Proceedings of Machine Learning Research 277:1–25, 2025 Machine Learning Meets Differential Equations: From Theory to Applications

ExtremONet: Extreme-Learning-based Neural Operator for identifying dynamical systems.

Jari Beysen [1], Floriano Tori [1]
[1] *Data Analytics Laboratory, Vrije Universiteit Brussel*
jari.beysen@gmail.com, floriano.tori@vub.be

Editors: Cecília Coelho, Bernd Zimmering, M. Fernanda P. Costa, Luís L. Ferrás, Oliver Niggemann

Abstract

The DeepONet, based on the Universal Approximation Theorem for Operators (UATO), made a significant impact on Deep Learning research due to its ability to learn maps between function spaces instead of between vector spaces like traditional Neural Networks. However, DeepONets are computationally expensive to train. To address this we introduce the *ExtremONet*: an Extreme-Learning-Machine-based variation of the DeepONet, capable of one-step learning maps between function spaces. We show that ExtremONets approach DeepONets in training error whilst displaying lower generalization error, and training between two to four orders of magnitude faster on small datasets. Our work represents an important step towards efficient dynamical modeling using Machine Learning. We conclude our analysis with an exploration of the ExtremONet's Out-of-Distribution Generalization capabilities.

Keywords: Neural Operator, Extreme-Learning-Machines, Ordinary Differential Equations, Partial Differential Equations, Out-of-Distribution Generalization

1. Introduction

Machine Learning has been a valuable tool for the exact sciences (Alves, 2006; Karagiorgi et al., 2021; Jumper et al., 2021; Huang et al., 2022). A notable example is the principle of Physics Informed Neural Networks (PINNs) (Raissi et al., 2019) which demonstrated that including knowledge of the underlying dynamics of a system in the loss function increases performance, and drastically reduces the amount of data necessary to generalize. PINNs bridged the gap between data-driven and model-driven approaches. They have proven to be reliable tools to solve differential equations and have therefore been used extensively to model Non-Linear Dynamics (Roehrl et al., 2020; Na et al., 2023; Antonelo et al., 2024). In its wake the new field of Neural Operators arose (Viswanath et al., 2023), established by the DeepONet architecture (Lu et al., 2021), which can be viewed as a generalization of Neural Networks. In the paradigm of Neural Operators, models learn a map between function spaces instead of between vector spaces. Where PINNs used information about the system to aid optimization, Neural Operators encode mathematical structure into the Neural Network architecture itself, creating an application-specific method. Neural Operator training is in general computationally expensive. In this work we propose a novel Extreme-Learning-Machine-based Neural Operator. We show that Extreme-Learning-Machines, which use the power of random networks, can be used to efficiently approximate operators whilst maintaining the performance of the DeepONet on small datasets.

This work is divided into 6 sections: Introduction, Related work, Methodology, Experiment details, Results, and Conclusion. A table of notations and symbols is included in Appendix A. Notational conventions & mathematical symbols.

2. Related work

Dynamics In this work we consider, first- and second order Ordinary Differential Equations (ODEs) and first- and second order Partial Differential Equations (PDEs). To solve such systems, some initial conditions (and/or boundary conditions for PDEs) need to be specified. These are referred to as initial value problems. In the case of an ODEs we impose that $y(t) : [a_t, b_t] \mapsto \mathbb{R}^d$; and in the case of PDEs we impose that $y(x,t) : [a_x, b_x]^n \times [a_t, b_t] \mapsto \mathbb{R}^d$. If the underlying structure in a differential equation is (partially) unknown, then numerical techniques are not easily applicable as they require complete knowledge of the system. System identification methods try to derive the underlying dynamics by studying input-output observations (Pillonetto et al., 2025). Predicting the behaviour of a system when the underlying dynamics are not identical between realizations is difficult. Such problems are much more general in nature, and can be denoted as:

$$\begin{cases} f(y(t), t, d_t y, d_t^2 y) = u(y,t), \\ y(a) = y_a, \\ d_y(t)_{t=b} = y'_b \end{cases}, \quad \begin{cases} f(y(x,t), x, t, \partial_t y, \partial_x y, \partial_t^2 y, \partial_x^2 y, \partial_t \partial_x y) = u(y,x,t), \\ y(x,a) = y_a(x) \mid y(a,t) = y_a(t), \\ \partial_t y(x,t)_{t=b} = y'_b(x) \mid \partial_x y(x,t)_{x=b} = y'_b(t) \end{cases} \tag{1}$$

in the case of ODEs and PDEs respectively. In these problem settings, we require that $u(\cdot) \in V$, which is a compact subset of $C[a_t, b_t]$, $C[a_x, b_x]^n$, or $C[a_y, b_y]^d$; the last constraint is hard to enforce, and therefore will be ignored in experimental situations. In the original DeepONet paper, only compact subsets of $C[a_t, b_t]$ or $C[a_x, b_x]^n$ were considered (Lu et al., 2021), which we will expand upon by studying the generalized problem defined above, where we also included functions acting on the dependent variable y. Note that the last type is not covered by the UATO, therefore there are no approximation guarantees. We imposed that the trajectory of $y(\cdot)$ is dependent on $u(\cdot)$, which cannot be learned via traditional means. An operator is a map defined on a compact set $V \subset C(K_1)$, where $C(K_1)$ are all continuous functions defined on some compact set of a Banach space $K_1 \subset X$, where X is the Banach space in question. The operator $G : V \to C(K_2)$ maps the compact space V into another space of all continuous functions $C(K_2)$ of a compact space $K_2 \subset Y$.

Neural Operators Neural Operators require a function $u(\cdot)$ as input. However, function spaces are infinite-dimensional vector spaces and thus cannot be trained on directly. The approximation theorem (Tianping Chen and Hong Chen, 1995) shows that a finite-dimensional sampling suffices (see Appendix B). These sampling points are called *sensor locations*, and the function values at those points are called *sensors*. The features s denote the sensor locations, with sensor values $u(s)$. The theorem implies the existence of basis functions $\psi(\cdot)$, $\phi(\cdot)$, integers p, m, and real numbers θ_{ijk} such that for a d-dimensional output $y(t)$:

$$y_i(t) = G_i(u(s))(t) \approx \mathscr{G}_i(u(s))(t) = \sum_{j=1}^{p} \sum_{k=1}^{m} \theta_{ijk} \psi_{ijk}(u(s)) \phi_{ik}(t). \tag{2}$$

If $\psi_{ijk}(u(s)) = a_i$, this reduces to a function basis expansion. Here, $\psi(\cdot)$ represents the input basis and $\phi(\cdot)$ the output basis. In this work, the Normalized Mean Squared Error (NMSE) $\mathcal{L}$ is used as a loss function, providing scale-invariant performance comparison across problem domains. Operator samples are triplets $\{(t_i, u_i(s), y_i) \mid i \in [1, N]\}$ where $t \in \mathbb{R}^{N\times D}$ (with $D = 1$), $y \in \mathbb{R}^{N\times d}$, and N is the sample count. The input-output dimensionality varies by setting. As the theorem is non-constructive regarding sensor locations, we sample them uniformly from the function space's domain. Letting r be the number of sensors of dimension v, each mapped to w dimensions: then $u_i(s) \in \mathbb{R}^{r\times w}$. We unroll this as $u(s) \in \mathbb{R}^{N\times r\times w} \to u'(s) \in \mathbb{R}^{N\times r'}$, with $r' = rw$. For simplicity, we omit the prime hereafter. The original Neural Operator (DeepONet) (Lu et al., 2021), satisfies the approximation form of the UATO:

$$G_i(u(s))(t) \approx \mathscr{G}_i(u(s))(t) = \sum_{j=1}^{p} \mathscr{B}_{ij}(u(s))\mathscr{T}_{ij}(t) + \theta_i^{(1)}, \tag{3}$$

where two fully connected networks (Branch $\mathscr{B}(u(s))$ and Trunk $\mathscr{T}(t)$) produce $d \times p$-sized outputs per time point.[1] The Branch network requires finite-dimensional measurements of the function $u(\cdot)$ as input, and the Trunk network requires time points as input in the case of dynamics.[2] This *unstacked* DeepONet contrasts with a *stacked* variant, where each j in p corresponds to a separate subnetwork producing rank-1 tensors. Numerous variations of Neural Operators have been developed. Among integral-operator-based architectures, differences lie in kernel parametrization (Kovachki et al., 2021). Notable examples include: Fourier Neural Operator (FNO) (Li et al., 2020), which leverages convolution in Fourier space; Laplace Neural Operator (LNO) (Cao et al., 2023), which uses a Laplace-transform-based kernel suitable for transient response modeling. The DeepONet's flexibility allows for the substitution feedforward networks with other modules in both Trunk and Branch, resulting in the emergence of various architectures: MultiAuto-DeepONet (Zhang et al., 2022), DeepGraphONet (Sun et al., 2022), sequential operator networks (He et al., 2024), and DeepOKAN (Abueidda et al., 2024).

Extreme-Learning-Machines Neural Networks are slow to train as they require many function calls for the optimizer to converge to a minimum. Extreme-Learning-Machines (ELMs), in contrast, can be trained in one step. ELMs (Huang et al., 2006) are very wide, randomly initialized single-layer Neural Networks; the only trainable parameters reside in the linear readout, which is trainable via Linear Regression (or a variation). The foundational proof for Extreme-Learning-Machines by Huang et al. (2006) is controversial, and was recently refuted by Perfilievaa et al. (2024). The work by Igelnik and Yoh-Han Pao (1995), which provided the foundation for Huang et al. did provide a non-constructive proof (regarding the training algorithm) showing that there exists some readout parametrization of random single-layer Neural Networks which can approximate any continuous function defined on a compact set, making them universal approximators (see Appendix B, Theorem 2). Extreme-Learning-Machines have empirically demonstrated remarkable performance when used as a Neural Network alternative (Huérfano-Maldonado et al., 2023). In this work,

1. Typically implemented via a flat dp-size output reshaped into $d \times p$.
2. This can be any other vector space $y(\cdot)$ has as domain.

we shall make reasonable theoretical assumptions which allow us to construct Extreme-Learning-based models. The standard ELM architecture takes the following form:

$$\hat{y}(x) = \mathscr{N}_E(x)\theta^{(0)} + \theta^{(1)} \quad \text{where} \quad \mathscr{N}_E(x) = f(xW + b)\,. \tag{4}$$

In this work W is a uniformly sampled matrix $W \sim \mathcal{U}(-\sigma/\sqrt{\dim(x)}, \sigma/\sqrt{\dim(x)})$, b is a normally sampled vector $b \sim \mathcal{N}(0, \sigma)$. The type of distribution the values should be sampled from depends on the situation; normal and uniform distributions are seen as the standard. Large values of σ are ideal for complex problems, and small values for less complex (almost linear) problems (Demidova and Zhuravlev, 2024). As mentioned before, the only trainable parameters are $\theta^{(0)}$ and $\theta^{(1)}$. This network lifts the input data to a latent output dimension m, also known as the width of the Extreme-Learning-Machine. ELMs are typically much higher in dimension than the input dimension to ensure good accuracy. In the case of functional data, one can reasonably assume that, if there are enough sensor locations, that there will be redundant features, which implies that there might be some unnecessary risk that the *curse of dimensionality* (Bellman and Kalaba, 1959) will prevent generalization due to a lack of data. One can artificially reduce the dimensionality of the input without removing features by imposing a sparsity on W, which will be initialized such that each latent dimension feature is connected to c input features, which can easily be imposed on the matrix W by setting $r - c$ random indices i to 0 for every j in W_{ij}, where r is the input dimension. One also has to adjust the initialization $W \sim \mathcal{U}(-\sigma/\sqrt{c}, \sigma/\sqrt{c})$ to accommodate the effective amount of input features. The main drawback of Extreme Learning is that the performance is dependent on the magnitude of σ and c, which are hyperparameters and therefore should be fine-tuned outside of the training loop. Another problem ELMs face is their tendency to overfit due to their large output dimension. The standard approach is to use Ridge Regression (Li and Niu, 2013) instead of Linear Regression, which can be defined as:

$$\underset{\theta}{\operatorname{argmin}}\, \mathbb{E}\{\|y(t) - [\mathscr{N}_E(t)||1]\theta^{(0)} - \theta^{(1)}\|^2 + \lambda\|[\theta^{(0)}|\theta^{(1)}]\|^2\} \tag{5}$$

$$\Rightarrow [\theta^{(0)}|\theta^{(1)}] = ([\mathscr{N}_E(t)||1]^t[\mathscr{N}_E(t)||1] + \lambda\mathbb{I})^{-1}[\mathscr{N}_E(t)||1]^t y(t), \tag{6}$$

where $[\cdot|\cdot]$ refers to a concatenation over the first index and $[\cdot||\cdot]$ over the second index. λ provides a penalty (regularization) on the l_2-norm of the parameters, penalizing overuse of $\mathscr{N}_E(t)$ features. Determining the optimal λ to optimally reduce the generalization error cannot be done internally, and it is therefore a hyperparameter. This parameter can have a large impact on the performance and should always be optimized for the problem setting by selecting the parameter that resulted in the lowest validation loss. A heuristic approach will be implemented in this work to minimize validation.[3] Brent's method (Brent, 1972) is a popular hybrid/heuristic scalar minimization algorithm based on inverse quadratic interpolation, the bisection method, and the secant method; it is pre-implemented in many scientific coding libraries.

3. 20% Split of the training set.

3. Methodology

The DeepONet in its standard form unfortunately does not permit Extreme Learning because one cannot apply Linear/Ridge Regression to $\mathscr{B}_E$ and $\mathscr{T}_E$ separately. Therefore, we suggest a new Neural Operator that does allow Extreme-Learning-Machines to be inserted. We propose the novel *Extreme Operator Network* (ExtremONet):

$$\begin{aligned}\mathscr{B}_E(u(s)) &= f^{(B)}(u(s)W^{(B)} + b^{(B)})\\ \mathscr{T}_E(t) &= f^{(T)}(tW^{(T)} + b^{(T)})\\ \mathscr{G}(u(s))(t) &= (\mathscr{B}_E(u(s)) \odot \mathscr{T}_E(t))\theta^{(0)} + \theta^{(1)},\end{aligned} \tag{7}$$

which is similar to the DeepONet, but can be trained via Ridge Regression instead. The derivation of the ExtremONet can be found in Appendix C. This novel model has 5 hyperparameters, namely width m, trunk scale σ_t, trunk connectivity c_t, branch scale σ_b, and branch connectivity c_b (dependence are explored in Appendix F). Because time is one-dimensional there is no option but setting $c_t = 1$. The reconfiguration of Eq. 2 can be viewed as removing bias from the architecture, and the introduction of more variance as dictated by the *Bias-Variance trade-off* (BVT). More recent investigations of this hypothesis have shown that it is not per se a zero-sum game (Rocks and Mehta, 2022), as is also demonstrated by the *double-descent* phenomenon. In this work we allowed for Linear Regression via overparametrization (of the architecture) because we ignored the conditions of $\theta^{(0)}$. This might have negative effects on the generalization performance of the derived model. The work by Rocks and Mehta (2022) demonstrated that in the overparametrized region random non-linear feature models (ELMs) do benefit from double-descent. In the underparametrized region, Ridge Regression functions as a generalization error minimization technique, and is able to control the bias and variance by adjusting λ (Feng et al., 2022). In our case it will handle generalization error caused by the extreme width, and the overparametrization of the UATO.

4. Experiment details

ODE systems We define parametrized systems as initial value problems where $u(\cdot) \to u_\alpha(\cdot)$ depends on a finite number a of parameters α, forming a compact set $V_\alpha \subset C(K_1)$ or $V_\alpha \subset C(2)$ with $\alpha \in A \subset \mathbb{R}^a$; for instance, the polynomials $u_\alpha(y) \subset K^q_\alpha[y]$. This polynomial structure generates $a = \sum_{j=0}^{q} \binom{d+j-1}{j} = \binom{d+q}{q}$ parameters for a d-dimensional input. For testing, we consider $d_t y = u(y)$ with a compact subset $u \in K^2_\alpha[y]$, exemplified by the Lorenz-63 system (Lorenz, 1963):

$$d_t y_0 = \alpha_0(y_1 - y_0), \quad d_t y_1 = y_0(\alpha_1 - y_2) - y_1, \quad d_t y_2 = y_0 y_1 - \alpha_2 y_2. \tag{8}$$

We evaluate both chaotic and non-chaotic regimes. Additionally, we learn operators on non-parametric spaces like Gaussian Random Fields (GRFs) (Rasmussen and Williams, 2005), where $u_t \sim \mathcal{N}(0, k^l(t,t))$, with the kernel $k^l_{ij}(t,t) = e^{-(t_i - t_j)^2/2l^2}$. Functions are generated via cubic interpolation. Mixed kernels $k^l(y,t)$ are not considered. Solutions from Lorenz-63 are harder to learn due to sensitivity to α in the chaotic regime, and the fact that is not covered under the UATO. For temporal function spaces, we consider a driven pendulum:

$$d_t^2 y = -\sin(y) + u(t). \tag{9}$$

PDE systems We analogously treat PDE operators and consider the 1D Sine-Gordon[4] (Kuksin and Bambusi, 2006) and diffusion equations.

$$\partial_t^2 y - \partial_x^2 y = u_\alpha(y), \quad u_\alpha(y) = \sum_{i=1}^{q} \sin(\alpha_i \pi y), \tag{10}$$

$$\partial_t y + \partial_x^2 y = u(x), \quad u(x) \sim \mathcal{N}(0, k_l(x,x)). \tag{11}$$

5. Results

General comparison small (1)-, medium (2)-, and large (3) DeepONet was compared against a small (1)-, medium (2)-, and large (3) ExtremONet (see Appendix E, Table 2). The models were trained on 10 000 total samples (see Appendix D) with 120 sensors[5], which was repeated 10 times. The DeepONets were single-layer 100-wide, 2-layers 200-wide, 3-layers 500 wide , with a p-size of 10, 10, and 20 respectively. All DeepONets are in the overparametrized region because the Branch networks require a lot of parameters to accommodate the large input size, meanwhile half of the ExtremONets are in the under-parametrized region (also one model with a ratio of approximately 1). We trained until convergence with the AdamW optimizer (Loshchilov and Hutter, 2017) with a learning rate of $2 \cdot 10^{-2}$, $5 \cdot 10^{-3}$, and $1 \cdot 10^{-3}$ and weight decay of 10^{-5}. In this work we defined convergence via a stopping condition. For the DeepONet, training is stopped early if the following condition is no longer met:

$$\frac{1}{k} \log_{10} \left(\Pi_{i=0}^{k-1} \frac{\mathcal{L}_{n-i}^{val}}{\mathcal{L}_{n-k-1-i}^{val}} \right) < -\epsilon. \tag{12}$$

Intuitively this forces the training loop to stop when the magnitude of the slope of the current validation loss is less than ϵ, which will be set to 0 and $k = 500$ to ensure a robust estimate of the onset of overfitting. The ExtremONets were 100-, 300-, and 1000-wide respectively with $\sigma_t = 10$, $\sigma_b = 0.1$, and $c_b = 1$, which was the standard parametrization unless specified otherwise. These standard values were chosen to reflect the hyperparameter exploration findings in Appendix F. λ was found via Brent's algorithm, which ran for 100 iterations, or until $\Delta\lambda < 10^{-2}$. For all models we used a hyperbolic tangent as activation function and the DeepONets had a linear readout. All experiments were repeated 30 times. Our results show that the ExtremONet outperformed the DeepONet on the pendulum- and Sine-Gordon dataset (all model sizes). The DeepONet outperformed the ExtremONet on the Lorenz-63 (only the medium-sized DeepONet) and on the diffusion dataset (see Appendix E; Appendix F, Figure 2). The DeepONets training cycles frequently ended prematurely because it was only able to marginally reduce the validation error before the model started overfitting, and therefore triggering the stopping condition. An important behavioural distinction is the generalization error throughout their respective training iterations. The ExtremONet validation error is almost perfectly correlated to the training error, which is not the case for the large DeepONet, which overfit on all problem settings. This is most likely because the much smaller hypothesis space of the ExtremONet. The

4. Also not covered under the UATO.
5. The amount of sensors used is always 120, the maximum needed according to Appendix F, Figure 12.

lower relative generalization error (see Appendix F, Figure 4) motivates an exploration of Out-of-Distribution Generalization which we discuss in the paragraphs below. Finally, our results highlight one of the key benefits of ExtremONets. Over all problem settings ExtremONets took between 100-10000x less time to train, whilst keeping the prediction time almost identical (see Appendix F, Figure 3). Additionally, Ridge Regression is a solved optimization problem, which guarantees that the found solutions are global minima. If the loss landscape is complex enough, then the guaranteed global minimum in combination with the smaller hypothesis space might explain the performance increase over the DeepONet on small datasets.

Width dependence Ridge regression is used to minimize validation loss with respect to λ. This creates a plateau in the train- and test loss with respect to the model width making overfitting almost impossible; this needs to be tested together with the train- and prediction time with respect to the model width m (see Appendix F, Figure 11). The train- and test loss should display power-law behaviour in the bottlenecked region (Maloney et al., 2022; Atanasov et al., 2025); we assume that a power-law behaviour (i.e. $(ax^b + c)$) occurs up until some lower bound at $m \to \infty$. In Table 5 the power-law fits and the relative generalization scaling demonstrates that Ridge Regression is an effective method to prevent overfitting. These fits allow us to approximate the minimum possible train- and test error ($\mathcal{L}(\infty) \to c$) for every system,[6] and therefore an approximate relative generalization error $(c_{test} - c_{train})/c_{train}$. The approximate relative generalization errors are: 0.04 ± 0; 0.10 ± 0; 0.21 ± 0; 0.21 ± 0, which are less than the median relative generalization error of the medium-sized DeepONet. In Theorem 3 by Igelnik and Yoh-Han Pao (1995) it was demonstrated that

	$\mathcal{L}_{train}$	$\mathcal{L}_{test}$	$b_{test} - b_{train}$
Lorenz-63	$(0.45\pm0)m^{-0.75\pm0} + 0.17 \pm 0$	$(0.46\pm0)m^{-0.73\pm0} + 0.18 \pm 0$	0.02 ± 0
Pendulum	$(0.69\pm0)m^{-1.16\pm0} + 0 \pm 0$	$(0.66\pm0)m^{-1.14\pm0} + 0 \pm 0$	0.02 ± 0
Sine-Gordon	$(0.99\pm0)m^{-0.75\pm0} + 0 \pm 0$	$(0.99\pm0)m^{-0.73\pm0} + 0 \pm 0$	0.02 ± 0
Diffusion	$(0.92\pm0)m^{-1.26\pm0} + 0 \pm 0$	$(0.92\pm0)m^{-1.25\pm0} + 0 \pm 0$	0.01 ± 0

Table 1: Train- and test NMSE power-law and relative generalization error scaling for the ExtremONet.

the absolute error for random single-layer Neural Networks asymptotically scales at least like $\sim 1/\sqrt{m}$, which is in line with $\sim 1/m^{0.58}$ and $\sim 1/m^{0.63}$ for the pendulum and diffusion system, but not for the Lorenz-63 and Sine-Gordon systems with $\sim 1/m^{0.375}$ and $\sim 1/m^{375}$ respectively for our random single-layer Neural Operator. Additionally, the dependency of the train and prediction time with respect to the model width were also approximated by a power-law function to assess the scalability of the approach. We report the results of fits aggregated over every system:

$$t_{\text{train}}(m) \sim m^{1.09\pm0.30}\ (s), \quad t_{\text{predict}}(m) \sim m^{0.97\pm0.04}\ (s)\,. \tag{13}$$

The large variance on the train time is due to the stopping condition of Brent's algorithm. One should also consider the impact of the amount of data points (N) on the performance

6. If we assume that the behaviour is consistent between the under- and overparametrized regions.

of the ExtremONet. This is explored in Figure 16 in Appendix F. The figure shows that 2000 data points are enough to saturate the training sets in the considered systems.

Function space complexity & out-of-distribution learning With the rise of domain-specific architectures, interest in out-of-distribution (OOD) generalization has increased for robust real-world deployment (Hendrycks and Dietterich, 2019; Lu et al., 2020; Kornblith et al., 2020; Koh et al., 2020). Many tasks violate the IID assumption, especially when test data is unavailable during training, such as in online learning. A broad overview is provided in Liu et al. (2021). One class of OOD is conceptual drift, where the relationship $X \sim Y$ changes over time. In this work we will discuss two separate instances of out-of-distribution learning:

- **In-function-space dynamical drift:** Dynamics evolve within the same function space sampled during training. The trajectory data of $y(t)$ is no longer IID.
- **Out-of-function-space sampling:** Dynamics of the test set are not sampled from the training distribution. The input function $u(\cdot)$- and $y(t)$ data are no longer IID.

Note that these two concepts are identical for Neural Networks since they do not learn an input function. We shall study the first type for PDEs and the second type for PDEs and ODEs. Dynamical drift manifests as $F_{u(x)} : (x,t) \mapsto y(x,t)$, with an unknown shift in $u(x)$ over time, effectively translating to some unknown $u(x) \to u_t(x)$. Since Neural Operators observe the input function only at initialization, they miss post-initialization drift. This can be mitigated by transitioning to phase space like a Recurrent Neural Network (RNN), but without memory:

$$\mathscr{G}_{\Delta t} : (u(s), y(t)) \mapsto y(t+\Delta t), \quad \text{yielding} \quad y(n\Delta t) = (\circ_{k=1}^{n} \mathscr{G}_{\Delta t}(u(s))) \circ y(0). \tag{14}$$

To model drift, we adjust $u(s)$ per step:

$$y(n\Delta t) = (\circ_{k=1}^{n} \mathscr{G}_{\Delta t}(u(s, k\Delta t))) \circ y(0). \tag{15}$$

This accounts for function space drift. However, phase space drift persists: evolving dynamics may push the system into previously unvisited phase regions, where accurate prediction requires generalization beyond the training manifold. To evaluate the effect of having an OOD trajectory in the case of *In-function-space dynamical drift*, we trained an ExtremONet in phase space on PDEs with drifting GRF inputs. Drift is modelled via a spherical interpolation between two GRF samples:

$$u_{\text{drift}}(x,t) = \sqrt{d\frac{t}{t_f}}u_0(x) + \sqrt{1 - d\frac{t}{t_f}}u_1(x), \quad d \in [0,1], \tag{16}$$

with t_f the final time. Since u_0, u_1 are independent GRF samples with the same kernel k_l, their normalized combination satisfies:

$$\text{Cov}(u_{\text{drift}}(x,t), u_{\text{drift}}(x',t)) = d\frac{t}{t_f}k_l(x,x') + (1 - d\frac{t}{t_f})k_l(x,x') = k_l(x,x'). \tag{17}$$

We trained the previously tested large DeepONet and large ExtremONet) on 1000 GRF-driven diffusion systems per length scale and drift level (6×6 grid). Each used 100 time steps from $t \in [0, 1]$, repeated 30 times. From Appendix F (Figure 5,6) one can conclude that with sufficient training complexity the ExtremONet becomes drift-robust (likely by increasing coverage of phase space regions). Low-complexity training data yields sparse phase space measurements, limiting generalization. Therefore, Recurrent ExtremONets can be used as self-correcting RNNs. The DeepONet is not able to learn the recurrent map without overfitting, and therefore triggering the stopping condition. For *Out-of-function-space dynamical drift*, we tested whether DeepONets and ExtremONets can generalize beyond their training function space.[7] This was done by training a standard DeepONet and ExtremONet on pendulum and diffusion systems driven by GRFs of one length scale, and testing on others. This also reveals how input space complexity impacts performance (see Appendix F, Figure 9,10). For each GRF scale (6×6 grid), 1000 systems were generated using 10 time steps from $t \in [0, 1]$, repeated 30 times. Our results show that (see Appendix F, Figure 7,8) the ExtremONet trained on complex (small l) GRFs could generalize to simpler (larger l) ones, showing generalization from complex-to-simple dynamics. The DeepONet is able to generalize to lower complexity function spaces (to a lesser extent than the ExtremONet) on the pendulum system; it dramatically overfits on the diffusion system, and is not able to generalize in any region.

6. Conclusion

In this paper, we demonstrate that an Extreme-Learning-based variation of the DeepONet is two to four orders of magnitude faster to train, and is able to outperform large DeepONets in half of the problem settings when only small datasets are available. Our approach allows for rapid deployment of Neural Operators in situations where time is most valuable. We demonstrate that the ExtremONet generalizes more effectively than the DeepONet in every situation due to the regularization term in Ridge Regression. More complex situations require a multi-layered structure, which is not compatible with ELMs; Multi-layer ELMs (Kale and Karakuzu, 2022) exist, but they are not equivalent to multi-layer NNs.[8] We also show that the ExtremONet can generalize out-of-function-space to lower complexity, which the DeepONet is not able to reproduce to the same extent. The ExtremONet can also behave like a self-correcting RNN in the case of in-function-space dynamical drift when the model is moved into phase space. In situations where the model needs to be adjusted as new data arrives, traditional DeepONets have to rely on on-line learning (Hoi et al., 2018) to adjust their parameters. In contrast, ExtremONets might be fast enough to essentially retrain the model without relying on on-line learning. One can also include an on-line ELM strategy, as mentioned in Huynh and Won (2011). We also demonstrated that function spaces of the differential equations dependent variable only marginally benefited from operator architectures, which is to be expected since it is not covered by the UAFO. Finally, as climate change and AI's energy consumption (Aquino-Brítez et al., 2025; Wright) become ever more relevant geo-political topics, the ExtremONet also provides a responsible alternative to DeepONets. In conclusion, the ExtremONet is a robust, fast-to-train Neural Operator, suitable for simple or moderately complex real-world problem settings, or when there is limited access to compute or data.

7. We only focus on OOD function spaces, and we therefore did not consider models in phase space.
8. The boundary of standard ELM applicability could be studied in some future work.

References

Diab W. Abueidda, Panos Pantidis, and Mostafa E. Mobasher. DeepOKAN: Deep Operator Network Based on Kolmogorov Arnold Networks for Mechanics Problems, 2024. URL https://arxiv.org/abs/2405.19143. Version Number: 3.

E. Ivo Alves. Earthquake Forecasting Using Neural Networks: Results and Future Work. *Nonlinear Dynamics*, 44(1-4):341–349, June 2006. ISSN 0924-090X, 1573-269X. doi: 10.1007/s11071-006-2018-1. URL http://link.springer.com/10.1007/s11071-006-2018-1.

Eric Aislan Antonelo, Eduardo Camponogara, Laio Oriel Seman, Jean Panaioti Jordanou, Eduardo Rehbein De Souza, and Jomi Fred Hübner. Physics-informed neural nets for control of dynamical systems. *Neurocomputing*, 579:127419, April 2024. ISSN 09252312. doi: 10.1016/j.neucom.2024.127419. URL https://linkinghub.elsevier.com/retrieve/pii/S0925231224001905.

Sergio Aquino-Brítez, Pablo García-Sánchez, Andrés Ortiz, and Diego Aquino-Brítez. Towards an Energy Consumption Index for Deep Learning Models: A Comparative Analysis of Architectures, GPUs, and Measurement Tools. *Sensors*, 25(3):846, January 2025. ISSN 1424-8220. doi: 10.3390/s25030846. URL https://www.mdpi.com/1424-8220/25/3/846.

Alexander Atanasov, Jacob A. Zavatone-Veth, and Cengiz Pehlevan. Scaling and renormalization in high-dimensional regression, June 2025. URL http://arxiv.org/abs/2405.00592. arXiv:2405.00592 [stat].

Richard Bellman and Robert Kalaba. A MATHEMATICAL THEORY OF ADAPTIVE CONTROL PROCESSES. *Proceedings of the National Academy of Sciences*, 45(8):1288–1290, August 1959. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.45.8.1288. URL https://pnas.org/doi/full/10.1073/pnas.45.8.1288.

R. P. Brent. *Algorithms for minimization without derivatives.* Prentice-Hall series in automatic computation. Prentice-Hall, Englewood Cliffs, N.J, 1972. ISBN 978-0-13-022335-7.

Qianying Cao, Somdatta Goswami, and George Em Karniadakis. LNO: Laplace Neural Operator for Solving Differential Equations, 2023. URL https://arxiv.org/abs/2303.10528. Version Number: 2.

Liliya A. Demidova and Vladimir E. Zhuravlev. The Study on Initialization Aspects of the Extreme Learning Machine Parameters by Random Values. In *2024 6th International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA)*, pages 364–369, Lipetsk, Russian Federation, November 2024. IEEE. ISBN 979-8-3315-3217-8. doi: 10.1109/SUMMA64428.2024.10803774. URL https://ieeexplore.ieee.org/document/10803774/.

J.R. Dormand and P.J. Prince. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, March 1980. ISSN 03770427. doi:

10.1016/0771-050X(80)90013-3. URL https://linkinghub.elsevier.com/retrieve/pii/0771050X80900133.

Yiding Feng, Ronen Gradwohl, Jason Hartline, Aleck Johnsen, and Denis Nekipelov. Bias-Variance Games, February 2022. URL http://arxiv.org/abs/1909.03618. arXiv:1909.03618 [cs].

Junyan He, Shashank Kushwaha, Jaewan Park, Seid Koric, Diab Abueidda, and Iwona Jasiuk. Sequential Deep Operator Networks (S-DeepONet) for predicting full-field solutions under time-dependent loads. *Engineering Applications of Artificial Intelligence*, 127:107258, January 2024. ISSN 09521976. doi: 10.1016/j.engappai.2023.107258. URL https://linkinghub.elsevier.com/retrieve/pii/S0952197623014422.

Dan Hendrycks and Thomas Dietterich. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations, 2019. URL https://arxiv.org/abs/1903.12261. Version Number: 1.

Steven C. H. Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online Learning: A Comprehensive Survey, October 2018. URL http://arxiv.org/abs/1802.02871. arXiv:1802.02871 [cs].

Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1-3):489–501, December 2006. ISSN 09252312. doi: 10.1016/j.neucom.2005.12.126. URL https://linkinghub.elsevier.com/retrieve/pii/S0925231206000385.

Thomas Huang, Cedric David, Catalina Oadia, Joe T. Roberts, Sujay V. Kumar, Paul Stackhouse, David Borges, Simon Baillarin, Gwendoline Blanchet, and Peter Kettig. An Earth System Digital Twin for Flood Prediction and Analysis. In *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*, pages 4735–4738, Kuala Lumpur, Malaysia, July 2022. IEEE. ISBN 978-1-6654-2792-0. doi: 10.1109/IGARSS46834.2022.9884830. URL https://ieeexplore.ieee.org/document/9884830/.

Hieu Trung Huynh and Yonggwan Won. Regularized online sequential learning algorithm for single-hidden layer feedforward neural networks. *Pattern Recognition Letters*, 32(14):1930–1935, October 2011. ISSN 0167-8655. doi: 10.1016/j.patrec.2011.07.016. URL https://linkinghub.elsevier.com/retrieve/pii/S0167865511002297. Publisher: Elsevier BV.

Yoleidy Huérfano-Maldonado, Marco Mora, Karina Vilches, Ruber Hernández-García, Rodrigo Gutiérrez, and Miguel Vera. A comprehensive review of extreme learning machine on medical imaging. *Neurocomputing*, 556:126618, November 2023. ISSN 09252312. doi: 10.1016/j.neucom.2023.126618. URL https://linkinghub.elsevier.com/retrieve/pii/S0925231223007415.

B. Igelnik and Yoh-Han Pao. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE Transactions on Neural Networks*, 6

(6):1320–1329, November 1995. ISSN 10459227. doi: 10.1109/72.471375. URL http://ieeexplore.ieee.org/document/471375/.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873): 583–589, August 2021. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-021-03819-2. URL https://www.nature.com/articles/s41586-021-03819-2.

Gizem Atac Kale and Cihan Karakuzu. Multilayer extreme learning machines and their modeling performance on dynamical systems. *Applied Soft Computing*, 122:108861, June 2022. ISSN 15684946. doi: 10.1016/j.asoc.2022.108861. URL https://linkinghub.elsevier.com/retrieve/pii/S1568494622002472.

Georgia Karagiorgi, Gregor Kasieczka, Scott Kravitz, Benjamin Nachman, and David Shih. Machine Learning in the Search for New Fundamental Physics, 2021. URL https://arxiv.org/abs/2112.03769. Version Number: 1.

Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton A. Earnshaw, Imran S. Haque, Sara Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. WILDS: A Benchmark of in-the-Wild Distribution Shifts, 2020. URL https://arxiv.org/abs/2012.07421. Version Number: 3.

Simon Kornblith, Ting Chen, Honglak Lee, and Mohammad Norouzi. Why Do Better Loss Functions Lead to Less Transferable Features?, 2020. URL https://arxiv.org/abs/2010.16402. Version Number: 2.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural Operator: Learning Maps Between Function Spaces. 2021. doi: 10.48550/ARXIV.2108.08481. URL https://arxiv.org/abs/2108.08481. Publisher: arXiv Version Number: 6.

Sergei B. Kuksin and Dario Bambusi. Hamiltonian PDEs. In *Handbook of Dynamical Systems*, volume 1, pages 1087–1133. Elsevier, 2006. ISBN 978-0-444-52055-5. doi: 10.1016/S1874-575X(06)80040-8. URL https://linkinghub.elsevier.com/retrieve/pii/S1874575X06800408.

Guoqiang Li and Peifeng Niu. An enhanced extreme learning machine based on ridge regression for regression. *Neural Computing and Applications*, 22(3-4):803–810, March 2013. ISSN 0941-0643, 1433-3058. doi: 10.1007/s00521-011-0771-7. URL http://link.springer.com/10.1007/s00521-011-0771-7.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations, 2020. URL https://arxiv.org/abs/2010.08895. Version Number: 3.

Jiashuo Liu, Zheyan Shen, Yue He, Xingxuan Zhang, Renzhe Xu, Han Yu, and Peng Cui. Towards Out-Of-Distribution Generalization: A Survey, 2021. URL https://arxiv.org/abs/2108.13624. Version Number: 2.

Edward N. Lorenz. Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, March 1963. ISSN 0022-4928, 1520-0469. doi: 10.1175/1520-0469(1963)020⟨0130:DNF⟩2.0.CO;2. URL http://journals.ametsoc.org/doi/10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization, 2017. URL https://arxiv.org/abs/1711.05101. Version Number: 3.

Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under Concept Drift: A Review. 2020. doi: 10.48550/ARXIV.2004.05785. URL https://arxiv.org/abs/2004.05785. Publisher: arXiv Version Number: 1.

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, March 2021. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5. URL https://www.nature.com/articles/s42256-021-00302-5.

Alexander Maloney, Daniel A. Roberts, and James Sully. A Solvable Model of Neural Scaling Laws, October 2022. URL http://arxiv.org/abs/2210.16859. arXiv:2210.16859 [cs].

Xiaodong Na, Yuan Li, Weijie Ren, and Min Han. Physics-informed hierarchical echo state network for predicting the dynamics of chaotic systems. *Expert Systems with Applications*, 228:120155, October 2023. ISSN 09574174. doi: 10.1016/j.eswa.2023.120155. URL https://linkinghub.elsevier.com/retrieve/pii/S0957417423006577.

Irina Perfilievaa, Nicolas Madrid, Manuel Ojeda-Aciego, Piotr Artiemjew, and Agnieszka Niemczynowicz. A Critical Analysis of the Theoretical Framework of the Extreme Learning Machine, June 2024. URL http://arxiv.org/abs/2406.17427. arXiv:2406.17427 [cs].

Gianluigi Pillonetto, Aleksandr Aravkin, Daniel Gedon, Lennart Ljung, Antônio H. Ribeiro, and Thomas B. Schön. Deep networks for system identification: A survey. *Automatica*, 171:111907, January 2025. ISSN 00051098. doi: 10.1016/j.automatica.2024.111907. URL https://linkinghub.elsevier.com/retrieve/pii/S0005109824004011.

M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, February 2019.

ISSN 00219991. doi: 10.1016/j.jcp.2018.10.045. URL https://linkinghub.elsevier.com/retrieve/pii/S0021999118307125.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, November 2005. ISBN 978-0-262-25683-4. doi: 10.7551/mitpress/3206.001.0001. URL https://direct.mit.edu/books/book/2320/Gaussian-Processes-for-Machine-Learning.

Jason W. Rocks and Pankaj Mehta. Memorizing without overfitting: Bias, variance, and interpolation in over-parameterized models. *Physical Review Research*, 4(1):013201, March 2022. ISSN 2643-1564. doi: 10.1103/PhysRevResearch.4.013201. URL http://arxiv.org/abs/2010.13933. arXiv:2010.13933 [stat].

Manuel A. Roehrl, Thomas A. Runkler, Veronika Brandtstetter, Michel Tokic, and Stefan Obermayer. Modeling System Dynamics with Physics-Informed Neural Networks Based on Lagrangian Mechanics. *IFAC-PapersOnLine*, 53(2):9195–9200, 2020. ISSN 24058963. doi: 10.1016/j.ifacol.2020.12.2182. URL https://linkinghub.elsevier.com/retrieve/pii/S2405896320328354.

Yixuan Sun, Christian Moya, Guang Lin, and Meng Yue. DeepGraphONet: A Deep Graph Operator Network to Learn and Zero-shot Transfer the Dynamic Response of Networked Systems, 2022. URL https://arxiv.org/abs/2209.10622. Version Number: 1.

Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, July 1995. ISSN 10459227. doi: 10.1109/72.392253. URL http://ieeexplore.ieee.org/document/392253/.

Hrishikesh Viswanath, Md Ashiqur Rahman, Abhijeet Vyas, Andrey Shor, Beatriz Medeiros, Stephanie Hernandez, Suhas Eswarappa Prameela, and Aniket Bera. Neural Operator: Is data all you need to model the world? An insight into the impact of Physics Informed Machine Learning, 2023. URL https://arxiv.org/abs/2301.13331. Version Number: 2.

Ian Wright. ChatGPT Energy Consumption Visualized.

Jiahao Zhang, Shiqi Zhang, and Guang Lin. MultiAuto-DeepONet: A Multi-resolution Autoencoder DeepONet for Nonlinear Dimension Reduction, Uncertainty Quantification and Operator Learning of Forward and Inverse Stochastic Problems, 2022. URL https://arxiv.org/abs/2204.03193. Version Number: 1.

Appendix A. Notational conventions & mathematical symbols

$\mathbb{R}^d$	d-dimensional Euclidean space
$\mapsto, \Rightarrow$	Map to, implies
d_t	Derivative
$\partial_t, \partial_x, \partial_y$	Partial derivative
$f(\cdot), u(\cdot)$	Functions with arbitrary domain
X	Banach space with norm $\|\cdot\|_X$
$C[a,b]$	Compact space within interval $[a,b]$
$C(K)$	All continuous functions on $K \subset X$ with norm $\|\cdot\|_{C(K)} = \max_{x\in K}\{\lvert f(x)\rvert\}$
V	Arbitrary compact set of $C(K)$
G	Operator
$s, u(s)$	Sensor location and sensor measurement
$\psi(\cdot), \phi(\cdot)$	Basis or activation functions
p, m	Integers (network width and p-size)
α, θ, W, b	Rank-n tensors (trainable depending on context), rank-2 tensor, rank-1 tensor
$\mathscr{N}, \mathscr{N}_E$	Neural Network, Extreme-Learning-Machine
$\mathscr{G}, \mathscr{G}_E$	Neural Operator, Extreme Neural Operator
$\mathcal{L}$	Loss function (NMSE)
N	Number of data samples
r, w, r'	Number of sensors, sensor dimension, unrolled number of sensors
$\hat{y}_i(t)$	Prediction of trajectory at time t
$\mathscr{B}, \mathscr{T}, \mathscr{B}_E, \mathscr{T}_E$	Branch- and Trunk network, NN or ELM
$\odot$	Hadamard product (element-wise multiplication)
$\mathcal{U}, \mathcal{N}$	Uniform-, and normal distribution
$\sim$	Sampled from or asymptotic
$\dim(\cdot)$	Dimension of object (last index in rank-n tensor)
σ, c	Standard deviation, connectivity
$\mathrm{argmin}_\theta\, \mathbb{E}\{\cdot\}$	Minimize expectation value with respect to θ
$[\cdot\lvert\cdot], [\cdot\|\cdot]$	Concatenation of rank-n tensors over 1st/2nd index.
GRF	Gaussian Random Field
$k_l(\cdot,\cdot)$	Kernel with parameter l
$\circ_{k=1}^{n}$	Sequence of n compositions
Cov	Covariance
$\equiv$	Define
$\partial\Omega$	Boundary of PDE problem
ρ_n	Sequence of n elements
$\inf_\alpha$	Infimum with respect to α
$\lceil\cdot\rceil$	Ceiling function
$a \bmod b$	Remainder of a/b

Appendix B. Theorems, lemmas, & corollaries

Theorem 1 (UAT for Operators (rank-2)) *For compact sets $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$, $V \subset C(K_1)$, a non-linear operator $G : V \to C(K_2)$ between function spaces V and $C(K_2)$ can be approximated arbitrarily close by some Neural-Network-like structure. For any $\epsilon > 0$, there exist some Tauber-Wiener functions $\psi(\cdot)$, $\phi(\cdot)$, features $s \in K_1$, integers p, m, r, and parameters $\theta^{(R)}_{(W)}$, $\theta^{(B)}_{(W)}$, $\theta^{(B)}_{(b)}$, $\theta^{(T)}_{(W)}$, and $\theta^{(T)}_{(b)}$, such that:*[9]

$$\big|G(u(s))(t) - \sum_{i=1}^{p}\sum_{j=1}^{m} \theta^{i(R)}_{j(W)}\psi\big(\sum_{l=1}^{r}\theta^{i(B)}_{jl(W)}u(s_l) + \theta^{i(B)}_{j(b)}\big)\phi\big(\theta^{(T)}_{j(W)}t + \theta^{(T)}_{j(b)}\big)\big| < \epsilon \tag{18}$$

holds for all $u(\cdot) \in V$ and $y \in C(K_2)$.

(Tianping Chen and Hong Chen, 1995)

Theorem 2 (UAT for single-layer random Neural Networks) *For every function $f(\cdot) \in C(K_1)$ defined on some compact set $K_1 \subset [0,1]^D$, and any function whose derivative is L_2-integrable, there exist parameters $\theta^{(R)}_{(W)}$, a sequence of single-layer Neural Network distance measurements:*

$$\rho_n = \Big\{\sqrt{\mathrm{E}\big\{\int_{K_1}\big|f(t) - \sum_{i=1}^{m}\theta^{(R)}_{i(W)}\phi(\theta^{j}_{i(W)}t + \theta^{j}_{i(b)})\big|^2 dt\big\}} \mid j \in [1,n]\Big\}, \tag{19}$$

and a sequence of random $\theta^{j}_{i(W)}$ and $\theta^{j}_{i(b)}$ such that $\lim_{n\to\infty}\rho_n = 0$.[10]

(Igelnik and Yoh-Han Pao, 1995)

Lemma 3 (rank-1 UATO (Novel)) *For compact sets $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$, $V \subset C(K_1)$, a non-linear operator $G : V \to C(K_2)$ between function spaces V and $C(K_2)$, the following inequality:*

$$\begin{aligned}&\inf_{\alpha}\big|G(u(s))(t) - \sum_{k=1}^{m'}\alpha^{(R)}_{k(W)}\psi\big(\sum_{l=1}^{r}\alpha^{(B)}_{kl(W)}u(s_l) + \alpha^{(B)}_{k(b)}\big)\phi\big(\alpha^{(T)}_{k(W)}t + \alpha^{(T)}_{k(b)}\big)\big| \leq \\ &\inf_{\theta}\big|G(u(s))(t) - \sum_{i=1}^{p}\sum_{j=1}^{m}\theta^{i(R)}_{j(W)}\psi\big(\sum_{l=1}^{r}\theta^{i(B)}_{jl(W)}u(s_l) + \theta^{i(B)}_{j(b)}\big)\phi\big(\theta^{(T)}_{j(W)}t + \theta^{(T)}_{j(b)}\big)\big|\end{aligned} \tag{20}$$

holds; where $m' = pm$, $\psi(\cdot)$, $\phi(\cdot)$, features s, parameters θ, and integer r are defined as in the UATO. This demonstrates that an overparametrization of the UATO can be reduced to a form that lets $\alpha^{(R)}$ be rank-1 tensors, compatible with Linear Regression.

Proof According to the UATO, define $\psi^i_j(u(s),\theta) \equiv \psi\big(\sum_{l=1}^{r}\alpha^{i(B)}_{jl(W)}u(s_l) + \alpha^{i(B)}_{j(b)}\big)$, and $\phi_j(t,\alpha) \equiv \phi\big(\alpha^{(T)}_{j(W)}t + \alpha^{(T)}_{j(b)}\big)$; similarly let us define $\psi_k(u(s),\alpha) \equiv \psi\big(\sum_{l=1}^{r}\alpha^{(B)}_{kl(W)}u(s_l) +$

9. $(R), (T), (B), (W), (b)$ is a naming convention which is relevant in the discussion of the DeepONet.
10. Note that this theorem is non-constructive regarding $\theta^{(R)}_{(W)}$.

$\alpha^{(B)}_{k(b)})$, and $\phi_k(t,\alpha) \equiv \phi\big(\alpha^{(T)}_{k(W)}t + \alpha^{(T)}_{k(b)}\big)$; If we can find a parametrization α such that

$$\sum_{k=1}^{m'} \alpha^{(R)}_{k(W)} \psi_k(u(s),\alpha)\phi_k(t,\alpha) = \sum_{i=1}^{p}\sum_{j=1}^{m} \theta^{i(R)}_{j(W)} \psi^i_j((u(s),\theta)\phi_j(t,\theta) \tag{21}$$

then the proof is complete. Let $\alpha_k = \theta^{k \bmod p}_{\lceil k/p \rceil}$, let $\psi_k(t,\alpha) = \phi^{k \bmod p}_{\lceil k/p \rceil}(t,\theta)$, and let $\phi_k(t,\alpha) = \phi_{\lceil k/p \rceil}(t,\theta)$ then the two formulations are identical, which completes the proof. ■

Corollary 4 (Novel) *Lemma 3 also implies that for any parameters* $\theta^{(R)}_{(W)}$, $\theta^{(B)}_{(W)}$, $\theta^{(B)}_{(b)}$, $\theta^{(T)}_{(W)}$, *and* $\theta^{(T)}_{(b)}$*:*

$$\inf_{\alpha} \Big| \sum_{k=1}^{m'} \alpha^{(R)}_{k(W)} \psi\big(\sum_{l=1}^{r} \alpha^{(B)}_{kl(W)} u(s_l) + \alpha^{(B)}_{k(b)}\big)\phi\big(\alpha^{(T)}_{k(W)}t + \alpha^{(T)}_{k(b)}\big) \tag{22}$$

$$- \sum_{i=1}^{p}\sum_{j=1}^{m} \theta^{i(R)}_{j(W)} \psi\big(\sum_{l=1}^{r} \theta^{i(B)}_{jl(W)} u(s_l) + \theta^{i(B)}_{j(b)}\big)\phi\big(\theta^{(T)}_{j(W)}t + \theta^{(T)}_{j(b)}\big)\Big| = 0 \tag{23}$$

Proof See the proof of Lemma 3. ■

Appendix C. ExtremONet derivation

To construct such an Extreme-Learning-compatible operator, Lemma 3 can be used as a template. By assuming that, just like for random single-layer Neural Networks, there exists a randomly sampled set of non-trainable parameters that approximates an operator arbitrarily close, then we are allowed to replace the basis functions with two random single-layer Neural Networks: $\psi(u(s)), \phi(t) \to \mathscr{B}_E(u(s)), \mathscr{T}_E(t)$. By using the ELM parameter naming convention, adding a bias term $\theta^{(1)}$, and training them via Ridge Regression, then one retrieves the ExtremONet.

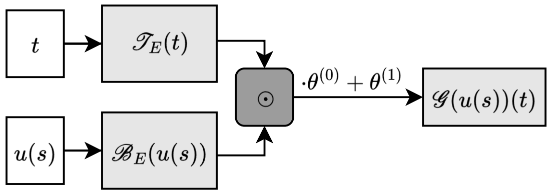

Figure 1: ExtremONet diagram.

Appendix D. Datasets, experimental setup & code depository

ODE datasets Lorenz-63 and pendulum datasets were generated similarly; test sets are 10× larger. Lorenz-63: 1000 realizations of 10 time points in $t \in [0,1]$, with parameters $\alpha_0 \in [0,10]$, $\alpha_1 \in [0,28]$, $\alpha_2 \in [0,8/3]$, and 40 sensor locations in $y \in [-30,30]^3$. Initial state: $y_0 \sim \mathscr{N}(0,1)$. Pendulum: 1000 realizations over $t \in [0,1]$, kernel scale $l = 0.1$, 120 sensor locations in $y \in [0,1]$, and initial conditions $y_0, y'_0 \sim \mathscr{N}(0,0.1)$.

PDE datasets All PDE datasets share structure; test sets are 10× larger. **Sine-Gordon:** 1000 samples, 10 time points in $t \in [0, 3]$, spatial grid: 100 points in $x \in [0, 1]$ (finite differences). Inputs: concatenated $y(x, t)$ and $\partial_t y(x, t)$. Parameters: $\alpha \in [0, 1]^3$. 120 sensor locations in $y \in [-10, 10]$. Initial conditions: $y_0(x) = k_{0.1}(1/\sqrt{2}, x)$, $y_0'(x) = 0$, toroidal boundary. **Diffusion:** 1000 samples, 10 time points in $t \in [0, 1]$, grid of 100 x points in $[0, 1]$, $l = 0.1$, 120 sensor locations in $y \in [0, 1]$, initial condition $y(x) = k_{0.1}(0.75, x)$, with Dirichlet boundary condition $\partial\Omega = 0$.

Experiment setup The tests were executed on Windows 11 Pro in a python 3.11 environment using cudaNN version 9.7 and Pytorch version 2.5.1 on a desktop equipped with an AMD Ryzen 7 7700 and an Nvidia RTX 4070 Ti SUPER. All mathematical operations were performed on the GPU where possible. The differential equations were solved using Runge-Kutta 5(4) (Dormand and Prince, 1980)) in Scipy version 1.15.1; spatial coordinates were treated with a finite differences method in the case of PDEs. The experiment code is available at `https://github.com/JariBey`.

Appendix E. DeepONet-ExtremONet comparison results

Table 2: Trainable parameters per system and model size for DeepONet and ExtremONet.

	DeepONet			ExtremONet		
	Small	Medium	Large	Small	Medium	Large
Lorenz-63	18 363	117 063	1 123 623	303	903	3 003
Pendulum	16 342	113 042	1 103 582	202	602	2 002
Sine-Gordon	464 500	1 005 200	5 311 700	20 200	60 200	200 200
Diffusion	262 400	603 100	3 307 600	10 100	30 100	100 100

Table 3: Train-, test NMSE, and train time for the largest DeepONet (top) and the largest ExtremONet (bottom).

	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	$t_{train}(s)$
Lorenz-63	$(1.8 \pm 0.0) \cdot 10^{-1}$	$(2.0 \pm 0.1) \cdot 10^{-1}$	$(2.6 \pm 3.8) \cdot 10^{2}$
	$(3.0 \pm 0.0) \cdot 10^{-1}$	$(3.4 \pm 0.1) \cdot 10^{-1}$	$(8.0 \pm 1.0) \cdot 10^{-2}$
Pendulum	$(3.3 \pm 0.0) \cdot 10^{-2}$	$(5.0 \pm 0.4) \cdot 10^{-2}$	$(4.8 \pm 1.6) \cdot 10^{1}$
	$(1.3 \pm 0.3) \cdot 10^{-4}$	$(1.4 \pm 0.4) \cdot 10^{-4}$	$(8.5 \pm 1.1) \cdot 10^{-2}$
Sine-Gordon	$(9.8 \pm 0.0) \cdot 10^{-1}$	$(9.9 \pm 0.0) \cdot 10^{-1}$	$(2.2 \pm 0.2) \cdot 10^{1}$
	$(4.7 \pm 3.6) \cdot 10^{-3}$	$(5.0 \pm 0.4) \cdot 10^{-3}$	$(8.7 \pm 1.3) \cdot 10^{-2}$
Diffusion	$(9.9 \pm 0.3) \cdot 10^{-6}$	$(1.7 \pm 0.1) \cdot 10^{-5}$	$(1.2 \pm 0.0) \cdot 10^{3}$
	$(3.4 \pm 0.3) \cdot 10^{-4}$	$(3.6 \pm 0.4) \cdot 10^{-4}$	$(8.0 \pm 1.0) \cdot 10^{-2}$

Appendix F. Figures

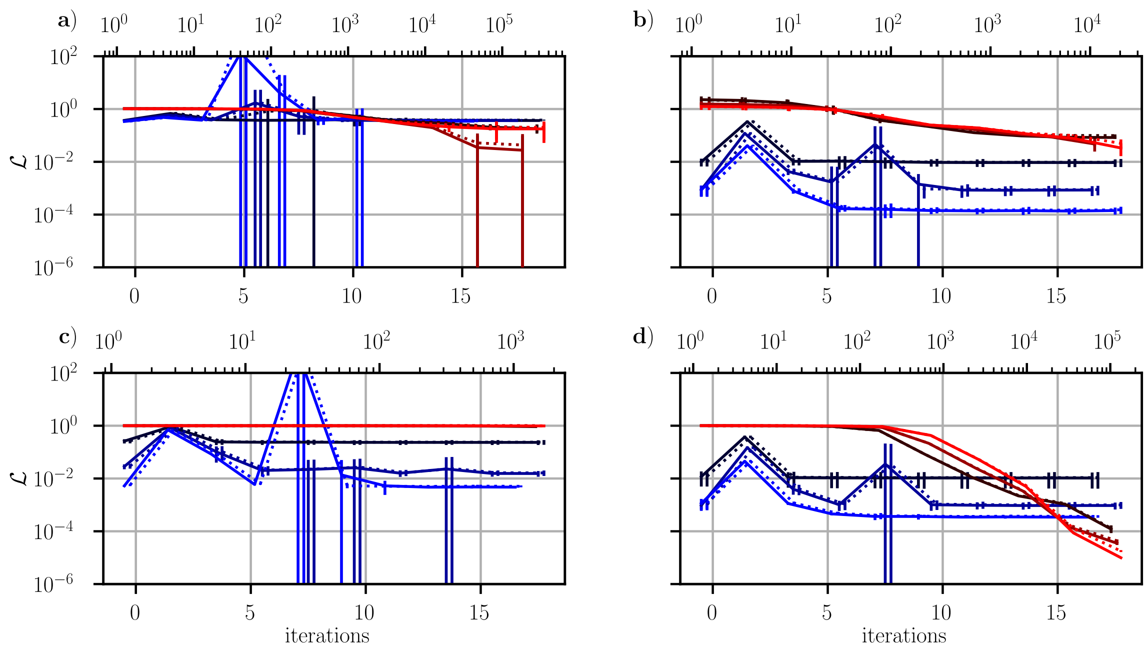

Figure 2: Train- and validation loss history (filled and dotted line) mean and standard deviation of three DeepONets (red) and three ExtremONets (blue) (30 repeats). Model size is tied to colour saturation. The upper x-axis represent DeepONet training iterations, lower x-axis represents Ridge Regression regularization parameter search iterations. The y-axis is $\log_{10}$-scale NMSE. **a)** Lorenz-63 system. **b)** Gravity pendulum ODE with GRF (t). **c)** Parametrized Sine-Gordon PDE. **d)** Diffusion PDE with GRF (x).

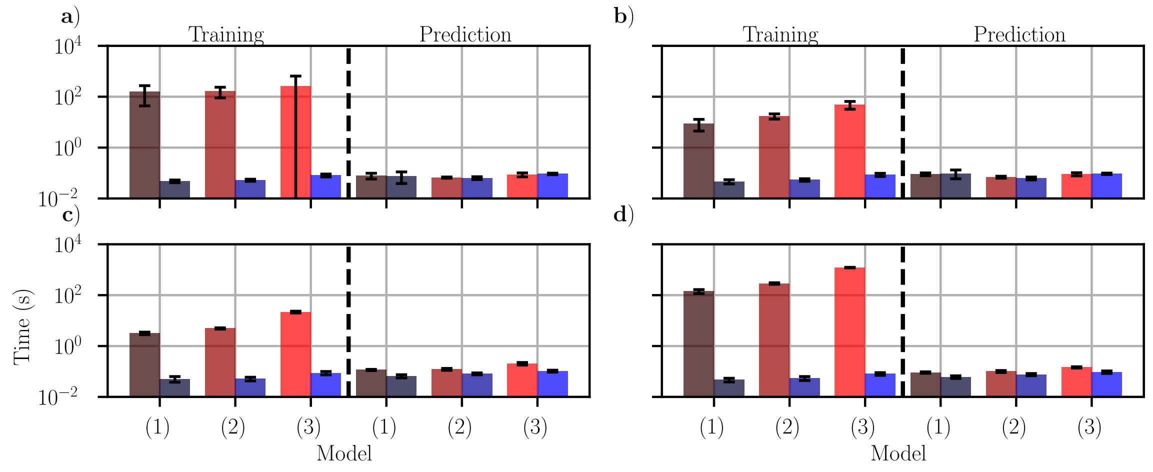

Figure 3: Train- and prediction mean and standard deviation time of three DeepONets (red) and three ExtremONets (blue) (30 repeats). Model sizes are small (1), medium (2), and large (3) (also indicated by bar colour saturation). **a)** Lorenz-63 system. **b)** Gravity pendulum ODE with GRF (t). **c)** Parametrized Sine-Gordon PDE. **d)** Diffusion PDE with GRF (x).

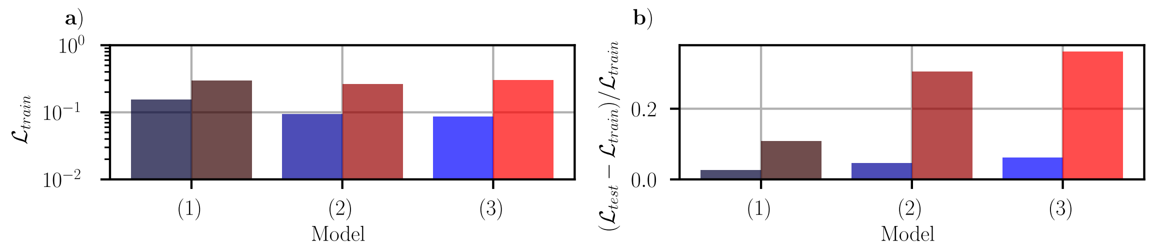

Figure 4: **a)** Mean of the training error. **b)** Mean of the relative generalization error. DeepONet (red) and the ExtremONet (blue) for the increasing model sizes; model sizes are small (1), medium (2), and large (3) (also indicated by bar colour saturation).

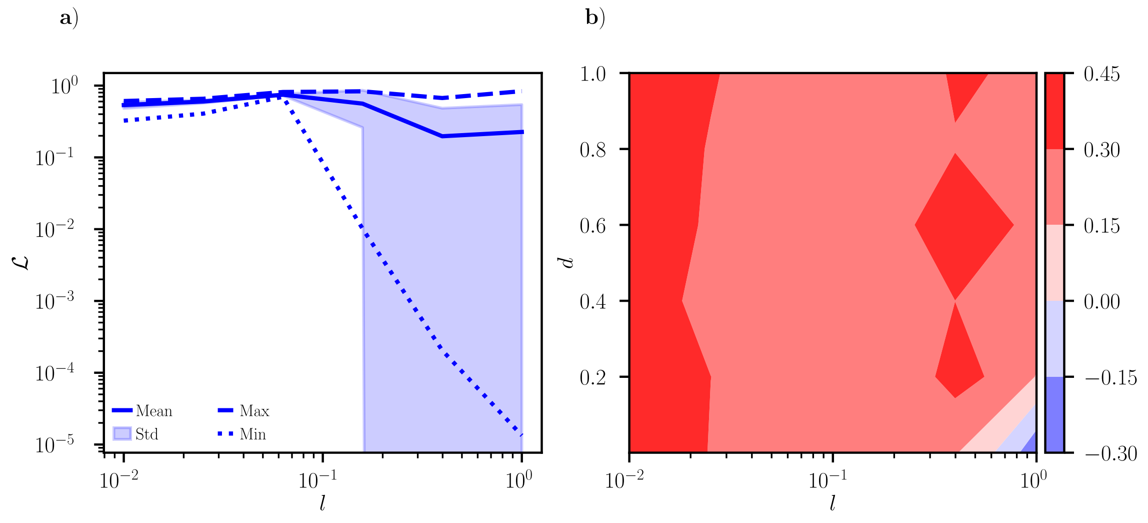

Figure 5: DeepONet in phase space trained on the diffusion with spatial GRF dataset; 30 repeats. **a)** Train loss statistics vs. GRF length scale $l \in [0.01, 1]$; y-axis is $\log_{10}$-NMSE. **b)** $\log_{10}(\mathcal{L}_{\text{test}}/\mathcal{L}_{\text{train}})$ for varying GRF length scales (x-axis) and drift $d \in [0, 1]$ (y-axis).

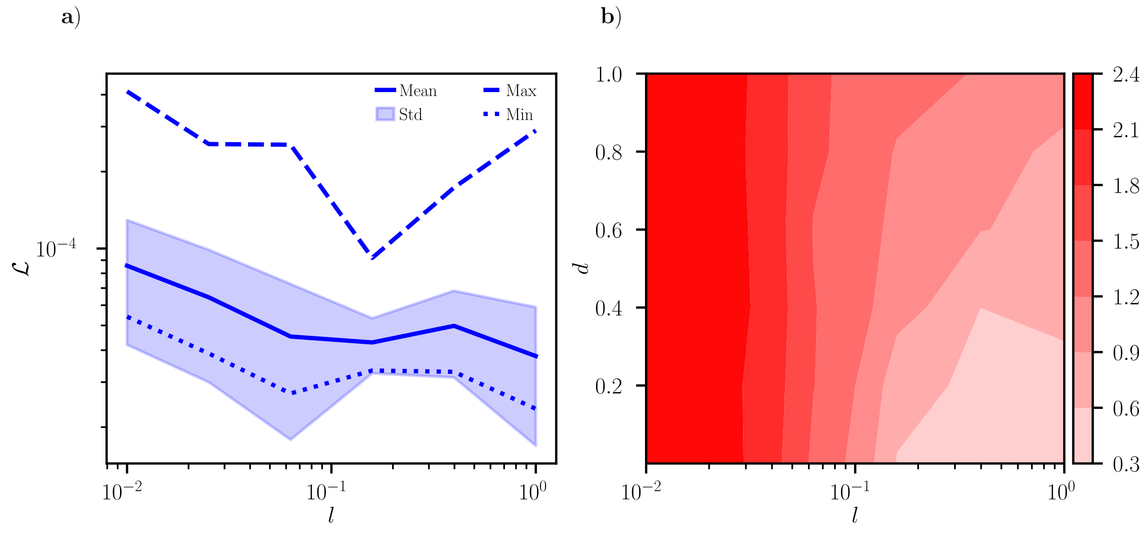

Figure 6: ExtremONet in phase space trained on the diffusion with spatial GRF dataset; 30 repeats. **a)** Train loss statistics vs. GRF length scale $l \in [0.01, 1]$; y-axis is $\log_{10}$-NMSE. **b)** $\log_{10}(\mathcal{L}_{\text{test}}/\mathcal{L}_{\text{train}})$ for varying GRF length scales (x-axis) and drift $d \in [0, 1]$ (y-axis).

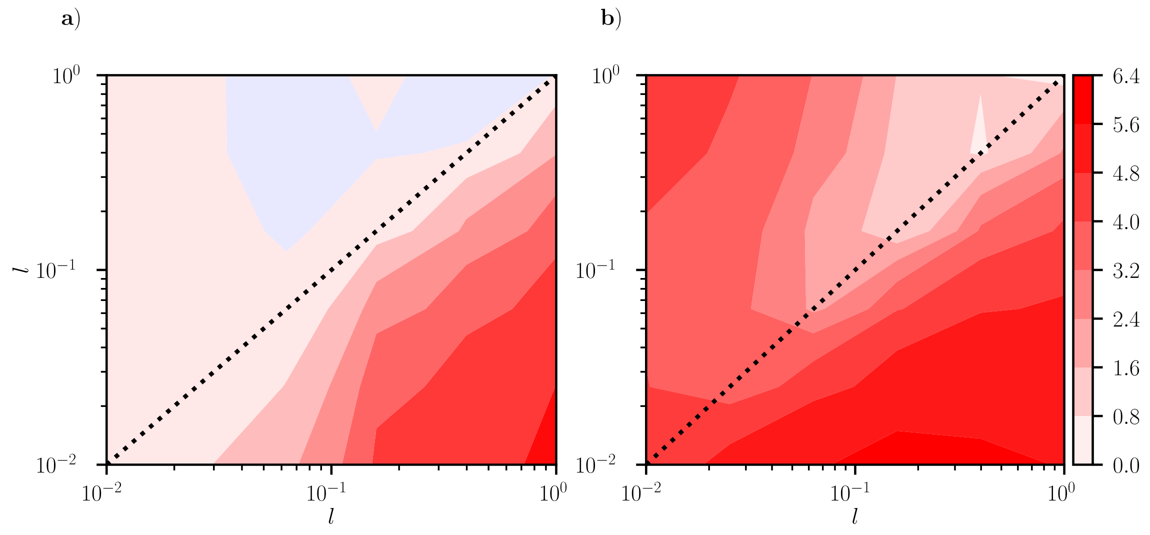

Figure 7: DeepONet out-of-function-space drift: $\log_{10}(\mathcal{L}_{\text{test}}/\mathcal{L}_{\text{train}})$ vs. GRF length scale. x-axis: training scale, y-axis: test scale; 30 repeats. **a)** Pendulum ODE with GRF (t). **b)** Diffusion PDE with GRF (x).

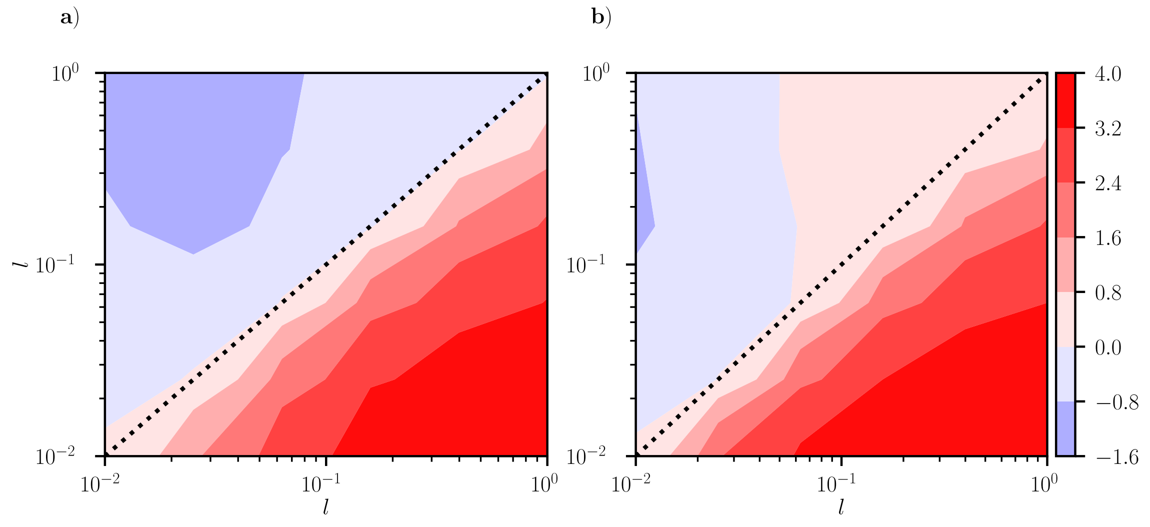

Figure 8: ExtremONet out-of-function-space drift: $\log_{10}(\mathcal{L}_{\text{test}}/\mathcal{L}_{\text{train}})$ vs. GRF length scale. x-axis: training scale, y-axis: test scale; 30 repeats. **a)** Pendulum ODE with GRF (t). **b)** Diffusion PDE with GRF (x).

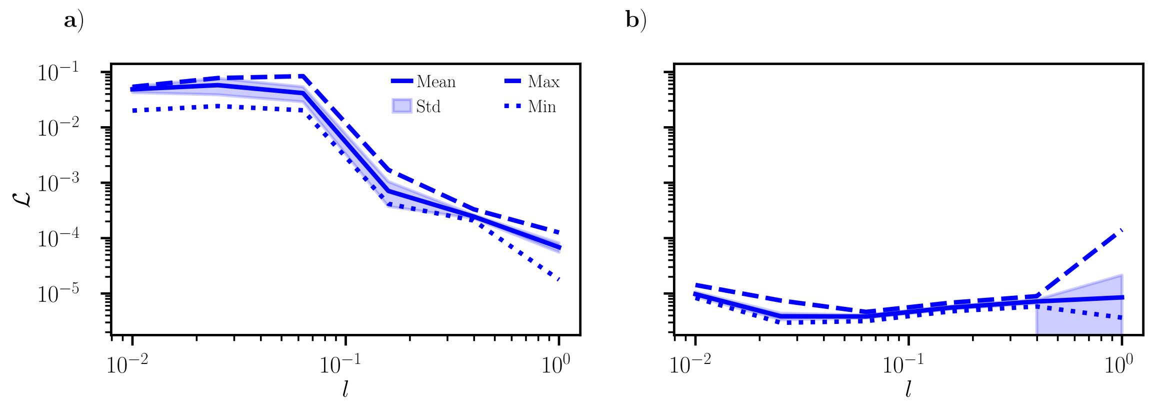

Figure 9: DeepONet train loss statistics vs. GRF length scale $l \in [0.01, 1]$; 30 repeats. **a)** Pendulum ODE with temporal GRF. **b)** Diffusion PDE with spatial GRF. y-axis is $\log_{10}$-NMSE.

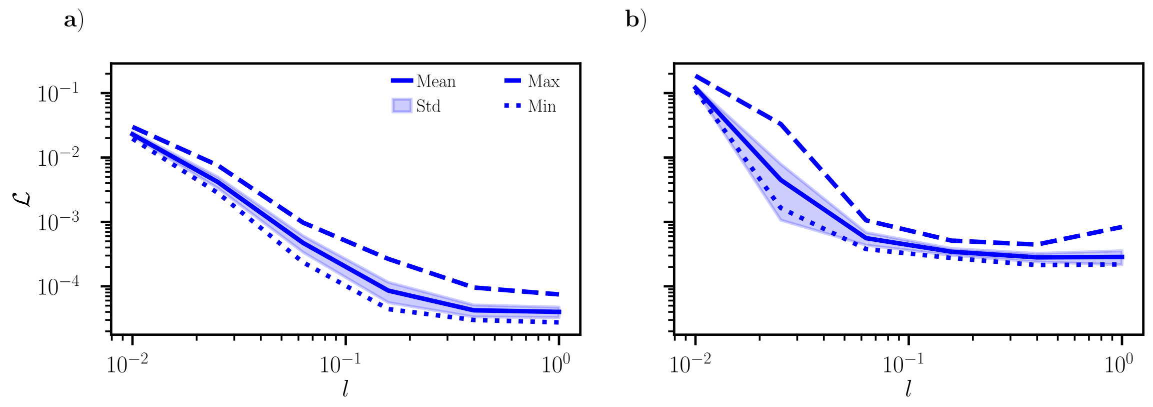

Figure 10: ExtremONet train loss statistics vs. GRF length scale $l \in [0.01, 1]$; 30 repeats. **a)** Pendulum ODE with temporal GRF. **b)** Diffusion PDE with spatial GRF. y-axis is $\log_{10}$-NMSE.

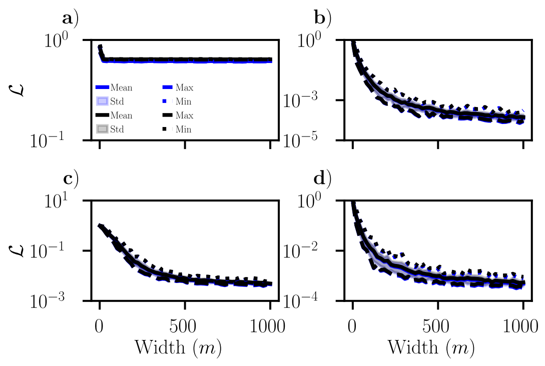

Figure 11: ExtremONet train loss- (blue) and test loss (black) mean, standard deviation, maximum, and minimum with respect to width m; 30 repeats. The y-axis is $\log_{10}$-scale NMSE. **a)** Lorenz-63 system. **b)** Gravity pendulum ODE with GRF (t). **c)** Parametrized Sine-Gordon PDE. **d)** Diffusion PDE with GRF (x).

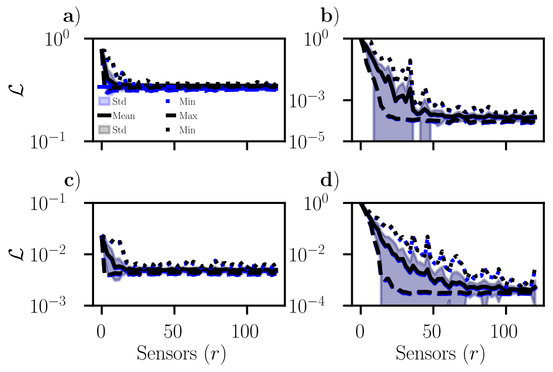

Figure 12: ExtremONet train loss- (blue) and test loss (black) mean, standard deviation, maximum, and minimum with respect to the amount of sensors (30 repeats). The y-axis is $\log_{10}$-scale NMSE. **a)** Lorenz-63 system. **b)** Gravity pendulum ODE with GRF (t). **c)** Parametrized Sine-Gordon PDE. **d)** Diffusion PDE with GRF (x).

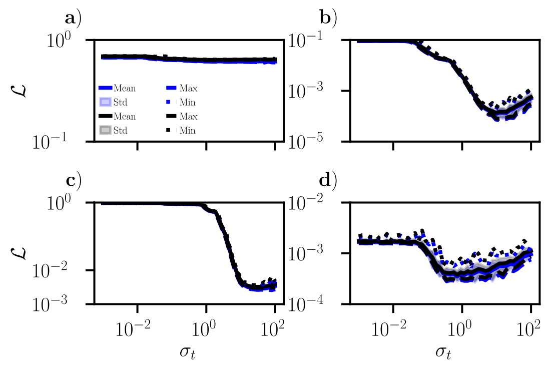
Figure 13: ExtremONet train loss- (blue) and test loss (black) mean, standard deviation, maximum, and minimum with respect to σ_t; 30 repeats. -

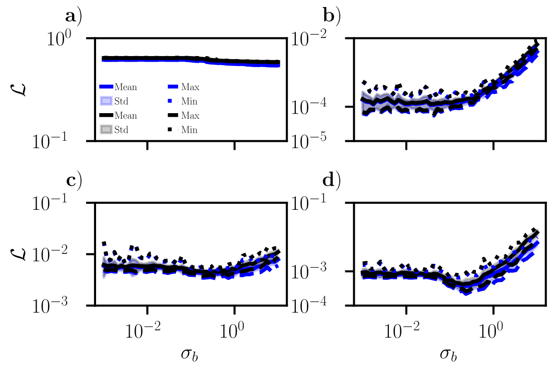
Figure 14: ExtremONet train loss- (blue) and test loss (black) mean, standard deviation, maximum, and minimum with respect to σ_b; 30 repeats. -

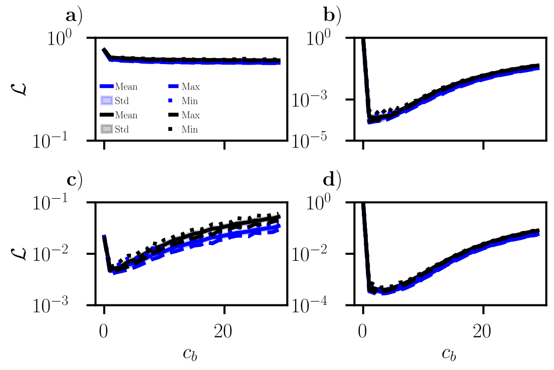
Figure 15: ExtremONet train loss- (blue) and test loss (black) mean, standard deviation, maximum, and minimum with respect to c_b; 30 repeats. -

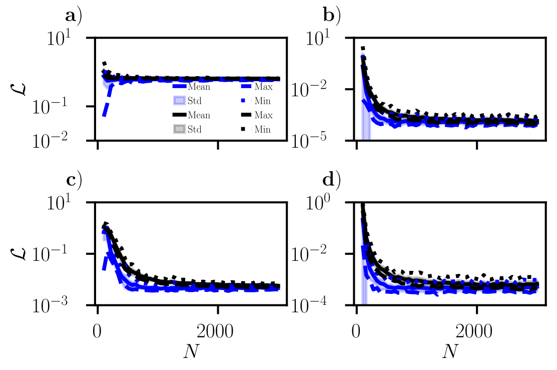
Figure 16: ExtremONet train loss- (blue) and test loss (black) mean, standard deviation, maximum, and minimum with respect to N; 30 repeats. -