Data-driven Lie point symmetry detection for continuous dynamical systems

Gabel, A.; Quax, R.; Gavves, E.

DOI
10.1088/2632-2153/ad2629

Publication date
2024

Document Version
Final published version

Published in
Machine Learning: Science and Technology

IOP Publishing Mach. Learn.: Sci. Technol. 5 (2024) 015037 https://doi.org/10.1088/2632-2153/ad2629

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
2 September 2023

REVISED
6 January 2024

ACCEPTED FOR PUBLICATION
5 February 2024

PUBLISHED
1 March 2024

PAPER

Data-driven Lie point symmetry detection for continuous dynamical systems

Alex Gabel*, Rick Quax and Efstratios Gavves

University of Amsterdam, Informatics Institute VIS Lab & Computational Science Lab, Amsterdam, The Netherlands
* Author to whom any correspondence should be addressed.

E-mail: a.gabel@uva.nl

Keywords: differential equations, symmetry detection, Lie theory

Abstract

Symmetry detection, the task of discovering the underlying symmetries of a given dataset, has been gaining popularity in the machine learning community, particularly in science and engineering applications. Most previous works focus on detecting 'canonical' symmetries such as translation, scaling, and rotation, and cast the task as a modeling problem involving complex inductive biases and architecture design of neural networks. We challenge these assumptions and propose that instead of constructing biases, we can learn to detect symmetries from raw data without prior knowledge. The approach presented in this paper provides a flexible way to scale up the detection procedure to non-canonical symmetries, and has the potential to detect both known and unknown symmetries alike. Concretely, we focus on predicting the generators of Lie point symmetries of partial differential equations, more specifically, evolutionary equations for ease of data generation. Our results demonstrate that well-established neural network architectures are capable of recognizing symmetry generators, even in unseen dynamical systems. These findings have the potential to make non-canonical symmetries more accessible to applications, including model selection, sparse identification, and data interpretability.

1. Introduction

Symmetry detection, the task of discovering invariant properties from raw data, has been of increasing interest to the machine learning community [1–4], in particular for science and engineering applications [5, 6]. For one, detecting symmetries allows for better interpretability of data [7] or even the sparse identification of underlying mechanisms [8]. Furthermore, incorporating symmetries into the learning algorithm improves robustness and trustworthiness [9] by making sure that predictions obey properties that the underlying mechanism obeys as well (e.g. task-specific invariances or physical conservation laws). Finally, incorporating the right symmetries into learning algorithms leads to much better sample efficiency [10]. As a consequence, methods for detecting symmetries and incorporating them into the deep learning architectures has turned into an enterprise on its own, generating a large family of 'symmetry-aware' [11] and 'physics-aware' [12, 13] neural networks (see figure 1).

Despite various attempts at detecting symmetries algorithmically and exploiting them in a neural setting, three major challenges hamper making symmetry detection a practical objective. The first challenge is that many existing methods require *a priori* manual and explicit definition of symmetries. These manually defined symmetries must then be translated into some form of inductive bias that can, thereafter, be implemented into a neural network architecture. Even if we could have algorithms that automate the design of optimal neural architectures for a given data setting and class of symmetries [11, 14], the practical applicability of these algorithms would be constrained by requiring explicit prior knowledge of what symmetries exist in the data, as well as requiring an explicit definition of them. The second challenge is that for continuous symmetries beyond the usual ones considered in other works, such as translation, scaling, and rotation, which we refer to as the canonical symmetries, defining succinct inductive biases and

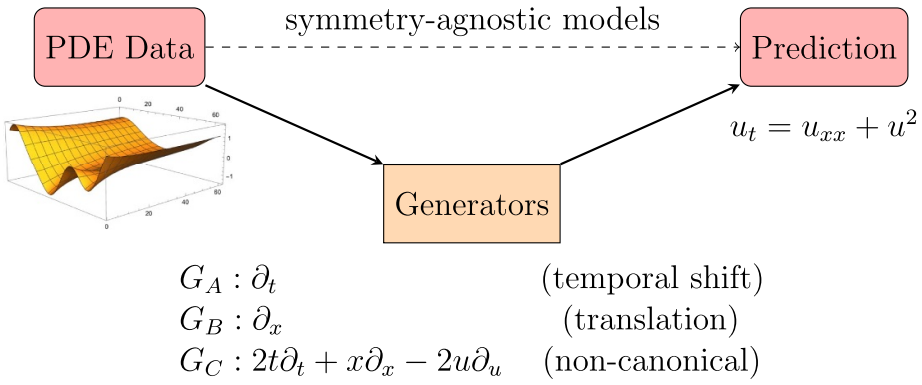

Figure 1. Circumventing the direct approach to making predictions by using the generators of the symmetry group of the data. Here, one starts off with raw PDE time-series data of a heat equation with quadratic source and attempt to identify the generators of the symmetry group before making any predictions, reducing the complexity of the hypothesis space substantially.

optimized neural network implementations is still non-trivial. As a consequence, a large class of symmetries is potentially missed.

The third challenge is that symmetry detection tasks are prone to high computational demand, which in realistic settings with high-dimensional data is prohibitive, for instance, having to learning permutation maps of factorial complexity for images [15]. All in all, despite their theoretical appeal, symmetries are still hard to deploy beyond idealized settings.

In this paper, we are interested in learning symmetry detection directly from data in a practical and scalable manner. We propose to exploit the benefits of deep learning methods to learn symmetries in data given past examples, in which this signal is present. Our conjecture is that for any system (i.e. a data-generating process) that obeys certain symmetries, these latent symmetries must leave a pattern in the observed data. Additionally, we conjecture that if a classifier can be successfully trained for this dataset, a symmetry detector can easily be trained as well. Trivially, it would map data points that share the same label to a symmetry transformation that maps them to each other in a supervised way.

Extending the conjecture, we expect these symmetry patterns to be shareable and recognizable, and thus learnable, because in practice the number of possible dynamical systems is much larger than the set of symmetries that they obey. This is especially true if the set of symmetries is constrained to be interpretable to the engineer or computationally useful. To this end, we propose a straightforward strategy: collect data from systems for which we already know the symmetries, then train neural network classifiers that can predict these symmetries automatically as well as generalize to unseen data.

The first benefit of our approach is that one no longer needs to derive complex inductive biases and correspondingly specialized network implementations to describe a class of symmetries. Instead, one can simply rely on past examples for which the symmetries are already known, and learn to recognize them in novel systems. One way of collecting such examples is by enumerating systems in the form of differential equations, for many of which the corresponding symmetries groups are either already well-documented or could be derived analytically. Whenever a new symmetry group is identified for a particular system, it can be added to the training data and thus enhance the symmetry detection algorithm. Second, we are no longer *de facto* constrained to certain classes of symmetries anymore. Instead, discovering more and more complex symmetries is analogous to collecting more past examples of such complex symmetries. Third, the parametrization proposed in this paper alleviates the computational issue of expressing symmetry transformations for a much broader class of functions while retaining its predictive power for canonical symmetries. This was done by treating the transformations as differential operators [6, 16, 17] rather than the usual matrix group approach [1, 2, 7]. Overall, assuming the symmetries can be predicted accurately, the proposed algorithm allows for scaling up and simplifying the process of discovering symmetries. This is because this algorithm transforms symmetry discovery from a problem of how to design complex and generalized convolutional operations to a problem of how to construct datasets that contain the symmetries of interest. The latter paradigm is arguably simple and more straightforward, since we can exploit on one hand the extensive existing knowledge of dynamical systems and their symmetries, and on the other hand the availability of sophisticated simulation software packages that can generate data for any given dynamical system.

In summary, with this work we make the following three contributions. First, we create a dataset for the detection of these symmetries, which ideally can be a benchmark to compare future algorithms. Second, we describe a practical data-driven algorithm for how to detect continuous non-canonical symmetries using classification, showcasing on partial differential equation (PDE) data for which symmetries are known. Third, we show that the proposed approach allows for reasonable zero-shot generalization even with previously unseen data and symmetries. We expect better architectures and learning algorithms to boost performance.

1.1. Related work

Symmetries in neural networks. Work has been done on studying group convolutional neural networks and using Lie groups to generalize beyond discrete and compact symmetry groups using localized kernels [7]. Although inspiring, this work is specialized to convolutions and kernel-based methods; in addition, the transformation groups to be sought are specified beforehand. In a recent work, a trained neural network was given which had implicitly already learned certain symmetries given a custom learning objective defined on the data, after which they infer the corresponding Lie algebras and their generators [18]. Both works discover symmetries in an indirect manner: they first construct generalized convolutions, or they assume the existing convolutions are already general enough. Then, they train the neural network such that its optimized solution eventually converges to the underlying symmetry. In the context of neural networks, such an implicit approach can be challenging when confronted with more complex symmetries in the data. This can be either due to complex optimization issues [4], or simply because the already trained neural networks are optimized with existing canonical symmetries in mind, and all other symmetries are not relevant and thus ignored during training. For instance, in the realm of image data applications, standard convolutional networks have translational symmetry built-in, while rotational and scaling symmetries are completely ignored [19]. Ideally, there exists a computationally cheap algorithm that predicts the underlying symmetries in the data even when they are arbitrarily complex, generalizing to symmetries with respect to other transformation functions beyond the ones seen during training. Concretely, what is currently missing in related work is a method for parametrizing group transformations *efficiently*, in order to allow for more general symmetries than the usual ones, which we will refer to as 'canonical symmetries', namely rotation, translation, and scaling.

Neural networks and Lie algebras. Most related work is interested in continuous symmetries, which naturally leads to Lie groups and Lie algebras [16]. Our motivation for detecting general symmetries using the formalism of Lie point symmetrie (LPSs) is that we expect that they possess a great potential to simplify learning tasks. Our reasoning is twofold. For one, the appeal of LPSs is that they reduce the number of degrees of freedom of the corresponding differential equations [16, 20, 21] and the usefulness of their application to neural PDE solvers has been shown [17]. Specifically, they have the potential to transform certain non-linear systems into their linear form, e.g. via a hodograph transformation. Second, they allow us to learn properties of the system itself, especially in a setting in which the data was generated from an unknown dynamical system. For instance, one could deduce that the data obeys some sort of conservation law, contains inherent symmetries, or respects some other relationships that are hard to spot by eye and are not directly obvious to the engineer. A framework for parametrizing and detecting LPSs has been explored for basic machine learning tasks [15], but not for PDE data, for which a vast collection of non-canonical symmetries is known. In an engineering context, these properties are already exploited in popular software packages for solving differential equations in a *symbolic* setting. In this article, however, we focus on detecting the symmetries from *raw data* using deep learning, and leave their exploitation for simplifying learning tasks for future work.

2. Theory

This section gives an overview of the concepts and basic definitions of PDEs, their solutions, and their LPSs. We follow the notation and overall logic of the introductions to the subject by Olver [16, 20] and recent work in applying machine learning techniques to this field [17].

2.1. PDEs

In essence, a PDE of integer order n can be defined as a relation between an unknown function u and its partial derivatives with respect to its independent variables. More generally, we can write a PDE as a function of independent variables $\boldsymbol{x}$ and all order k partial derivatives $u^{(k)}$ as follows:

$$\Delta\left(\boldsymbol{x}, u, u^{(1)}, u^{(2)}, \ldots, u^{(n)}\right) = 0, \quad \forall \boldsymbol{x} \in D \tag{1}$$

where D is the domain of the unknown function, such that $u : D \to \mathbb{R}$. In the following, the PDE will also be written using the succinct notation $\Delta_u = 0$. A function u is a solution of the PDE described by Δ if and only

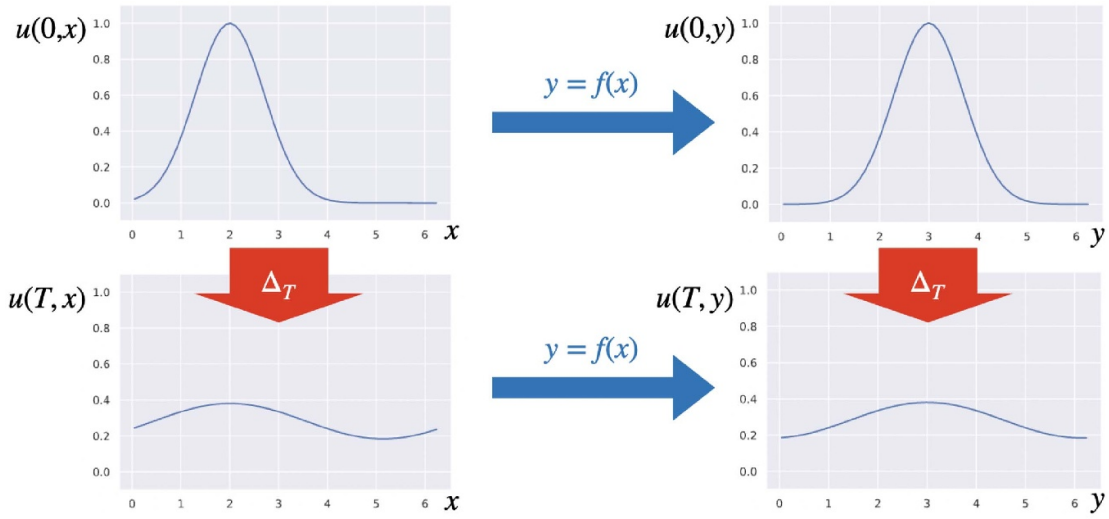

Figure 2. The diffusion equation admits a translational symmetry (see 'shift-equivariance' in machine learning). Shifting any initial condition along the periodic x-axis and then evolving it according to the dynamics of the PDE for some time T (operator Δ_T) yields the same result as first evolving the initial condition and then shifting. Formally, this means the PDE admits an LPS with generator ∂_x and $y = x + \epsilon$, since we can shift the time-series along the x-axis and obtain a new solution.

if it satisfies (1). The solution set of Δ can be defined by $S_\Delta = \{u|\Delta_u = 0\}$, i.e. the set of functions that satisfy the equation of the PDE.

Example: a simple example is the one-dimensional diffusion equation, which can be written as $\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2}$. Its solution function $u(x,t)$ describes the behavior of the concentration of an ink blob suspended in water over time, or how heat diffuses through a metal, among other phenomena. In the equation, the independent variables are x (space) and t (time); the order of this PDE is 2 because the highest derivative is of second order.

2.2. Transformations of PDE solutions

We will focus on *point transformations* (or *coordinate transformations*, see figure 2) that map solution functions to other solutions in the solution set S_Δ. That is, we consider a transformation g that maps solutions to solutions, such that $g \cdot S_\Delta \subseteq S_\Delta$. Transformation functions are defined in the natural way, such that a solution written as a function of the independent variables $u = f(\boldsymbol{x})$ transforms by composition $\bar{u} = g \cdot f(\boldsymbol{x}) = \bar{f}(\bar{\boldsymbol{x}})$. As these transformations are continuous, they can always be parametrized. For infinitesimal transformations, i.e. small values of the parameter, this symmetry transformation defines a differential operator, also known as the generator of the symmetry transformation. In order to recover the transformation function, the differential operator would need to be integrated (also known as applying the exponential map). The constant of integration is then our parameter (angles of rotation, distance translates, scaling parameters etc).

Example: it is well known that the classic diffusion equation admits a variety of symmetries. Almost trivially, it is symmetric up to constant shifts in x and t. Concretely, for any solution $u(x,t)$ and real number $\epsilon \in \mathbb{R}$, the transformations $u(x,t) \to u(x+\epsilon,t)$ and $u(x,t) \to u(x,t+\epsilon)$ generate new solutions. Also, as this equation is linear, it has both scaling and additive symmetries. Formulated as point transformations, this means that $e^\epsilon u(x,t)$ and $u(x,t) + \epsilon\alpha(x,t)$ are also solutions (for $\alpha(x,t)$ also a solution to the diffusion equation). Finally, it admits a non-isotropic scaling $(x,t,u) \to (e^\epsilon x, e^{2\epsilon}t, u)$ and some more complicated transformations.

2.3. LPSs

In the context of PDEs, an LPS describes *continuous* symmetries. Its main properties are that the relevant transformations form a Lie group (closed, invertible, and smooth, i.e. sufficiently differentiable[1]), act point-wise, and do not depend on derivatives of the unknown function $u(x,t)$. An LPS can act on previously found solutions in order to obtain new, possibly not yet known solutions. In addition, an LPS can be used to break down complicated PDEs into simpler ones. This always implies reducing the number of degrees of freedom, and sometimes (but not always) implies reducing the complexity of the equations, such as going

[1] Usually, infinitely differentiable is used as the required property, labeled as C^∞.

from non-linear to linear. Simplifying systems of PDEs using their symmetries and consequent coordinate transformations was, in fact, the primary reason Sophus Lie discovered Lie groups [20].

In the field of machine learning, a symmetry of a task is a transformation of the input data that leaves a certain aspect of the output unchanged. This can range from an *invariance*, in which the expected output of the task is independent of the transformation, to the more general *equivariance*, where a transformation to the input leads to the same transformation being applied to the output. Knowing a symmetry relation between data points leads to a simpler learning task, greatly reducing the number of parameters that need to be learned and hence the size of the hypothesis space of functions modeled by the neural network. Using LPSs should therefore lead to better generalizability of symmetry-aware neural networks.

We emphasize that LPSs were chosen as they contain all the canonical symmetries widely discussed in the machine learning literature and simultaneously provide a bridge to detecting symmetries in scientific data, especially in the form of differential equations. It should be mentioned that beyond Lie point there are other types of symmetries of dynamical systems, such as Lie contact, Lie–Bäcklund, and Noether symmetries (the latter being well-known to physicists and related to solutions of Euler–Lagrange equations). These symmetries are beyond the scope of our current work, and we refer the interested reader to the resources referenced at the start of this section for more information.

2.4. Generators of LPSs

The generators of a PDE, describing its symmetries, form a basis of a vector space that is tangent to a single solution. These generators, then, are the objects we wish to learn from one or more time series of which we do not know the underlying PDE.

Example: take the abovementioned, one-dimensional diffusion equation with diffusivity constant $k \in \mathbb{R}$. This equation admits a number of LPSs. We will look at three different ways to interpret these symmetries. First, we provide a list of the generators as *differential operators*:

$$\begin{aligned}
&G_1 : \partial_t \\
&G_2 : \partial_x \\
&G_3 : 2t\partial_t + x\partial_x \\
&G_4 : 2t\partial_x - \frac{1}{k}xu\partial_u \\
&G_5 : 4t^2\partial_t + 4xt\partial_x - \frac{1}{k}\left(x^2 + 2kt\right)u\partial_u \\
&G_6 : u\partial_u \\
&G_7 : b\left(t,x\right)\partial_u .
\end{aligned}$$

Here, $b(t,x)$ is a solution to the PDE itself, i.e. $b_t = kb_{xx}$. As discussed above, each of the generators can also be thought of as a transformation that maps solutions to other solutions, with the constant of integration functioning as the transformation parameter. Note that while some of these generators are easily interpretable, such as G_1 being a shift along the time axis and G_2 being a shift along the x-axis (or 'translation'), others are more involved and might require integration before the nature of the transformation becomes clear. G_3 is a non-isotropic scaling, in which the scaling factor along the t-axis is double in magnitude than along the x-axis. G_4 and G_5 are more obscure, while G_6 generates a simple scaling in the u direction (i.e. the height of the graph in the (t,x,u)-space). The final generator G_7 is quite special, as it reflects the possibility of allowing for the addition of solutions to each other. This generator furnishes an infinite-dimensional Lie algebra, as there are no countable 'directions' in which to transform (this depends on the choice of $b(t,x)$, of which there are an infinite number). In combination with G_6, it tells us that the PDE is in fact linear. One can readily see the way in which this formalism generalizes the concept of a symmetry transformation, effortlessly allowing for generators beyond the canonical ones considered in many works, such as G_1, G_2, and G_6.

Alternatively, we can turn to the differential geometry perspective of the operators and explicitly rewrite these as vector fields, in the (t,x,u)-basis:

$$G_1 : \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad G_3 : \begin{bmatrix} 2t \\ x \\ 0 \end{bmatrix}, \quad G_4 : \begin{bmatrix} 0 \\ 2t \\ -\frac{1}{k}xu \end{bmatrix}, \quad \text{and} \quad G_5 : \begin{bmatrix} 4t^2 \\ 4xt \\ -\frac{1}{k}\left(x^2 + 2kt\right)u \end{bmatrix}.$$

Intuitively, the vector fields define a 'flow' along which the graph of the solution of the PDE can be transformed such that the graph of a new solution is obtained. This leads to the third interpretation of the LPSs, which is the explicit coordinate transformation discussed in section 2.2.

3. Method

In this section, we describe the full process for obtaining an algorithm that can automatically discover complex symmetries in unseen data. We do this by learning coefficients of a set of predefined terms for the functions that describe the symmetry transformation. First, we discuss the data generation process used. Then, we describe the proposed encoding scheme for generators. Lastly, we describe the final neural network algorithm to predict symmetries.

3.1. Data generation with evolutionary equations

To simplify the process of sampling and due to label availability, we will restrict our attention to one-dimensional evolutionary equations. These equations have the form $\frac{\partial u}{\partial t} = \tilde{\Delta}(\boldsymbol{x}, u^{(1)}, u^{(2)}, \ldots, u^{(n)})$ in which we single out the first partial derivative with respect to time, and do not allow for other higher order or mixed partial derivatives with respect to time. Evolutionary equations allow for efficient sampling and roll-out. A superposition of randomly selected Fourier modes is used to generate one-dimensional, periodic initial conditions, a distribution which will be labeled as $\mathcal{F}$ (B), as is done in similar work [17, 22]. Then a solver is used to calculate the evolution of the function through time. This approach ensures data of the same dimensionality and initial conditions sampled i.i.d. for each PDE [23].

We pick 25 PDEs that have evolutionary dynamics in order to be able to sample an initial condition and calculate its time-series accordingly using *Mathematica*'s built-in solver[2] Furthermore, their LPS generators are known and tabulated in differential equation handbooks [24]. In order to encode the vector field generators for these generalized symmetries, we decided on learning the normalized coefficients.

Example: let us use the running example of the linear heat equation. Then we obtain the following generators, that we later use as labels in our classification setting:

$$G_1: \begin{matrix} \partial_t & \partial_x & \partial_u & \\ \end{matrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{matrix} c \\ t \\ x \\ u \\ tu \end{matrix} \quad \text{and} \quad G_3: \begin{matrix} \partial_t & \partial_x & \partial_u & \\ \end{matrix} \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{matrix} c \\ t \\ x \\ u \\ tu \end{matrix} .$$

Note that, depending on the set of terms used for the regression, certain generators cannot be represented at all. Additionally, as generators are like basis vectors of the Lie algebra (or directions), it is justifiable to normalize the vectorized matrix in order to remove this degree of freedom.

In table 1 the full list of PDEs (some well-known, others more obscure) and the chosen LPS generator are shown. The first PDEs are the diffusion (or linear heat) equation with different diffusion constants (1)–(3). The next ones are a non-linear heat equation (4), two non-linear filtration equations (5)–(6), and a potential filtration equation (7), respectively. What follows is a series of non-linear heat equations with various sources (8)–(21), Burger's equation (22), the potential form of Burger's equation (23), a generalized Hopf equation (24), and the Benjamin–Bona–Mahony (BBM) equation (25). The generators are expressed as vector fields using differential operators. Additionally, the encoding of these generators as coefficient matrices is done as was shown above. This encoding is quite straightforward, as we can normalize the coefficient of the vector field by normalizing the vectorized coefficient matrices. E.g. the normalization constants for G_1 and G_3 above are 1 and $1/\sqrt{3}$ respectively.

The generators were chosen such that are maximally shared across PDEs; in other words, we avoided generators that are unique in the sense that only a single PDE has them. Also, translations in x and t-directions were ignored for the sake of simplicity, as all PDEs considered share these particular symmetries due to their standardized form. Interestingly, while the time series are often similar to the naked eye, standard multi-layer perceptron (MLP) could still discern a meaningful signal for the classification of symmetries and even the type of PDE.

3.2. Symmetry detection via supervised learning

It is important to keep in mind that symmetries are not unique to PDEs, and each PDE can have an arbitrary number of generators. However, for simplicity in training the neural network, we randomly assign each symmetry to a single PDE. Specifically, in our experiments, we pick one generator per PDE. This randomly chosen generator is the target output of our symmetry detector. Picking one generator per PDE prevents

[2] The Wolfram Language `NDSolve` function ultimately reLies on the numerical method of lines. (For more information, see: https://reference.wolfram.com/language/tutorial/NDSolvePDE.html).

Table 1. List of the PDEs and generator labels used in the supervised training of the LPS detector. To avoid one-to-one mappings of PDE to LPS generator, maximal overlap was chosen among the labels. Note that some are shared.

Index	PDE	Generator
1	$u_t = 0.1u_{xx}$	$[2t, x, 0]$
2	$u_t = 1.0u_{xx}$	$[2t, x, 0]$
3	$u_t = 10.0u_{xx}$	$[2t, x, 0]$
4	$u_t = (e^u u_x)_x$	$[2t, x, 0]$
5	$u_t = e^{u_x} u_{xx}$	$[0, 0, 1]$
6	$u_t = \frac{u_{xx}}{(u_x)^2+1} e^{3\arctan(u_x)}$	$[0, 0, 1]$
7	$u_t = \arctan(u_{xx})$	$[0, 0, 1]$
8	$u_t = (e^u u_x)_x + e^{-2u}$	$[2t, 3x/2, 1]$
9	$u_t = (e^u u_x)_x + e^{-u}$	$[t, x, 1]$
10	$u_t = (e^u u_x)_x - e^u$	$[t, 0, -1]$
11	$u_t = (e^u u_x)_x - e^{2u}$	$[2t, x/2, -1]$
12	$u_t = (e^u u_x)_x + 1$	$[0, x, 2]$
13	$u_t = (e^u u_x)_x - 1$	$[0, x, 2]$
14	$u_t = u_{xx} - e^u$	$[2t, x, -2]$
15	$u_t = u_{xx} + u^{-1}$	$[2t, x, u]$
16	$u_t = u_{xx} + u^2$	$[2t, x, -2u]$
17	$u_t = u_{xx} - u^2$	$[2t, x, -2u]$
18	$u_t = u_{xx} + u$	$[2t, x, 2tu]$
19	$u_t = u_{xx} - u$	$[2t, x, -2tu]$
20	$u_t = u_{xx} + 1$	$[2t, x, 2tu]$
21	$u_t = u_{xx} - 1$	$[2t, x, -2tu]$
22	$u_t = uu_x + u_{xx}$	$[2t, x, -u]$
23	$u_t = u_{xx} + (u_x)^2$	$[0, 0, 1]$
24	$u_t = (e^u u_x)_x - uu_x$	$[t, t+x, 1]$
25	$u_t = uu_x + u_{txx}$	$[t, 0, -u]$

overfitting in our limited data set. Our symmetry detector is implemented by a fully-connected neural network and various neural network models will be compared in order to investigate the effect of inductive bias on the symmetry detector's performance.

The supervised learning task is defined as follows. For each PDE Δ^N, labeled by $N \in \{1, 2 \ldots 25\}$, the time-evolution of a sampled initial condition $u(x, 0) \sim \mathcal{F}$ is used as the input. The neural network f_θ with parameters θ is trained to return the symmetry generator that was assigned to the underlying PDE in table 1. The loss is simply the mean squared error (MSE) such that

$$\mathcal{L} = \mathrm{MSE}\left(f_\theta\left(u\right), G_N\right), \quad \Delta_u^N = 0 \tag{2}$$

is minimized over all samples from the 25 different PDEs using stochastic gradient descent in the usual way. Note that a crucial assumption has been made, namely that the presence of periodic boundary conditions will not affect the symmetry generators. This is not necessarily true in all cases and is important to keep in mind.

This seemingly incomplete supervised task has the interesting consequence that it will be possible to judge the performance of our model by checking how it generalizes to unseen PDEs and whether the training scheme allows for predicting correct generators beyond the ones that were intended, i.e. the assigned labels in table 1.

3.3. Zero-shot symmetry detection

In addition to learning and testing on the full data-set, we performed a zero-shot learning approach, in which we omitted various numbers of PDEs from training. This experiment was designed in order to check the generalizability of the symmetry detector to unseen PDEs and their symmetry generators. The results of this type of zero-shot test should be informative, as there exists the possibility that time-series data from one PDE might result in a correct generator, different from the one expected in table A1. A non-exhaustive list of exact generators for additional LPSs is given in A (table A1).

4. Results

The neural network model used is a five-layer deep, fully-connected, MLP with the following number of neurons in each layer: 3968, 256, 128, 64, 32, and 15. The input size accommodates for the size of the time-series, which are all discretized by N in time and N in space. The output size corresponds to the length of the flattened coefficient matrix, which is represented by a 3-by-5 matrix in section 3.1. We choose a batch

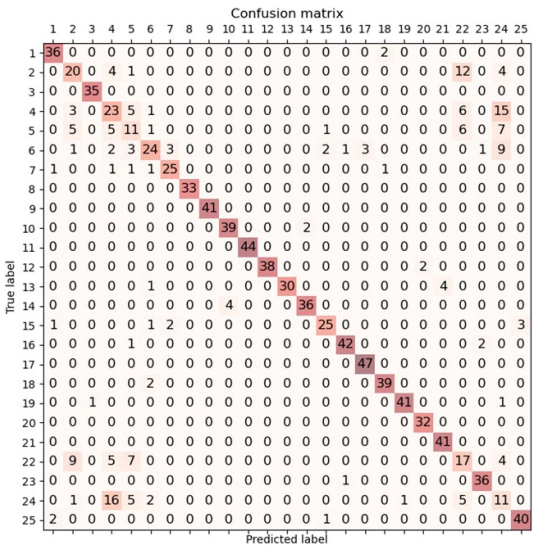

Figure 3. Confusion matrix for the 25-way PDE classification problem. Noticeably, time-series obtained from PDE 24 seems to be mistaken for PDE 4.

size of 128 and the Adam optimizer with a learning rate of 0.001 including a scheduler with step-size 50 and $\gamma = 0.1$.

4.1. Experiment 1: can the neural network predict known PDEs from novel time-series?

As an initial task, the model is trained to predict to which PDE a given time-series belongs. This is done in order to understand the predictive power of a neural network in the context of the PDE dataset that was generated for the subsequent tasks. As hypothesized (see section 3.1), this task being learned successfully by the model implies that the supervised task we are interested in can be easily accomplished. All that is required, in theory, is that the model also maps the correct PDE to the chosen label. An alternative perspective: it also means that the numerical quality of the dataset and the complexity of the initial conditions that were sampled is high enough for a distinction on the basis of the underlying dynamics is possible.

With the model described above, but adjusted for the 25-way classification problem of predicting the correct PDE given a time-series, an accuracy of 80.6% and an average F1-score of 81.5% was obtained (figure 3). The only differences are the size of the output layer (25 instead of 15) and the loss function (LogSoftmax and NLL instead of linear and MSE). From the predictions, it seems like the biggest errors were made for PDEs with similar symbolic forms. In particular, we see confusion between classifying PDEs 4 and 24, which differ up to a term, i.e. uu_x, resulting in similar behavior for small values of the function or its spatial derivative compared to the exponential term.

4.2. Experiment 2: can the neural network predict the generators of known PDEs on novel time-series?

For this task, the model is trained to output a symmetry generator of the input's underlying PDE. This task was designed to check the regression method implemented as the encoding of the LPS on unseen time-series (figure 4). The results show that with the MSE loss, the model is good at learning the encoded generator if the PDE is present in the training set. Note that, in order to normalize the predicted generator, the output of the model was vectorized and normalized using the usual inner-product. This ensures that the degree of freedom associated with scaling the generator (similar to that of a basis vector in linear algebra) is removed, as the targets were also normalized according to the inner-product.

4.3. Experiment 3: can the neural network zero-shot predict generators of unknown PDEs on novel time-series?

In order to investigate the generalizability of the model, each PDE is left out during training and the performance is tested on data generated from this unseen PDE (figure 4). Depending on which PDE was left out, different performance drops are observed (figure 5). Some PDEs have a performance drop of more than 0.01, which is about double the average MSE across all PDEs (PDEs 3, 7, 12, 13, 20, 21, and 23). Others retain their predictive performance (notably, PDEs 9, 10, 11, and 14). It is worth pointing out that the predictions for PDE 23 look very similar to a different generator that is also admissible, namely: $[2t, x, 0]$.

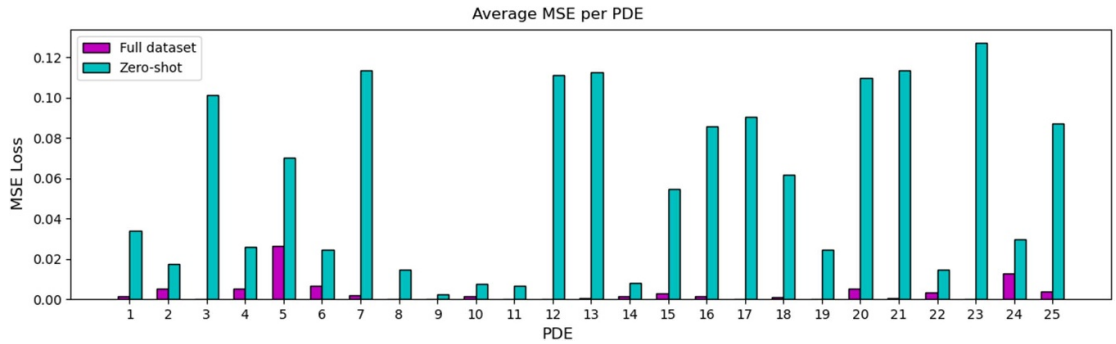

Figure 4. Neural networks can predict the generators of known PDEs (blue) on novel time-series, never seen during training (green). As expected, performance drops when a particular PDE is left out of the training set (red).

Different deep learning models were used in the above experiments, with similar results. Various sizes of MLPs, ResNets [25], and small transformer models [26] were compared and the MLP was chosen for its simplicity. Results for this comparison can be found in C.

5. Discussion

Various interesting insights arise from the previous results. Firstly, we note the success of the model to classify PDEs from their time-series data. This shows that this novel type of task is well-defined and worth improving on, especially in the field of model selection or group-equivariant architectures. Second, we observe that while the supervised task works well, as expected, the zero-shot task performed exceedingly well in some cases, while failing in others. We note that other generators beyond the ones in table 1 are admissible, and we refer the interested reader to table A1 in A for additional ones. Since LPSs form a Lie algebra, linear combinations of generators are also generators. A full list of symmetries can be found in tables in differential equation handbooks [24]. Some questions regarding how to measure the degree of success of this task remain open.

Not only do we propose a broader, more complex symmetry detection task for machine learning algorithms that claim to exploit equivariances in real-world settings, but also a way of encoding these non-canonical symmetries in a tractable manner. Additionally, we suggest considering a wider range of dynamical systems and differential equations rather than the usual ones, especially given the variety in the nature of the underlying symmetries. There are some issues left that must be addressed before this becomes a fully successful task. Namely: how can a model allow for a variable number of generators? What other tasks can be performed using these LPS generators? How can LPS generators be integrated into an equivariant model? One potential way of using the learned symmetry generator to do model selection involves exploiting the exponential map in order to determine a suitable connectivity matrix for a neural network that is to be used in a downstream task [4]. This is a promising avenue for future research.

We point out two limitations of using neural networks in this setting. Firstly, the MLP used in the experiments only works for $(1+1)$-dimensional PDEs that are similar enough in nature to the ones in the training set (e.g. evolutionary equations). In the zero-shot experiment, we were able to show some form of generalizability to novel PDEs. Nevertheless, this is not a 'universal' symmetry detector that works on other types of data without training the model to do so. The biggest contribution is the encoding of the symmetry generators and showing its usefulness in symmetry detection for a specific 'family' of PDEs, which can easily be extended to other settings.

Secondly, due to the fixed output size and supervised training, only one symmetry at a time can be detected. This is a reflection of the way in which the output neurons encode the symmetry generator, namely, using symbolic regression. We believe the latter not to be a big limitation, as this set-up can easily be extended to a multi-label output (e.g. by predicting a sequence of generators as output). Many factors need to be taken into account, however, the main ones being the permutation invariance of the labels, the variable number of labels per PDE, and the fact that some generators are admissible by all the PDEs in the dataset. The real issue here is whether the model will eventually learn and perform well on unseen data. This is a question for further work. From our motivation, in the context of current machine learning research

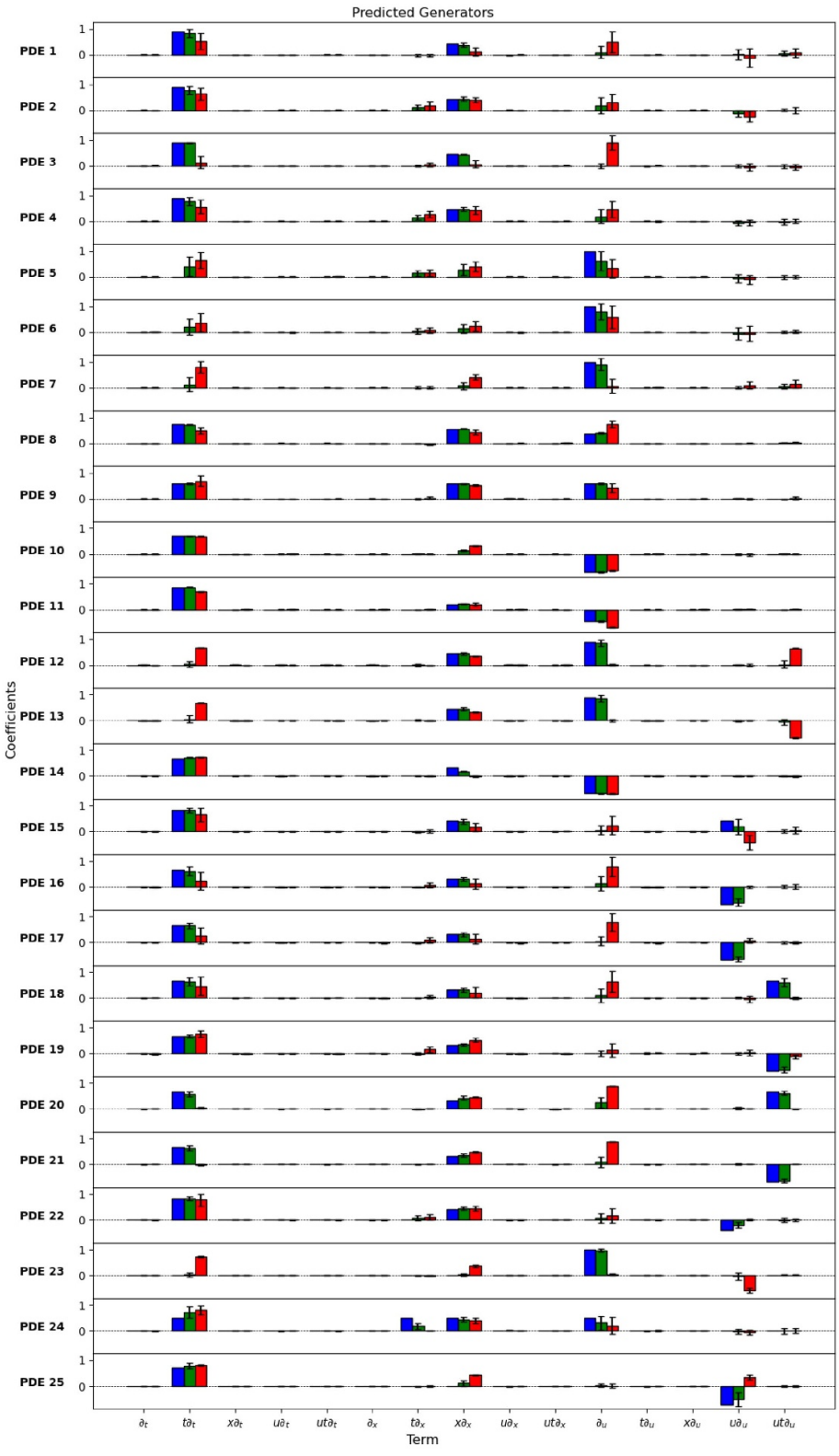

Figure 5. Performance drop for each PDE depending on whether it was included during training or not.

(inspired by group convolutions and geometric deep learning), we argue that just one symmetry (i.e. inductive bias, structural prior, or restriction of the hypothesis space) should suffice for a single task.

6. Conclusion

In this paper, we propose a novel, pragmatic way to encode and learn symmetries in the context of neural symmetry detection. We show that using this framework, canonical symmetries as well as more complex

symmetries can be learned effectively from time-series data. We also find signs of generalizability to generators which are not seen during training. All in all this suggests that our method may be used to construct a centralized symmetry detection algorithm which is continually trained on a library of existing and new found symmetries and can be applied in a wide variety of applications that involve differential equations. This is a marked step forward in the automation of symmetry detection that is broadly accessible.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: 10.5281/zenodo.8311061.

Appendix A. List of PDEs

The chosen evolutionary equations are listed in table A1, including a selection of interesting generators. Note that there are some overlapping generators, as well as the ∂_t and ∂_x which are shared among all PDEs in the list.

Table A1. PDE table.

Index	PDE	Generators
1	$u_t = 0.1u_{xx}$	$[2t, x, 0], [0, 2t, -10xu], [0, 0, u]$
2	$u_t = 1.0u_{xx}$	$[2t, x, 0], [0, 2t, -xu], [0, 0, u]$
3	$u_t = 10.0u_{xx}$	$[2t, x, 0], [0, 2t, -0.1xu], [0, 0, u]$
4	$u_t = (e^u u_x)_x$	$[2t, x, 0], [0, x, 2]$
5	$u_t = e^{u_x} u_{xx}$	$[0, 0, 1], [2t, x, u], [0, 0, 1]$
6	$u_t = \frac{u_{xx}}{(u_x)^2+1} e^{3\arctan(u_x)}$	$[0, 0, 1], [2t, x, u], [3t, u, -x]$
7	$u_t = \arctan(u_{xx})$	$[0, 0, 1]$
8	$u_t = (e^u u_x)_x + e^{-2u}$	$[2t, 3x/2, 1]$
9	$u_t = (e^u u_x)_x + e^{-u}$	$[t, x, 1]$
10	$u_t = (e^u u_x)_x - e^u$	$[t, 0, -1]$
11	$u_t = (e^u u_x)_x - e^{2u}$	$[2t, x/2, -1]$
12	$u_t = (e^u u_x)_x + 1$	$[0, x, 2], [e^{-t}, 0, e^{-t}]$
13	$u_t = (e^u u_x)_x - 1$	$[0, x, 2], [e^t, 0, e^{-t}]$
14	$u_t = u_{xx} - e^u$	$[2t, x, -2]$
15	$u_t = u_{xx} + u^-1$	$[2t, x, u]$
16	$u_t = u_{xx} + u^2$	$[2t, x, -2u]$
17	$u_t = u_{xx} - u^2$	$[2t, x, -2u]$
18	$u_t = u_{xx} + u$	$[2t, x, 2tu]$
19	$u_t = u_{xx} - u$	$[2t, x, -2tu]$
20	$u_t = u_{xx} + 1$	$[2t, x, 2tu], [0, 0, u - t]$
21	$u_t = u_{xx} - 1$	$[2t, x, -2tu], [0, 0, u + t], [0, 2t, -xt - xu]$
22	$u_t = uu_x + u_{xx}$	$[2t, x, -u], [0, t, -1]$
23	$u_t = u_{xx} + (u_x)^2$	$[0, 0, 1], [2t, x, 0], [0, 2t, -x]$
24	$u_t = (e^u u_x)_x - uu_x$	$[t, t + x, 1]$
25	$u_t = uu_x + u_{txx}$	$[t, 0, -u]$

Appendix B. Sampling initial conditions

The dataset generated consists of 200 time-series per PDE (a list of the PDEs can be found in A). The `NDSolve` function was used with the method of lines, and both `MaxPoints` and `MinPoints` were set to 800. In order to generate periodic initial conditions for the solver, we used a similar procedure as in [22, 27] for the driving term, namely:

$$u(t = 0, x) = \sum_{i=1}^{N} A_i \sin(\omega_i x + \phi_i)$$

for $A_i \sim [-0.5, 0.5]$, $\omega_i \sim \{1, 2, 3\}$, and $\phi_i \sim [0, 2\pi]$ sampled uniformly. Additionally, periodic boundary conditions were enforced.

Table C2. Model comparison (TF denotes a transformer model trained from scratch).

Model	Test Loss	Parameters ($\times 10^6$)
MLP	0.009 645	1.1
ResNet18	0.001 904	11.7
TF (1 layer, 1 head)	0.011 046	3.9
TF (1 layer, 4 heads)	0.012 796	3.9

Appendix C. Other models

We performed the symmetry detection experiment on the full dataset for multiple models (experiment 2). Although the ResNet18 model performs the best, it has an order of magnitude more parameters.

ORCID iDs

Alex Gabel https://orcid.org/0009-0008-0964-8918
Rick Quax https://orcid.org/0000-0002-0299-0074
Efstratios Gavves https://orcid.org/0000-0001-8947-1332

References

[1] Rao R and Ruderman D 1998 Learning Lie groups for invariant visual perception *Advances in Neural Information Processing Systems* vol 11 , ed M Kearns, S Solla and D Cohn (MIT Press)
[2] Sohl-Dickstein J, Wang C M and Olshausen B A 2010 An unsupervised algorithm for learning Lie group transformations (arXiv:1001.1027)
[3] Cohen T and Welling M 2014 Learning the irreducible representations of commutative Lie groups (arXiv:1402.4437)
[4] Dehmamy N, Walters R, Liu Y, Wang D and Yu R 2021 Automatic symmetry discovery with Lie algebra convolutional network *Advances in Neural Information Processing Systems* ed A Beygelzimer, Y Dauphin, P Liang and J W Vaughan
[5] Krippendorf S and Syvaeri M Detecting symmetries with neural networks 2020 (arXiv:2003.13679)
[6] Liu Z and Tegmark M Machine-learning hidden symmetries CoRR (arXiv:2109.09721)
[7] van der Ouderaa T F A, Romero D W, van der Wilk M 2022 Relaxing equivariance constraints with non-stationary continuous filters (arXiv:2204.07178)
[8] Brunton S L, Proctor J L and Kutz J N 2016 Discovering governing equations from data by sparse identification of nonlinear dynamical systems *Proc. Natl Acad. Sci* **113** 3932–7
[9] Mattheakis M, Protopapas P, Sondak D, Di Giovanni M and Kaxiras E 2019 Physical symmetries embedded in neural networks (arXiv:1904.08991)
[10] Cohen T S and Welling M 2016 Group equivariant convolutional networks (arXiv:0801.4134)
[11] Finzi M, Welling M and Wilson A G 2021 A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups (arXiv:2104.09459)
[12] Raissi M, Perdikaris P and Karniadakis G E 2019 Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations *J. Comput. Phys.* **378** 686–707
[13] Karniadakis G E, Kevrekidis I G, Lu L, Perdikaris P, Wang S and Yang L 2021 Physics-informed machine learning *Nat. Rev. Phys.* **3** 422–40
[14] Kondor R and Trivedi S 2018 On the generalization of equivariance and convolution in neural networks to the action of compact groups (arXiv:2104.09459)
[15] Gabel A, Klein V, Valperga R, Lamb J S W, Webster K and Quax R 2023 Learning Lie group symmetry transformations with neural networks
[16] Olver P J 2012 Applications of Lie groups to differential equations *Graduate Texts in Mathematics* (Springer)
[17] Brandstetter J, Welling M, Worrall D E 2022 Lie point symmetry data augmentation for neural PDE solvers (arXiv:2202.07643)
[18] Moskalev A, Sepliarskaia A, Sosnovik I and Smeulders A 2022 LieGG: studying learned Lie rators (arXiv:2210.04345)
[19] Alcorn M A, Li Q, Gong Z, Wang C, Mai L, Ku W and Nguyen A 2019 Strike (with) a pose: neural networks are easily fooled by strange poses of familiar objects *IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)* pp 4840–9
[20] Oliveri F 2010 Lie symmetries of differential equations: classical results and recent contributions *Symmetry.* **06** 2
[21] Paliathanasis A and Tsamparlis M 2018 Lie symmetries for systems of evolution equations *J. Geom. Phys.* **124** 165–9
[22] Bar-Sinai Y, Hoyer S, Hickey J and Brenner M P 2019 Learning data-driven discretizations for partial differential equations *Proc. Natl Acad. Sci.* **116** 15344–9
[23] Gabel A 2023 PDE_LPS *Zenodo* (https://doi.org/10.5281/zenodo.8311061)
[24] Ibragimov N H 1993 *CRC Handbook of Lie Group Analysis of Differential Equations: Symmetries, Exact Solutions and Conservation Laws* vol 1 (Taylor & Francis)
[25] He K, Zhang X, Ren S and Sun J 2015 Deep residual learning for image recognition *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (*Las Vegas, NV, USA, 2016*) pp 770–8
[26] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L and Gomez A N 2017 Attention is all you need *Neural Information Processing Systems* (*Long Beach, CA, USA, 2017*) vol 30
[27] Brandstetter J, Worrall D and Welling M 2022 Message passing neural PDE solvers *Int. Conf. on Learning Representations (ICLR)* (*February 2022*)