WARM-STARTING PDE SOLVERS WITH ANY-DIMENSIONAL MACHINE LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Any-dimensional machine learning models, such as graph neural networks (GNNs), can be naturally trained and evaluated on inputs of different sizes and dimensions. Inspired by the GNN transferability literature, we show mathematical conditions under which a partial differential equation (PDE) learning-based solver can be trained in small dimensions and directly applied to solve a higher dimensional PDE in a zero-shot fashion. These conditions are based on symmetries in both the partial differential equation and the initial data. When the equations satisfy the symmetries but the data does not, which is the case for many PDEs arising from physics, we show that our theory gives a principled way of warm-starting low-dimensional PDE solvers for higher dimensional PDEs. We apply this method on the heat equation, Burgers' equation, and the compressible Navier–Stokes equations, improving the performance in both zero-shot and typical training regimes on high dimensional data. For example, we train a surrogate model on 2D Navier–Stokes data and achieve better results on 3D test data than a baseline surrogate model trained on 3D data, while only using 12% of the flops and 20% of the total data size.

1 INTRODUCTION

In machine learning practice, it is common to use models that are defined on a fixed set of parameters and can be evaluated on inputs of arbitrary size or dimension. These were formalized as *any-dimensional machine learning models* in Levin (2026). They are often defined on objects modulo certain symmetries, and can have uniform expressivity across input sizes or dimensions due to a phenomenon known as representation stability (Church & Farb, 2013; Yao et al., 2026). For example, graph neural networks operate on graphs of arbitrary size and structure, and transformer models can handle variable-length sequences, reusing the same weights across positions, where the positional encoding adapts to length. There is a vast literature that explains the mechanism that allows any-dimensional models to generalize across input sizes or dimensions. This started with the study of transferability or size generalization of graph neural networks (Ruiz et al., 2021; Levie et al., 2021; Levie, 2024; Velasco et al., 2024; Maskey et al., 2023), and it was recently generalized beyond graphs (Levin et al., 2025; Levin & Chandrasekaran, 2026).

In the literature on machine learning for PDEs, neural operators were designed with the purpose of generalizing across resolutions (Lu et al., 2021; Li et al., 2020; Karniadakis et al., 2021), but there is not much theory explaining if, how, or why a trained PDE solver can generalize to different spatial domains. The closest setting to the one considered here aggregates data from different dimensions and/or coordinate systems for PDE identification (Phan et al., 2025; Psarellis et al., 2024). Some empirical success in generalizing across dimensions has been made in the realm of foundational models for physics. For example, the Walrus model (McCabe et al., 2025) approaches the problem by embedding low dimensional data in a higher dimensional space, and treating all inputs as higher dimensional. However, this approach introduces a computational hurdle. Accurate PDE simulations often require data sets with high resolution (Li et al., 2008; Yeung et al., 2025), which is a challenge when the images scale as $\mathcal{O}(N^n)$ with side length N and dimension n. Ideally, we would like a machine learning model that exploits the efficiency benefits of a low dimensional space, and could be adapted to higher dimensions.

In this work, we use the any-dimensional machine learning framework to identify mathematical conditions under which PDE solvers can exactly generalize across spatial dimensions. Using the intuition gained from this exercise, we develop a method that also applies when these conditions do not hold: namely, using a pretrained PDE solver in low dimensions to warm-start the learning of a

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

PDE solver in higher dimensions. We empirically show that this method provides a computational advantage in more general settings.

This work is inspired by the transferability framework for graph neural networks, and more generally the any-dimensional learning and representation stability literature (see Appendix A for a brief literature review). This framework relies on some form of symmetry: the function class must be constrained so it does not grow without bounds as the input grows. The main assumption we make is that the PDE is symmetric with respect to spatial Euclidean transformations. For physics applications, Euclidean symmetry is a natural assumption because the fundamental laws of physics are homogeneous and isotropic. Therefore, many PDEs arising from physics satisfy Euclidean symmetries (e.g., Navier–Stokes, the Boltzmann equation, the heat equation, etc.). Note that for our warm-starting procedure, only the equation needs to fulfill a symmetry assumption—not the initial data, boundary conditions, or domain.

We remark that the mathematical framework would allow us to potentially replace the Euclidean symmetry assumptions with other symmetries (coordinate permutation, Poincaré symmetry, or affine symplectic symmetries, for example) with additional work. However, some form of assumption constraining the function class will always be required to be able to generalize across dimensions.

Our contributions

(a) We develop a framework in which a PDE can be defined across dimensions if it is expressed in terms of orthogonally invariant differential operators on jet space. Examples of such operators include the Laplacian and the Dirichlet energy density.

(b) We explicitly describe conditions under which the solution of a PDE in $\mathbb{R}^n$ expressed in terms of $\mathrm{O}(n)$-invariant differential operators (as in (a)) can be lifted to a solution of a corresponding PDE in $\mathbb{R}^{n+1}$.

(c) We propose a mathematically principled method to lift learning-based PDE solvers from dimension n to dimension $n+1$. Under the assumptions of (b), we show that it maps PDE solutions in n dimensions to PDE solutions in $n+1$ dimensions. When the assumptions of (b) are not satisfied, we develop the Nepo warm-start method to provide an advantageous starting point for $(n+1)$-dimensional solvers.

(d) We demonstrate the Nepo warm-start approach with the heat equation, Burgers' equation, and the compressible Navier–Stokes equations. The warm-start allows, for example, a Navier–Stokes model trained only on 2D data to outperform one trained on 3D data while using 12% of the flops and 20% of the total data size.

2 Any-dimensional PDEs

Consider a partial differential equation

$$\frac{\partial u}{\partial t} = D\left(x_1, \dots, x_n, u, \frac{\partial u}{\partial x_1}, \dots, \frac{\partial u}{\partial x_n}, \dots\right) \tag{1}$$

characterizing the time evolution of some smooth function $u : \mathbb{R}^n \times [0, \infty) \to \mathbb{R}$. We denote $u_t : \mathbb{R}^n \to \mathbb{R}$ a point in the time evolution, but we sometimes drop the subscript t for notational convenience (so that u can also denote just a smooth function $\mathbb{R}^n \to \mathbb{R}$).

A priori, it is necessary to pin down n before it is even possible to write down such an equation: it controls the inputs that appear on the right side. On the other hand, the starting point for our investigation is the observation that certain important partial differential equations are written in a language that is meaningful without committing to some specific n. For example, the heat equation has the form

$$\frac{\partial u}{\partial t} = \Delta u, \tag{2}$$

and the Laplacian operator Δ is defined on (adequately differentiable) functions on $\mathbb{R}^n$ for any n. We begin by seeking a general conceptual framework into which such examples fit.

If the equation has partial derivatives of order at most k, then the function D defining the partial differential equation is defined on the kth-order jet space

$$J^k(\mathbb{R}^n) := \left\{\left(x_1, \dots, x_n, u, \frac{\partial u}{\partial x_1}, \dots, \frac{\partial u}{\partial x_n}, \frac{\partial^2 u}{\partial x_1^2}, \frac{\partial^2 u}{\partial x_1 \partial x_2}, \dots, \frac{\partial^k u}{\partial x_n^k}\right)\right\} = \mathbb{R}^n \times U_{n,k} \tag{3}$$

where $\mathbb{R}^n$ is the space domain of the partial differential equation, with coordinates $x_1, \ldots, x_n$, and $U_{n,k} := \mathbb{R}^{N_{n,k}}$ has coordinates corresponding to the function u and all its partial derivatives up to kth order, of which together there are $N_{n,k} := \binom{n+k}{k}$ in total. The space $J^k(\mathbb{R}^n)$ is a (trivial) fiber bundle (the *jet bundle*) with base $\mathbb{R}^n$ and fiber $U_{n,k}$.

We want to consider differential operators that, like the Laplacian, are defined in every dimension. To do so, we consider an n-indexed sequence of functions $D_n : J^k(\mathbb{R}^n) \to \mathbb{R}$, giving us an n-indexed family of partial differential equations

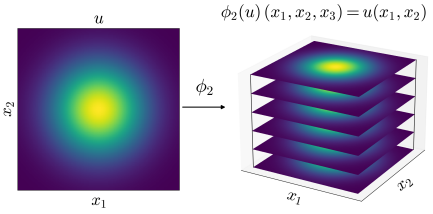
$$\frac{\partial u}{\partial t} = D_n\left(\mathbf{x}, u, \frac{\partial u}{\partial \mathbf{x}}, \ldots, \frac{\partial^k u}{\partial \mathbf{x}^k}\right) \tag{4}$$

(where $\mathbf{x}$ abbreviates $x_1, \ldots, x_n$ and $\partial^j u / \partial \mathbf{x}^j$ abbreviates the collection of partial derivatives of order j). We will define a condition (the *compatibility condition*) under which we may regard the D_n as "the same".

Definition 2.1. *Let $\pi_n : \mathbb{R}^{n+1} \to \mathbb{R}^n$ be the projection $(x_1, \ldots, x_n, x_{n+1}) \mapsto (x_1, \ldots, x_n)$ to the first n coordinates. Given a smooth function $u \in C^\infty(\mathbb{R}^n)$, we can pull it back along π_n, yielding a function $\phi_n(u) \in C^\infty(\mathbb{R}^{n+1})$:*

$$\phi_n(u) := u \circ \pi_n. \tag{5}$$

This defines an embedding $\phi_n : C^\infty(\mathbb{R}^n) \to C^\infty(\mathbb{R}^{n+1})$.



We may think of the embedding ϕ_n as a 'duplication' embedding on smooth functions, as visualized in Definition 2.1, where we extend a 2D scalar field as a constant in the new coordinate.

Given a smooth function $u \in C^\infty(\mathbb{R}^n)$, we can construct a section of the jet bundle $J^k(\mathbb{R}^n)$, i.e. a function in $\mathbb{C}^\infty(\mathbb{R}^n, U_{n,k})$ that sends $\mathbf{x} \in \mathbb{R}^n$ to the evaluations of u and all its partial derivatives up to the kth order, at $\mathbf{x}$. This section is called the kth *prolongation* of u, and we will denote it s_u. The embedding ϕ_n just defined can be extended to an induced embedding, which we also call ϕ_n, from the set $C^\infty(\mathbb{R}^n, U_{n,k})$ of smooth sections of $J^k(\mathbb{R}^n)$ into the set $C^\infty(\mathbb{R}^{n+1}, U_{n+1,k})$ of smooth sections of $J^k(\mathbb{R}^{n+1})$, by extending the effect of ϕ_n to prolongations. By construction, the pulled-back function $\phi_n(u)$ does not depend on the last coordinate x_{n+1}, so all partial derivatives involving x_{n+1} vanish, while the values of all partial derivatives involving only $x_1, \ldots, x_n$ coincide with the corresponding partial derivative of u evaluated at $(x_1, \ldots, x_n)$. Therefore we make the following definition:

Definition 2.2. *We define an embedding of the fiber of $J^k(\mathbb{R}^n)$*

$$\tilde{\phi}_n : U_{n,k} \to U_{n+1,k} \tag{6}$$

by zero-padding in all the coordinates corresponding to a partial derivative involving x_{n+1}. We define an embedding of sections of $J^k(\mathbb{R}^n)$ in sections of $J^k(\mathbb{R}^{n+1})$ by

$$\phi_n : C^\infty(\mathbb{R}^n, U_{n,k}) \to C^\infty(\mathbb{R}^{n+1}, U_{n+1,k}) \tag{7}$$

$$s_u \mapsto (\mathbf{x} \mapsto \tilde{\phi}_n(s_u(\pi_n(\mathbf{x})))). \tag{8}$$

Note that the ϕ_n in (8) specializes to the ϕ_n in (5) when $k = 0$, so the notation is justified. An n-indexed family of functions $D_n : J^k(\mathbb{R}^n) \to \mathbb{R}$, defining a sequence of differential equations as in (4), also induces an n-indexed family of maps

$$\widehat{D}_n : C^\infty(\mathbb{R}^n, U_{n,k}) \to C^\infty(\mathbb{R}^n), \tag{9}$$

$$\widehat{D}_n(s_u)(\mathbf{x}) := D_n(\mathbf{x}, s_u(\mathbf{x})). \tag{10}$$

We are interested in establishing a theory under which the following diagram commutes:

$$\begin{array}{ccccccccc}
\xhookrightarrow{\cdots} & C^\infty(\mathbb{R}^n, U_{n,k}) & \xhookrightarrow{\phi_n} & C^\infty(\mathbb{R}^{n+1}, U_{n+1,k}) & \xhookrightarrow{\phi_{n+1}} & C^\infty(\mathbb{R}^{n+2}, U_{n+2,k}) & \xhookrightarrow{\cdots} \\
& \downarrow \widehat{D}_n & & \downarrow \widehat{D}_{n+1} & & \downarrow \widehat{D}_{n+2} & \\
\xhookrightarrow{\cdots} & C^\infty(\mathbb{R}_n) & \xhookrightarrow{\phi_n} & C^\infty(\mathbb{R}^{n+1}) & \xhookrightarrow{\phi_{n+1}} & C^\infty(\mathbb{R}^{n+2}) & \xhookrightarrow{\cdots}
\end{array} \tag{11}$$

This is a bare minimum requirement if we hope to think of the D_n as being "the same differential equation in different dimensions". But it is not enough by itself. This diagram depends on the choice of the particular orthogonal projection $\pi_n : \mathbb{R}^{n+1} \to \mathbb{R}^n$ mentioned in Definition 2.1; composing either the domain or codomain with an isometry would lead to a different projection $\pi'_n : \mathbb{R}^{n+1} \to \mathbb{R}^n$, which would in turn produce a different embedding $\phi'_n : C^\infty(\mathbb{R}^n) \hookrightarrow C^\infty(\mathbb{R}^{n+1})$ defined by pullback along π'_n, and different embeddings $\tilde{\phi}'_n : U_{n,k} \hookrightarrow U_{n+1,k}$ and $\phi'_n : C^\infty(\mathbb{R}^n, U_{n,k}) \hookrightarrow C^\infty(\mathbb{R}^{n+1}, U_{n+1,k})$ defined by the effect of this new pullback on prolongations. Therefore the sense in which the D_n's are "the same" is not forced to depend on a choice of coordinates, we ask that (11) commute for these alternative ϕ'_n's as well.

We give a sufficient condition on the D_n that guarantees this coordinate-free strengthening of (11):

Proposition 2.3. *If a family of functions $D_n : J^k(\mathbb{R}^n) \to \mathbb{R}$ makes* (11) *commute as written (i.e., with the family of embeddings ϕ_n as above), and the D_n do not depend on $\mathbf{x} \in \mathbb{R}^n$ and thus are functions only of $U_{n,k}$, and if furthermore they are invariant with respect to the natural action of the orthogonal group $\mathrm{O}(n)$ on $U_{n,k}$, then* (11) *would also commute with any choice of ϕ'_n as above in place of ϕ_n.*

The hypothesis of Proposition 2.3 will be called the *compatibility condition*. The proof is in Appendix B. The action of $\mathrm{O}(n)$ on $U_{n,k}$ mentioned in Proposition 2.3 is isomorphically its action on

$$\mathbb{R} \oplus \mathbb{R}^n \oplus \mathrm{Sym}^2(\mathbb{R}^n) \oplus \cdots \oplus \mathrm{Sym}^k(\mathbb{R}^n), \tag{12}$$

where $\mathrm{Sym}^j(\mathbb{R}^n)$ is the set of symmetric tensors of order j—this is because the collection of partial derivatives of order j forms a symmetric tensor.

Remark 2.1. The compatibility condition given here is the point at which a symmetry requirement enters our theory. The proposition says that this condition guarantees that the equations D_n satisfy (11) without being committed to a particular choice of origin and orthonormal coordinates in each $\mathbb{R}^n$. In this sense, it is a natural condition in the context of equations modeling physical phenomena, since the laws of physics have no such commitment.

Remark 2.2. The compatibility condition only imposes a symmetry constraint on the differential equation itself, not the initial data or the boundary conditions.

The compatibility condition gives us the general framework that the heat equation illustrates: any D_n satisfying it can meaningfully be applied to $J^k(\mathbb{R}^n)$ for all sufficiently high n. In fact, if one fixes the degree and order of the PDE, then there is a fixed, finite basis of differential operators, independent of n, in which the D_n's can be expressed uniformly in n. The Laplacian Δu (order 2, degree 1) and the Dirichlet energy density $\|\nabla u\|^2$ (order 1, degree 2) are representative examples of such operators (see Example C.1 for more examples). This can be understood in terms of *representation stability* for the orthogonal group, as explained in Appendix C.

3 Any-dimensional solution operators

We now address the question of compatibility for solution operators of differential equations. As one might expect, even if a PDE is compatible in the sense of Proposition 2.3, the associated solution operator is generically not. Nevertheless, there are families of PDEs whose solution operator does have a simple parametric dependence on the dimension of the space they are defined on; in such cases, one may hope that a lower-dimensional solution operator can be 'lifted' to a higher dimensional space. For example, the heat equation $\frac{\partial u}{\partial t} = \Delta u$ on $\mathbb{R}^n$ satisfies

$$u(x,t) = \frac{1}{(4\pi t)^{n/2}} \int_{\mathbb{R}^n} \exp\Big(-\|x-y\|_2^2/4t\Big) u_0(y) dy \,, \tag{13}$$

with the dimension dependence appearing purely as a scaling factor outside the integral. If such a straightforward dependence on dimension (as a parameter) is known, it would in principle allow us to transfer the solution operator between $\mathbb{R}^n$ subsets for varying dimension n. However, even if a PDE admits a solution operator in the form of a convolution, the dimension-dependent terms rarely factor outside the domain-dependent integral, and direct transferability cannot be expected. Instead, we consider a weaker equivalence framework outlined below. All the proofs of the statements of this section are in Appendix E.1.

Definition 3.1. If D_n satisfies the compatibility condition of Proposition 2.3, then we say that $F_n(u) = \frac{\partial u}{\partial t} - \widehat{D}_n(s_u)$ is a *compatible scalar PDE*.

Definition 3.2. Let $F_n(u) = 0$ be a scalar PDE defined on a set $U \subseteq \mathbb{R}^n$. The associated solution operator $\mathcal{S}^t_{F_n} : (u_0) \mapsto (u_t)$ is the mapping from initial conditions to the solution of the equation at time t.

We then have the following result:

Lemma 3.1 (Operator Symmetry)**.** *Suppose $F_n(u) = 0$ is a compatible scalar PDE with associated solution operator $\mathcal{S}^t_{F_n}(u_0) \mapsto u_t$. If there is a uniqueness theorem for the equation $F_n = 0$, then $\mathcal{S}^t_{F_n}$ is $\mathrm{E}(n)$-equivariant, i.e. for all initial conditions u_0 and $g \in \mathrm{E}(n)$,*

$$\mathcal{S}^t_{F_n}(g \cdot u_0) = g \cdot \mathcal{S}^t_{F_n}(u_0)\,. \tag{14}$$

Next, we note that any solution u satisfying $F_n(u) = 0$ in $\mathbb{R}^n$ can be embedded to the solution space of F_{n+1} in $\mathbb{R}^{n+1}$ as follows:

Lemma 3.2 (Solution Embeddings)**.** *For any $u \in \mathbb{C}^\infty(\mathbb{R}^n \times [0, \infty))$ satisfying the compatible scalar PDE $F_n(u) = 0$ on $\mathbb{R}^n$, we have that $F_{n+1}(\phi_n(u)) = 0$ on $\mathbb{R}^{n+1}$. Furthermore, the same statement holds with ϕ_n replaced by arbitrary ϕ'_n as in Section 2.*

For example, If $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x_1^2}$ on $\mathbb{R}$, the embedded $\phi_n(u)$ satisfies $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}$ on $\mathbb{R}^2$, since the last term $\frac{\partial^2 u}{\partial x_2^2} = 0$ is identically zero.

The previous two technical lemmas can be used to show that, under this 'duplication' embedding framework, the solution operators for $\mathcal{S}_F$ for n and $n+1$ dimensions 'agree', namely:

Proposition 3.3. *Let F_n be a compatible scalar PDE. Let u_0 be any initial conditions on $U \subseteq \mathbb{R}^n$, and let ϕ_n be the embedding from Definition 2.1. Then, $\mathcal{S}^t_{F_{n+1}}(\phi_n(u_0)) = \phi_n(\mathcal{S}^t_{F_n}(u_0))$. Furthermore, if there is a uniqueness theorem for $F_n = 0$, then the same statement holds with ϕ_n replaced by arbitrary ϕ'_n as in Section 2.*

Remark 3.1 (Perturbations)**.** If a solution operator satisfies a Lipschitz condition, it is possible to control the error between solutions arising from exact- and slightly perturbed compatible embeddings. See Appendix E.

Remark 3.2 (Vector-valued PDEs)**.** A similar proof strategy could be followed for vector-valued evolution PDEs, where the 'lift' and 'projection' operations apply to each vector individually. In that case u is vector-valued, and the right-hand-side D_n of the PDE is equivariant with respect to $\mathrm{O}(n)$.

Thus, for $(n+1)$-dimensional initial conditions that are a ϕ_n embedding of n-dimensional initial conditions, the solution operator $\mathcal{S}^t_{F_{n+1}}$ of a higher-dimensional problem can be identified with that of a lower one $\mathcal{S}^t_{F_n}$, after an appropriate projection π_n. This *does not* give us a strong equivalence between operators in different dimensions. Instead, it suggests that the parametrization for a lower solution operator can be transferred to a higher-dimensional space *while agreeing with the true high dimensional solution operator* for a subset of initial data. Thus, it provides a possible warm-start strategy for learning high-dimensional operators, which can be 'initialized' by training in lower dimensions. We discuss this approach theoretically and implement it computationally in the following sections.

4 Lifting PDE solvers to higher dimensions

In this section, we extend the ideas for transferring PDE solution operators to transferring machine learning models that emulate solution operators. In place of continuous fields, the dynamics are modeled in a grid where each pixel can have scalar, (pseudo-)vector, and (pseudo-)tensor features that discretize the corresponding fields. To this end, we consider the models from Gregory et al. (2025), which extend equivariant convolutional neural networks to vector and tensor images.

4.1 Geometric convolutional neural networks

We first define the space of (pseudo-)vectors and tensors, which will correspond to the features at each pixel (Definition 4.1). We then define the tensor images (Definition 4.2).

Definition 4.1 ($k_{(p)}$-tensors)**.** A $1_{(p)}$*-tensor* is an element of $\mathbb{R}^n$ equipped with the action of $\mathrm{O}(n)$

$$g \cdot v = \det(M(g))^{\frac{1-p}{2}}\, M(g)\, v, \tag{15}$$

where the parity p is either $+1$ for vectors or -1 for pseudovectors. Given k $1_{(p_i)}$-tensors denoted v_i, then $T := v_1 \otimes \ldots \otimes v_k$ is a *rank-1 $k_{(p)}$-tensor*, where $p = \prod_{i=1}^k p_i$ and the action of $\mathrm{O}(n)$ is

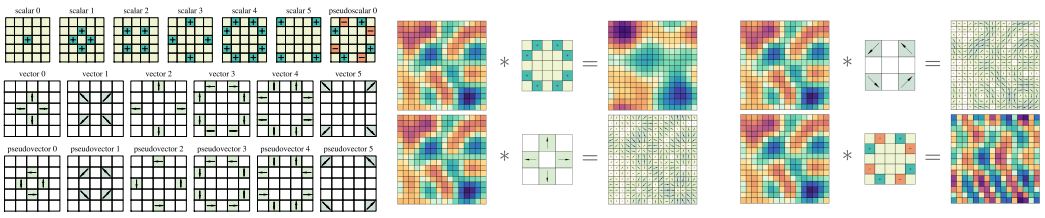


Figure 1: (Left) The 5x5 scalar, pseudoscalar, vector, and pseudovector symmetric filters for 2-D images. (Right) Convolution of a scalar image with different geometric filters. The convolution with a scalar filter can be seen as the discretization of a diffusion operator, and the convolution with the vector filter approximates a discretization of the gradient. In general any symmetric discretization of any coordinate-free operator (gradient, divergence, curl) could be expressed in terms of these symmetric filters.

defined as

$$g \cdot (v_1 \otimes \ldots \otimes v_k) = (g \cdot v_1) \otimes \ldots \otimes (g \cdot v_k)\,. \tag{16}$$

Thus a tensor T is an element of a vector space $(\mathbb{R}^n)^{\otimes k}$, which we denote $\mathcal{T}_k(\mathbb{R}^n, p)$. To get higher rank tensors, we can add tensors of the same order k and parity p, and the action of $\mathrm{O}(n)$ extends linearly.

To get a discretized version of the scalar, vector, and tensor fields in the previous sections, we construct a tensor image, a grid or lattice of tensors.

Definition 4.2 ($k_{(p)}$-tensor image)**.** Consider images supported in $(\mathbb{Z}/N\mathbb{Z})^n$, the n-dimensional discrete torus. Then a $k_{(p)}$*-tensor image* is a function $A : (\mathbb{Z}/N\mathbb{Z})^n \to \mathcal{T}_k(\mathbb{R}^n, p)$. The vector space of $k_{(p)}$-tensor images with side lengths N is denoted $\mathcal{A}_{N,n,k,p}$.

Instead of being equipped with the action of the continuous $\mathrm{E}(n)$, $\mathcal{A}_{N,n,k,p}$ is equipped with the action of the discrete group G_n, the semidirect product of discrete translations $(\mathbb{Z}/N\mathbb{Z})^n$ and B_n, the group of rotations of 90 degrees and reflections. Using tensor images, Gregory et al. (2025) defines geometric convolutional neural networks analogous to standard convolutional networks. The geometric convolution, denoted $A * C$ for input image A and filter C, is the same as standard image convolution except that the image-filter pixel multiplication is implemented as a tensor product, followed by a tensor contraction. These geometric convolutions characterize linear G_d-equivariant functions, and hence preserve the output as a tensor image, if the filters are symmetric with respect B_d. See Appendix D for the mathematical details, and see Fig. 1 for an illustration. Further layers such as point-wise nonlinearities and max pooling can be defined for tensor images (Appendix D).

4.2 Embedding of tensor images

It is straightforward to extend the projection π_n and embedding ϕ_n to the tensor image case.

Definition 4.3. The *tensor projection* $\pi_n : \mathcal{T}_k\big(\mathbb{R}^{n+1}, p\big) \to \mathcal{T}_k(\mathbb{R}^n, p)$ is defined for all $a \in \mathcal{T}_k\big(\mathbb{R}^{n+1}, p\big)$ as $[\pi_n(a)]_{i_1,\ldots,i_k} := [a]_{i_1,\ldots,i_k}$, for $i_1, \ldots, i_k$ ranging from 1 to n. The pseudoinverse will be the zero padding map $\tilde{\phi}_n : \mathcal{T}_k(\mathbb{R}^n, p) \to \mathcal{T}_k\big(\mathbb{R}^{n+1}, p\big)$, analogous to Definition 2.2.

Definition 4.4 (Tensor image embedding)**.** Let $A \in \mathcal{A}_{N,n,k,p}$. Then the *tensor image embedding from n to $n+1$* of A, denoted $\phi_n : \mathcal{A}_{N,n,k,p} \to \mathcal{A}_{N,n+1,k,p}$, is defined for all $\bar{\imath}_{n+1} \in (\mathbb{Z}/N\mathbb{Z})^{n+1}$ as $\phi_n(A)(\bar{\imath}_{n+1}) = \tilde{\phi}_n(A(\pi_n(\bar{\imath}_{n+1})))$. The pseudoinverse $\rho_n : \mathcal{A}_{N,n+1,k,p} \to \mathcal{A}_{N,n,k,p}$ takes an n-dimensional slice: $\rho_n(A')(\bar{\imath}_n) = \pi_n\Big(A'\Big(\tilde{\phi}_n(\bar{\imath}_n)\Big)\Big)$, for $A' \in \mathcal{A}_{N,n+1,k,p}$ and all $\bar{\imath}_n \in (\mathbb{Z}/N\mathbb{Z})^n$.

We say that a sequence of functions $\{f_n\}$ are compatible with the embeddings $\{\phi_n\}$ if the following diagram commutes:

$$\begin{array}{ccccccccc}
\ldots & \xhookrightarrow{\cdots} & \mathcal{A}_{N,n,k,p} & \xhookrightarrow{\phi_n} & \mathcal{A}_{N,n+1,k,p} & \xhookrightarrow{\phi_{n+1}} & \mathcal{A}_{N,n+2,k,p} & \xhookrightarrow{\cdots} & \ldots \\
 & & \downarrow{\scriptstyle f_n} & & \downarrow{\scriptstyle f_{n+1}} & & \downarrow{\scriptstyle f_{n+2}} & & \\
\ldots & \xhookrightarrow{\cdots} & A_{N,n,k',p'} & \xhookrightarrow{\phi_n} & A_{N,n+1,k',p'} & \xhookrightarrow{\phi_{n+1}} & A_{N,n+2,k',p'} & \xhookrightarrow{\cdots} & \ldots
\end{array} \tag{17}$$

We consider our entire neural networks as the functions f_n, and we enforce overall compatibility by requiring compatibility of every layer in the network. The following theorem specifies when compatibility with respect to (17) holds for geometric convolutions with a particular filter.

Theorem 4.1 (Geometric convolution compatibility). *Let $C \in \mathcal{A}_{M,n,k+k',p\,p'}$ and $C' \in \mathcal{A}_{M,n+1,k+k',p\,p'}$ be convolution filters. Let the functions $f_n : \mathcal{A}_{N,n,k,p} \to \mathcal{A}_{N,n,k',p'}$ and $f_{n+1} : \mathcal{A}_{N,n+1,k,p} \to \mathcal{A}_{N,n+1,k',p'}$ be defined respectively as $f_n(A) = A * C$ for all $A \in \mathcal{A}_{N,n,k,p}$ and $f_{n+1}(A') = A' * C'$ for all $A' \in \mathcal{A}_{N,n+1,k,p}$. Then f_n, f_{n+1} are compatible with respect to* (17) *if*

$$C(\bar{\imath}_n) = \sum_{\bar{\imath}_{n+1} \in \pi_n^{-1}(\bar{\imath}_n)} \pi_n(C'(\bar{\imath}_{n+1}))\,, \tag{18}$$

for all $\bar{\imath}_n \in (\mathbb{Z}/N\mathbb{Z})^n$, where $\pi_n^{-1}(\bar{\imath}_n)$ is the pre-image of the tensor projection to $\bar{\imath}_n$.

The proof is given in Appendix E.2. Since tensor images form a vector space, we can think of either side of (18) as a linear combination of filters, and the same result holds (see Corollary E.1). It is additionally straightforward to show that all other common layers in the network (point-wise nonlinearities, max pooling, etc.) are compatible with our embedding ϕ_n (Appendix E.2).

4.3 Algorithm for warm-starting PDEs

The previous section proves conditions under which the various layers of a CNN exactly satisfy compatibility with the tensor image embedding (17). This would be the conclusion to the story if we expected all our higher dimensional images to look like the embedding: N copies of the lower dimensional image stacked on top of each other. This is unlikely to be true in practice, so we would like our lifted models to work for generic initial conditions in higher dimensions. In this section we will provide an algorithm for transferring filters and their coefficients in a way that satisfies compatibility (18), while simultaneously achieving better performance on generic input images.

Let A' be an $(n+1)$-dimensional image, let C' be an $(n+1)$-dimensional filter, and let C be an n-dimensional filter. We will say C' and C are close if the convolution $A' * C'$ followed by taking any n-dimensional slice is close to taking the same n-dimensional slice of A', then convolving with C. We combine ρ with a random isometry to get a random slice. This random slice projection is similar to the 2D to 3D embedding used in the Walrus model (McCabe et al., 2025).

Definition 4.5 (Warm-start coefficients). Let $C_0, \ldots, C_{L_n} \in \mathcal{A}_{M,n,k,p}$ be a basis of B_n-invariant filters with coefficients $\alpha_0, \ldots, \alpha_{L_n}$ and let $C = \sum_{i=0}^{L_n} \alpha_i C_i$. Let $C'_0, \ldots, C'_{L_{n+1}} \in \mathcal{A}_{M,n+1,k,p}$ be a basis of B_{n+1}-invariant filters, and let $\mathcal{A}'$ be some distribution of $(n+1)$-dimensional images. Then the *warm-start coefficients for $\mathcal{A}'$* are the coefficients $\alpha'_0, \ldots, \alpha'_{L_{n+1}}$ with $C' = \sum_{i=0}^{L_{n+1}} \alpha'_i C'_i$ that satisfy

$$\underset{\alpha'_i, \ldots, \alpha'_{L_{n+1}}}{\arg\min} \underset{h \in G_{n+1}}{\mathbb{E}} \Big[\underset{A' \sim \mathcal{A}'}{\mathbb{E}} [\mathcal{L}((\rho_n(h \cdot A') * C) - \rho_n(h \cdot (A' * C')))] \Big]\,, \tag{19}$$

subject to the constraint that C, C' satisfy the compatibility condition (17). $\mathcal{L}$ is the MSE loss.

Equation 19 can be thought of as a soft compatibility constraint, whereas (17) is a hard constraint. Solving (19) exactly for particular distributions of input images $\mathcal{A}'$ could be an involved problem by itself, further complicated by the fact that we would need to solve this problem for each layer of our network. For a warm start however, we only require a reasonable starting guess from which we will further train our network end-to-end on fine-tuning data. Thus we consider $\mathcal{A}'$ as a distribution of scalar images with simplifying assumptions, and we derive a convenient form of (19) to solve.

Proposition 4.2. *Let the filters and coefficients be as defined in Definition 4.5. If $\mathcal{A}'$ is the distribution of scalar images with independent identically distributed (iid) pixels with $\underset{A' \sim \mathcal{A}'}{E}[A'(\bar{\imath})] = 0$ and $\underset{A' \sim \mathcal{A}'}{E}\big[A'(\bar{\imath})^2\big] = 1$, then* (19) *is equivalent to*

$$\underset{\alpha'_0, \ldots, \alpha'_{L_{n+1}}}{\arg\min} \mathcal{L}(\pi_n(\varphi_n(C) - C'))\,, \tag{20}$$

subject to the constraint that C, C' satisfy the compatibility condition (17)*.*

Note that π_n is acting on tensor images here by projecting all the pixels, and φ_n embeds an n-dimensional filter in the center slice of an $(n+1)$-dimensional filter with zeros everywhere else. The proof and full details are given in Appendix F. We can now describe the full algorithm. We call it the *Nepo Warm-start* because we use our influence and connections, the n-dimensional data, to gain a huge advantage for our offspring, the $(n+1)$-dimensional model. The second step of Algorithm 1, computing the coefficients from (20), can be done in advance analytically based on the coefficients for the n-dimensional filters. These values are given in Table 2 in Appendix F.

Algorithm 1: Nepo Warm-start

Input: $k_{(p)}$-tensor filters $C_0, C_1, \ldots, C_{L_n}$ in n dimensions and $k_{(p)}$-tensor filters $C'_0, C'_1, \ldots, C'_{L_{n+1}}$ in $n+1$ dimensions for each required tensor order k and parity p. Data $\mathcal{D}_n$ in n dimensions and optionally data $\mathcal{D}_{n+1}$ in $n+1$ dimensions.

1. Train a ginjax model Gregory et al. (2025) with the n-dimensional $k_{(p)}$-tensor filters $C_0, \ldots, C_{L_n}$ on $\mathcal{D}_n$.
2. For each tensor order k and parity p, solve (20) to convert the coefficients $\alpha_0, \ldots, \alpha_{L_n}$ for the n-dimensional filters to the coefficients $\alpha'_0, \ldots, \alpha'_{L_{n+1}}$ for the $(n+1)$-dimensional filters.
3. For each tensor order k and parity p, replace the n-dimensional filters $C_0, \ldots, C_{L_n}$ with the $(n+1)$-dimensional filters $C'_0, \ldots, C'_{L_{n+1}}$.
4. Optionally, continue to fine-tune the model on $\mathcal{D}_{n+1}$ data.

Output: The $(n+1)$-dimensional ginjax model.

5 Numerical Experiments

We demonstrate our method on three PDEs: the heat equation, Burgers' equation, and the compressible Navier–Stokes equations. For each equation, our ultimate goal is to train a neural network to approximate the solution operator $\mathcal{S}^t_{F_{n+1}}(u_0) \mapsto u_t$. To do this, we use the Nepo Warm-start (Algorithm 1) to train a model on $\mathcal{D}_n$ data, convert the model to work with $\mathcal{D}_{n+1}$ data, and fine-tune on data sets of varying size. We then compare each of these models with a held-out data set to a baseline model with identical architecture that was only trained on the fine-tuning data.

For the heat equation, we used 128 1D pre-training data points, $0, 1, 4, 8, 32$, and 128 2D fine-tuning data points, and 128 2D test data points. For Burgers' equation, we used 8 2D trajectories of pre-training data, $0, 1, 4$, and 8 3D trajectories of fine-tuning data, and 8 3D trajectories of test data. Each trajectory had 50 time steps which became 49 input-output data points. Finally, for the Navier–Stokes equations, we used 128 2D trajectories of pre-training data, $0, 1, 4, 8$, and 32 3D trajectories of fine-tuning data, and 32 3D trajectories of test data. Each trajectory had 21 time steps which became 17 input-output data points, using 4 time steps for input and 1 time step for the output. Full data and training details are in Appendix G.

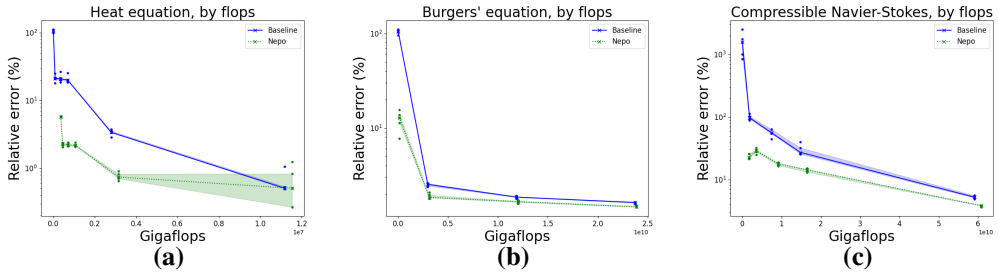


Figure 2: Relative error performance of **(a)** the heat equation, **(b)** Burgers' equation, and **(c)** the compressible Navier–Stokes equations over varying numbers of fine tuning points, indexed by the total number of gigaflops to pre-train and fine-tune the models. Note the order of magnitude of the x-tick values, and the log values of the y-axis. We ran 5 trials, with the lines representing the median value, the shaded region representing the interquartile range, and a dot for each value.

The results under the relative error metric are shown in Fig. 2, and the values are in Table 1. We first note that the pre-train results are more accurate than the baseline results in 12/13 cases, with a negligible difference in the last case. The improvement is most dramatic for the most challenging PDE, the compressible Navier–Stokes equations. The warm-start model with no additional fine-tuning performs better than the baseline model trained with 8 trajectories while using 12% of the flops (Table 6) and 20% of the total data size. We see a mild overfitting effect when fine-tuning the Nepo warm-start for Navier–Stokes with 1 training trajectory. Although pre-training the model requires some additional computational time, in order to achieve comparable performance, the baseline model requires significantly more computational time.

PDE	Models	0	1	4	8	32	128
Heat	Baseline	105.3%	21.0%	20.8%	19.8%	3.3%	**0.5%**
	Nepo	**5.7%**	**2.2%**	**2.2%**	**2.1%**	**0.7%**	**0.5%**
Burgers	Baseline	105.3%	2.5%	1.8%	1.6%	Time out	Time out
	Nepo	**12.8%**	**1.8%**	**1.6%**	**1.4%**	Time out	Time out
N-S	Baseline	1,565.7%	97.0%	56.3%	27.2%	5.2%	Time out
	Nepo	**22.8%**	**28.9%**	**17.8%**	**14.3%**	**3.8%**	Time out

Table 1: The median relative error performance over 5 trials of each PDE over varying numbers of fine tuning points. N-S refers to compressible Navier–Stokes. For the heat equation, the column labels represent the number of fine-tuning points, while for Burgers and Navier–Stokes, they refer to the number of trajectories of 17 and 49 time steps each, respectively. The positions in the table labeled "Time out" were training loops that would have taken longer than 15 hours per trial. With higher precision, the baseline method does better than Nepo for the heat equation by 0.001%.

Aside from computational time, memory is often an additional bottleneck in simulating higher dimensions. When the data is too large to be processed all at once, it can require additional cumbersome strategies, such as splitting the spatial domain into sections to be processed individually Bolton & Zanna (2019). For the heat equation, a single tensor image in dimension n with side length N requires $\mathcal{O}(N^n)$ memory, while for Burgers' equation and the Navier–Stokes equations, it requires $\mathcal{O}(n\,N^n)$. Our any-dimensional method therefore provides memory benefits in addition to the computational benefits.

To confirm that Definition 4.5 is an effective method of selecting coefficients for higher dimensional filters, we perform an ablation over other reasonable strategies in Appendix G.3. Finally, additional plots and a table with the l_2 error are shown in Appendix G.4 along with a table of the flops values.

6 Discussion and Limitations

Inspired by the graph neural network transferability literature, this paper focuses on the transferability of PDE solvers across ambient dimensions. One key aspect for transferability in this framework is the existence of symmetries that maintain the expressive power as the dimension increases. In this work we assume that the right hand side of the PDE we are solving is orthogonally symmetric, but this could be changed to other symmetries (permutations or Lorentz for example) with further theoretical development. However, some amount of symmetry in the equations will be an unavoidable assumption for this framework to apply. This assumption covers many applications of interest because any PDEs arising from the laws of physics will naturally have these symmetries. Another limitation of this work is that it only focuses on flat geometries. A priori we do not know whether the approach could be adapted to general manifolds.

Finally, the main limitation is that the conditions under which the solver generalizes exactly correspond to a subspace where the problem is basically a n-dimensional problem in $(n+1)$-dimensions. That is also the case in the graph neural networks setting, but they can show that difference goes to zero as the number of nodes in the graph goes to infinity. In our PDE application, it is impractical to consider the dimension growing to infinity, but we show that the solution in n dimensions can be used as a viable warm-start for $n+1$ dimensions in practice. Our results numerically demonstrate that this approach gives a significant computational advantage for many problems in many settings.

AI use statement

We only used AI to produce the figure next to Definition 2.1.

Reproducibility statement

The details to reproduce all experiments are provided in Appendix G. The data sets, described in Appendix G.1, are either publicly available, or generated with open source code. Our code is open source and will be released after anonymous review.

REFERENCES

Yulia Alexandr, Joe Kileel, and Bernd Sturmfels. Moment varieties for mixtures of products. In *Proceedings of the 2023 International Symposium on Symbolic and Algebraic Computation*, pp. 53–60, 2023.

Eray Unsal Atay, Eitan Levin, Venkat Chandrasekaran, and Victoria Kostina. Poset-markov channels: Capacity via group symmetry. *IEEE Transactions on Information Theory*, 2025.

Pablo Barceló, Egor V Kostylev, Mikaël Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations (ICLR 2020)*, Virtual conference, Ethiopia, April 2020. URL https://hal.science/hal-03356968.

Ben Blum-Smith and Soledad Villar. Machine learning and invariant theory. *Notices of the American Mathematical Society*, 70(8):1205–1213, 2023.

Ben Blum-Smith, Marco Cuturi, Soledad Villar, et al. Learning functions on symmetric matrices and point clouds via lightweight invariant features. *arXiv preprint arXiv:2405.08097*, 2024.

Jan Böker, Ron Levie, Ningyuan Huang, Soledad Villar, and Christopher Morris. Fine-grained expressivity of graph neural networks. *Advances in Neural Information Processing Systems*, 36, 2024.

Thomas Bolton and Laure Zanna. Applications of deep learning to ocean data inference and subgrid parameterization. *Journal of Advances in Modeling Earth Systems*, 11(1):376–399, 2019. doi: https://doi.org/10.1029/2018MS001472. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2018MS001472.

Thomas Church. Homological stability for configuration spaces of manifolds. *Inventiones mathematicae*, 188(2):465–504, 2012.

Thomas Church and Benson Farb. Representation theory and homological stability. *Advances in Mathematics*, 245:250–314, 2013. ISSN 0001-8708. doi: https://doi.org/10.1016/j.aim.2013.06.016. URL https://www.sciencedirect.com/science/article/pii/S0001870813002259.

Thomas Church, Jordan S Ellenberg, and Benson Farb. Representation stability in cohomology and asymptotics for families of varieties over finite fields. *arXiv preprint arXiv:1309.6038*, 2013.

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

Aldo Conca, Sandra Di Rocco, Jan Draisma, June Huh, Bernd Sturmfels, Filippo Viviani, and Jan Draisma. Noetherianity up to symmetry. *Combinatorial Algebraic Geometry: Levico Terme, Italy 2013, Editors: Sandra Di Rocco, Bernd Sturmfels*, pp. 33–61, 2014.

Congyue Deng, Or Litany, Yueqi Duan, Adrien Poulenard, Andrea Tagliasacchi, and Leonidas Guibas. Vector neurons: A general framework for so(3)-equivariant networks, 2021. URL https://arxiv.org/abs/2104.12229.

Mateo Díaz, Dmitriy Drusvyatskiy, Jack Kendrick, and Rekha R Thomas. Invariant kernels: Rank stabilization and generalization across dimensions. *arXiv preprint arXiv:2502.01886*, 2025.

Mario Geiger and Tess Smidt. e3nn: Euclidean neural networks. *arXiv preprint arXiv:2207.09453*, 2022.

Wilson G. Gregory, David W. Hogg, Ben Blum-Smith, Maria Teresa Arias, Kaze W. K. Wong, and Soledad Villar. Equivariant geometric convolutions for dynamical systems on vector and tensor images. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 383(2298):20240247, 2025. doi: 10.1098/rsta.2024.0247. URL https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2024.0247.

Martin Grohe. The logic of graph neural networks. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 1–17. IEEE, 2021.

Sékou-Oumar Kaba, Arnab Kumar Mondal, Yan Zhang, Yoshua Bengio, and Siamak Ravanbakhsh. Equivariance with learned canonicalization functions. In *International Conference on Machine Learning*, pp. 15546–15566. PMLR, 2023.

George Em 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.

Sammy Khalife and Josué Tonelli-Cueto. Is uniform expressivity too restrictive? towards efficient expressivity of GNNs. In *The Thirteenth International Conference on Learning Representations*, 2024.

Felix Koehler, Simon Niedermayr, Rudiger Westermann, and Nils Thuerey. APEBench: A benchmark for autoregressive neural emulators of PDEs. *Advances in Neural Information Processing Systems (NeurIPS)*, 38, 2024.

Risi Kondor. N-body networks: a covariant hierarchical neural network architecture for learning atomic potentials. *arXiv preprint arXiv:1803.01588*, 2018.

Risi Kondor and Shubhendu Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. In *International conference on machine learning*, pp. 2747–2755. PMLR, 2018.

Ron Levie. A graphon-signal analysis of graph neural networks. *Advances in Neural Information Processing Systems*, 36, 2024.

Ron Levie, Wei Huang, Lorenzo Bucci, Michael Bronstein, and Gitta Kutyniok. Transferability of spectral graph convolutional neural networks. *Journal of Machine Learning Research*, 22(272): 1–59, 2021.

Eitan Levin. *Any-Dimensional Data Science: Learning, Optimization, and Sampling*. PhD thesis, CALIFORNIA INSTITUTE OF TECHNOLOGY, 2026.

Eitan Levin and Venkat Chandrasekaran. Free descriptions of convex sets. *arXiv preprint arXiv:2307.04230*, 2023.

Eitan Levin and Venkat Chandrasekaran. Any-dimensional polynomial optimization via de finetti theorems. *arXiv preprint arXiv:2507.15632*, 2025.

Eitan Levin and Venkat Chandrasekaran. Any-dimensional learning by sampling. *arXiv preprint arXiv:2607.07680*, 2026.

Eitan Levin and Mateo Díaz. Any-dimensional equivariant neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 2773–2781. PMLR, 2024.

Eitan Levin, Yuxin Ma, Mateo Diaz Diaz, and Soledad Villar. On transferring transferability: Towards a theory for size generalization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Yi Li, Eric Perlman, Minping Wan, Yunke Yang, Charles Meneveau, Randal Burns, Shiyi Chen, Alexander Szalay, and Gregory Eyink. A public turbulence database cluster and applications to study lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence*, 9:N31, January 2008. ISSN 1468-5248. doi: 10.1080/14685240802376389. URL `http://dx.doi.org/10.1080/14685240802376389`.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017. URL `https://arxiv.org/abs/1608.03983`.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

Yuxin Ma, Nan Chen, Mateo Díaz, Soufiane Hayou, Dmitriy Kunisky, and Soledad Villar. μpscaling small models: Principled warm starts and hyperparameter transfer. *arXiv preprint arXiv:2602.10545*, 2026.

Sohir Maskey, Ron Levie, and Gitta Kutyniok. Transferability of graph neural networks: an extended graphon approach. *Applied and Computational Harmonic Analysis*, 63:48–83, 2023.

Michael McCabe, Payel Mukhopadhyay, Tanya Marwah, Bruno Regaldo-Saint Blancard, Francois Rozet, Cristiana Diaconu, Lucas Meyer, Kaze W. K. Wong, Hadi Sotoudeh, Alberto Bietti, Irina Espejo, Rio Fear, Siavash Golkar, Tom Hehir, Keiya Hirashima, Geraud Krawezik, Francois Lanusse, Rudy Morel, Ruben Ohana, Liam Parker, Mariel Pettee, Jeff Shen, Kyunghyun Cho, Miles Cranmer, and Shirley Ho. Walrus: A cross-domain foundation model for continuum dynamics, 2025. URL https://arxiv.org/abs/2511.15684.

Trung V Phan, George A Kevrekidis, Soledad Villar, Yannis G Kevrekidis, and Juan M Bello-Rivas. Towards coordinate-and dimension-agnostic machine learning for partial differential equations. *arXiv preprint arXiv:2505.16549*, 2025.

Yorgos M Psarellis, Seungjoon Lee, Tapomoy Bhattacharjee, Sujit S Datta, Juan M Bello-Rivas, and Ioannis G Kevrekidis. Data-driven discovery of chemotactic migration of bacteria via coordinate-invariant machine learning. *BMC bioinformatics*, 25(1):337, 2024.

Eric Ramos and Graham White. Families of markov chains with compatible symmetric-group actions. *arXiv preprint arXiv:1810.08475*, 2018.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. URL https://arxiv.org/abs/1505.04597.

Eran Rosenbluth, Jan Toenshoff, and Martin Grohe. Some might say all you need is sum. *arXiv preprint arXiv:2302.11603*, 2023.

Luana Ruiz, Fernando Gama, and Alejandro Ribeiro. Graph neural networks: Architectures, stability, and transferability. *Proceedings of the IEEE*, 109(5):660–682, 2021.

Steven Sam and Andrew Snowden. Gl-equivariant modules over polynomial rings in infinitely many variables. *Transactions of the American Mathematical Society*, 368(2):1097–1158, 2016.

Steven Sam and Andrew Snowden. Gröbner methods for representations of combinatorial categories. *Journal of the American Mathematical Society*, 30(1):159–203, 2017.

Steven V Sam and Andrew Snowden. Stability patterns in representation theory. In *Forum of Mathematics, Sigma*, volume 3, pp. e11. Cambridge University Press, 2015.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. Pdebench: An extensive benchmark for scientific machine learning, 2024. URL https://arxiv.org/abs/2210.07182.

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

Mauricio Velasco, Kaiying O'Hare, Bernardo Rychtenberg, and Soledad Villar. Graph neural networks and non-commuting operators. *Advances in neural information processing systems*, 37: 95662–95691, 2024.

Soledad Villar, David W Hogg, Kate Storey-Fisher, Weichi Yao, and Ben Blum-Smith. Scalars are universal: Equivariant machine learning, structured like classical physics. *Advances in Neural Information Processing Systems*, 34:28848–28863, 2021.

Soledad Villar, Weichi Yao, David W Hogg, Ben Blum-Smith, and Bianca Dumitrascu. Dimensionless machine learning: Imposing exact units equivariance. *Journal of Machine Learning Research*, 24 (109):1–32, 2023.

Shengtai Yao, Eitan Levin, and Mateo Díaz. Any-dimensional invariant universality. *arXiv preprint arXiv:2605.23156*, 2026.

P.K. Yeung, Kiran Ravikumar, Rohini Uma-Vaideswaran, Daniel L. Dotson, Katepalli R. Sreenivasan, Stephen B. Pope, Charles Meneveau, and Stephen Nichols. Small-scale properties from exascale computations of turbulence on a $\mathbf{32\,768^3}$ periodic cube. *Journal of Fluid Mechanics*, 1019:R2, 2025. doi: 10.1017/jfm.2025.10493.

A Related work

Any-dimensional optimization and machine learning with representation stability. A recent line of research shows how ideas from representation stability (Church, 2012; Church et al., 2013) can be applied to optimization (Levin & Chandrasekaran, 2023; 2025) and machine learning (Levin & Díaz, 2024; Levin et al., 2025). This framework has been recently applied to PDE identification (Phan et al., 2025), kernel regression (Díaz et al., 2025), and information theory (Atay et al., 2025). These ideas in particular explain why finitely-parameterized equivariant machine learning models can have uniform expressivity across growing input sizes. In GNNs, the notion of uniform expressivity has been explored with other techniques in Grohe (2021); Barceló et al. (2020); Khalife & Tonelli-Cueto (2024); Rosenbluth et al. (2023); Böker et al. (2024). Beyond learning, representation stability has been used to study limits of a number of mathematical objects, see for example Sam & Snowden (2017; 2015; 2016); Conca et al. (2014); Ramos & White (2018); Alexandr et al. (2023). Closest to our work is the recent work Ma et al. (2026), which combines ideas of any-dimensional machine learning and μP hyperparameter transfer to warm-start large models from pretrained smaller models.

Equivariant machine learning. The theory of any-dimensional machine learning applies to equivariant machine learning models, i.e., neural networks with symmetries imposed. There is a wide class of equivariant models, including the ones expressed in terms of group convolutions (Cohen & Welling, 2016; Kondor & Trivedi, 2018), representation theory (Kondor, 2018; Thomas et al., 2018; Geiger & Smidt, 2022), canonicalization (Kaba et al., 2023), and invariant theory (Villar et al., 2021; Blum-Smith & Villar, 2023; Villar et al., 2023; Blum-Smith et al., 2024), among others. In this work, we use the equivariant models on scalars and tensor fields designed to solve PDEs from Gregory et al. (2025).

Graph neural network transferability. The GNN transferability literature explains why one can train a GNN on small graphs and generalize to larger graphs (Ruiz et al., 2021; Levie et al., 2021; Levie, 2024; Velasco et al., 2024; Maskey et al., 2023). These results were extended beyond graphs (to all any-dimensional models) in Levin et al. (2025). The key idea is that a GNN implicitly identifies graphs of different sizes as the same object, and this induces a metric that we can use to compare graphs of different sizes under which the GNN is continuous. This allows us to derive size-generalization theoretical guarantees. In this paper, we develop a similar principle for PDE learning across dimensions.

B Proof of Proposition 2.3

Here we prove Proposition 2.3. Recall that the proposition states that if an n-indexed family of differential equations D_n:

1. makes diagram (11) commute as written, with ϕ_n the standard projections $\mathbb{R}^{n+1} \to \mathbb{R}^n$, and
2. the D_n do not depend on the space coordinate $\mathbf{x}$ and so are functions of $U_{n,k}$ alone, and
3. furthermore they are invariant with respect to the natural action of $\mathrm{O}(n)$ on $U_{n,k}$,

then they would also make diagram (11) commute if the ϕ_n were replaced by any other affine-orthogonal projections $\mathbb{R}^{n+1} \to \mathbb{R}^n$. We use this numbering for the hypotheses of the theorem in the proof. The proof itself is routine: the majority of the work is in unwinding definitions.

A notational note: it will be convenient in this proof, in order to avoid a proliferation of parentheses, to be able to write both evaluations and compositions of maps using juxtaposition, as needed. Parentheses are therefore only used when there is a danger of ambiguity, and of necessity in maps that take multiple arguments.

Proof of Proposition 2.3. We begin by unwinding definitions to translate the commutativity of diagram (11) into a statement about the D_n, in terms of the explicit maps π_n (Definition 2.1) and $\tilde{\phi}_n$ (Definition 2.2). Prima facie, the diagram says that $\widehat{D}_{n+1}\phi_n = \phi_n \widehat{D}_n$ for all n. Applying both sides to an arbitrary $s_u \in C^\infty(\mathbb{R}^n, U_{n,k})$ and then evaluating at an arbitrary $\mathbf{x} \in \mathbb{R}^n$, we obtain the statement that

$$(\widehat{D}_{n+1}(\phi_n s_u))(\mathbf{x}) = (\phi_n(\widehat{D}_n s_u))(\mathbf{x}) \tag{21}$$

for all such n, s_u, $\mathbf{x}$. Applying the definition first of $\widehat{D}_{n+1}$ and then of $\phi_n : C^\infty(\mathbb{R}^n, U_{n,k}) \to C^\infty(\mathbb{R}^n, U_{n,k})$ on the left side, and the definition first of $\phi_n : C^\infty(\mathbb{R}^n) \to C^\infty(\mathbb{R}^n)$ and then of $\widehat{D}_n$

on the right, we get

$$D_{n+1}(\mathbf{x}, \tilde{\phi}_n s_u \pi_n \mathbf{x}) = D_n(\pi_n \mathbf{x}, s_u \pi_n \mathbf{x}). \tag{22}$$

This equation, quantified over all n, s_u, and $\mathbf{x}$, expresses hypothesis 1 above. We now assume this along with hypotheses 2 and 3, and aim to conclude that (22) would also hold with the π_n replaced with an arbitrary affine-orthogonal projection $\mathbb{R}^{n+1} \to \mathbb{R}^n$, and a corresponding modification to $\tilde{\phi}_n$.

An arbitrary affine-orthogonal projection $\pi'_n : \mathbb{R}^{n+1} \to \mathbb{R}^n$ has the form $\pi'_n = \pi_n g$, where $g \in \mathrm{E}(n+1)$ is an arbitrary Euclidean-isometric self-map of $\mathbb{R}^{n+1}$. The action of $\mathrm{E}(n+1)$ on $\mathbb{R}^{n+1}$ also induces a corresponding action on $U_{n+1,k}$, which we denote by $\tilde{g}$. Then the induced embedding $\tilde{\phi}'_n : U_{n,k} \hookrightarrow U_{n+1,k}$ has the form $\tilde{\phi}'_n = \tilde{g}\tilde{\phi}_n$. Translations act trivially on the fiber of jet space, so the action of $\mathrm{E}(n+1)$ on $U_{n+1,k}$ factors through $\mathrm{O}(n+1)$, and $\tilde{g}$ can be seen as an element of $\mathrm{O}(n+1)$ in its action on $U_{n+1,k}$.

After this setup, the proof is very short. Applying hypothesis 2 to (22), we can write D_{n+1} and D_n in terms of the second argument only, so we have

$$D_{n+1}(\tilde{\phi}_n s_u \pi_n \mathbf{x}) = D_n(s_u \pi_n \mathbf{x}), \tag{23}$$

for all $\mathbf{x} \in \mathbb{R}^{n+1}$ and all s_u. For any $g \in \mathrm{E}(n+1)$ (and so any $\pi'_n = \pi_n g$ and corresponding $\tilde{\phi}'_n = \tilde{g}\tilde{\phi}_n$), we thus have

$$\begin{aligned}
D_{n+1}(\mathbf{x}, \tilde{\phi}'_n s_u \pi'_n \mathbf{x}) &= D_{n+1}(\tilde{g}\tilde{\phi}_n s_u \pi_n g \mathbf{x}) && (24)\\
&= D_{n+1}(\tilde{\phi}_n s_u \pi_n g \mathbf{x}) && (25)\\
&= D_n(s_u \pi_n g \mathbf{x}) && (26)\\
&= D_n(\pi_n g \mathbf{x}, s_u \pi_n g \mathbf{x}) && (27)\\
&= D_n(\pi'_n \mathbf{x}, s_u \pi'_n \mathbf{x}), && (28)
\end{aligned}$$

where the first equality is hypothesis 2 and the definitions of π'_n and $\tilde{\phi}'_n$, the second equality is hypothesis 3, the third is from (23) with $g\mathbf{x}$ in place of $\mathbf{x}$, the fourth is again hypothesis 2, and the fifth is the definition of π'_n. This establishes (22) with π'_n in place of π_n and $\tilde{\phi}'_n$ in place of $\tilde{\phi}_n$, and thus it establishes (11) with the corresponding ϕ'_n in place of ϕ_n, completing the proof. □

C Representation stability

Representation stability is a phenomenon first noted in connection with algebraic topology (Church et al., 2013; Church, 2012). First, given a naturally n-indexed sequence of groups G_n (such as the symmetric groups S_n, the general linear groups $GL(n, \mathbb{R})$, or the orthogonal groups $\mathrm{O}(n)$), there is often a natural way to identify the irreducible representations of G_n with some of the irreducible representations of G_{n+1}. Then, given a natural n-indexed sequence of G_n-representations V_n and G_n-equivariant embeddings $V_n \to V_{n+1}$, it can come to pass under good circumstances that for sufficiently high n, the irreducible representations of G_n appearing in V_n are all the same with respect to this identification, and with the same multiplicities, for sufficiently high n. Such a sequence of G_n representations V_n is said to be *representation stable*.

The canonical action of $\mathrm{O}(n)$ on $\mathbb{R}^n$ is an example of a representation-stable sequence of representations. From general theory, it follows that $E_{n,k} \cong \mathbb{R} \oplus \mathbb{R}^n \oplus \mathrm{Sym}^2(\mathbb{R}^n) \oplus \cdots \oplus \mathrm{Sym}^k(\mathbb{R}^n)$ is representation-stable as well. In turn, it follows that the space of homogeneous degree-d polynomial functions on $E_{n,k}$, which is isomorphic (as a representation of $\mathrm{O}(n)$) to $\mathrm{Sym}^d(E_{n,k})$, is representation-stable as well. In particular, for all sufficiently high n, the multiplicity of the trivial subrepresentation, i.e., the dimension of the space of $\mathrm{O}(n)$-invariants inside $\mathrm{Sym}^d(E_{n,k})$, does not depend on n.

In fact, we can identify a basis of invariants in $\mathrm{Sym}^d(E_{n,k})$ for any given d, whose description does not depend on n. Any element of this basis yields a sequence of D_n's satisfying the compatibility condition, and every such sequence can be described in terms of this basis.

The Laplacian Δu is an example of such an invariant. Other examples include u itself and the squared norm of the gradient of u (also known as the Dirichlet energy density of u). We illustrate further by describing all of the $\mathrm{O}(n)$-invariants on order 2 jet space that are of degree 1, 2, or 3 as polynomials in u and its derivatives:

Example C.1. Given a smooth function $u : \mathbb{R}^n \to \mathbb{R}$, view the gradient ∇u as a column vector, and denote by $\operatorname{Hess} u$ the Hessian matrix of u. Then every nonconstant $\mathrm{O}(n)$-invariant polynomial of degree at most 3 in u and its first and second derivatives, is in fact a polynomial in the following:

- u itself
- Δu, the Laplacian
- $\nabla u \cdot \nabla u$, the Dirichlet energy density
- $e_2(\operatorname{Hess} u)$, the coefficient of the t^2 term in the characteristic polynomial $\det(I - t \operatorname{Hess} u)$ of the Hessian
- $\nabla u^\top (\operatorname{Hess} u) \nabla u$, the norm of the gradient with respect to the symmetric bilinear form defined by the Hessian
- $e_3(\operatorname{Hess} u)$, the coefficient of the t^3 term in the characteristic polynomial of the Hessian

The important takeaway is that this list of invariants does not depend on n (although some of the list items are zero for very low n). Note that the invariants listed here satisfy the compatibility requirement: for any invariant on the list, its instantiation for any n is equal to its instantiation for $n + 1$ upon zeroing out the derivatives that depend on x_{n+1}, so the diagram in (11) commutes; the $\mathrm{O}(n)$-symmetry then implies it would also commute beginning from a different sequence of orthogonal projections π'_n.

Remark C.1. The determinant of the Hessian is an invariant for any n. However, unlike the invariants listed in Example C.1, it is not naturally identified by the above theory across n-values, as it does not satisfy the compatibility condition—in particular, (11) does not commute for this choice $D_n(u) := \det(\operatorname{Hess} u)$ (since its degree depends on n). What is true is that for any fixed j, the coefficient of the degree j term of the characteristic polynomial, $e_j(\operatorname{Hess} u)$, satisfies compatibility across values of n. When $n < j$ it is zero; when $n = j$ it is equal to the determinant.

D Tensor Image and Geometric Convolution Background

In this section we will introduce the key concepts of tensor images and geometric convolution. For narrative flow we will include the definitions for some terms previously defined in Section 4.1.

We first define the space of (pseudo-)vectors and tensors, which will correspond to the features at each pixel (Definition D.1). We then define the geometric images (Definition D.2).

Definition D.1 ($k_{(p)}$-tensors)**.** A $1_{(p)}$*-tensor* is an element of $\mathbb{R}^n$ equipped with the action of $\mathrm{O}(n)$

$$g \cdot v = \det(M(g))^{\frac{1-p}{2}} M(g)\, v, \tag{29}$$

where the parity p is either $+1$ for vectors or -1 for pseudovectors. Given k $1_{(p_i)}$-tensors denoted v_i, then $T := v_1 \otimes \ldots \otimes v_k$ is a *rank-1* $k_{(p)}$*-tensor*, where $p = \prod_{i=1}^k p_i$ and the action of $\mathrm{O}(n)$ is defined as

$$g \cdot (v_1 \otimes \ldots \otimes v_k) = (g \cdot v_1) \otimes \ldots \otimes (g \cdot v_k)\,. \tag{30}$$

Thus a tensor T is an element of a vector space $(\mathbb{R}^n)^{\otimes k}$, which we denote $\mathcal{T}_k(\mathbb{R}^n, p)$. To get higher rank tensors, we can add tensors of the same order k and parity p, and the action of $\mathrm{O}(n)$ extends linearly.

The space $\mathcal{T}_k(\mathbb{R}^n, p)$ naturally has addition and scalar multiplication, but also it has tensor products and tensor contractions. The tensor product takes a $k_{(p)}$-tensor and a $k'_{(p')}$-tensor and outputs a $(k + k')_{(p\,p')}$-tensor whose components are defined as:

$$[a \otimes b]_{i_1,\ldots,i_{k+k'}} = [a]_{i_1,\ldots i_k} [b]_{i_{k+1},\ldots,i_{k+k'}}\,. \tag{31}$$

The tensor contraction ι_k can be though of as analogous to the matrix trace where axes $i_1, \ldots, i_k$ are paired with $i_{k+1}, \ldots, i_{2k}$ and summed over where those axis indices are equal. Written in Einstein summation notation, the k-contraction of a $(2k + k')_{(p)}$-tensor is defined as:

$$[\iota_k(a)]_{i_{2k+1},\ldots,i_{2k+k'}} = [a]_{i_1,\ldots,i_{2k+k'}}\,. \tag{32}$$

To create a geometric image, we align tensors into a grid or lattice structure.

Definition D.2 ($k_{(p)}$-tensor image)**.** Consider images supported in $(\mathbb{Z}/N\mathbb{Z})^n$, the n-dimensional discrete torus. Then a *$k_{(p)}$-tensor image* is a function $A : (\mathbb{Z}/N\mathbb{Z})^n \to \mathcal{T}_k(\mathbb{R}^n, p)$. The set of $k_{(p)}$-tensor images with side lengths N is denoted $\mathcal{A}_{N,n,k,p}$.

For the four tensor operations described above—addition, scalar multiplication, tensor product, and tensor contraction—we have analogous definitions for the geometric image where the operation is performed pixel-wise. For example, if $A, B \in \mathcal{A}_{N,n,k,p}$, then for all $\bar{\imath} \in (\mathbb{Z}/N\mathbb{Z})^n$ we have $(A + B)(\bar{\imath}) = A(\bar{\imath}) + B(\bar{\imath})$.

In this space, the geometric convolutional neural networks are analogous to standard convolutional networks, except that the image-filter pixel multiplication is implemented as tensor products, followed by contractions. For example, if $n = 2$ (i.e. 2D images) the filters are symmetric with respect to 90 degree rotations, and either symmetric ($p = +1$) or anti-symmetric ($p = -1$) with respect to reflections. For general dimension n, this is the hyperoctahedral group, denoted B_n.

Definition D.3 (Geometric convolution)**.** Let N, M be side lengths where $M = 2m + 1$ is odd, let $A \in \mathcal{A}_{N,n,k,p}$ be the input image, and let $C \in \mathcal{A}_{M,n,k+k',p\,p'}$ be the convolution filter. Then the *geometric convolution* is a tensor image $(A * C) \in \mathcal{A}_{N,n,k',p'}$ such that:

$$(A * C)(\bar{\imath}) = \sum_{\bar{a} \in (\mathbb{Z}/M\mathbb{Z})^n} \iota_k(A(\bar{\imath} - c(\bar{a})) \otimes C(\bar{a})) \,, \tag{33}$$

where $c : (\mathbb{Z}/M\mathbb{Z})^n \to \mathbb{Z}^n$ is the pixel centering function $[c(\bar{a})]_i = [\bar{a}]_i - \frac{M-1}{2}$. Note that the pixel centering is not on the torus, but $\bar{\imath} - c(\bar{a})$ is on the torus of $(\mathbb{Z}/N\mathbb{Z})^n$.

Geometric convolutions characterize linear G_d-equivariant functions Gregory et al. (2025), but we also need nonlinear functions to build expressive networks. For this task, we will use the Vector neurons nonlinearity, originally from Deng et al. (2021) and extended to tensors in Gregory et al. (2025).

Definition D.4 (Vector Neuron nonlinearity for tensors)**.** Let A_i for $i = 1, \ldots, r$ be input $k_{(p)}$-tensor image channels. Let $\alpha_i \in \mathbb{R}$ be learned scalar parameters, let $B = \sum_{i=1}^r \alpha_i A_i$, and let $\hat{B} = \frac{B}{\|B\|_2}$, where $\|\cdot\|_2$ is the typical tensor norm, also known as the Frobenius norm. The projection of A_i to $\hat{B}$ is $A_{i,\|} := \iota_k\Big(A_i \otimes \hat{B}\Big)\hat{B}$. Finally, let $\tilde{\sigma}$ be a scalar nonlinearity defined on images pixel-wise. Then the *Vector Neuron nonlinearity* $\sigma : (\mathcal{A}_{N,n,k,p})^r \to (\mathcal{A}_{N,n,k,p})^r$ for the ith output σ_i is defined:

$$\sigma_i\Big((A_j)_{j=1}^r\Big) = A_i - A_{i,\|} + \frac{\tilde{\sigma}\Big(\iota_k\Big(A_i \otimes \hat{B}\Big)\Big)}{\Big|\iota_k\Big(A_i \otimes \hat{B}\Big)\Big|} A_{i,\|} \tag{34}$$

Additionally, we can define pooling layers such as max pooling, where the max is calculated using the ℓ_2 norm of a tensor as follows:

Definition D.5 (max pool)**.** Let $A \in \mathcal{A}_{N,n,k,p}$ and let b be a positive integer that divides N. Then max pool is a function whose output is $\mathcal{A}_{N/b,n,k,p}$ defined for each pixel index $\bar{\imath}$:

$$\text{max pool}(A, b)(\bar{\imath}) = A\left(b\,\bar{\imath} + \operatorname*{arg\,max}_{\bar{a} \in (\mathbb{Z}/b\mathbb{Z})^n} \|A(b\,\bar{\imath} + \bar{a})\|_2\right) \tag{35}$$

E Compatibility

E.1 Compatibility of Solution Operators

Lemma (Restatement of 3.1)**.** *Suppose $F_n(u) = 0$ is a compatible scalar PDE with associated solution operator $\mathcal{S}^t_{F_n}(u_0) \mapsto u_t$. If there is a uniqueness theorem for the equation $F_n = 0$, then $\mathcal{S}^t_{F_n}$ is $\mathrm{E}(n)$-equivariant, i.e. for all initial conditions u_0 and $g \in \mathrm{E}(n)$,*

$$\mathcal{S}^t_{F_n}(g \cdot u_0) = g \cdot \mathcal{S}^t_{F_n}(u_0) \,. \tag{36}$$

Proof. Let $g \in \mathrm{E}(n)$. Let u_t be the evaluation of the function u at time t. Then consider $\mathcal{S}^t_{F_n}(g \cdot u_0)$. Because $\mathcal{S}^t_{F_n}$ is a solution operator, this is the evaluation at time t of a solution to $F_n = 0$ with

initial condition $g \cdot u_0$. Meanwhile, consider $g \cdot \mathcal{S}^t_{F_n}(u_0)$. Because F_n is $\mathrm{E}(n)$-invariant and $F_n(\mathcal{S}^t_{F_n}(u_0)) = F_n(u_t) = 0$, we have

$$0 = F_n(u_t) = F_n(g \cdot u_t) = F_n(g \cdot \mathcal{S}^t_{F_n}(u_0)) \,. \tag{37}$$

Thus, $g \cdot \mathcal{S}^t_n(u_0)$ is also a solution to $F_n = 0$, and its initial condition is $\sigma \cdot G_0(u_0) = g \cdot u_0$. Because we assumed a uniqueness theorem for the equation $F_n = 0$, we conclude that

$$\mathcal{S}^t_{F_n}(g \cdot u_0) = g \cdot \mathcal{S}^t_{F_n}(u_0), \tag{38}$$

so $\mathcal{S}^t_{F_n}$ is $\mathrm{E}(n)$-equivariant. □

Lemma (Restatement of 3.2)**.** *For any $u \in \mathbb{C}^\infty(\mathbb{R}^n \times [0,\infty))$ satisfying the compatible scalar PDE $F_n(u) = 0$ on $\mathbb{R}^n$, we have that $F_{n+1}(\phi_n(u)) = 0$ on $\mathbb{R}^{n+1}$. Furthermore, the same statement holds with ϕ_n replaced by arbitrary ϕ'_n as in Section 2.*

Proof. Compatibility implies that (11) commutes. Thus, $\widehat{D}_{n+1} \circ \phi_n = \phi_n \circ \widehat{D}_n$. Applying both sides of the equality $\widehat{D}_{n+1} \circ \phi_n = \phi_n \circ \widehat{D}_n$ to the prolongation s_u of the time-evolving function u, and then evaluating at a point $(\mathbf{x}, t)$ in space and time, we get

$$\widehat{D}_{n+1}(\phi_n(s_u))(\mathbf{x}, t) = \phi_n(\widehat{D}_n(s_u))(\mathbf{x}, t) = \widehat{D}_n(s_u)(\pi_n(\mathbf{x}), t). \tag{39}$$

Also, $\frac{d(\phi_n(u))}{dt}(\mathbf{x}, t) = \frac{du}{dt}(\pi_n(\mathbf{x}), t)$ by the chain rule. Since

$$\left(\frac{du}{dt} - \widehat{D}_n(s_u)\right)(\pi_n(\mathbf{x}), t) = F_n(u)(\pi_n(\mathbf{x}), t) = 0 \tag{40}$$

by assumption, it follows that

$$0 = F_{n+1}(\phi_n(u))(\mathbf{x}, t) = \left(\frac{d(\phi_n(u))}{dt} - \widehat{D}_{n+1}(\phi_n(s_u))\right)(\mathbf{x}, t) \tag{41}$$

as well. This proves the first statement. The statement for ϕ'_n follows from Proposition 2.3. □

Proposition (Restatement of 3.3)**.** *Let F_n be a compatible scalar PDE. Let u_0 be any initial conditions on $U \subseteq \mathbb{R}^n$, and let ϕ_n be the embedding from Definition 2.1. Then, $\mathcal{S}^t_{F_{n+1}}(\phi_n(u_0)) = \phi_n(\mathcal{S}^t_{F_n}(u_0))$. Furthermore, if there is a uniqueness theorem for $F_n = 0$, then the same statement holds with ϕ_n replaced by $g \circ \phi_n$, for any Euclidean isometry g of $\mathbb{R}^{n+1}$.*

Proof. (sketch) We have $\phi_n(S^t_{F_n}(u_0)) = \phi_n u_t$ by definition of the solution operator. The right-hand-side satisfies $F_{n+1}(\phi_n u_t) = 0$ by Lemma 3.2. The restriction of the embedding map at time $t = 0$ yields $\phi_n(u_0) = (\phi_n(u))_0$, and therefore $(\phi_n(u))_t = \mathcal{S}^t_{F_{n+1}}((\phi_n(u))_0)$. The statement for $g \circ \phi_n$ follows from Lemma 3.1. □

On Remark 3.1 An evolutionary PDE induces a solution operator $S^t_{F_n} : X \to Y$ where X_n and Y_n are Banach spaces of initial conditions and solutions at time t over the corresponding domain. Solution operators are often Lipschitz for many linear (and sometimes nonlinear) PDEs, with respect to the appropriate corresponding norms, satisfying:

$$\left\|\mathcal{S}^t_{F_n}(u) - \mathcal{S}^t_{F_n}(v)\right\|_{Y_n} \leq C_{F^t_n} \|u_0 - v_0\|_{X_n}$$

Thus, in higher dimension, if we embed $w_0 = \phi_n(u_0)$, and define a slightly perturbed initial condition $\tilde{w}_0 = w_0 + \delta$, we have:

$$\left\|\mathcal{S}^t_{F_{n+1}}(w_0) - \mathcal{S}^t_{F_{n+1}}(\tilde{w}_0)\right\|_{Y_{n+1}} \leq C_{F^t_{n+1}} \|\delta\|_{X_{n+1}}.$$

Then, since $\mathcal{S}^t_{F_{n+1}}(w_0) = \phi_n(\mathcal{S}^t_{F_n}(u_0))$, we have that integrating the "compatible part" of a perturbed initial condition and subsequently lifting remains a good approximation for the full perturbed solution. We note that many of the quantities here are dependent on the PDE and corresponding solution operator, and the norms over the corresponding Banach spaces. In particular the constants $C_{F^t_n}$ can be dimension- and time-dependent, possibly resulting in vacuous bounds from a practical approximation perspective. At the same time, they can be more forgiving, and force small perturbations to become smaller over time, thus yielding a robust approximation result.

E.2 Compatibility of Tensor Image Functions

Now we prove Theorem 4.1, compatibility for geometric convolutions.

Theorem (Restatement of 4.1)**.** *Let $C \in \mathcal{A}_{M,n,k+k',p\,p'}$ and $C' \in \mathcal{A}_{M,n+1,k+k',p\,p'}$ be convolution filters. Let the functions $f_n : \mathcal{A}_{N,n,k,p} \to \mathcal{A}_{N,n,k',p'}$ and $f_{n+1} : \mathcal{A}_{N,n+1,k,p} \to \mathcal{A}_{N,n+1,k',p'}$ be defined respectively as $f_n(A) = A * C$ for all $A \in \mathcal{A}_{N,n,k,p}$ and $f_{n+1}(A') = A' * C'$ for all $A' \in \mathcal{A}_{N,n+1,k,p}$. Then f_n are compatible with respect to* (17) *if*

$$C(\bar{\imath}_n) = \sum_{\bar{\imath}_{n+1} \in \pi_n^{-1}(\bar{\imath}_n)} \pi_n(C'(\bar{\imath}_{n+1}))\,, \tag{42}$$

for all $\bar{\imath}_n \in (\mathbb{Z}/N\mathbb{Z})^n$, where $\pi_n^{-1}(\bar{\imath}_n)$ is the pre-image of the tensor projection to $\bar{\imath}_n$.

Proof. Let C, C', f_n, f_{n+1} be defined as in the statement of the theorem. Suppose that $C(\bar{\imath}_n) = \sum_{\bar{\imath}_{n+1} \in \pi_n^{-1}(\bar{\imath}_n)} \pi_n(C'(\bar{\imath}_{n+1}))$ for all $\bar{\imath}_n \in (\mathbb{Z}/N\mathbb{Z})^n$. Thus,

$$\phi_n(f_n(A))(\bar{\imath}_{n+1}) \tag{43}$$
$$= \tilde{\phi}_n(f_n(A)(\pi_n(\bar{\imath}_{n+1}))) \tag{44}$$
$$= \tilde{\phi}_n\left(\sum_{\bar{a}_n \in (\mathbb{Z}/M\mathbb{Z})^n} \iota_k(A(\pi_n(\bar{\imath}_{n+1}) - c(\bar{a}_n)) \otimes C(\bar{a}_n))\right) \tag{45}$$
$$= \sum_{\bar{a}_n \in (\mathbb{Z}/M\mathbb{Z})^n} \tilde{\phi}_n(\iota_k(A(\pi_n(\bar{\imath}_{n+1}) - c(\bar{a}_n)) \otimes C(\bar{a}_n))) \tag{46}$$
$$= \sum_{\bar{a}_n \in (\mathbb{Z}/M\mathbb{Z})^n} \tilde{\phi}_n\left(\iota_k\left(A(\pi_n(\bar{\imath}_{n+1}) - \pi_n(c(\bar{a}_{n+1}))) \otimes \sum_{\bar{a}_{n+1} \in \pi_n^{-1}(\bar{a}_n)} \pi_n(C'(\bar{a}_{n+1}))\right)\right) \tag{47}$$
$$= \sum_{\bar{a}_n \in (\mathbb{Z}/M\mathbb{Z})^n} \tilde{\phi}_n\left(\iota_k\left(\pi_n(\phi_n(A)(\bar{\imath}_{n+1} - c(\bar{a}_{n+1}))) \otimes \sum_{\bar{a}_{n+1} \in \pi_n^{-1}(\bar{a}_n)} \pi_n(C'(\bar{a}_{n+1}))\right)\right) \tag{48}$$
$$= \sum_{\bar{a}_n \in (\mathbb{Z}/M\mathbb{Z})^n} \sum_{\bar{a}_{n+1} \in \pi_n^{-1}(\bar{a}_n)} \tilde{\phi}_n(\iota_k(\pi_n(\phi_n(A)(\bar{\imath}_{n+1} - c(\bar{a}_{n+1}))) \otimes \pi_n(C'(\bar{a}_{n+1})))) \tag{49}$$
$$= \sum_{\bar{a}_{n+1} \in (\mathbb{Z}/M\mathbb{Z})^n} \iota_k\Big(\tilde{\phi}_n(\pi_n(\phi_n(A)(\bar{\imath}_{n+1} - c(\bar{a}_{n+1})) \otimes C'(\bar{a}_{n+1})))\Big) \tag{50}$$
$$= \sum_{\bar{a}_{n+1} \in (\mathbb{Z}/M\mathbb{Z})^n} \iota_k(\phi_n(A)(\bar{\imath}_{n+1} - c(\bar{a}_{n+1})) \otimes C'(\bar{a}_{n+1})) \tag{51}$$
$$= f_{n+1}(\phi_n(A))(\bar{\imath}_{n+1}) \tag{52}$$

In (47) we substitute $C(\bar{a}_n)$ based on our assumption, and additionally say $c(\bar{a}_n) = \pi_n(c(\bar{a}_{n+1})$, where a_{n+1} is any pixel index in the pre-image $\pi_n^{-1}(\bar{a}_n)$. In (50) we commute $\tilde{\phi}_n$ with $\iota_k(\cdot)$ since the pseudoinverse fills the extra indices that would be added with zeros. Finally, in (51) we can remove $\tilde{\phi}_n(\pi_n(\cdot))$ because the embedded image $\phi_n(A)$ itself undergoes the projection pseudoinverse, so in effect we are saying that $\tilde{\phi}_n(\pi_n(\tilde{\phi}_n(a))) = \tilde{\phi}_n(a)$ which is true. The embedding and the function commute, so f_n, f_{n+1} are compatible with (17), and this completes the proof. □

The following is a simple corollary of Theorem 4.1 for a linear combination of filters.

Corollary E.1 (Compatibility for a linear combination of filters)**.** *Let $C_0, C_1, \ldots, C_{L_n} \in \mathcal{A}_{M,n,k+k',p\,p}$ be a set of n-dimensional convolution filters with associated coefficients $\alpha_0, \alpha_1, \ldots, \alpha_{L_n} \in \mathbb{R}$. Let $C'_0, C'_1, \ldots, C'_{L_{n+1}} \in \mathcal{A}_{M,n+1,k+k',p\,p'}$ be a set of $(n+1)$-dimensional convolution filters with associated coefficients $\alpha'_0, \ldots, \alpha'_{L_{n+1}} \in \mathbb{R}$. Let the functions $f_n : \mathcal{A}_{N,n,k,p} \to \mathcal{A}_{N,n,k',p'}$ and $f_{n+1} : \mathcal{A}_{N,n+1,k,p} \to \mathcal{A}_{N,n+1,k',p'}$ be defined respectively as $f_n(A) = A * \left(\sum_{\ell=0}^{L_n} \alpha_\ell C_\ell\right)$ and $f_{n+1}(A) = A * \left(\sum_{\ell=0}^{L_{n+1}} \alpha'_\ell C'_\ell\right)$ for all $A \in \mathcal{A}_{N,n,k,p}$. Then*

f_n, f_{n+1} are compatible with (17) *if*

$$\sum_{\ell=0}^{L_n} \alpha_\ell C_\ell(\bar{\imath}_n) = \sum_{\bar{\imath}_{n+1} \in \pi_n^{-1}(\bar{\imath}_n)} \sum_{\ell=0}^{L_{n+1}} \pi_n(\alpha'_\ell C'_\ell(\bar{\imath}_{n+1})) \,, \tag{53}$$

for all $\bar{\imath}_n \in (\mathbb{Z}/N\mathbb{Z})^n$, where $\pi_n^{-1}(\bar{\imath}_n)$ is the pre-image of the tensor projection to $\bar{\imath}_n$.

Proof. This follows immediately from Theorem 4.1. □

Proposition E.2 (Compatibility of point-wise functions)**.** *Let $f_n : \mathcal{A}_{N,n,k,p} \to \mathcal{A}_{N,n,k',p'}$ be a function defined point-wise, that is there exists some $h_n : \mathcal{T}_k(\mathbb{R}^n, p) \to \mathcal{T}_{k'}(\mathbb{R}^n, p')$ such that $f_n(A)(\bar{\imath}) = h_n(A(\bar{\imath}))$ for all $A \in \mathcal{A}_{N,n,k,p}$ and all $\bar{\imath} \in (\mathbb{Z}/N\mathbb{Z})^n$. Then f_n, f_{n+1} with associated h_n, h_{n+1} are compatible with* (17) *if $h_{n+1}(\tilde{\phi}_n(a)) = \tilde{\phi}_n(h_n(a))$ for all $a \in \mathcal{T}_k(\mathbb{R}^n, p)$.*

Proof. Let $f_n, f_{n+1}, h_n, h_{n+1}$ be as described in the statement of the proposition, and suppose that $h_{n+1}(\tilde{\phi}_n(a)) = \tilde{\phi}_n(h_n(a))$ for all $a \in \mathcal{T}_k(\mathbb{R}^n, p)$. Let $A \in \mathcal{A}_{N,n+1,k,p}$ and let $\bar{\imath}_{n+1} \in (\mathbb{Z}/N\mathbb{Z})^{n+1}$. Thus:

$$\begin{aligned}
f_{n+1}(\phi_n(A))(\bar{\imath}_{n+1}) &= h_{n+1}(\phi_n(A)(\bar{\imath}_{n+1})) && (54)\\
&= h_{n+1}(\tilde{\phi}_n(A(\pi_n(\bar{\imath}_{n+1})))) && (55)\\
&= \tilde{\phi}_n(h_n(A(\pi_n(\bar{\imath}_{n+1})))) && (56)\\
&= \tilde{\phi}_n(f_n(A)(\pi_n(\bar{\imath}_{n+1}))) && (57)\\
&= \phi_n(f_n(A))(\bar{\imath}_{n+1}) && (58)
\end{aligned}$$

This completes the proof. □

Now we would like to confirm that the Vector Neuron nonlinearity Deng et al. (2021); Gregory et al. (2025) used in our models satisfies the tensor embedding condition and is therefore compatible.

Corollary E.3 (Compatibility of Vector Neuron nonlinearity)**.** *The Vector Neuron nonlinearity (Definition D.4) is compatible with* (17)*.*

Proof. The Vector Neuron nonlinearity is defined for (multiple channels of) an entire tensor image because the function is the same at each pixel. Thus it suffices to show that the definition on a single pixel commutes with $\tilde{\phi}_n$. Let $a_i \in \mathcal{T}_k(\mathbb{R}^n, p)$ for $i = 1, \ldots, r$ be the channels of tensors at a particular pixel index. If we consider $b = \sum_{i=1}^r \alpha_i a_i$ as a function of the a_i, then clearly by linearity

$$b(\tilde{\phi}_n(a_i)) = \sum_{i=1}^r \alpha_i \tilde{\phi}_n(a_i) = \tilde{\phi}_n\left(\sum_{i=1}^r \alpha_i a_i\right) = \tilde{\phi}_n(b(a_i)) \,. \tag{59}$$

Also, $\left\|\tilde{\phi}_n(b)\right\|_2 = \|b\|_2$ because zero-padding does not change the tensor norm, so

$$\hat{b}(\tilde{\phi}_n(a_i)) = \frac{b\left(\tilde{\phi}_n(a_i)\right)}{\left\|b\left(\tilde{\phi}_n(a_i)\right)\right\|_2} = \frac{\tilde{\phi}_n(b(a_i))}{\|b(a_i)\|_2} = \tilde{\phi}_n\left(\frac{b(a_i)}{\|b(a_i)\|_2}\right) = \tilde{\phi}_n\left(\hat{b}(a_i)\right) \,. \tag{60}$$

Thus,

$$\iota_k\left(\tilde{\phi}_n(a_i) \otimes \hat{b}(\tilde{\phi}_n(a_i))\right) = \iota_k\left(\tilde{\phi}_n(a_i) \otimes \tilde{\phi}_n\left(\hat{b}(a_i)\right)\right) = \iota_k\left(a_i \otimes \hat{b}(a_i)\right) \tag{61}$$

Finally, we conclude that,

$$\sigma\left(\left(\tilde{\phi}_n(a_j)\right)_{j=1}^{r}\right) = \tilde{\phi}_n(a_i) - \tilde{\phi}_n(a_{i,\|}) + \frac{\tilde{\sigma}\left(\iota_k\left(\tilde{\phi}_n(a_i) \otimes \hat{b}\left(\tilde{\phi}_n(a_i)\right)\right)\right)}{\left|\iota_k\left(\tilde{\phi}_n(a_i) \otimes \hat{b}\left(\tilde{\phi}_n(a_i)\right)\right)\right|}\tilde{\phi}_n(a_{i,\|}) \tag{62}$$

$$= \tilde{\phi}_n(a_i) - \tilde{\phi}_n(a_{i,\|}) + \frac{\tilde{\sigma}\left(\iota_k\left(a_i \otimes \hat{b}(a_i)\right)\right)}{\left|\iota_k\left(a_i \otimes \hat{b}(a_i)\right)\right|}\tilde{\phi}_n(a_{i,\|}) \tag{63}$$

$$= \tilde{\phi}_n\left(a_i - a_{i,\|} + \frac{\tilde{\sigma}\left(\iota_k\left(a_i \otimes \hat{b}(a_i)\right)\right)}{\left|\iota_k\left(a_i \otimes \hat{b}(a_i)\right)\right|}a_{i,\|}\right) \tag{64}$$

$$= \tilde{\phi}_n\left(\sigma\left((a_j)_{j=1}^{r}\right)\right) . \tag{65}$$

Thus the single pixel function commutes with $\tilde{\phi}_n$, so the Vector neuron nonlinearity is compatible. □

Next we will show that max pooling is compatible.

Proposition E.4. *The max pool layer (Definition D.5) is compatible with* (17).

Proof. Let $A \in \mathcal{A}_{N,n,k,p}$, let b be an integer that divides N, and let $\bar{\imath}_{n+1} \in (\mathbb{Z}/N\mathbb{Z})^{n+1}$. Then

$$\phi_n(\text{max pool}(A, b))(\bar{\imath}_{n+1}) \tag{66}$$

$$= \tilde{\phi}_n(\text{max pool}(A, b)(\pi_n(\bar{\imath}_{n+1}))) \tag{67}$$

$$= \tilde{\phi}_n\left(A\left(b\,\pi_n(\bar{\imath}_{n+1}) + \underset{\bar{a}\in(\mathbb{Z}/b\mathbb{Z})^n}{\arg\max} \|A(b\,\pi_n(\bar{\imath}_{n+1}) + \bar{a})\|_2\right)\right) \tag{68}$$

$$= \tilde{\phi}_n\left(A\left(b\,\pi_n(\bar{\imath}_{n+1}) + \pi_n\left(\underset{\bar{a}\in(\mathbb{Z}/b\mathbb{Z})^{n+1}}{\arg\max} \|A(b\,\pi_n(\bar{\imath}_{n+1}) + \pi_n(\bar{a}))\|_2\right)\right)\right) \tag{69}$$

$$= \tilde{\phi}_n\left(A\left(\pi_n\left(b\,\bar{\imath}_{n+1} + \underset{\bar{a}\in(\mathbb{Z}/b\mathbb{Z})^{n+1}}{\arg\max} \left\|\tilde{\phi}_n(A(\pi_n(b\,\bar{\imath}_{n+1} + \bar{a}))\right\|_2\right)\right)\right) \tag{70}$$

$$= \phi_n(A)\left(b\,\bar{\imath}_{n+1} + \underset{\bar{a}\in(\mathbb{Z}/b\mathbb{Z})^{n+1}}{\arg\max} \|\phi_n(A)(b\,\bar{\imath}_{n+1} + \bar{a})\|_2\right) \tag{71}$$

$$= \text{max pool}(\phi_n(A), b)(\bar{\imath}_{n+1}) . \tag{72}$$

The key insight of this proof is line (69) where we note that the projection π_n of the $\arg\max$ over the $n+1$-dimensional hypercube of side length b pixels is equivalent to the $\arg\max$ over the n-dimensional hypercube. This completes the proof. □

F LIFTING FILTERS

We first define the zero-pad embedding.

Definition F.1 (Zero-pad embedding)**.** The *zero-pad embedding*, denoted $\varphi_n : \mathcal{A}_{M,n,k,p} \to \mathcal{A}_{M,n+1,k,p}$, is defined for all images $C \in \mathcal{A}_{M,n,k,p}$ and all $\bar{\imath}_{n+1} \in (\mathbb{Z}/M\mathbb{Z})^{n+1}$ as

$$\varphi_n(C)(\bar{\imath}_{n+1}) = \begin{cases} \tilde{\phi}_n(C(\pi_n(\bar{\imath}_{n+1}))) & \text{for } \tilde{\phi}_n(\pi_n(\bar{\imath}_{n+1})) = \bar{\imath}_{n+1} \\ 0 & \text{otherwise} \end{cases} . \tag{73}$$

Note that for filters, the indices are centered so that $\bar{\imath} = \{0, \ldots, 0\}$ is the center pixel.

Proposition (Restatement of Proposition 4.2)**.** *Let filters and coefficients be as defined in Definition 4.5. If we let $\mathcal{A}'$ be the distribution of scalar images with independent identically distributed (iid) pixels with $\underset{A'\sim\mathcal{A}'}{E}[A'(\bar{\imath})] = 0$ and $\underset{A'\sim\mathcal{A}'}{E}\left[A'(\bar{\imath})^2\right] = 1$, then* (19) *is equivalent to $\mathcal{L}(\pi_n(\varphi_n(C) - C'))$.*

Proof. Let filters and coefficients be as defined in Definition 4.5. Thus

$$\underset{h\in G_{n+1}}{E}\Big[\underset{A'\sim\mathcal{A}'}{E}[\mathcal{L}((\rho_n(h\cdot A')*C)-\rho_n(h\cdot(A'*C')))]\Big] \tag{74}$$

$$=\underset{h\in G_{n+1}}{E}\Big[\underset{A'\sim\mathcal{A}'}{E}[\mathcal{L}((\rho_n(h\cdot A')*C)-\rho_n((h\cdot A')*C'))]\Big] \tag{75}$$

$$=\underset{A'\sim\mathcal{A}'}{E}[\mathcal{L}((\rho_n(A')*C)-\rho_n(A'*C'))] \tag{76}$$

$$=\underset{A'\sim\mathcal{A}'}{E}[\mathcal{L}(\rho_n(A'*\varphi_n(C))-\rho_n(A'*C'))] \tag{77}$$

$$=\underset{A'\sim\mathcal{A}'}{E}[\mathcal{L}(\rho_n(A'*(\varphi_n(C)-C')))]\,. \tag{78}$$

Now we further break down the loss function $\mathcal{L}$:

$$\underset{A'\sim\mathcal{A}'}{E}[\mathcal{L}(\rho_n(A'*(\varphi_n(C)-C')))] \tag{79}$$

$$=\underset{A'\sim\mathcal{A}'}{E}\left[\frac{1}{N^n}\sum_{\bar{\imath}_n\in(\mathbb{Z}/N\mathbb{Z})^n}\|\rho_n(A'*(\varphi_n(C)-C'))(\bar{\imath}_n)\|_2^2\right] \tag{80}$$

$$=\underset{A'\sim\mathcal{A}'}{E}\left[\frac{1}{N^n}\sum_{\bar{\imath}_n\in(\mathbb{Z}/N\mathbb{Z})^n}\|\rho_n(A'*(\varphi_n(C)-C'))(\bar{\imath}_n)\|_2^2\right] \tag{81}$$

$$=\underset{A'\sim\mathcal{A}'}{E}\left[\frac{1}{N^n}\sum_{\bar{\imath}_d\in(\mathbb{Z}/N\mathbb{Z})^n}\left\|\pi_n\Big((A'*(\varphi_n(C)-C'))\Big(\tilde{\phi}_n(\bar{\imath}_n)\Big)\Big)\right\|_2^2\right] \tag{82}$$

$$=\underset{A'\sim\mathcal{A}'}{E}\left[\frac{1}{N^n}\sum_{\bar{\imath}_n\in(\mathbb{Z}/N\mathbb{Z})^n}\left\|\pi_n\left(\sum_{\bar{a}_{n+1}\in(\mathbb{Z}/M\mathbb{Z})^{n+1}}A'\Big(\tilde{\phi}_n(\bar{\imath}_n)-\bar{a}_{n+1}\Big)\otimes(\varphi_n(C)-C')(\bar{a}_{n+1})\right)\right\|_2^2\right] \tag{83}$$

$$=\underset{A'\sim\mathcal{A}'}{E}\left[\frac{1}{N^n}\sum_{\bar{\imath}_n\in(\mathbb{Z}/N\mathbb{Z})^n}\left\|\sum_{\bar{a}_{n+1}\in(\mathbb{Z}/M\mathbb{Z})^{n+1}}\pi_n\Big(A'\Big(\tilde{\phi}_n(\bar{\imath}_n)-\bar{a}_{n+1}\Big)\Big)\otimes\pi_n((\varphi_n(C)-C')(\bar{a}_{n+1}))\right\|_2^2\right]. \tag{84}$$

Now we note that the squared norm of a sum of vectors can be decomposed as two sums, the first given by a norm on the matching vectors, and the second a double sum over the inner product of the different vectors. In our setting, the norm is still the tensor norm, and the inner product is the full k-contraction. Thus, the first sum is,

$$\sum_{\bar{a}_{n+1}\in(\mathbb{Z}/M\mathbb{Z})^{n+1}}\left\|\pi_n\Big(A'\Big(\tilde{\phi}_n(\bar{\imath}_n)-\bar{a}_{n+1}\Big)\Big)\otimes\pi_n((\varphi_n(C)-C')(\bar{a}_{n+1}))\right\|_2^2 \tag{85}$$

$$=\sum_{\bar{a}_{n+1}\in(\mathbb{Z}/M\mathbb{Z})^{n+1}}A'\Big(\tilde{\phi}_n(\bar{\imath}_n)-\bar{a}_{n+1}\Big)^2\|\pi_n((\varphi_n(C)-C')(\bar{a}_{n+1}))\|_2^2\,, \tag{86}$$

with the equality following from the assumption that A' is a scalar image. Likewise, the second sum, after moving the scalars outside the contraction, is

$$=2\sum_{\bar{a}_{n+1}\neq\bar{b}_{n+1}\in(\mathbb{Z}/M\mathbb{Z})^{n+1}}A'\Big(\tilde{\phi}_n(\bar{\imath}_n)-\bar{a}_{n+1}\Big)A'\Big(\tilde{\phi}_n(\bar{\imath}_n)-\bar{b}_{n+1}\Big) \tag{87}$$

$$\iota_k\big(\pi_n((\varphi_n(C)-C')(\bar{a}_{n+1}))\otimes\pi_n\big((\varphi_n(C)-C')(\bar{b}_{n+1})\big)\big) \tag{88}$$

By linearity we can now directly apply the expectation to the terms with A'. Since $\underset{A'\sim\mathcal{A}'}{E}\Big[A'\Big(\tilde{\phi}_n(\bar{\imath}_n)-\bar{a}_{n+1}\Big)\Big]=0$ by assumption, the second sum is 0. By assumption $\underset{A'\sim\mathcal{A}'}{E}\Big[A'\Big(\tilde{\phi}_n(\bar{\imath}_n)-\bar{a}_{n+1}\Big)^2\Big]=1$ in the first sum. If we plug these results back into (84) we

have:

$$\underset{A'\sim\mathcal{A}'}{E}\left[\frac{1}{N^n}\sum_{\bar{\imath}_n\in(\mathbb{Z}/N\mathbb{Z})^n}\left\|\sum_{\bar{a}_{n+1}\in(\mathbb{Z}/M\mathbb{Z})^{n+1}}\pi_n\Big(A'\Big(\tilde{\phi}_n(\bar{\imath}_n)-\bar{a}_{n+1}\Big)\Big)\otimes\pi_n((\varphi_n(C)-C')(\bar{a}_{n+1}))\right\|_2^2\right] \tag{89}$$

$$=\frac{1}{N^n}\sum_{\bar{\imath}_n\in(\mathbb{Z}/N\mathbb{Z})^n}\sum_{\bar{a}_{n+1}\in(\mathbb{Z}/M\mathbb{Z})^{n+1}}\|\pi_n((\varphi_n(C)-C')(\bar{a}_{n+1}))\|_2^2 \tag{90}$$

$$=\sum_{\bar{a}_{n+1}\in(\mathbb{Z}/M\mathbb{Z})^{n+1}}\|\pi_n((\varphi_n(C)-C')(\bar{a}_{n+1}))\|_2^2 \tag{91}$$

$$=\mathcal{L}(\pi_n(\varphi_n(C)-C'))\,. \tag{92}$$

This completes the proof. □

We now show the coefficients derived from Proposition 4.2.

Type	Dimension	Weights		
Scalar filters 1D	α_0	α_1		
2D Nepo	$\alpha_0-\frac{2}{3}\alpha_1$	$\frac{1}{3}\alpha_1$	$\frac{1}{3}\alpha_1$	
2D Copy	$\alpha_0+2\alpha_1$	$-\alpha_1$	α_1	
2D Zeros	$\alpha_0-2\alpha_1$	α_1	0	
Scalar filters 2D	α_0	α_1	α_2	
3D	$\alpha_0-\frac{10}{9}\alpha_1+\frac{4}{9}\alpha_2$	$\frac{5}{9}\alpha_1-\frac{2}{9}\alpha_2$	$\frac{2}{9}\alpha_1+\frac{1}{9}\alpha_2$	$-\frac{1}{9}\alpha_1+\frac{4}{9}\alpha_2$
Vector filters 2D	α_0	α_1		
3D	$\alpha_0-\frac{2}{3}\alpha_1$	$\frac{1}{3}\alpha_1$	$\frac{1}{3}\alpha_1$	
2-tensor filters 2D	α_0	α_1	α_2	α_3
3D	$\alpha_0-\frac{2}{3}\alpha_1+\frac{2}{3}\alpha_3$	$\frac{5}{9}\alpha_1-\frac{2}{9}\alpha_2+\frac{1}{9}\alpha_3$	$\frac{1}{9}\alpha_1+\frac{2}{9}\alpha_2-\frac{1}{9}\alpha_3$	$\frac{1}{3}\alpha_2$
2D	α_4			
3D	α_4	0	$-\frac{4}{9}\alpha_1+\frac{4}{9}\alpha_2-\frac{8}{9}\alpha_3$	$\frac{2}{9}\alpha_1-\frac{2}{9}\alpha_2-\frac{2}{9}\alpha_3$

Table 2: Coefficients for positive parity filters with side length 3. For each set of two, the first row is the weights for the lower dimensional filters, in order from the center of the filter outwards. The second row is the weights for the higher dimensional filters in terms of the weights of the lower dimensional filters. The 2D Copy and 2D Zeros coefficient transfer strategies are explained in Appendix G.3.

G Experimental Details

G.1 Data

We use data from three PDEs, the heat equation (left), Burgers' equation (right):

$$\frac{du}{dt}=\Delta u\,,\qquad\qquad \frac{du}{dt}=-b\frac{1}{2}\nabla\cdot(u\otimes u)+\nu\nabla^2u\,, \tag{93}$$

and the compressible Navier–Stokes equations:

$$\partial_t\rho+\nabla\cdot(\rho\vec{v})=0,\quad \rho(\partial_t\vec{v}+\vec{v}\cdot\nabla\vec{v})=-\nabla p+\eta\Delta\vec{v}+(\zeta+\eta/3)\nabla(\nabla\cdot\vec{v}), \tag{94}$$

$$\partial_t\big(\epsilon+\rho v^2/2\big)+\nabla\cdot\big[\big(p+\epsilon+\rho v^2/2\big)\vec{v}-\vec{v}\cdot\sigma'\big]=0 \tag{95}$$

These are equations are used to generate fields which we discretize to images. All images had a side length of 64 pixels. The initial heat data u_0 is generated with a uniform distribution from $-\sqrt{3}$ to $\sqrt{3}$ in each pixel, and then the value at time $t=1$ was calculated using the analytic heat kernel with all points, where the distance to each point was the closest distance on the torus.

The data for Burgers' equation was generated by the Apebench package Koehler et al. (2024) using the difficulty scenario. The diffusion coefficient for 2D was 0.75 and for 3D was 2.25, while the

convection coefficient was -0.75 for 2D and -2.25 for 3D. This was to undue the automatic scaling of the coefficients by dimension in the difficulty scenario so that the same parameters were used across dimensions. Each trajectory was run for 50 time steps with 0 warmup steps. Each trajectory of 50 steps was then split into 49 input output pairs of one timestep as part of the training data.

The data for the compressible Navier–Stokes equations is generated by PDEBench Takamoto et al. (2024). Although PDEBench has many datasets including Navier–Stokes in 2D and 3D, we require that exactly the same hyperparameters are used to generate data across dimensions. Thus we used their code to generate additional data with shear and bulk viscosity of 1e-08, and Mach number 0.1. Each trajectory was run for 21 time steps which where split into 17 data points of 4 input steps and 1 output step.

The size of the datasets in number of float values for n dimensions, b trajectories, t time steps, image side length N, s scalar channels, and v vector channels is $btN^n(s + vn)$.

G.2 Model and Training

We used a UNet Ronneberger et al. (2015) as that was the best performing model in Gregory et al. (2025). We used filter side lengths of 3 with four downsampling blocks by max pooling and two convolution layers between each downsample. For nonlinearities we used the Vector Neuron Deng et al. (2021) described in Gregory et al. (2025) with GeLU nonlinearity. We used 48 channels as the convolution default which doubles every downsample.

We used two metrics during training and evaluation. The first is the sum of mean squared error (SMSE) which takes the mean squared error over each field, then takes the sum over the fields. If A_j is the true $k_{(p)}$-tensor image for field j and $\hat{A}_j$ is our predicted $k_{(p)}$-tensor image for field j, then the $\mathcal{L}_{\text{SMSE}}$ is defined as,

$$\mathcal{L}_{\text{SMSE}}\left(\{A_j\}_{j=1}^J, \left\{\hat{A}_j\right\}_{j=1}^J\right) = \sum_j \frac{1}{N^n} \sum_{\bar{\imath}} \left\|A_j(\bar{\imath}) - \hat{A}_j(\bar{\imath})\right\|_2^2, \tag{96}$$

where $\|\cdot\|_2$ is the tensor norm. The second metric is the relative error which is defined as,

$$\mathcal{L}_{\text{rel}}\left(\{A_j\}_{j=1}^J, \left\{\hat{A}_j\right\}_{j=1}^J\right) = \frac{1}{J} \sum_j \frac{\sqrt{\sum_{\bar{\imath}} \left\|A_j(\bar{\imath}) - \hat{A}_j(\bar{\imath})\right\|_2^2}}{\sqrt{\sum_{\bar{\imath}} \|A_j(\bar{\imath})\|_2^2 + \epsilon}} * 100\,, \tag{97}$$

where ϵ is some fixed small value to prevent division by 0 numerical issues, and we multiply by 100 to get the error as a percent. We set $\epsilon = \texttt{1e-5}$. We found that calculating the relative error per pixel led to numerical instability during training, so we use the above version which can be viewed as taking the norm over the entire image for a field. We train all models with the relative error, and report results for the relative error and SMSE.

We followed a similar training regime as in Gregory et al. (2025). We train for 50 epochs using the AdamW optimizer Loshchilov & Hutter (2019) with a weight decay of `1e-5` and a cosine decay schedule Loshchilov & Hutter (2017) to `1e-7` with 5 epochs of warmup from `1e-8`. Peak learning rates were tuned for each problem for each dimension and for each different number of tuning points. Learning rates were chosen from the set $\{1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3\}$ based on performance on a validation set of 128 points for the heat equation, 8 trajectories for Burgers' equation, or 32 trajectories for Navier–Stokes. For the ablation over the alternative coefficient transfer strategies, the same learning rates tuned for our warm-start strategy were used for all the strategies. It is possible that the performance could be improved by individually tuning the learning rates for each strategy, but the results for 0 tuning points shows that our warm-start strategy is likely to remain the best. The resulting learning rates are shown in Table 3.

We trained on a single RTX 6000 Ada GPU with a batch size of 8 for all experiments except for the Burgers' and Navier–Stokes 3D experiments where we used 2 GPUs with a batch size of 2 per GPU for an effective batch size of 4.

G.3 Warm-start Ablation

We explored whether the warm-start coefficient transfer, Definition 4.5 and Proposition 4.2, is the correct equation to optimize. We experimented with two reasonable alternative strategies that satisfied

PDE	Models	Params	1	4	8	32	128
Heat	1D Pre-train	3,926,737					1e−3
	2D Baseline	5,691,697	5e−4	5e−4	5e−4	5e−4	5e−4
	2D Fine-tune	5,691,697	5e−4	5e−4	5e−4	5e−4	5e−4
Burgers	2D Pre-train	11,576,305			1e−4		
	3D Baseline	16,871,185	1e−4	1e−4	1e−4		
	3D Fine-tune	16,871,185	1e−4	1e−4	1e−4		
N-S	2D Pre-train	25,114,131					1e−4
	3D Baseline	35,705,235	5e−4	1e−4	1e−4	1e−4	
	3D Fine-tune	35,705,235	1e−4	1e−4	1e−4	5e−5	

Table 3: Pre-training and fine-tuning learning rates for the number of data points or trajectories.

compatibility but made different choices with the free parameter. In the Copy strategy, we set the new coefficient for the outermost weight equal to the old coefficient for the outermost weight, then adjusted the other weights to maintain compatibility. In the Zeros strategy, we set the new weight to 0 and adjusted the others likewise. We tried these strategies for the heat equation, with the results shown in Fig. 3 and Table 4. Clearly our warm-start strategy performs better. It is also interesting to note that in some cases, and particularly for the Copy strategy, a naive weight transfer strategy is worse than no pre-training at all. This is true even though all strategies satisfy compatibility.

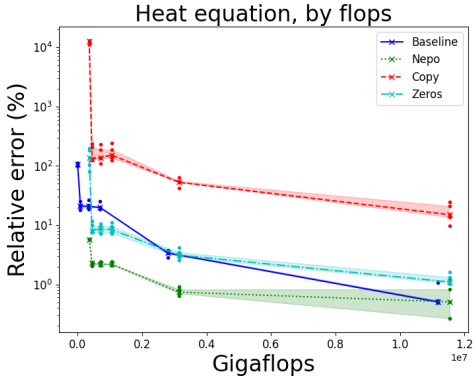


Figure 3: The median relative error performance of each warm-start strategy with the heat equation over varying numbers of fine tuning points, indexed by the total number of gigaflops to pre-train and fine-tune the models. Note the order of magnitude of the x-tick values, and the log values of the y-axis. Each experiment was run 5 times, with the lines representing the median value, the shaded region representing the interquartile range, and a dot for each value.

PDE	Models	0	1	4	8	32	128
Heat	Baseline	105.3%	21.0%	20.8%	19.8%	3.3%	**0.5%**
	Copy	12,797.3%	131.6%	138.5%	152.5%	52.7%	15.0%
	Zeros	138.3%	8.1%	8.1%	8.4%	3.1%	1.0%
	Nepo	**5.7%**	**2.2%**	**2.2%**	**2.1%**	**0.7%**	**0.5%**

Table 4: The median relative error performance over 5 trials of each warm-start strategy with the heat equation over varying numbers of fine tuning points. The column labels represent the number of fine-tuning points. With higher precision, the baseline method does better than Nepo by 0.001%.

G.4 Additional Results

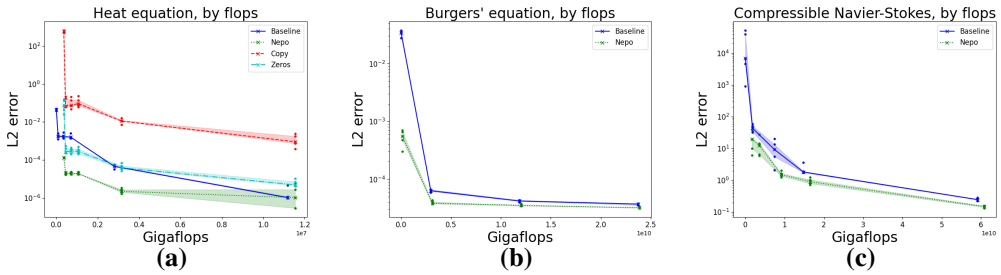


Figure 4: ℓ_2 error performance of each PDE over varying numbers of fine tuning points, indexed by the total number of gigaflops to pre-train and fine-tune the models. Note the order of magnitude of the x-tick values, and the log values of the y-axis. Each experiment was run 5 times, with the lines representing the median value, the shaded region representing the interquartile range, and a dot for each value. **(a)** The heat equation trained on 0, 1, 4, 8, 32, and 128 fine-tuning points. **(b)** Burgers' equation trained on 0, 1, 4, and 8 trajectories. **(c)** The compressible Navier–Stokes equations trained on 0, 1, 4, 8, and 32 trajectories.

PDE	Models	0	1	4	8	32	128
Heat	Baseline	4.4e-02	1.7e-03	1.7e-03	1.5e-03	4.5e-05	**1.0e-06**
	Copy	6.5e+02	6.9e-02	7.6e-02	9.3e-02	1.1e-02	9.0e-04
	Zeros	7.6e-02	2.6e-04	2.9e-04	2.8e-04	3.8e-05	4.8e-06
	Nepo	**1.3e-04**	**2.0e-05**	**1.9e-05**	**1.9e-05**	**2.2e-06**	**1.0e-06**
Burgers	Baseline	3.3e-02	6.3e-05	4.1e-05	3.7e-05	Time out	Time out
	Nepo	**5.5e-04**	**3.8e-05**	**3.5e-05**	**3.2e-05**	Time out	Time out
N-S	Baseline	6.9e+03	4.3e+01	9.3e+00	1.8e+00	2.4e-01	Time out
	Nepo	**1.9e+01**	**1.2e+01**	**1.5e+00**	**9.0e-01**	**1.4e-01**	Time out

Table 5: The median ℓ_2 error performance over 5 trials of each PDE over varying numbers of fine tuning points. N-S refers to compressible Navier–Stokes. For the heat equation, the column labels represent the number of fine-tuning points, while for Burgers and Navier–Stokes, they refer to the number of trajectories of 17 and 49 time steps each, respectively. The positions in the table labeled "Time out" were training loops that would have taken longer than 15 hours per trial. With higher precision, the baseline method does better than Nepo for the heat equation by 1e-09.

PDE	Models	n=0	n=1	n=4	n=8	n=32	n=128
Heat	Baseline	0	8.732e+04	3.493e+05	6.986e+05	2.794e+06	1.117e+07
	Nepo	3.594e+05	4.467e+05	7.087e+05	1.058e+06	3.153e+06	1.153e+07
Burgers	Baseline	0	2.965e+09	1.186e+10	2.372e+10		
	Nepo	1.396e+08	3.105e+09	1.200e+10	2.386e+10		
N-S	Baseline	0	1.846e+09	7.386e+09	1.477e+10	5.908e+10	
	Nepo	1.724e+09	3.571e+09	9.110e+09	1.649e+10	6.081e+10	

Table 6: Estimated gigaflops of training process. Nepo warm-start includes the time to pre-train on lower dimensional data and convert the model.