q. \tag{10}$$

In other words, a given solution Γ_f forms a p-dimensional submanifold of the space $\mathbb{R}^p \times \mathbb{R}^q$.

The n-th **prolongation** of a given smooth function Γ_f expands or "prolongs" the graph of the solution into a larger space to include derivatives up to the n-th order. More precisely, if $\mathcal{U} = \mathbb{R}^q$ is the solution space of a given function and $f : \mathbb{R}^p \to \mathcal{U}$, then we introduce the Cartesian product space of the prolongation:

$$\mathcal{U}^{(n)} = \mathcal{U} \times \mathcal{U}_1 \times \mathcal{U}_2 \times \cdots \times \mathcal{U}_n, \tag{11}$$

where $\mathcal{U}_k = \mathbb{R}^{\dim(k)}$ and $\dim(k) = \binom{p+k-1}{k}$ is the dimension of the so-called *jet space* consisting of all k-th order derivatives. Given any solution $f : \mathbb{R}^p \to \mathcal{U}$, the prolongation can be calculated by simply calculating the corresponding derivatives up to order n (e.g., via a Taylor expansion at each point). For a given function $\boldsymbol{u} = f(\boldsymbol{x})$, the n-th prolongation is denoted as $\boldsymbol{u}^{(n)} = \mathrm{pr}^{(n)} f(\boldsymbol{x})$. As a simple example, for the case of $p = 2$ with independent variables x and y and $q = 1$ with a single dependent variable f, the second prolongation is

$$\begin{aligned} \boldsymbol{u}^{(2)} = \mathrm{pr}^{(2)} f(x,y) &= (u; u_x, u_y; u_{xx}, u_{xy}, u_{yy}) \\ &= \left(f; \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}; \frac{\partial^2 f}{\partial x^2}, \frac{\partial^2 f}{\partial x \partial y}, \frac{\partial^2 f}{\partial y^2}\right) \in \mathbb{R}^1 \times \mathbb{R}^2 \times \mathbb{R}^3, \end{aligned} \tag{12}$$

which is evaluated at a given point (x, y) in the domain. The complete space $\mathbb{R}^p \times \mathcal{U}^{(n)}$ is often called the n-th order jet space [11].

A system of differential equations is a set of l differential equations $\Delta : \mathbb{R}^p \times \mathcal{U}^{(n)} \to \mathbb{R}^l$ of the independent and dependent variables with dependence on the derivatives up to a maximum order of n:

$$\Delta_\nu(\boldsymbol{x}, \boldsymbol{u}^{(n)}) = 0, \quad \nu = 1, \dots, l. \tag{13}$$

A smooth solution is thus a function f such that for all points in the domain of $\boldsymbol{x}$:

$$\Delta_\nu(\boldsymbol{x}, \mathrm{pr}^{(n)} f(\boldsymbol{x})) = 0, \quad \nu = 1, \dots, l. \tag{14}$$

In geometric terms, the system of differential equations states where the given map Δ vanishes on the jet space, and forms a subvariety

$$Z_\Delta = \{(\boldsymbol{x}, \boldsymbol{u}^{(n)}) : \Delta(\boldsymbol{x}, \boldsymbol{u}^{(n)}) = 0\} \subset \mathbb{R}^p \times \mathcal{U}^{(n)}. \tag{15}$$

Therefore to check if a solution is valid, one can check if the prolongation of the solution falls within the subvariety Z_Δ. As an example, consider the one dimensional heat equation

$$\Delta = u_t - c u_{xx} = 0. \tag{16}$$

We can check that $f(x,t) = \sin(x)e^{-ct}$ is a solution by forming its prolongation and checking if it falls withing the subvariety given by the above equation:

$$\begin{aligned} &\mathrm{pr}^{(2)} f(x,t) = \left(\sin(x)e^{-ct}; \cos(x)e^{-ct}, -c\sin(x)e^{-ct}; -\sin(x)e^{-ct}, -c\cos(x)e^{-ct}, c^2\sin(x)e^{-ct}\right), \\ &\Delta(x, t, \boldsymbol{u}^{(n)}) = -c\sin(x)e^{-ct} + c\sin(x)e^{-ct} = 0. \end{aligned} \tag{17}$$

A.1 Symmetry Groups and Infinitesimal Invariance

A symmetry group G for a system of differential equations is a set of local transformations to the function which transform one solution of the system of differential equations to another. The group takes the form of a Lie group, where group operations can be expressed as a composition of one-parameter transforms. More rigorously, given the graph of a solution Γ_f as defined in Eq. (10), a group operation $g \in G$ maps this graph to a new graph

$$g \cdot \Gamma_f = \{(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}}) = g \cdot (\boldsymbol{x}, \boldsymbol{u}) : (\boldsymbol{x}, \boldsymbol{u}) \in \Gamma_f\}, \tag{18}$$

where $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{u}})$ label the new coordinates of the solution in the set $g \cdot \Gamma_f$. For example, if $\boldsymbol{x} = (x, t)$, $u = u(x,t)$, and g acts on $(\boldsymbol{x}, u)$ via

$$(x, t, u) \mapsto (x + \epsilon t, t, u + \epsilon),$$

then $\tilde{u}(\tilde{x}, \tilde{t}) = u(x,t) + \epsilon = u(\tilde{x} - \epsilon\tilde{t}, \tilde{t}) + \epsilon$, where $(\tilde{x}, \tilde{t}) = (x + \epsilon t, t)$.

Note, that the set $g \cdot \Gamma_f$ may not necessarily be a graph of a new $\boldsymbol{x}$-valued function; however, since all transformations are local and smooth, one can ensure transformations are valid in some region near the identity of the group.

As an example, consider the following transformations which are members of the symmetry group of the differential equation $u_{xx} = 0$. $g_1(t)$ translates a single spatial coordinate x by an amount t and g_2 scales the output coordinate u by an amount e^r:

$$\begin{aligned} g_1(t) \cdot (x, u) &= (x + t, u), \\ g_2(r) \cdot (x, u) &= (x, e^r \cdot u). \end{aligned} \tag{19}$$

It is easy to verify that both of these operations are local and smooth around a region of the identity, as sending $r, t \to 0$ recovers the identity operation. Lie theory allows one to equivalently describe the potentially nonlinear group operations above with corresponding infinitesimal generators of the group action, corresponding to the Lie algebra of the group. Infinitesimal generators form a vector field over the total space $\Omega \times \mathcal{U}$, and the group operations correspond to integral flows over that vector field. To map from a single parameter Lie group operation to its corresponding infinitesimal generator, we take the derivative of the single parameter operation at the identity:

$$\boldsymbol{v}_g|_{(x,u)} = \frac{d}{dt} g(t) \cdot (x, u) \bigg|_{t=0}, \tag{20}$$

where $g(0) \cdot (x, u) = (x, u)$.

To map from the infinitesimal generator back to the corresponding group operation, one can apply the exponential map

$$\exp(t\boldsymbol{v}) \cdot (\boldsymbol{x}, \boldsymbol{u}) = g(t) \cdot (\boldsymbol{x}, \boldsymbol{u}), \tag{21}$$

where $\exp : \mathfrak{g} \to G$. Here, $\exp(\cdot)$ maps from the Lie algebra, $\mathfrak{g}$, to the corresponding Lie group, G. This exponential map can be evaluated using various methods, as detailed in Appendix B and Appendix E.

Returning to the example earlier from Equation (19), the corresponding Lie algebra elements are

$$\begin{aligned} \boldsymbol{v}_{g_1} &= \partial_x \leftrightarrow g_1(t) \cdot (x, u) = (x + t, u), \\ \boldsymbol{v}_{g_2} &= u\partial_u \leftrightarrow g_2(r) \cdot (x, u) = (x, e^r \cdot u). \end{aligned} \tag{22}$$

Informally, Lie algebras help simplify notions of invariance as it allows one to check whether functions or differential equations are invariant to a group by needing only to check it at the level of the derivative of that group. In other words, for any vector field corresponding to a Lie algebra element, a given function is invariant to that vector field if the action of the vector field on the given function evaluates to zero everywhere. Thus, given a symmetry group, one can determine a set of invariants using the vector fields corresponding to the infinitesimal generators of the group. To determine whether a differential equation is in such a set of invariants, we extend the definition of a prolongation to act on vector fields as

$$\mathrm{pr}^{(n)}\, \boldsymbol{v}\big|_{(\boldsymbol{x}, \boldsymbol{u}^{(n)})} = \frac{d}{d\epsilon}\bigg|_{\epsilon=0} \mathrm{pr}^{(n)} \left[\exp(\epsilon \boldsymbol{v})\right] (\boldsymbol{x}, \boldsymbol{u}^{(n)}). \tag{23}$$

A given vector field $\boldsymbol{v}$ is therefore an infinitesimal generator of a symmetry group G of a system of differential equations Δ_ν indexed by $\nu \in \{1, \dots, l\}$ if the prolonged vector field of any given solution is still a solution:

$$\mathrm{pr}^{(n)}\, \boldsymbol{v}[\Delta_\nu(\boldsymbol{x}, \boldsymbol{u}^{(n)})] = 0, \quad \nu = 1, \dots, l, \quad \text{whenever } \Delta(\boldsymbol{x}, \boldsymbol{u}^{(n)}) = 0. \tag{24}$$

For sake of convenience and brevity, we leave out many of the formal definitions behind these concepts and refer the reader to [11] for complete details.

A.2 Deriving Generators of the Symmetry Group of a PDE

Since symmetries of differential equations correspond to smooth maps, it is typically easier to derive the particular symmetries of a differential equation via their infinitesimal generators. To derive such generators, we first show how to perform the prolongation of a vector field. As before, assume we have p independent variables $x^1, \dots, x^p$ and l dependent variables $u^1, \dots, u^l$, which are a function of the dependent variables. Note that we use superscripts to denote a particular variable. Derivatives with respect to a given variable are denoted via subscripts corresponding to the indices. For example, the variable u^1_{112} denotes the third order derivative of u^1 taken twice with respect to the variable x^1 and once with respect to x^2. As stated earlier, the prolongation of a vector field is defined as the operation

$$\mathrm{pr}^{(n)}\, \boldsymbol{v}\big|_{(\boldsymbol{x}, \boldsymbol{u}^{(n)})} = \frac{d}{d\epsilon}\bigg|_{\epsilon=0} \mathrm{pr}^{(n)} \left[\exp(\epsilon \boldsymbol{v})\right] (\boldsymbol{x}, \boldsymbol{u}^{(n)}). \tag{25}$$

To calculate the above, we can evaluate the formula on a vector field written in a generalized form. *I.e.*, any vector field corresponding to the infinitesimal generator of a symmetry takes the general form

$$\boldsymbol{v} = \sum_{i=1}^{p} \xi^i(\boldsymbol{x}, \boldsymbol{u}) \frac{\partial}{\partial x^i} + \sum_{\alpha=1}^{q} \phi_\alpha(\boldsymbol{x}, \boldsymbol{u}) \frac{\partial}{\partial u^\alpha}. \tag{26}$$

Throughout, we will use Greek letter indices for dependent variables and standard letter indices for independent variables. Then, we have that

$$\mathrm{pr}^{(n)}\, \boldsymbol{v} = \boldsymbol{v} + \sum_{\alpha=1}^{q} \sum_{\boldsymbol{J}} \phi_\alpha^{\boldsymbol{J}}(\boldsymbol{x}, \boldsymbol{u}^{(n)}) \frac{\partial}{\partial u_{\boldsymbol{J}}^\alpha}, \tag{27}$$

where $\boldsymbol{J}$ is a tuple of dependent variables indicating which variables are in the derivative of $\frac{\partial}{\partial u_{\boldsymbol{J}}^\alpha}$. Each $\phi_\alpha^{\boldsymbol{J}}(\boldsymbol{x}, \boldsymbol{u}^{(n)})$ is calculated as

$$\phi_\alpha^{\boldsymbol{J}}(\boldsymbol{x}, \boldsymbol{u}^{(n)}) = \prod_{i \in \boldsymbol{J}} \boldsymbol{D}_i \left(\phi_a - \sum_{i=1}^{p} \xi^i u_i^\alpha \right) + \sum_{i=1}^{p} \xi^i u_{\boldsymbol{J},i}^\alpha, \tag{28}$$

where $u_{\boldsymbol{J},i}^\alpha = \partial u_{\boldsymbol{J}}^\alpha / \partial x^i$ and $\boldsymbol{D}_i$ is the total derivative operator with respect to variable i defined as

$$\boldsymbol{D}_i P(x, u^{(n)}) = \frac{\partial P}{\partial x^i} + \sum_{i=1}^{q} \sum_{\boldsymbol{J}} u_{\boldsymbol{J},i}^\alpha \frac{\partial P}{\partial u_{\boldsymbol{J}}^\alpha}. \tag{29}$$

After evaluating the coefficients, $\phi_\alpha^{\boldsymbol{J}}(x, u^{(n)})$, we can substitute these values into the definition of the vector field's prolongation in Equation (27). This fully describes the infinitesimal generator of the given PDE, which can be used to evaluate the necessary symmetries of the system of differential equations. An example for Burgers' equation, a canonical PDE, is presented in the following.

A.3 Example: Burgers' Equation

Burgers' equation is a PDE used to describe convection-diffusion phenomena commonly observed in fluid mechanics, traffic flow, and acoustics [41]. The PDE can be written in either its "potential" form or its "viscous" form. The potential form is

$$u_t = u_{xx} + u_x^2. \tag{30}$$

Cautionary note: We derive here the symmetries of Burgers' equation in its potential form since this form is more convenient and simpler to study for the sake of an example. The equation we consider in our experiments is the more commonly studied Burgers' equation in its standard form which does not have the same Lie symmetry group (see Table 3). Similar derivations for Burgers' equation in its standard form can be found in example 6.1 of [42].

Following the notation from the previous section, $p = 2$ and $q = 1$. Consequently, the symmetry group of Burgers' equation will be generated by vector fields of the following form

$$\boldsymbol{v} = \xi(x,t,u) \frac{\partial}{\partial x} + \tau(x,t,u) \frac{\partial}{\partial t} + \phi(x,t,u) \frac{\partial}{\partial u}, \tag{31}$$

where we wish to determine all possible coefficient functions, $\xi(t,x,u)$, $\tau(x,t,u)$, and $\phi(x,t,u)$ such that the resulting one-parameter sub-group $\exp(\varepsilon \mathbf{v})$ is a symmetry group of Burgers' equation.

To evaluate these coefficients, we need to prolong the vector field up to 2^{nd} order, given that the highest-degree derivative present in the governing PDE is of order 2. The 2^{nd} prolongation of the vector field can be expressed as

$$\mathrm{pr}^{(2)}\, \boldsymbol{v} = \boldsymbol{v} + \phi^x \frac{\partial}{\partial u_x} + \phi^t \frac{\partial}{\partial u_t} + \phi^{xx} \frac{\partial}{\partial u_{xx}} + \phi^{xt} \frac{\partial}{\partial u_{xt}} + \phi^{tt} \frac{\partial}{\partial u_{tt}}. \tag{32}$$

Applying this prolonged vector field to the differential equation in Equation (30), we get the infinitesimal symmetry criteria that

$$\mathrm{pr}^{(2)}\, \boldsymbol{v}[\Delta(x,t,\boldsymbol{u}^{(2)})] = \phi^t - \phi^{xx} + 2u_x \phi^x = 0. \tag{33}$$

To evaluate the individual coefficients, we apply Equation (28). Next, we substitute every instance of u_t with $u_x^2 + u_{xx}$, and equate the coefficients of each monomial in the first and second-order derivatives of u to find the pertinent symmetry groups. Table 2 below lists the relevant monomials as well as their respective coefficients.

Table 2: Monomial coefficients in vector field prolongation for Burgers' equation.

Monomial	Coefficient
1	$\phi_t = \phi_{xx}$
u_x	$2\phi_x + 2(\phi_{xu} - \xi_{xx}) = -\xi_t$
u_x^2	$2(\phi_u - \xi_x) - \tau_{xx} + (\phi_{uu} - 2\xi_{xu}) = \phi_u - \tau_t$
u_x^3	$-2\tau_x - 2\xi_u - 2\tau_{xu} - \xi_{uu} = -\xi_u$
u_x^4	$-2\tau_u - \tau_{uu} = -\tau_u$
u_{xx}	$-\tau_{xx} + (\phi_u - 2\xi_x) = \phi_u - \tau_t$
$u_x u_{xx}$	$-2\tau_x - 2\tau_{xu} - 3\xi_u = -\xi_u$
$u_x^2 u_{xx}$	$-2\tau_u - \tau_{uu} - \tau_u = -2\tau_u$
u_{xx}^2	$-\tau_u = -\tau_u$
u_{xt}	$-2\tau_x = 0$
$u_x u_{xt}$	$-2\tau_u = 0$

Using these relations, we can solve for the coefficient functions. For the case of Burgers' equation, the most general infinitesimal symmetries have coefficient functions of the following form:

$$\xi(t, x) = k_1 + k_4 x + 2k_5 t + 4k_6 xt \tag{34}$$

$$\tau(t) = k_2 + 2k_4 t + 4k_6 t^2 \tag{35}$$

$$\phi(t, x, u) = (k_3 - k_5 x - 2k_6 t - k_6 x^2)u + \gamma(x, t) \tag{36}$$

where $k_1, \ldots, k_6 \in \mathbb{R}$ and $\gamma(x, t)$ is an arbitrary solution to Burgers' equation. These coefficient functions can be used to generate the infinitesimal symmetries. These symmetries are spanned by the six vector fields below:

$$\boldsymbol{v}_1 = \partial_x \tag{37}$$

$$\boldsymbol{v}_2 = \partial_t \tag{38}$$

$$\boldsymbol{v}_3 = \partial_u \tag{39}$$

$$\boldsymbol{v}_4 = x\partial_x + 2t\partial_t \tag{40}$$

$$\boldsymbol{v}_5 = 2t\partial_x - x\partial_u \tag{41}$$

$$\boldsymbol{v}_6 = 4xt\partial_x + 4t^2\partial_t - (x^2 + 2t)\partial_u \tag{42}$$

as well as the infinite-dimensional subalgebra: $\boldsymbol{v}_\gamma = \gamma(x, t)e^{-u}\partial_u$. Here, $\gamma(x, t)$ is any arbitrary solution to the heat equation. The relationship between the Heat equation and Burgers' equation can be seen, whereby if u is replaced by $w = e^u$, the Cole–Hopf transformation is recovered.

B Exponential map and its approximations

As observed in the previous section, symmetry groups are generally derived in the Lie algebra of the group. The exponential map can then be applied, taking elements of this Lie algebra to the corresponding group operations. Working within the Lie algebra of a group provides several benefits. First, a Lie algebra is a vector space, so elements of the Lie algebra can be added and subtracted to yield new elements of the Lie algebra (and the group, via the exponential map). Second, when generators of the Lie algebra are closed under the Lie bracket of the Lie algebra (*i.e.*, the generators form a basis for the structure constants of the Lie algebra), any arbitrary Lie point symmetry can be obtained via an element of the Lie algebra (i.e. the exponential map is surjective onto the connected component of the identity) [11]. In contrast, composing group operations in an arbitrary, *fixed* sequence is not guaranteed to be able to generate any element of the group. Lastly, although not extensively detailed here, the "strength," or magnitude, of Lie algebra elements can be measured using an appropriately selected norm. For instance, the operator norm of a matrix could be used for matrix Lie algebras.

In certain cases, especially when the element $\boldsymbol{v}$ in the Lie algebra consists of a single basis element, the exponential map $\exp(\boldsymbol{v})$ applied to that element of the Lie algebra can be calculated explicitly. Here, applying the group operation to a tuple of independent and dependent variables results in the so-called Lie point transformation, since it is applied at a given point $\exp(\epsilon \boldsymbol{v}) \cdot (x, f(x)) \mapsto (x', f(x)')$. Consider the concrete example below from Burger's equation.

Example B.1 (Exponential map on symmetry generator of Burger's equation)**.** *The Burger's equation contains the Lie point symmetry* $\boldsymbol{v}_\gamma = \gamma(x,t)e^{-u}\partial_u$ *with corresponding group transformation* $\exp(\epsilon \boldsymbol{v}_\gamma) \cdot (x, t, u) = (x, t, \log\left(e^u + \epsilon\gamma\right))$.

Proof. This transformation only changes the u component. Here, we have

$$\begin{aligned} \exp\left(\epsilon\gamma e^{-u}\partial_u\right) u &= u + \sum_{k=1}^{n} \left(\epsilon\gamma e^{-u}\partial_u\right)^k \cdot u \\ &= u + \epsilon\gamma e^{-u} - \frac{1}{2}\epsilon^2\gamma^2 e^{-2u} + \frac{1}{3}\epsilon^3\gamma^3 e^{-3u} + \cdots \end{aligned} \tag{43}$$

Applying the series expansion $\log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots$, we get

$$\begin{aligned} \exp\left(\epsilon\gamma e^{-u}\partial_u\right) u &= u + \log\left(1 + \epsilon\gamma e^{-u}\right) \\ &= \log\left(e^u\right) + \log\left(1 + \epsilon\gamma e^{-u}\right) \\ &= \log\left(e^u + \epsilon\gamma\right). \end{aligned} \tag{44}$$

□

In general, the output of the exponential map cannot be easily calculated as we did above, especially if the vector field $\boldsymbol{v}$ is a weighted sum of various generators. In these cases, we can still apply the exponential map to a desired accuracy using efficient approximation methods, which we discuss next.

B.1 Approximations to the exponential map

For arbitrary Lie groups, computing the exact exponential map is often not feasible due to the complex nature of the group and its associated Lie algebra. Hence, it is necessary to approximate the exponential map to obtain useful results. Two common methods for approximating the exponential map are the truncation of Taylor series and Lie-Trotter approximations.

Taylor series approximation Given a vector field $\boldsymbol{v}$ in the Lie algebra of the group, the exponential map can be approximated by truncating the Taylor series expansion of $\exp(\boldsymbol{v})$. The Taylor series expansion of the exponential map is given by:

$$\exp(\boldsymbol{v}) = \mathrm{Id} + \boldsymbol{v} + \frac{1}{2}\boldsymbol{v} \cdot \boldsymbol{v} + \cdots = \sum_{n=0}^{\infty} \frac{\boldsymbol{v}^n}{n!}. \tag{45}$$

To approximate the exponential map, we retain a finite number of terms in the series:

$$\exp(\boldsymbol{v}) = \sum_{n=0}^{k} \frac{\boldsymbol{v}^n}{n!} + o(\|\boldsymbol{v}\|^k), \tag{46}$$

where k is the order of the truncation. The accuracy of the approximation depends on the number of terms retained in the truncated series and the operator norm $\|\boldsymbol{v}\|$. For matrix Lie groups, where $\boldsymbol{v}$ is also a matrix, this operator norm is equivalent to the largest magnitude of the eigenvalues of the matrix [43]. The error associated with truncating the Taylor series after k terms thus decays exponentially with the order of the approximation.

Two drawbacks exist when using the Taylor approximation. First, for a given vector field $\boldsymbol{v}$, applying $\boldsymbol{v} \cdot f$ to a given function f requires algebraic computation of derivatives. Alternatively, derivatives can also be approximated through finite difference schemes, but this would add an additional source of error. Second, when using the Taylor series to apply a symmetry transformation of a PDE to a starting solution of that PDE, the Taylor series truncation will result in a new function, which is not necessarily a solution of the PDE anymore (although it can be made arbitrarily close to a solution by increasing the truncation order). Lie-Trotter approximations, which we study next, approximate the exponential map by a composition of symmetry operations, thus avoiding these two drawbacks.

Lie-Trotter series approximations The Lie-Trotter approximation is an alternative method for approximating the exponential map, particularly useful when one has access to group elements directly, i.e. the closed-form output of the exponential map on each Lie algebra generator), but they are non-commutative. To provide motivation for this method, consider two elements $\boldsymbol{X}$ and $\boldsymbol{Y}$ in the Lie algebra. The Lie-Trotter formula (or Lie product formula) approximates the exponential of their sum [22, 44].

$$\exp(\boldsymbol{X}+\boldsymbol{Y}) = \lim_{n\to\infty}\left[\exp\left(\frac{\boldsymbol{X}}{n}\right)\exp\left(\frac{\boldsymbol{Y}}{n}\right)\right]^n \approx \left[\exp\left(\frac{\boldsymbol{X}}{k}\right)\exp\left(\frac{\boldsymbol{Y}}{k}\right)\right]^k, \tag{47}$$

where k is a positive integer controlling the level of approximation.

The first-order approximation above can be extended to higher orders, referred to as the Lie-Trotter-Suzuki approximations.Though various different such approximations exist, we particularly use the following recursive approximation scheme [45, 23] for a given Lie algebra component $\boldsymbol{v} = \sum_{i=1}^{p} \boldsymbol{v}_i$.

$$\begin{aligned}
\mathcal{T}_2(\boldsymbol{v}) &= \exp\left(\frac{\boldsymbol{v}_1}{2}\right)\cdot\exp\left(\frac{\boldsymbol{v}_2}{2}\right)\cdots\exp\left(\frac{\boldsymbol{v}_p}{2}\right)\exp\left(\frac{\boldsymbol{v}_p}{2}\right)\cdot\exp\left(\frac{\boldsymbol{v}_{p-1}}{2}\right)\cdots\exp\left(\frac{\boldsymbol{v}_1}{2}\right), \\
\mathcal{T}_{2k}(\boldsymbol{v}) &= \mathcal{T}_{2k-2}(u_k\boldsymbol{v})^2\cdot\mathcal{T}_{2k-2}((1-4u_k)\boldsymbol{v})\cdot\mathcal{T}_{2k-2}(u_k\boldsymbol{v})^2, \\
u_k &= \frac{1}{4-4^{1/(2k-1)}}.
\end{aligned} \tag{48}$$

To apply the above formula, we tune the order parameter p and split the time evolution into r segments to apply the approximation $\exp(\boldsymbol{v}) \approx \prod_{i=1}^{r}\mathcal{T}_p(\boldsymbol{v}/r)$. For the p-th order, the number of stages in the Suzuki formula above is equal to $2\cdot 5^{p/2-1}$, so the total number of stages applied is equal to $2r\cdot 5^{p/2-1}$.

These methods are especially useful in the context of PDEs, as they allow for the approximation of the exponential map while preserving the structure of the Lie algebra and group. Similar techniques are used in the design of splitting methods for numerically solving PDEs [46, 47]. Crucially, these approximations will always provide valid solutions to the PDEs, since each individual group operation in the composition above is itself a symmetry of the PDE. This is in contrast with approximations via Taylor series truncation, which only provide approximate solutions.

As with the Taylor series approximation, the p-th order approximation above is accurate to $o(\|\boldsymbol{v}\|^p)$ with suitably selected values of r and p [23]. As a cautionary note, the approximations here may fail to converge when applied to unbounded operators [48, 49]. In practice, we tested a range of bounds to the augmentations and tuned augmentations accordingly (see Appendix E).

C VICReg Loss

In our implementations, we use the VICReg loss as our choice of SSL loss [9]. This loss contains three different terms: a variance term that ensures representations do not collapse to a single point, a covariance term that ensures different dimensions of the representation encode different data, and an invariance term to enforce similarity of the representations for pairs of inputs related by an augmentation. We go through each term in more detail below. Given a distribution $\mathcal{T}$ from which to draw augmentations and a set of inputs $\boldsymbol{x}_i$, the precise algorithm to calculate the VICReg loss for a batch of data is also given in Algorithm 1.

Formally, define our embedding matrices as $\boldsymbol{Z}, \boldsymbol{Z}' \in \mathbb{R}^{N\times D}$. Next, we define the similarity criterion, $\mathcal{L}_{\text{sim}}$, as

$$\mathcal{L}_{\text{sim}}(\boldsymbol{u},\boldsymbol{v}) = \|\boldsymbol{u}-\boldsymbol{v}\|_2^2,$$

which we use to match our embeddings, and to make them invariant to the transformations. To avoid a collapse of the representations, we use the original variance and covariance criteria to define our

Algorithm 1 VICReg Loss Evaluation

Hyperparameters: $\lambda_{var}, \lambda_{cov}, \lambda_{inv}, \gamma \in \mathbb{R}$
Input: N inputs in a batch $\{\boldsymbol{x}_i \in \mathbb{R}^{D_{in}}, i = 1, \dots, N\}$
VICRegLoss($N, \boldsymbol{x}_i, \lambda_{var}, \lambda_{cov}, \lambda_{inv}, \gamma$):

Apply augmentations $t, t' \sim \mathcal{T}$ to form embedding matrices $\boldsymbol{Z}, \boldsymbol{Z}' \in \mathbb{R}^{N \times D}$:

$$\boldsymbol{Z}_{i,:} = h_\theta\left(f_\theta\left(t \cdot \boldsymbol{x}_i\right)\right) \text{ and } \boldsymbol{Z}'_{i,:} = h_\theta\left(f_\theta\left(t' \cdot \boldsymbol{x}_i\right)\right)$$

Form covariance matrices $\mathrm{Cov}(\boldsymbol{Z}), \mathrm{Cov}(\boldsymbol{Z}') \in \mathbb{R}^{D \times D}$:

$$\mathrm{Cov}(\boldsymbol{Z}) = \frac{1}{N-1} \sum_{i=1}^{N} \left(\boldsymbol{Z}_{i,:} - \overline{\boldsymbol{Z}}_{i,:}\right)\left(\boldsymbol{Z}_{i,:} - \overline{\boldsymbol{Z}}_{i,:}\right)^\top, \quad \overline{\boldsymbol{Z}}_{i,:} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{Z}_{i,:}$$

Evaluate loss: $\mathcal{L}(\boldsymbol{Z}, \boldsymbol{Z}') = \lambda_{var} \mathcal{L}_{var}(\boldsymbol{Z}, \boldsymbol{Z}') + \lambda_{cov} \mathcal{L}_{cov}(\boldsymbol{Z}, \boldsymbol{Z}') + \lambda_{inv} \mathcal{L}_{inv}(\boldsymbol{Z}, \boldsymbol{Z}')$

$$\mathcal{L}_{var}(\boldsymbol{Z}, \boldsymbol{Z}') = \frac{1}{D} \sum_{i=1}^{N} \max(0, \gamma - \sqrt{\mathrm{Cov}(\boldsymbol{Z})_{ii}}) + \max(0, \gamma - \sqrt{\mathrm{Cov}(\boldsymbol{Z}')_{ii}}),$$

$$\mathcal{L}_{cov}(\boldsymbol{Z}, \boldsymbol{Z}') = \frac{1}{D} \sum_{i,j=1, i\neq j}^{N} [\mathrm{Cov}(\boldsymbol{Z})_{ij}]^2 + [\mathrm{Cov}(\boldsymbol{Z}')_{ij}]^2,$$

$$\mathcal{L}_{inv}(\boldsymbol{Z}, \boldsymbol{Z}') = \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{Z}_{i,:} - \boldsymbol{Z}_{i',:}\|^2$$

Return: $\mathcal{L}(\boldsymbol{Z}, \boldsymbol{Z}')$

regularisation loss, $\mathcal{L}_{\text{reg}}$, as

$$\begin{aligned}
\mathcal{L}_{\text{reg}}(\boldsymbol{Z}) &= \lambda_{cov}\ C(\boldsymbol{Z}) + \lambda_{var}\ V(\boldsymbol{Z}), \quad \text{with} \\
C(\boldsymbol{Z}) &= \frac{1}{D} \sum_{i \neq j} \mathrm{Cov}(\boldsymbol{Z})_{i,j}^2 \quad \text{and} \\
V(\boldsymbol{Z}) &= \frac{1}{D} \sum_{j=1}^{D} \max\left(0, 1 - \sqrt{\mathrm{Var}(\boldsymbol{Z}_{:,j})}\right).
\end{aligned}$$

The variance criterion, $V(\boldsymbol{Z})$, ensures that all dimensions in the representations are used, while also serving as a normalization of the dimensions. The goal of the covariance criterion is to decorrelate the different dimensions, and thus, spread out information across the embeddings.

The final criterion is

$$\mathcal{L}_{\text{VICReg}}(\boldsymbol{Z}, \boldsymbol{Z}') = \lambda_{\text{inv}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{\text{sim}}(\boldsymbol{Z}_{i,\text{inv}}, \boldsymbol{Z}'_{i,\text{inv}}) + \mathcal{L}_{\text{reg}}(\boldsymbol{Z}') + \mathcal{L}_{\text{reg}}(\boldsymbol{Z}).$$

Hyperparameters $\lambda_{var}, \lambda_{cov}, \lambda_{inv}, \gamma \in \mathbb{R}$ weight the contributions of different terms in the loss. For all studies conducted in this work, we use the default values of $\lambda_{var} = \lambda_{\text{inv}} = 25$ and $\lambda_{cov} = 1$, unless specified. In our experience, these default settings perform generally well.

D Expanded related works

Machine Learning for PDEs Recent work on machine learning for PDEs has considered both invariant prediction tasks [50] and time-series modelling [51, 52]. In the fluid mechanics setting, models learn dynamic viscosities, fluid densities, and/or pressure fields from both simulation and real-world experimental data [53, 54, 55]. For time-dependent PDEs, prior work has investigated the efficacy of convolutional neural networks (CNNs), recurrent neural networks (RNNs), graph neural networks (GNNs), and transformers in learning to evolve the PDE forward in time [34, 56, 57, 58]. This has invoked interest in the development of reduced order models and learned representations for time integration that decrease computational expense, while attempting to maintain solution accuracy. Learning representations of the governing PDE can enable time-stepping in a latent space, where the computational expense is substantially reduced [59]. Recently, for example, Lusch et al. have studied learning the infinite-dimensional Koopman operator to globally linearize latent space dynamics [60]. Kim et al. have employed the Sparse Identification of Nonlinear Dynamics (SINDy) framework to parameterize latent space trajectories and combine them with classical ODE solvers to integrate latent space coordinates to arbitrary points in time [51]. Nguyen et al. have looked at the development of foundation models for climate sciences using transformers pre-trained on well-established climate datasets [7]. Other methods like dynamic mode decomposition (DMD) are entirely data-driven, and find the best operator to estimate temporal dynamics [61]. Recent extensions of this work have also considered learning equivalent operators, where physical constraints like energy conservation or the periodicity of the boundary conditions are enforced [29].

Self-supervised learning All joint embedding self-supervised learning methods have a similar objective: forming representations across a given domain of inputs that are invariant to a certain set of transformations. Contrastive and non-contrastive methods are both used. Contrastive methods [21, 62, 63, 64, 65] push away unrelated pairs of augmented datapoints, and frequently rely on the InfoNCE criterion [66], although in some cases, squared similarities between the embeddings have been employed [67]. Clustering-based methods have also recently emerged [68, 69, 6], where instead of contrasting pairs of samples, samples are contrasted with cluster centroids. Non-contrastive methods [10, 38, 9, 70, 71, 72, 37] aim to bring together embeddings of positive samples. However, the primary difference between contrastive and non-contrastive methods lies in how they prevent representational collapse. In the former, contrasting pairs of examples are explicitly pushed away to avoid collapse. In the latter, the criterion considers the set of embeddings as a whole, encouraging information content maximization to avoid collapse. For example, this can be achieved by regularizing the empirical covariance matrix of the embeddings. While there can be differences in practice, both families have been shown to lead to very similar representations [16, 73]. An intriguing feature in many SSL frameworks is the use of a projector neural network after the encoder, on top of which the SSL loss is applied. The projector was introduced in [21]. Whereas the projector is not necessary for these methods to learn a satisfactory representation, it is responsible for an important performance increase. Its exact role is an object of study [74, 15].

Equivariant networks and geometric deep learning In the past several years, an extensive set of literature has explored questions in the so-called realm of geometric deep learning tying together aspects of group theory, geometry, and deep learning [75]. In one line of work, networks have been designed to explicitly encode symmetries into the network via equivariant layers or explicitly symmetric parameterizations [76, 77, 78, 79]. These techniques have notably found particular application in chemistry and biology related problems [80, 81, 82] as well as learning on graphs [83]. Another line of work considers optimization over layers or networks that are parameterized over a Lie group [84, 85, 86, 87, 88]. Our work does not explicitly encode invariances or structurally parameterize Lie groups into architectures as in many of these works, but instead tries to learn representations that are approximately symmetric and invariant to these group structures via the SSL. As mentioned in the main text, perhaps more relevant for future work are techniques for learning equivariant features and maps [39, 40].

E Details on Augmentations

The generators of the Lie point symmetries of the various equations we study are listed below. For symmetry augmentations which distort the periodic grid in space and time, we provide inputs x and t to the network which contain the new spatial and time coordinates after augmentation.

E.1 Burgers' equation

As a reminder, the Burgers' equation takes the form

$$u_t + uu_x - \nu u_{xx} = 0. \tag{49}$$

Lie point symmetries of the Burgers' equation are listed in Table 3. There are five generators. As we will see, the first three generators corresponding to translations and Galilean boosts are consistent with the other equations we study (KS, KdV, and Navier Stokes) as these are all flow equations.

Table 3: Generators of the Lie point symmetry group of the Burgers' equation in its standard form [42, 89].

	Lie algebra generator	Group operation $(x, t, u) \mapsto$
g_1 (space translation)	$\epsilon\partial_x$	$(x + \epsilon, t, u)$
g_2 (time translation)	$\epsilon\partial_t$	$(x, t + \epsilon, u)$
g_3 (Galilean boost)	$\epsilon(t\partial_x + \partial_u)$	$(x + \epsilon t, t, u + \epsilon)$
g_4 (scaling)	$\epsilon(x\partial_x + 2t\partial_t - u\partial_u)$	$(e^{\epsilon}x, e^{2\epsilon}t, e^{-\epsilon}u)$
g_5 (projective)	$\epsilon(xt\partial_x + t^2\partial_t + (x - tu)\partial_u)$	$\left(\frac{x}{1-\epsilon t}, \frac{t}{1-\epsilon t}, u + \epsilon(x - tu)\right)$

Comments and errata in [12] As a cautionary note, the symmetry group given in Table 1 of [12] for Burgers' equation is incorrectly labeled for Burgers' equation in its standard form. Instead, these augmentations are those for Burgers' equation in its potential form, which is given as:

$$u_t + \frac{1}{2}u_x^2 - \nu u_{xx} = 0. \tag{50}$$

The potential form is often more convenient for analyzing symmetries of Burgers' equation. Burgers' equation in its standard form is $v_t + vv_x - \nu v_{xx} = 0$, which can be obtained from the transformation $v = u_x$. The Lie point symmetry group of the equation in its potential form contains more generators than that of the standard form. This is because translating all of these generators into the standard form can lose the smoothness and locality of the transformations (some are no longer Lie point transformations).

Fortunately, this error does not carry through in their experiments: [12] only consider input data as solutions to Heat equation, which they subsequently transform into solutions of Burgers' equation via a Cole-Hopf transform. Therefore, in their code, they apply augmentations using the symmetry group of the Heat equation for which they have the correct symmetry group. We opted only to work with solutions to Burgers' equations itself for a fairer comparison to real-world settings, where a convenient transform to a linear PDE such as the Cole-Hopf transform is generally not available.

E.2 KdV

Lie point symmetries of the KdV equation are listed in Table 4. Though all the operations listed are valid generators of the symmetry group, only g_1 and g_3 are invariant to the downstream task of the inverse problem. (Notably, these parameters are independent of any spatial shift). Consequently, during SSL pre-training for the inverse problem, only g_1 and g_3 were used for learning representations. In contrast, for time-stepping, all listed symmetry groups were used.

Table 4: Generators of the Lie point symmetry group of the KdV equation. The only symmetries used in the inverse task of predicting initial conditions are g_1 and g_3 since the other two are not invariant to the downstream task.

	Lie algebra generator	Group operation $(x, t, u) \mapsto$
g_1 (space translation)	$\epsilon\partial_x$	$(x + \epsilon, t, u)$
g_2 (time translation)	$\epsilon\partial_t$	$(x, t + \epsilon, u)$
g_3 (Galilean boost)	$\epsilon(t\partial_x + \partial_u)$	$(x + \epsilon t, t, u + \epsilon)$
g_4 (scaling)	$\epsilon(x\partial_x + 3t\partial_t - 2u\partial_u)$	$(e^{\epsilon}x, e^{3\epsilon}t, e^{-2\epsilon}u)$

E.3 KS

Lie point symmetries of the KS equation are listed in Table 5. All of these symmetry generators are shared with the KdV equation listed in Table 3. Similar to KdV, only g_1 and g_3 are invariant to the downstream regression task of predicting the initial conditions. In addition, for time-stepping, all symmetry groups were used in learning meaningful representations.

Table 5: Generators of the Lie point symmetry group of the KS equation. The only symmetries used in the inverse task of predicting initial conditions are g_1 and g_3 since g_2 is not invariant to the downstream task.

	Lie algebra generator	Group operation $(x, t, u) \mapsto$
g_1 (space translation)	$\epsilon\partial_x$	$(x + \epsilon, t, u)$
g_2 (time translation)	$\epsilon\partial_t$	$(x, t + \epsilon, u)$
g_3 (Galilean boost)	$\epsilon(t\partial_x + \partial_u)$	$(x + \epsilon t, t, u + \epsilon)$

E.4 Navier Stokes

Lie point symmetries of the incompressible Navier Stokes equation are listed in Table 6 [90]. As pressure is not given as an input to any of our networks, the symmetry g_q was not included in our implementations. For augmentations g_{E_x} and g_{E_y}, we restricted attention only to linear $E_x(t) = E_y(t) = t$ or quadratic $E_x(t) = E_y(t) = t^2$ functions. This restriction was made to maintain invariance to the downstream task of buoyancy force prediction in the linear case or easily calculable perturbations to the buoyancy by an amount 2ϵ to the magnitude in the quadratic case. Finally, we fix both order and steps parameters in our Lie-Trotter approximation implementation to 2 for computationnal efficiency.

F Experimental details

Whereas we implemented our own pretraining and evaluation (kinematic viscosity, initial conditions and buoyancy) pipelines, we used the data generation and time-stepping code provided on Github by [12] for Burgers', KS and KdV, and in [18] for Navier-Stokes (MIT License), with slight modification to condition the neural operators on our representation. All our code relies relies on Pytorch. Note that the time-stepping code for Navier-Stokes uses Pytorch Lightning. We report the details of the training cost and hyperparameters for pretraining and timestepping in Table 7 and Table 8 respectively.

F.1 Experiments on Burgers' Equation

Solutions realizations of Burgers' equation were generated using the analytical solution [32] obtained from the Heat equation and the Cole-Hopf transform. During generation, kinematic viscosities, ν, and initial conditions were varied.

Representation pretraining We pretrain a representation on subsets of our full dataset containing $10,000$ 1D time evolutions from Burgers equation with various kinematic viscosities, ν, sampled uniformly in the range $[0.001, 0.007]$, and initial conditions using a similar procedure to [12]. We generate solutions of size 224×448 in the spatial and temporal dimensions respectively, using the default parameters from [12]. We train a ResNet18 [17] encoder using the VICReg [9] approach to joint embedding SSL, with a smaller projector (width 512) since we use a smaller ResNet than in the original paper. We keep the same variance, invariance and covariance parameters as in [9]. We use the following augmentations and strengths:

- Crop of size $(128, 256)$, respectively, in the spatial and temporal dimension.
- Uniform sampling in $[-2, 2]$ for the coefficient associated to g_1.
- Uniform sampling in $[0, 2]$ for the coefficient associated to g_2.
- Uniform sampling in $[-0.2, 0.2]$ for the coefficient associated to g_3.

Table 6: Generators of the Lie point symmetry group of the incompressible Navier Stokes equation. Here, u, v correspond to the velocity of the fluid in the x, y direction respectively and p corresponds to the pressure. The last three augmentations correspond to infinite dimensional Lie subgroups with choice of functions $E_x(t), E_y(t), q(t)$ that depend on t only. For invariant tasks, we only used settings where $E_x(t), E_y(t) = t$ (linear) or $E_x(t), E_y(t) = t^2$ (quadratic) to ensure invariance to the downstream task or predictable changes in the outputs of the downstream task. These augmentations are listed as numbers 6 to 9.

	Lie algebra generator	Group operation $(x, y, t, u, v, p) \mapsto$
g_1 (time translation)	$\epsilon\partial_t$	$(x, y, t+\epsilon, u, v, p)$
g_2 (x translation)	$\epsilon\partial_x$	$(x+\epsilon, y, t, u, v, p)$
g_3 (y translation)	$\epsilon\partial_y$	$(x, y+\epsilon, t, u, v, p)$
g_4 (scaling)	$\epsilon(2t\partial_t + x\partial_x + y\partial_y - u\partial_u - v\partial_v - 2p\partial_p)$	$(e^\epsilon x, e^\epsilon y, e^{2\epsilon} t, e^{-\epsilon} u, e^{-\epsilon} v, e^{-2\epsilon} p)$
g_5 (rotation)	$\epsilon(x\partial_y - y\partial_x + u\partial_v - v\partial_u)$	$(x\cos\epsilon - y\sin\epsilon, x\sin\epsilon + y\cos\epsilon, t, u\cos\epsilon - v\sin\epsilon, u\sin\epsilon + v\cos\epsilon, p)$
g_6 (x linear boost)^1	$\epsilon(t\partial_x + \partial_u)$	$(x+\epsilon t, y, t, u+\epsilon, v, p)$
g_7 (y linear boost)^1	$\epsilon(t\partial_y + \partial_v)$	$(x, y+\epsilon t, t, u, v+\epsilon, p)$
g_8 (x quadratic boost)^2	$\epsilon(t^2\partial_x + 2t\partial_u - 2x\partial_p)$	$(x+\epsilon t^2, y, t, u+2\epsilon t, v, p-2x)$
g_9 (y quadratic boost)^2	$\epsilon(t^2\partial_y + 2t\partial_v - 2y\partial_p)$	$(x, y+\epsilon t^2, t, u, v+2\epsilon t, p-2y)$
g_{E_x} (x general boost)^3	$\epsilon(E_x(t)\partial_x + E'_x(t)\partial_u - xE''_x(t)\partial_p)$	$(x+\epsilon E_x(t), y, t, u+\epsilon E'_x(t), v, p-E''x(t)x)$
g_{E_y} (y general boost)^3	$\epsilon(E_y(t)\partial_y + E'y(t)\partial_v - yE''y(t)\partial_p)$	$(x, y+\epsilon E_y(t), t, u, v+\epsilon E'y(t), p-E''y(t)y)$
g_q (additive pressure)^3	$\epsilon q(t)\partial_p$	$(x, y, t, u, v, p+q(t))$

^{1} case of g_{E_x} or g_{E_y} where $E_x(t) = E_y(t) = t$ (linear function of t)
^{2} case of g_{E_x} or g_{E_y} where $E_x(t) = E_y(t) = t^2$ (quadratic function of t)
^{3} $E_x(t), E_y(t), q(t)$ can be any given smooth function that only depends on t

- Uniform sampling in $[-1, 1]$ for the coefficient associated to g_4.

We pretrain for 100 epochs using AdamW [33] and a batch size of 32. Crucially, we assess the quality of the learned representation via linear probing for kinematic viscosity regression, which we detail below.

Kinematic viscosity regression We evaluate the learned representation as follows: the ResNet18 is frozen and used as an encoder to produce features from the training dataset. The features are passed through a linear layer, followed by a sigmoid to constrain the output within $[\nu_{\text{min}}, \nu_{\text{max}}]$. The learned model is evaluated against our validation dataset, which is comprised of $2,000$ samples.

Time-stepping We use a 1D CNN solver from [12] as our baseline. This neural solver takes T_p previous time steps as input, to predict the next T_f future ones. Each channel (or spatial axis, if we view the input as a 2D image with one channel) is composed of the realization values, u, at T_p times, with spatial step size dx, and time step size dt. The dimension of the input is therefore $(T_p + 2, 224)$, where the extra two dimensions are simply to capture the scalars dx and dt. We augment this input with our representation. More precisely, we select the encoder that allows for the most accurate linear regression of ν with our validation dataset, feed it with the CNN operator input and reduce the resulting representation dimension to d with a learned projection before adding it as supplementary channels to the input, which is now $(T_p + 2 + d, 224)$.

We set $T_p = 20$, $T_f = 20$, and $n_{\text{samples}} = 2,000$. We train both models for 20 epochs following the setup from [12]. In addition, we use AdamW with a decaying learning rate and different configurations of 3 runs each:

- Batch size $\in \{16, 64\}$.
- Learning rate $\in \{0.0001, 0.00005\}$.

F.2 Experiments on KdV and KS

To obtain realizations of both the KdV and KS PDEs, we apply the method of lines, and compute spatial derivatives using a pseudo-spectral method, in line with the approach taken by [12].

Representation pretraining To train on realizations of KdV, we use the following VICReg parameters: $\lambda_{var} = 25$, $\lambda_{inv} = 25$, and $\lambda_{cov} = 4$. For the KS PDE, the λ_{var} and λ_{inv} remain unchanged, with $\lambda_{cov} = 6$. The pre-training is performed on a dataset comprised of $10,000$ 1D time evolutions of each PDE, each generated from initial conditions described in the main text. Generated solutions were of size 128×256 in the spatial and temporal dimensions, respectively. Similar to Burgers' equation, a ResNet18 encoder in conjunction with a projector of width 512 was used for SSL pre-training. The following augmentations and strengths were applied:

- Crop of size $(32, 256)$, respectively, in the spatial and temporal dimension.
- Uniform sampling in $[-0.2, 0.2]$ for the coefficient associated to g_3.

Initial condition regression The quality of the learned representations is evaluated by freezing the ResNet18 encoder, training a separate regression head to predict values of A_k and ω_k, and comparing the NMSE to a supervised baseline. The regression head was a fully-connected network, where the output dimension is commensurate with the number of initial conditions used. In addition, a range-constrained sigmoid was added to bound the output between $[-0.5, 2\pi]$, where the bounds were informed by the minimum and maximum range of the sampled initial conditions. Lastly, similar to Burgers' equation, the validation dataset is comprised of $2,000$ labeled samples.

Time-stepping The same 1D CNN solver used for Burgers' equation serves as the baseline for time-stepping the KdV and KS PDEs. We select the ResNet18 encoder based on the one that provides the most accurate predictions of the initial conditions with our validation set. Here, the input dimension is now $(T_p + 2, 128)$ to agree with the size of the generated input data. Similarly to Burgers' equation, $T_p = 20$, $T_f = 20$, and $n_{\text{samples}} = 2,000$. Lastly, AdamW with the same learning rate and batch size configurations as those seen for Burgers' equation were used across 3 time-stepping runs each.

A sample visualization with predicted instances of the KdV PDE is provided in Fig. 7 below:

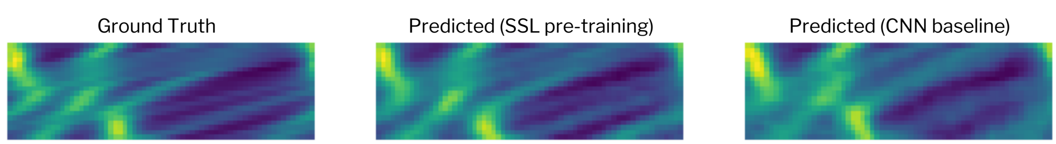

Figure 7: Illustration of the 20 predicted time steps for the KdV PDE. (**Left**) Ground truth data from PDE solver; (**Middle**) Predicted $u(x, t)$ using learned representations; (**Right**) Predicted output from using the CNN baseline.

F.3 Experiments on Navier-Stokes

We use the Conditioning dataset for Navier Stokes-2D proposed in [18], consisting of 26,624 2D time evolutions with 56 time steps and various buoyancies ranging approximately uniformly from 0.2 to 0.5.

Table 7: List of model hyperparameters and training details for the invariant tasks. Training time includes periodic evaluations during the pretraining.

Equation	**Burgers'**	**KdV**	**KS**	**Navier Stokes**
Network:				
Model	ResNet18	ResNet18	ResNet18	ResNet18
Embedding Dim.	512	512	512	512
Optimization:				
Optimizer	LARS [91]	AdamW	AdamW	AdamW
Learning Rate	0.6	0.3	0.3	3e-4
Batch Size	32	64	64	64
Epochs	100	100	100	100
Nb of exps	~ 300	~ 30	~ 30	~ 300
Hardware:				
GPU used	Nvidia V100	Nvidia M4000	Nvidia M4000	Nvidia V100
Training time	$\sim 5h$	$\sim 11h$	$\sim 12h$	$\sim 48h$

Table 8: List of model hyperparameters and training details for the timestepping tasks.

Equation	**Burgers'**	**KdV**	**KS**	**Navier Stokes**
Neural Operator:				
Model	CNN [12]	CNN [12]	CNN [12]	Modified U-Net-64 [18]
Optimization:				
Optimizer	AdamW	AdamW	AdamW	Adam
Learning Rate	1e-4	1e-4	1e-4	1e-3
Batch Size	16	16	16	64
Epochs	20	20	20	50
Hardware:				
GPU used	Nvidia V100	Nvidia M4000	Nvidia M4000	Nvidia V100 (8)
Training time	$\sim 1d$	$\sim 2d$	$\sim 2d$	$\sim 5d$

Representation pretraining We train a ResNet18 for 100 epochs with AdamW, a batch size of 64 and a learning rate of 3e-4. We use the same VICReg hyperparameters as for Burgers' Equation. We use the following augmentations and strengths (augmentations whose strength is not specified here are not used):

- Crop of size $(16, 128, 128)$, respectively in temporal, x and y dimensions.
- Uniform sampling in $[-1, 1]$ for the coefficients associated to g_2 and g_3 (applied respectively in x and y).
- Uniform sampling in $[-0.1, 0.1]$ for the coefficients associated to g_5.
- Uniform sampling in $[-0.01, 0.01]$ for the coefficients associated to g_6 and g_7 (applied respectively in x and y).
- Uniform sampling in $[-0.01, 0.01]$ for the coefficients associated to g_8 and g_9 (applied respectively in x and y).

Buoyancy regression We evaluate the learned representation as follows: the ResNet18 is frozen and used as an encoder to produce features from the training dataset. The features are passed through a linear layer, followed by a sigmoid to constrain the output within $[\text{Buoyancy}_{\text{min}}, \text{Buoyancy}_{\text{max}}]$. Both the fully supervised baseline (ResNet18 + linear head) and our (frozen ResNet18 + linear head) model are trained on $3,328$ unseen samples and evaluated against $6,592$ unseen samples.

Time-stepping We use smaller trajectories (32) as in [18] (56) to reduce computational burden. To condition on our representation, we simply replace the Fourier embedding of the buoyancy by a learned projection of our representation. We compare our conditioning to the parameter conditioning, and no conditioning. All methods are however conditioned on time, and use a single frame to predict a future one. We use the same base configuration as the one provided in [18] for conditioning with modified UNet-64, except we double the effective batch size (since we use 8 GPUs instead of 4) and thus increase the base learning rate to 1e-3. We also depart from [18] by evaluating the learned PDE surrogate at four subsequent time horizons: $\{1, 2, 4, 8\}$.

Table 9: Time-stepping MSE (↓) for Navier-Stokes on various time horizons.

Time horizon	1	2	4	8
Method:				
Time conditioned	0.0028 ± 0.0001	0.0035 ± 0.0001	0.0053 ± 0.0001	0.0106 ± 0.0001
Time + Rep. cond. (ours)	0.0008 ± 0.0001	0.0014 ± 0.0001	0.0032 ± 0.0001	0.0092 ± 0.0001
Time + Param. cond.	0.0006 ± 0.0001	0.0013 ± 0.0001	0.0027 ± 0.0001	0.0091 ± 0.0001

Time-stepping results. We report our complete results after 20k iterations in Table 9.

In order for the appendix to be self-contained, we include references again at the end of the appendix. This reference numbering includes references that appear in the appendix, but not the main body of the paper.

References

[1] Mazier Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. URL `https://doi.org/10.1016/j.jcp.2018.10.045`.

[2] George E. Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021. URL `https://doi.org/10.1038/s42254-021-00314-5`.

[3] Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science advances*, 3(4):e1602614, 2017.

[4] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016. URL `https://doi.org/10.1073/pnas.1517384113`.

[5] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *arXiv preprint arXiv:2103.00020*, 2021.

[6] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9650–9660, 2021.

[7] Tung Nguyen, Johannes Brandstetter, Ashish Kapoor, Jayesh K Gupta, and Aditya Grover. ClimaX: A foundation model for weather and climate. *arXiv preprint arXiv:2301.10343*, 2023.

[8] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, 6, 1993.

[9] Adrien Bardes, Jean Ponce, and Yann LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *arXiv preprint arXiv:2105.04906*, 2021.

[10] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *NeurIPS*, 2020.

[11] Peter J. Olver. Symmetry groups and group invariant solutions of partial differential equations. *Journal of Differential Geometry*, 14:497–542, 1979.

[12] Johannes Brandstetter, Max Welling, and Daniel E Worrall. Lie point symmetry data augmentation for neural pde solvers. *arXiv preprint arXiv:2202.07643*, 2022.

[13] Nail H Ibragimov. *CRC handbook of Lie group analysis of differential equations*, volume 3. CRC press, 1995.

[14] Gerd Baumann. *Symmetry analysis of differential equations with Mathematica®*. Springer Science & Business Media, 2000.

[15] Florian Bordes, Randall Balestriero, Quentin Garrido, Adrien Bardes, and Pascal Vincent. Guillotine regularization: Improving deep networks generalization by removing their head. *arXiv preprint arXiv:2206.13378*, 2022.

[16] Quentin Garrido, Yubei Chen, Adrien Bardes, Laurent Najman, and Yann Lecun. On the duality between contrastive and non-contrastive self-supervised learning. *arXiv preprint arXiv:2206.02574*, 2022.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[18] Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *arXiv preprint arXiv:2209.15616*, 2022.

[19] Victor Isakov. *Inverse problems for partial differential equations*, volume 127. Springer, 2006.

[20] Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Gregoire Mialon, Yuandong Tian, et al. A cookbook of self-supervised learning. *arXiv preprint arXiv:2304.12210*, 2023.

[21] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[22] Hale F Trotter. On the product of semi-groups of operators. *Proceedings of the American Mathematical Society*, 10(4):545–551, 1959.

[23] Andrew M Childs, Yuan Su, Minh C Tran, Nathan Wiebe, and Shuchen Zhu. Theory of trotter error with commutator scaling. *Physical Review X*, 11(1):011020, 2021.

[24] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[25] Nadav Brandes, Dan Ofer, Yam Peleg, Nadav Rappoport, and Michal Linial. Proteinbert: a universal deep-learning model of protein sequence and function. *Bioinformatics*, 38(8): 2102–2110, 2022.

[26] Jianyi Yang, Ivan Anishchenko, Hahnbeom Park, Zhenling Peng, Sergey Ovchinnikov, and David Baker. Improved protein structure prediction using predicted interresidue orientations. *Proceedings of the National Academy of Sciences*, 117(3):1496–1503, 2020.

[27] Rui Wang, Robin Walters, and Rose Yu. Incorporating symmetry into deep dynamics models for improved generalization. *arXiv preprint arXiv:2002.03061*, 2020.

[28] Jack Richter-Powell, Yaron Lipman, and Ricky TQ Chen. Neural conservation laws: A divergence-free perspective. *arXiv preprint arXiv:2210.01741*, 2022.

[29] Peter J. Baddoo, Benjamin Herrmann, Beverley J. McKeon, J. Nathan Kutz, and Steven L. Brunton. Physics-informed dynamic mode decomposition (pidmd), 2021. URL `https://arxiv.org/abs/2112.04307`.

[30] Marc Finzi, Ke Alexander Wang, and Andrew G Wilson. Simplifying hamiltonian and lagrangian neural networks via explicit constraints. *Advances in neural information processing systems*, 33:13880–13889, 2020.

[31] Yuhan Chen, Takashi Matsubara, and Takaharu Yaguchi. Neural symplectic form: learning hamiltonian equations on general coordinate systems. *Advances in Neural Information Processing Systems*, 34:16659–16670, 2021.

[32] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022.

[33] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[34] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019. URL `https://doi.org/10.1073/pnas.1814058116`.

[35] Mert Bulent Sariyildiz, Yannis Kalantidis, Karteek Alahari, and Diane Larlus. No reason for no supervision: Improved generalization in supervised models. In *International Conference on Learning Representations*, 2023.

[36] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[37] Adrien Bardes, Jean Ponce, and Yann LeCun. VICRegl: Self-supervised learning of local visual features. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=ePZsWeGJXyp`.

[38] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *CVPR*, 2020.

[39] Robin Winter, Marco Bertolini, Tuan Le, Frank Noe, and Djork-Arné Clevert. Unsupervised learning of group invariant and equivariant representations. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 31942–31956. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/cf3d7d8e79703fe947deffb587a83639-Paper-Conference.pdf`.

[40] Quentin Garrido, Laurent Najman, and Yann Lecun. Self-supervised learning of split invariant equivariant representations. *arXiv preprint arXiv:2302.10283*, 2023.

[41] Janpou Nee and Jinqiao Duan. Limit set of trajectories of the coupled viscous burgers' equations. *Applied mathematics letters*, 11(1):57–61, 1998.

[42] Peter J Olver. Symmetry groups and group invariant solutions of partial differential equations. *Journal of Differential Geometry*, 14(4):497–542, 1979.

[43] Andrew Baker. *Matrix groups: An introduction to Lie group theory*. Springer Science & Business Media, 2003.

[44] John D Dollard, Charles N Friedman, and Pesi Rustom Masani. *Product integration with applications to differential equations*, volume 10. Westview Press, 1979.

[45] Masuo Suzuki. General theory of fractal path integrals with applications to many-body theories and statistical physics. *Journal of Mathematical Physics*, 32(2):400–407, 1991.

[46] Robert I McLachlan and G Reinout W Quispel. Splitting methods. *Acta Numerica*, 11:341–434, 2002.

[47] Stéphane Descombes and Mechthild Thalhammer. An exact local error representation of exponential operator splitting methods for evolutionary problems and applications to linear schrödinger equations in the semi-classical regime. *BIT Numerical Mathematics*, 50(4):729–749, 2010.

[48] Klaus-Jochen Engel, Rainer Nagel, and Simon Brendle. *One-parameter semigroups for linear evolution equations*, volume 194. Springer, 2000.

[49] Claudia Canzi and Graziano Guerra. A simple counterexample related to the lie–trotter product formula. In *Semigroup Forum*, volume 84, pages 499–504. Springer, 2012.

[50] Mahdi Ramezanizadeh, Mohammad Hossein Ahmadi, Mohammad Alhuyi Nazari, Milad Sadeghzadeh, and Lingen Chen. A review on the utilized machine learning approaches for modeling the dynamic viscosity of nanofluids. *Renewable and Sustainable Energy Reviews*, 114:109345, 2019.

[51] William D Fries, Xiaolong He, and Youngsoo Choi. Lasdi: Parametric latent space dynamics identification. *Computer Methods in Applied Mechanics and Engineering*, 399:115436, 2022.

[52] Xiaolong He, Youngsoo Choi, William D Fries, Jon Belof, and Jiun-Shyan Chen. glasdi: Parametric physics-informed greedy latent space dynamics identification. *arXiv preprint arXiv:2204.12005*, 2022.

[53] Rahmad Syah, Naeim Ahmadian, Marischa Elveny, SM Alizadeh, Meysam Hosseini, and Afrasyab Khan. Implementation of artificial intelligence and support vector machine learning to estimate the drilling fluid density in high-pressure high-temperature wells. *Energy Reports*, 7:4106–4113, 2021.

[54] Ricardo Vinuesa and Steven L Brunton. Enhancing computational fluid dynamics with machine learning. *Nature Computational Science*, 2(6):358–366, 2022.

[55] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[56] Ehsan Adeli, Luning Sun, Jianxun Wang, and Alexandros A Taflanidis. An advanced spatio-temporal convolutional recurrent neural network for storm surge predictions. *arXiv preprint arXiv:2204.09501*, 2022.

[57] Pin Wu, Feng Qiu, Weibing Feng, Fangxing Fang, and Christopher Pain. A non-intrusive reduced order model with transformer neural network and its application. *Physics of Fluids*, 34 (11):115130, 2022.

[58] Léonard Equer, T. Konstantin Rusch, and Siddhartha Mishra. Multi-scale message passing neural pde solvers, 2023. URL `https://arxiv.org/abs/2302.03580`.

[59] Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. In *Computer graphics forum*, volume 38(2), pages 59–70. Wiley Online Library, 2019.

[60] Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):4950, 2018.

[61] Peter J Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 656:5–28, 2010.

[62] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[63] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[64] Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers. In *ICCV*, 2021.

[65] Chun-Hsiao Yeh, Cheng-Yao Hong, Yen-Chi Hsu, Tyng-Luh Liu, Yubei Chen, and Yann LeCun. Decoupled contrastive learning. *arXiv preprint arXiv:2110.06848*, 2021.

[66] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[67] Jeff Z HaoChen, Colin Wei, Adrien Gaidon, and Tengyu Ma. Provable guarantees for self-supervised deep learning with spectral contrastive loss. *NeurIPS*, 34, 2021.

[68] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning. In *ECCV*, 2018.

[69] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *NeurIPS*, 2020.

[70] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *ICML*, pages 12310–12320. PMLR, 2021.

[71] Aleksandr Ermolov, Aliaksandr Siarohin, Enver Sangineto, and Nicu Sebe. Whitening for self-supervised representation learning, 2021.

[72] Zengyi Li, Yubei Chen, Yann LeCun, and Friedrich T Sommer. Neural manifold clustering and embedding. *arXiv preprint arXiv:2201.10000*, 2022.

[73] Vivien Cabannes, Bobak T Kiani, Randall Balestriero, Yann LeCun, and Alberto Bietti. The ssl interplay: Augmentations, inductive bias, and generalization. *arXiv preprint arXiv:2302.02774*, 2023.

[74] Grégoire Mialon, Randall Balestriero, and Yann Lecun. Variance-covariance regularization enforces pairwise independence in self-supervised representations. *arXiv preprint arXiv:2209.14905*, 2022.

[75] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[76] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[77] Risi Kondor and Shubhendu Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. In *International Conference on Machine Learning*, pages 2747–2755. PMLR, 2018.

[78] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016.

[79] Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

[80] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[81] Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

[82] James Kirkpatrick, Brendan McMorrow, David HP Turban, Alexander L Gaunt, James S Spencer, Alexander GDG Matthews, Annette Obika, Louis Thiry, Meire Fortunato, David Pfau, et al. Pushing the frontiers of density functionals by solving the fractional electron problem. *Science*, 374(6573):1385–1389, 2021.

[83] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[84] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015.

[85] Jun Li, Li Fuxin, and Sinisa Todorovic. Efficient riemannian optimization on the stiefel manifold via the cayley transform. *arXiv preprint arXiv:2002.01113*, 2020.

[86] Bobak Kiani, Randall Balestriero, Yann Lecun, and Seth Lloyd. projunn: efficient method for training deep networks with unitary matrices. *arXiv preprint arXiv:2203.05483*, 2022.

[87] Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic recurrent neural networks. *arXiv preprint arXiv:1909.13334*, 2019.

[88] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International conference on machine learning*, pages 1120–1128. PMLR, 2016.

[89] Turgut Öziş and İSMAİL Aslan. Similarity solutions to burgers' equation in terms of special functions of mathematical physics. *Acta Physica Polonica B*, 2017.

[90] SP Lloyd. The infinitesimal group of the navier-stokes equations. *Acta Mechanica*, 38(1-2): 85–98, 1981.

[91] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.