SELF-SUPERVISED LEARNING WITH LIE SYMMETRIES FOR PARTIAL DIFFERENTIAL EQUATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Machine learning for differential equations paves the way for computationally efficient alternatives to numerical solvers, with potentially broad impacts in science and engineering. Though current algorithms typically require simulated training data tailored to a given setting, one may instead wish to learn useful information from heterogeneous sources, or from real dynamical systems observations that are messy or incomplete. In this work, we learn general-purpose representations of PDEs from heterogeneous data by implementing joint embedding methods for self-supervised learning (SSL), a framework for unsupervised representation learning that has had notable success in computer vision. Our representation outperforms baseline approaches to invariant tasks, such as regressing the coefficients of a PDE, while also improving the time-stepping performance of neural solvers. Data augmentation is central to SSL: although simple augmentation strategies such as cropping provide satisfactory results, our inclusion of transformations corresponding to the symmetry group of a given PDE significantly improves the quality of the learned representations.

1 INTRODUCTION

Dynamical systems governed by differential equations are ubiquitous in fluid dynamics, climate science, astrophysics, and beyond. Accurately analyzing and predicting the evolution of such systems is of paramount importance, inspiring decades of innovation in algorithms for numerical methods and inverse problems. However, high-accuracy solvers are often computationally expensive. Machine learning has recently arisen as an alternative method for analyzing differential equations at a fraction of the cost (Raissi et al., 2019; Karniadakis et al., 2021; Rudy et al., 2017). Typically, the neural network for a given equation is trained on simulations of that same equation, generated by numerical solvers that are high-accuracy but comparatively slow (Brunton et al., 2016). What if we instead wish to learn from heterogeneous data, e.g., data with missing information, or gathered from actual observation of varied physical systems rather than clean simulations?

For example, we may have access to a dataset of instances of time-evolution, stemming from a family of partial differential equations (PDEs) for which important characteristics, such as viscosity or reaction-rate constants, vary and are unknown. In this case, representations learned from such a large, "unlabeled" dataset could still prove useful in learning to identify unknown characteristics, given only a small dataset "labeled" with viscosities or reaction constants. Alternatively, the "unlabeled" dataset may contain evolutions over very short periods of time, or with missing time intervals; possible goals are then to learn representations that could be useful in filling in these time-steps, or regressing other quantities of interest.

To tackle these broader challenges, we take inspiration from the recent success of self-supervised learning (SSL) as a tool for learning powerful representations from large, unlabeled datasets of text and images (Radford et al., 2021; Caron et al., 2021). Building such representations from and for scientific data is a natural next step in the development of machine learning for science (Nguyen et al., 2023). In particular, we focus on the joint embedding framework for SSL, a popular paradigm for learning visual representations (Bardes et al., 2021; Grill et al., 2020). Here, an encoder aims to enforce similarity between embeddings of two augmented versions of a given sample guided by the principle that representations suited to downstream tasks, such as classification, should be invariant to the selected augmentations. Thus, data augmentation plays a crucial role in joint embedding

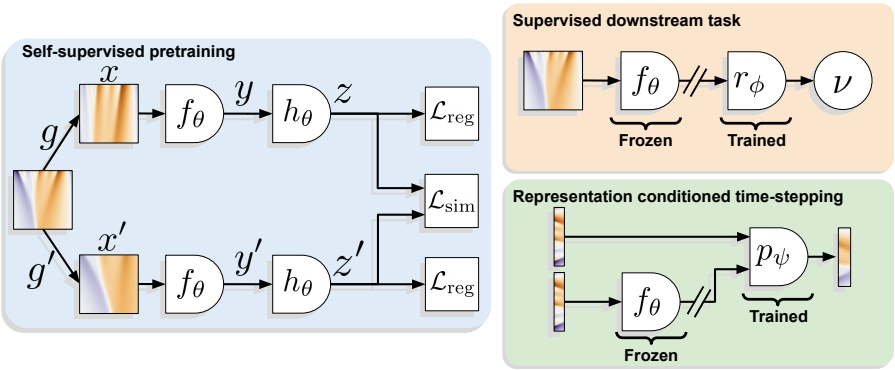

Figure 1: Pretraining and evaluation frameworks. **(Left)** Self-supervised pretraining. We generate augmented solutions x and x' using Lie symmetries parametrized by g and g' before passing them through an encoder f_θ, yielding representations y. The representations are then input to a projection head h_θ, yielding embeddings z, on which the SSL loss is applied. **(Right)** Evaluation protocols for our pretrained representations y. On new data, we use the computed representations to either predict characteristics of interest, or to condition a neural operator to improve time-stepping performance.

SSL. For symmetries of visual representations, SSL relies on human intuition to build hand-crafted augmentations (e.g. recoloring and cropping), which must be carefully tuned. In contrast, PDEs are endowed with a group of symmetries, which preserve the defining equations of the PDE. For example, solutions to certain PDEs with periodic boundary conditions are preserved by translations in time and space, but there exist more elaborate equation-specific transformations as well. Brandstetter et al. (2022) first applied infinitesimal Lie point symmetries to PDE data in a machine learning setting, demonstrating their effectiveness as an augmentation for learning to time-step. Symmetry groups are well-studied for common PDE families (Olver, 1979; Ibragimov, 1995), and in general can be derived using the language of Lie theory.

Contributions We present a general framework for performing SSL for PDEs using their corresponding symmetry groups. In particular, we show that by exploiting the analytic transformations from one PDE solution to another, we can use joint embedding methods to generate useful representations from large, heterogeneous PDE datasets. We demonstrate the broad utility of these representations on downstream tasks, including regressing key parameters and time-stepping, on simulated physically-motivated datasets. Our approach is applicable to any family of PDEs, harnesses the well-understood mathematical structure of the equations governing PDE data — a luxury not typically available in non-scientific domains — and demonstrates more broadly the promise of adapting self-supervision to the physical sciences. We hope this proof-of-concept will serve as a starting point for developing foundation models on more complex dynamical systems with our framework.

2 METHODOLOGY

Our approach combines joint embedding methods for SSL with Lie symmetry augmentations. We expand further on each of these two components below.

Self-Supervised Learning (SSL) The goal of SSL is to learn useful representations from large unlabeled datasets. In the joint-embedding framework, each input is transformed into two separate "views", using augmentations that preserve the underlying information in the data. The augmented views are then fed through a learnable encoder f_θ, producing representations that can be used for downstream tasks, and finally through a projection head h_θ, yielding embeddings on which the SSL loss — a similarity loss $\mathcal{L}_{\text{sim}}$ between the pairs of views, to make their representations invariant to augmentations — is applied. To avoid trivial solutions (such as the same representation for every input), a regularization loss $\mathcal{L}_{\text{reg}}$ is included as well. It can consist of a repulsive force between points, or a regularization on the covariance matrix of the embeddings. We illustrate this process in Fig. 1. In practice, we use VICReg (Bardes et al., 2021) as our SSL criterion, and provide more details on the experimental setup in Appendix C. The learned representation can then be leveraged to

solve various tasks of interest, such as predicting the viscosity coefficient from Burgers' equation in low-data regimes, without training an entirely new network from scratch.

Augmentations and PDE Symmetry Groups In the joint embedding framework, both symmetric and noisy augmentations are crucial to the construction of inputs from which to learn representations. Noise augmentations, such as masking or adding random noise to the input data, force a network to learn to de-noise the inputs, which improves its ability to generalize to new data (Chen et al., 2020a; Doersch et al., 2015; Brown et al., 2020). Such noise also helps enforce useful biases into the network, e.g.,cropping forces the network to learn global representations of the data (Chen et al., 2022). In parallel, symmetry augmentations enforce the invariance of representations to known symmetry groups of the data. For PDEs, these symmetry groups can be analytically derived (Appendix B), and have been categorized for several common PDEs (Olver, 1979). However, implementing these geometric augmentations also entails new difficulties. For example, they may transform a regular discretization of pixels to an irregular one. Overall, appropriately parameterizing and sampling from Lie point symmetries requires care, and we further describe our methodology in Appendix C.

3 Experimental Results

In this section, we empirically validate our framework on systems governed by the Schrödinger equation and Burgers' equation. We view these simple models as an encouraging proof of concept, and an invitation to advance to more challenging dynamical systems.

3.1 Schrödinger Evolution

We consider time evolution of the Schrödinger equation for a single particle in 1-D. Our task is to regress coefficients, c_k, of the time evolution (see Appendix B for more formal details)

$$\frac{\partial}{\partial t}\left|\phi(t)\right\rangle = -iH\left|\phi(t)\right\rangle, \quad H = \sum_{k=1}^{5} c_k \left|\psi_k\right\rangle\left\langle\psi_k\right|, \quad \left|\phi(0)\right\rangle = \frac{1}{Z}\sum_{k=1}^{20} v_k b_k \left|\psi_k\right\rangle, \tag{1}$$

where $c_k, v_k \sim \mathcal{N}(0,1)$ are drawn i.i.d. Gaussian, Z is a normalization constant, and $b_k \sim B(1, 0.5)$ are Bernoulli random variables. ψ_k are orthonormal eigenfunctions of the Mehler kernel (see Appendix C.2), which form the basis for the Hamiltonian (Mehler, 1866; Kibble, 1945). To make the task slightly harder, we perturb the labels for c_k by i.i.d. Gaussian noise with standard deviation 0.25. As explained in Appendix C.2, augmentations correspond to operations which commute with H, including scaling of the basis coefficients $v_k b_k \to \alpha v_k b_k$ for some $\alpha \in \mathbb{C}$.

	n_{pre} :	500		5000	
Method	n_{sup} :	50	500	50	500
SSL + Linear Head (augmentations)		**0.509**	0.323	0.604	0.269
SSL + Linear Head (augmentations + crop)		0.682	0.346	**0.538**	0.323
Baseline (supervised only)		0.970	0.897	0.970	0.897
Baseline (supervised only + augmenting data)		0.702	**0.157**	0.702	**0.157**

Table 1: Mean squared error in regressing the five coefficients of the Schrödinger time evolution (see Eq. (1)). Errors are reported relative to the optimum (a value of 1 is equivalent to the error of constantly outputting 0 for each c_k, and a relative error of 0 implies optimally predicting the coefficients). See Appendix C.2 for further details.

We use the VICReg loss for SSL pretraining with $n_{pre} \in \{500, 5000\}$ unlabeled samples (see Appendix C.1 for more details). The downstream supervised task is provided with labels for $n_{sup} = 500$ (10%) or $n_{sup} = 50$ (1%) of these samples. Results in Table 1 show that the SSL pretraining is quite effective when the ratio of unlabeled to labeled samples is very large. The baseline method only outperforms SSL pretraining when it is provided with augmentations of many labeled samples. One explanation is that in this contrived problem, augmentations are very powerful: they produce exact solutions, which are distributed similarly to the input distribution before augmentations.

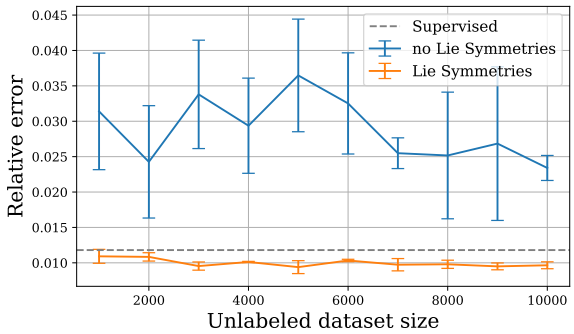

Figure 2: Regression of kinematic viscosity. When using Lie point symmetries (LPS) during pretraining, we are able to improve performance over the supervised baselines, even when using an unlabled dataset size that is half the size of the labeled one. As we increase the amount of unlabeled data that we use, the performance improves, further reinforcing the usefulness of self-supervised representations.

3.2 Burgers' Equation

We now consider the time evolution of Burgers' equation in one dimension, for which background can be found in Appendix B. It is defined as

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{2}$$

in its viscous form. Our goal is to leverage Lie point symmetries and joint embedding SSL to learn useful representations (i.e., to pretrain an encoder neural network) from a large dataset of time evolutions, with different kinematic viscosities, ν, and initial conditions. This encoder will be frozen and produce features, which allow one to infer the kinematic viscosity using a simple linear layer on a new set of realizations of Eq. (2), or to improve time-stepping with neural solvers. Crucially, we leverage the task of regressing ν as an indicator of the quality of the learned representation: we select and use for time-stepping the representation from which the kinematic viscosity can be regressed the most accurately on a validation set. See Appendix C for details.

Kinematic viscosity regression After pretraining an encoder on a large dataset (not annotated with viscosities), we freeze its weights and use the features to train a linear model on a smaller, labeled dataset of only 2000 samples. We compare to a supervised model (encoder and linear head) trained on the same labeled dataset. Figure 2 shows that, while using only a cropping augmentation already provides satisfactory results, using Lie point symmetries matches the supervised performance with relatively little unlabeled data. Performance further improves with the quantity of unlabeled data.

Time-stepping We explore whether the same representation can improve time-stepping neural operators. A neural operator typically uses a sequence of time steps (the "time history") of the equation to predict a future sequence of time steps. We compute a representation of the input time steps using our frozen encoder, and add it as a new channel. This concatenated input is fed to a CNN to predict the next sequence of timesteps, as in Brandstetter et al. (2022). As shown in Table 2, conditioning the CNN operator with our representation slightly reduces the error.

Training samples	2000
CNN baseline	0.13 ± 0.02
CNN (conditioned)	0.12 ± 0.02

Table 2: Normalized mean square error (NMSE) while time-stepping Burgers' equation after 20 training epochs. The NMSEs are averaged across 3 runs.

4 Conclusion

This work leverages Lie point symmetries for self-supervised representation learning from PDE data. Our preliminary experiments with Burgers' equation and the Schrödinger equation demonstrate the usefulness of the resulting representation for sample or compute efficient estimation of characteristics and time-stepping. Following this initial work, many exciting directions remain. Learning *equivariant* representations with SSL could be helpful for time-stepping, perhaps directly in the learned representation space. In future work, we plan to apply our methodology to more challenging tasks and datasets, such as the Navier-Stokes equation. We hope that our approach will provide a practical toolkit for learning from a wide variety of dynamical systems data.

REFERENCES

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.

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.

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.

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

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 (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=ePZsWeGJXyp.

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.

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

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, 2020. ISBN 9781713829546.

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.

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.

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

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.

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*, pp. 9650–9660, 2021.

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

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

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

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

Yubei Chen, Adrien Bardes, Zengyi Li, and Yann LeCun. Intra-instance vicreg: Bag of self-supervised image patch embedding. *arXiv preprint arXiv:2206.08954*, 2022.

Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pp. 1422–1430, 2015.

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

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

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

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.

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.

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

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*, pp. 770–778, 2016.

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*, pp. 9729–9738, 2020.

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.

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

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`.

WF Kibble. An extension of a theorem of mehler's on hermite polynomials. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 41(1), pp. 12–15. Cambridge University Press, 1945.

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), pp. 59–70. Wiley Online Library, 2019.

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

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

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

F Gustav Mehler. Ueber die entwicklung einer function von beliebig vielen variablen nach laplaceschen functionen höherer ordnung., 1866.

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

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

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.

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

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

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.

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.

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.

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.

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.

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

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.

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

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.

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.

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

Eberhard Zeidler. *Applied functional analysis: main principles and their applications*, volume 109. Springer Science & Business Media, 2012.

A Related Work

Machine Learning for PDEs Recent work on machine learning for PDEs has considered both invariant prediction tasks Ramezanizadeh et al. (2019) and time-series modelling Fries et al. (2022); He et al. (2022). In the fluid mechanics setting, models learn dynamic viscosities, fluid densities, and/or pressure fields from both simulation and real-world experimental data Syah et al. (2021); Vinuesa & Brunton (2022); Raissi et al. (2020). 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 Bar-Sinai et al. (2019); Adeli et al. (2022); Wu et al. (2022); Equer et al. (2023). 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 Kim et al. (2019). Recently, for example, Lusch et al. have studied learning the infinite-dimensional Koopman operator to globally linearize latent space dynamics Lusch et al. (2018). 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 Fries et al. (2022). Nguyen et al. have looked at the development of foundation models for climate sciences using transformers pre-trained on well-established climate datasets Nguyen et al. (2023). Other methods like dynamic mode decomposition (DMD) are entirely data-driven, and find the best operator to estimate temporal dynamics Schmid (2010). 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 Baddoo et al. (2021).

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 (Chen et al., 2020a; He et al., 2020; Chen et al., 2020b; 2021; Yeh et al., 2021) push away unrelated pairs of augmented datapoints, and frequently rely on the InfoNCE criterion (Oord et al., 2018), although in some cases, squared similarities between the embeddings have been employed HaoChen et al. (2021). Clustering-based methods have also recently emerged (Caron et al., 2018; 2020; 2021), where instead of contrasting pairs of samples, samples are contrasted with cluster centroids. Non-contrastive methods (Grill et al., 2020; Chen & He, 2020; Bardes et al., 2021; Zbontar et al., 2021; Ermolov et al., 2021; Li et al., 2022; Bardes et al., 2022) 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 (Garrido et al., 2022; Cabannes et al., 2023). 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 Chen et al. (2020a). 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 (Mialon et al., 2022; Bordes et al., 2022).

B Deriving Symmetry Groups

Symmetry augmentations enforce 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 Olver (1979). An example of the form of such augmentations is given in Figure 3 for a simple PDE which rotates a point in 2-D space. In this example, the PDE exhibits both rotational symmetry and scaling symmetry of the radius. For arbitrary PDEs, such symmetries can be derived as explained in more detail below.

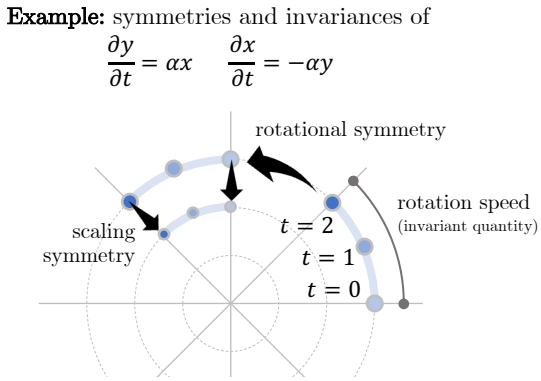

Figure 3: Illustration of the PDE symmetry group and invariances of a simple PDE, which rotates a point in 2-D space. The PDE symmetry group here corresponds to scalings of the radius of the rotation and fixed rotations of all the points over time. A sample invariant quantity is the rate of rotation (related to the parameter α in the PDE) which is fixed for any solution to this PDE.

Symmetries of differential equations carry a Lie group structure and transformations correspond to smooth maps. It is typically easier to derive the symmetries of a system of differential equations via their infinitesimal generators. By using these infinitesimal generators, one can replace nonlinear conditions for the invariance of a function under the group transformation by an equivalent linear condition of infinitesimal invariance under the respective generator of the group action Olver (1979).

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 (1979); Ibragimov (1995) among others. Lie point symmetries of differential equations can be performed from compositions of one parameter transforms, which are commonly used to describe dynamical systems Zeidler (2012). Each one parameter transform corresponds to a given infinitesimal generator in the Lie algebra of the group. To map between a Lie group operation and its corresponding infinitesimal generator, we can take the derivative of the single parameter group operation at the identity:

$$\boldsymbol{v}_g|_{(x,u)} = \frac{d}{dt} g(t) \cdot (x, u) \bigg|_{t=0}, \tag{3}$$

where $g(0) \cdot (x, u) = (x, u)$ and $\boldsymbol{v}_g$ is a vector field in the Lie algebra denoting the flow or infinitesimal change at any given coordinates (x, u).

In our setting, we consider a differential equation that has a set of p independent variables $x = (x^1, x^2, \ldots, x^p)$ and q dependent variables $u = (u^1, u^2, \ldots, u^q)$ which are coordinates in $\mathbb{R}^p$ and $\mathbb{R}^q$, respectively. Any given function has a corresponding graph Γ_f of the function

$$\Gamma_f = \{(x, f(x)) : x \in \Omega\} \subset \mathbb{R}^p \times \mathbb{R}^q. \tag{4}$$

Additionally, $g \in G$ is a group operation that maps solutions of a given PDE at one point to solutions of another PDE at a different set of points. Since all transformations are local and smooth, one can ensure transformations maintain the validity of a solution in some region near the identity of the group.

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

$$\exp(t\boldsymbol{v}) \cdot (x, u) = g(t) \cdot (x, u) \tag{5}$$

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. For example, in the case of matrix Lie groups, the exponential map can be evaluated using its corresponding Taylor series:

$$\exp(\varepsilon M) = \sum_{k=0}^{\infty} \frac{\varepsilon^k M^k}{k!} = I + \varepsilon M + \frac{1}{2}\varepsilon^2 M^2 + \ldots \tag{6}$$

where $\varepsilon \in \mathbb{R}$ and I is the identity.

To derive the infinitesimal generator of a given PDE, we need to define the prolongation of a vector field. The n-th prolongation of a given graph, Γ_f, expands or "prolongs" the graph of a solution to a larger space to include derivatives up to the n-th order. Here, we show how to perform the prolongation of a vector field to this higher-dimensional space, which is often referred to as the jet-space, $M^{(n)}$, where $M^{(n)} \subset \mathbb{R}^p \times U^{(n)}$.

To evaluate the n-th prolongation of the vector field, as previously, assume we have p independent variables $(x^1, \ldots, x^p)$ and q dependent variables $(u^1, \ldots, u^q)$ which are a function of the dependent variables. Note that we use superscripts to denote a particular variable, while derivatives with respect to a given variable are denoted via subscripts corresponding to the indices. For example, the variable $u^1_{x^1x^1x^2}$ denotes the third order derivative of u^1 taken twice with respect to the variable x^1 and once with respect to x^2. Using this notation, the prolongation of a vector field is defined as the operation

$$\mathrm{pr}^{(n)}\, \boldsymbol{v}\big|_{(x,u^{(n)})} = \frac{d}{d\epsilon}\bigg|_{\epsilon=0} \mathrm{pr}^{(n)} \left[\exp(\epsilon \boldsymbol{v})\right](x, u^{(n)}). \tag{7}$$

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

$$\boldsymbol{v} = \sum_{i=1}^{p} \xi^i(x,u)\frac{\partial}{\partial x^i} + \sum_{\alpha=1}^{q} \phi_\alpha(x,u)\frac{\partial}{\partial u^\alpha}. \tag{8}$$

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

$$\mathrm{pr}^{(n)}\, \boldsymbol{v} = \boldsymbol{v} + \sum_{\alpha=1}^{q}\sum_{\boldsymbol{J}} \phi^{\boldsymbol{J}}_\alpha(x, u^{(n)})\frac{\partial}{\partial u^\alpha_{\boldsymbol{J}}}, \tag{9}$$

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

$$\phi^{\boldsymbol{J}}_\alpha(x, u^{(n)}) = \prod_{i\in\boldsymbol{J}} \boldsymbol{D}_i \left(\phi_a - \sum_{i=1}^{p} \xi^i u^\alpha_i\right) + \sum_{i=1}^{p} \xi^i u^\alpha_{\boldsymbol{J},i}, \tag{10}$$

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

$$\boldsymbol{D}_i P(x, u^{(n)}) = \frac{\partial P}{\partial x^i} + \sum_{i=1}^{q}\sum_{\boldsymbol{J}} u^\alpha_{\boldsymbol{J},i}\frac{\partial P}{\partial u^\alpha_{\boldsymbol{J}}}. \tag{11}$$

After evaluating the coefficients, $\phi^{\boldsymbol{J}}_\alpha(x, u^{(n)})$, we can substitute these values into the definition of the vector field's prolongation in Eq. 9. 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.

B.1 Example: Burgers' Equation

Burgers' equation is a PDE used to describe convection-diffusion phenomena commonly observed in fluid mechanics, traffic flow, and acoustics Nee & Duan (1998). The PDE can be written in either its "conservative" form or its "viscous" form. In deriving the symmetry groups for Burgers' equation, it is more convenient to work with the former,

$$u_t = u_{xx} + u_x^2. \tag{12}$$

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

$$\boldsymbol{v} = \xi(x,t,u)\frac{\partial}{\partial x} + \tau(x,t,u)\frac{\partial}{\partial t} + \phi(x,t,u)\frac{\partial}{\partial u}, \tag{13}$$

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 2nd order, given that the highest-degree derivative present in the governing PDE is of order 2. The 2nd prolongation of the vector field can be expressed as

$$\text{pr}^{(2)}\, \boldsymbol{v} = \boldsymbol{v} + \phi^x \frac{\partial}{\partial u_x} + \phi^t \frac{\partial}{\partial u_t} + \phi^{xx} \frac{\partial}{\partial u_{xx}} + \phi^{xt} \frac{\partial}{\partial u_{xt}} + \phi^{tt} \frac{\partial}{\partial u_{tt}}. \tag{14}$$

Applying this prolonged vector field to the differential equation in Eq. 12, we get the infinitesimal symmetry criteria that

$$\text{pr}^{(2)}\, \boldsymbol{v}[\Delta(x, t, u^{(2)})] = \phi^t - \phi^{xx} + 2u_x \phi^x = 0. \tag{15}$$

To evaluate the individual coefficients, we apply Eq. 10. 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 3 below lists the relevant monomials as well as their respective coefficients.

Table 3: 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:

$$\xi(t, x) = k_1 + k_4 x + 2k_5 t + 4k_6 xt \tag{16}$$

$$\tau(t) = k_2 + 2k_4 t + 4k_6 t^2 \tag{17}$$

$$\phi(t, x, u) = (k_3 - k_5 x - 2k_6 t - k_6 x^2)u + \gamma(x, t) \tag{18}$$

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:

$$\boldsymbol{v}_1 = \partial_x \tag{19}$$

$$\boldsymbol{v}_2 = \partial_t \tag{20}$$

$$\boldsymbol{v}_3 = \partial_u \tag{21}$$

$$\boldsymbol{v}_4 = x\partial_x + 2t\partial_t \tag{22}$$

$$\boldsymbol{v}_5 = 2t\partial_x - x\partial_u \tag{23}$$

$$\boldsymbol{v}_6 = 4xt\partial_x + 4t^2\partial_t - (x^2 + 2t)\partial_u \tag{24}$$

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 Hopf–Cole transformation is recovered.

C EXPERIMENTAL DETAILS

C.1 VICREG LOSS

Here, we discuss the loss function used to train our models, VICReg (Bardes et al., 2021). We define our embedding matrices as $Z, Z' \in \mathbb{R}^{N \times D}$. Next, we define its similarity criterion, $\mathcal{L}_{\text{sim}}$, as

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

which we use to match our embeddings and make them invariant to the transformations. In order to avoid a collapse of the representations, we use the original variance and covariance criteria to define our regularisation loss, $\mathcal{L}_{\text{reg}}$, as

$$\begin{aligned} \mathcal{L}_{\text{reg}}(Z) &= \lambda_C \; C(Z) + \lambda_V \; V(Z), \quad \text{with} \\ C(Z) &= \frac{1}{d} \sum_{i \neq j} \text{Cov}(Z)_{i,j}^2 \quad \text{and} \\ V(Z) &= \frac{1}{d} \sum_{j=1}^{d} \max\left(0, 1 - \sqrt{\text{Var}(Z_{\cdot,j})}\right). \end{aligned}$$

The variance criterion, $V(Z)$, 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

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

For all studies conducted in this work, we use the default values of $\lambda_C = \lambda_{\text{inv}} = 25$ and $\lambda_V = 1$, unless specified.

C.2 EXPERIMENTS ON QUANTUM MECHANICS

Inputs to this task are the wavefunction of randomly drawn solutions, $|\phi(t)\rangle$, drawn at 8-time intervals from $t = 0$ to $t = 0.7$. Input states span over the top eigenfunctions of the Mehler kernel, visually shown in Figure 4. This eigenbasis takes the form

$$\psi_k(x) = \frac{H_k(x) \exp(-x^2/2)}{\sqrt{2^k k! \sqrt{\pi}}}, \tag{25}$$

where $H_k(x)$ are the Hermite polynomials. These are eigenfunctions of the quantum harmonic oscillator operator, D_x, defined as (Kibble, 1945)

$$D_x \phi = \frac{\partial^2 \phi}{\partial x^2} - x^2 \phi. \tag{26}$$

In this study, we use bra-ket notation to indicate the possible solutions of the quantum toy problem. For complex-valued functions, $\phi \in L^2(X)$, for some domain X, $|\phi\rangle$ embeds the function in a Hilbert space. The bra-ket notation allows access to an inner product of the complex Hilbert space with the notation

$$\langle \phi | \psi \rangle = \int \overline{\phi(x)} \psi(x) dx. \tag{27}$$

For any solution $|\phi(t)\rangle$, we divide by the norm of the coefficients $Z = \sqrt{\sum_{k=1}^{20} (v_k b_k)^2}$ to ensure all input states have norm 1. Since input states are in the same basis as the Hamiltonian, time evolution takes the simple form

$$|\phi(t)\rangle = \exp(-iHt) \, |\phi(0)\rangle = \frac{1}{Z} \sum_{k=1}^{20} v_k b_k \exp(-ic_k t) \, |\psi_k\rangle \, . \tag{28}$$

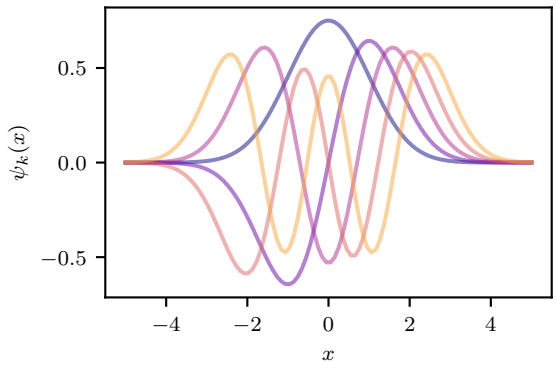

Figure 4: Top 5 eigenfunctions in the Mehler basis shown here for wavefunctions in the quantum mechanics model.

At each time interval, we sample the values of $|\phi(t)\rangle$ at 100 uniform locations equally-spaced between $x = -5$ and $x = 5$. Therefore, inputs have shape 8×100. As the above is a linear equation, augmentations correspond to operations which commute with H. Namely, any constant scaling of the coefficients $v_k b_k \to \alpha v_k b_k$ for some $\alpha \in \mathbb{C}$ will also be a valid solution of the time evolution.

$$|\phi(t)\rangle = \frac{1}{Z}\sum_{k=1}^{20} v_k b_k \exp(-ic_k t)\,|\psi_k\rangle \;\to\; |\phi'(t)\rangle = \frac{1}{Z'}\sum_{k=1}^{20} \alpha_k v_k b_k \exp(-ic_k t)\,|\psi_k\rangle\,, \tag{29}$$

where $|\phi'(t)\rangle$ is also a valid solution, and $\alpha_k \in \mathbb{C}$ is a scaling with random phase and norm drawn i.i.d. from a standard Gaussian distribution. Augmentations are applied by estimating the overlap of a given solution in the basis of $|\psi_k\rangle$ and applying the phases. In addition to the augmentation above, we also consider crops, which crop the data in the time dimension by taking a slice of size 6 across the time dimension.

We train both the SSL and baseline task with a convolutional network network consisting of the following layers:

```
  ConvNet(
 (conv1): Conv1d(8, 8, kernel_size=(5,), stride=(1,), padding=(1,))
 (conv2): Conv1d(8, 16, kernel_size=(5,), stride=(1,), padding=(1,))
 (pool1): AvgPool1d(kernel_size=(4,), stride=(2,), padding=(0,))
 (conv3): Conv1d(16, 32, kernel_size=(5,), stride=(1,), padding=(1,))
 (conv4): Conv1d(32, 32, kernel_size=(5,), stride=(1,), padding=(1,))
 (pool2): MaxPool1d(kernel_size=8, stride=2, padding=0, dilation=1)
 (fc1): Linear(in_features=576, out_features=256, bias=True)
 (fc2): Linear(in_features=256, out_features=128, bias=True)
)
```

For the baseline supervised network, the last layer only has five output features, corresponding to the five coefficients to be regressed. The ReLU nonlinearity is used between hidden layers.

We use the VICReg loss (Bardes et al., 2021) with a projector that takes the representation dimension of either dimension 128 or 256, and projects it into a space of dimension 256, 512, or 1024 with a single trainable linear layer (we found this choice to be the best as opposed to using a feedforward network). We used either a batch size of 32 or 256 in the VICReg loss. Representations obtained via SSL are trained with a linear head on the downstream regression task for up to 2000 epochs. Supervised data used in this downstream task is augmented in the same fashion as in the SSL task. We use the AdamW optimizer with default Pytorch hyperparameters. In general, we found that the performance was highly dependent on the projector and the representation dimension. We performed hyperparameter tuning on these two parameters along with the batch size. Performance was noticeably variable during this tuning procedure, and the best performance over hyperparameter options (with respect to test error on the supervised task) is indicated in the table for each method.

C.3 EXPERIMENTS ON BURGERS

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 Brandstetter et al. (2022). We generate solutions of size 224×448 in the spatial and temporal dimensions respectively, using the default parameters from Brandstetter et al. (2022). We train a ResNet18 (He et al., 2016) encoder using the VICReg (Bardes et al., 2021) approach to joint embedding SSL, with a smaller projector (width 512) since we use a smaller ResNet than in the original paper. We use the following augmentations:

- Crop of size $(128, 256)$, respectively in the spatial and temporal dimension.
- Uniform sampling in $[-2, 2]$ for the coefficient associated to v_1.
- Uniform sampling in $[0, 2]$ for the coefficient associated to v_2.
- Uniform sampling in $[-1000, 1000]$ for the coefficient associated to v_3.
- Uniform sampling in $[-1, 1]$ for the coefficient associated to v_4.
- Uniform sampling in $[-0.1, 0.1]$ for the coefficient associated to v_5.

We deliberately ignore symmetries that rely on the viscosity of the solution (v_γ and v_6) to avoid any leakage with our downstream tasks. We pretrain for 100 epochs using AdamW (Loshchilov & Hutter, 2017) 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 Brandstetter et al. (2022) 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 $T_p = 40$, $T_f = 20$, and $n_{\text{samples}} = 2000$. We train both models for 20 epochs following the setup from Brandstetter et al. (2022). In addition, we use Adam with a decaying learning rate and different configurations of 3 runs each:

- Batch size $\in [16, 256]$.
- Learning rate $\in [0.001, 0.0001]$.