Breaking Free: Decoupling Forced Systems with Laplace Neural Networks

Bernd Zimmering[1] (✉)[0000−0001−8292−9945], Cecília Coelho[1,2][0009−0009−4502−937X], Vaibhav Gupta[1][0009−0004−5359−6263], Maria Maleshkova[1][0000−0003−3458−4748], and Oliver Niggemann[1][0000−0001−8747−3596]

[1] Institute for Artificial Intelligence, Helmut Schmidt University Hamburg, Germany
{bernd.zimmering,cecilia.coelho,guptav,maleshkm,oliver.niggemann}@hsu-hh.de
[2] Centre of Mathematics (CMAT), University of Minho, Braga, Portugal

Abstract. Forecasting the behaviour of industrial robots, power grids or pandemics under changing external inputs requires accurate dynamical models that can adapt to varying signals and capture long-term effects such as delays or memory. While recent neural approaches address some of these challenges individually, their reliance on computationally intensive solvers and their black-box nature limit their practical utility. In this work, we propose Laplace-Net, a decoupled, solver-free neural framework for learning forced and delay-aware dynamical systems. It uses the Laplace transform to (i) bypass computationally intensive solvers, (ii) enable the learning of delays and memory effects and (iii) decompose each system into interpretable control-theoretic components. Laplace-Net also enhances transferability, as its modular structure allows for targeted re-training of individual components to new system setups or environments. Experimental results on eight benchmark datasets—including linear, nonlinear and delayed systems—demonstrate the method's improved accuracy and robustness compared to state-of-the-art approaches, particularly in handling complex and previously unseen inputs.

Keywords: Neural Networks · Scientific Machine Learning · Neural Differential Equations · Laplace Transform.

1 Introduction

In *forced* dynamical systems, an external controller C (e.g., a motor or human) drives a dynamical system S (e.g., a pendulum) using input signal $x(t)$ to achieve desired objectives over time t (Fig. 1). The system responds with outputs $y(t)$, evolving from its initial state y_0 based on the interaction of the forcing inputs with its internal dynamics [23]. These systems span various domains such as control engineering, robotics, finance, ecosystems and epidemiology, where external inputs (e.g., economic policies, climate, public health measures) strongly influence system behaviour [21,27,7]. Here, modelling S is essential for designing C, enabling effective control of these complex systems.

Traditionally, modelling such systems has relied on hand-crafted ordinary differential equations (ODEs) with predefined forcing terms [23]. These approaches

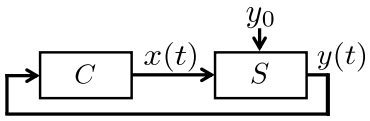

Fig. 1. Control loop with system S that responds with $y(t)$ and is forced by some controller C through excitations $x(t)$. y_0 is the initial state of the system.

require deep domain expertise and often struggle to generalise beyond their original design [32]. In recent years, data-driven methods like *Neural ODEs* [4] have emerged as powerful tools to learn the dynamics of the system directly from the data. Extensions of Neural ODEs [25,16,12,18] incorporate external excitations $x(t)$ into their models, but rely on iterative ODE solvers. Such solvers can be computationally expensive and prone to numerical drift over long prediction horizons [2]. Although alternative methods [1,14] have been proposed to address these challenges, they often fail to fully decouple the dynamics of the system S from the external controller C, limiting their flexibility in handling arbitrary forcing inputs.

To overcome these limitations, *Neural Operators* (NOs) have been introduced as a solver-free alternative for learning mappings directly from inputs $x(t)$ to outputs $y(t)$ [20,3]. However, NOs frequently struggle to capture critical *memory effects*, such as delays or non-local behaviours (i.e. fractional differential equations (FDEs) as described in [9]), which are essential for accurately modelling many real-world systems. Recent studies [28,6,22] emphasise the importance of addressing these memory effects to improve model fidelity.

Another promising avenue involves leveraging the Laplace Domain (LD) for learning forced dynamical systems. LD-based methods have shown strong performance in capturing both forced dynamics and delays [3,14]. Although the Laplace transform has long been a cornerstone in fields like control theory, fluid dynamics, and systems biology, its integration into modern machine learning frameworks remains relatively under-explored. A recent study [30] highlights the conceptual parallels between classical engineering approaches (e.g., transfer functions, ODE models) and current machine learning methods such as Neural ODEs and Neural Laplace. These similarities are discussed in detail using the Spring-Mass-Damper (SMD) system as a running example, which also serves as a benchmark in this study.

In many real-world scenarios - such as adapting robotic controllers to new hardware configurations or refining economic forecasts under changing regulations - modular and *decoupled* architectures offer significant advantages. By isolating specific components of a model (e.g., internal dynamics vs. external forcing), these architectures enable efficient adaptation to new scenarios without requiring a complete model retraining. This modularity not only reduces computational costs but also enhances interpretability by aligning learned components with classical theoretical insights.

Despite the progress made by existing methods, many remain constrained by specific assumptions about input signals, overlook delays or other memory effects or rely heavily on iterative solvers. These limitations can lead to numerical drift, high computational overheads, or inadequate modelling of long-term dependencies - especially in scenarios involving complex forcing signals or extended prediction horizons.

To address these challenges, we propose a novel modular Laplace-based framework that explicitly handles external forcing and memory effects without relying on iterative solvers (as summarised in Table 1). Our main contributions can be summarized as follows:

- **Decoupled Laplace Representation:** We introduce Laplace-Net, a decoupled NN architecture that employs the Laplace domain with an explicit factorization that separates internal system dynamics from external forcing and initial states, aligning with classical system theory.
- **Arbitrary Forcing and Intervention Handling:** Our approach accepts time-varying or previously unseen inputs without retraining the entire model, facilitating straightforward adaptation to new control signals or external perturbations.
- **Solver-Free, Memory-Aware Inference:** We follow Holt et al. [14] employing numerical inverse Laplace transforms and thus avoiding iterative integration. We mitigate numerical drift and capture memory effects (e.g., delays or fractional dynamics) within the same framework.
- **Enhanced Transferability and Interpretability:** The decoupling into well established subcomponents enables their reuse or pre-training and improves interpretability for experts.
- **Improved Performance:** Laplace-Net consistently outperforms LNO across all datasets and surpasses LSTM on 6 out of 8 datasets, demonstrating its effectiveness in capturing linear, non-linear, chaotic, and memory-dependent dynamics.

The remainder of this paper is structured as follows: Section 3 formally defines the problem and introduces key concepts related to Laplace transforms. Section 4 presents the proposed decoupled Laplace-based framework. Section 5 evaluates our approach on linear, non-linear and delayed systems.[3] The conclusions and future directions are discussed in Section 6.

2 Related Work

Data-driven modelling of dynamical systems encompasses a wide range of neural approaches, many of which differ in their reliance on time-stepping solvers, ability to model memory effects or inhomogeneous excitations, and degree of modular decomposition. Table 1 summarizes representative methods evaluated across four key aspects:

[3] Code available at `https://github.com/zimmer-ing/Laplace-Net`

(i) Solver-Free indicates whether the method avoids ODE solvers during training, which are compute intensive.
(ii) Memory assesses the capability to explicitly handle delays (as in Delay Differential Equations, DDEs) and non-local behavior (as found in Fractional Differential Equations, FDEs [8]).
(iii) Arbitrary Forcing evaluates the ability to generalize to unseen or non-parametric input trajectories $x(t)$.
(iv) Decoupled reflects the extent to which the approach separates system dynamics from external inputs and initial conditions, specifically into components that mimic theoretical concepts.

Table 1. Comparison of neural approaches for modelling dynamical systems along four key dimensions: solver-free inference; memory (e.g. delays) ; arbitrary forcing; modularity.

Method	Reference	Solver Free	Memory	Arbitrary Forcing	Decoupled
Neural ODE	[4]	×	×	×	×
Neural FDE	[6,29]	×	✓	×	×
Neural DDE	[28]	×	✓	×	×
ODE-RNN	[25]	×	×	✓	×
Fourier NODE	[18]	✓†	×	✓	×
Neural IM	[12]	×	×	✓	✓
Neural CDE	[16]	×	×	✓	×
Neural Flow	[1]	✓	×	×	×
Neural Laplace	[14]	✓	✓	×	×
Neural Laplace Control	[13]	✓	✓	✓	×
DeepONet	[20]	✓	×	✓	×‡
Laplace NO	[3]	✓	×	✓	✓
Laplace-Net	This Work	✓	✓	✓	✓

† Fourier NODEs eliminate the need for a solver during training but still require one for inference.
‡ DeepONet is partially decoupled: the branch network encodes both initial and external inputs, while the trunk network models system dynamics. However, the separation between initial conditions and external inputs within the branch network is not complete.

Neural ODEs and Solver-Based Extensions. *Neural ODEs* [4] and their immediate variants, such as *ODE-RNN* [25], *ANODE* [11], and *Neural IM* [12], parameterise ODE vector fields in a continuous latent space. These approaches have demonstrated their effectiveness in various scenarios, including homogeneous and some forced systems. However, they still rely on iterative numerical solvers such as Runge-Kutta, which can lead to substantial computational costs during

training and error accumulation over extended time horizons. Extensions of this framework include approaches for delayed [28,22] and fractional systems [8]. *Neural Delayed DEs* [28] and *Neural Fractional DEs* [5,29] explicitly embed delay or learn the amount of memory [6] but, like their predecessors, rely on solver-based training loops and do not explicitly separate forcing inputs.

Laplace-Based, Solver-Free Methods. *Neural Laplace* [14] revolutionizes the approach to learning DEs by operating in the Laplace domain, eliminating the need for stepwise integration during both training and prediction. This method employs a numerical inverse Laplace transform algorithm to generate time-domain predictions, enabling the capture of complex memory-like effects such as fractional-order behaviour and delay times. While *Neural Laplace* claims to handle forced differential equations, its implementation is limited to forcing functions that remain constant between training and testing phases, rather than accommodating arbitrary inputs. *Neural Laplace Control* [13] extends the framework for reinforcement learning scenarios. However, it only considers past actions during prediction, not accounting for future actions, restricting its applicability. *Fourier NODE* [18] offers an alternative approach, approximating state derivatives in the frequency domain to avoid using ODE solvers during training. This method explicitly incorporates control inputs for trajectory prediction, enhancing its versatility in handling forced systems. However, *Fourier NODE* still requires a numerical solver during inference. This requirement limits its applicability for resource-constrained scenarios (e.g. edge devices).

Operator Learning for Forced Systems. Modern approaches, such as *DeepONets* [20] and *Laplace Neural Operators* [3], enable *solver-free* evaluations by learning mappings from input (forcing) functions directly to solution functions. While initially developed for Partial Differential Equations (PDE), some works have successfully extended their application to ODEs [20,3]. *DeepONets*, for instance, employ a unique architecture consisting of two key components: a branch network that encodes inputs functions, such as initial conditions and forcing profiles, into a high-dimensional vector; a trunk network that processes evaluation points of the solution. This structure allows *DeepONets* to effectively capture complex solution behaviours across a wide range of input conditions. However, the method's unified approach to encoding both system dynamics and external influences can limit the interpretability and modular reuse of the learned representations. In contrast, *Laplace Neural Operators* introduce a degree of modularization by separating internal system dynamics from external inputs within the Laplace domain. By restricting themselves to a pole–residue representation (Eq. (5) in [3]), these operators enable the inverse Laplace transform to be carried out symbolically, simplifying implementation. However, such a representation struggles to capture delayed or fractional dynamics (Eq. (15) in [15], Theorem 5.4 in [19], and Example 1.25 in [26]), as their Laplace transforms do not generally reduce to simple pole–residue forms. Methods like *Neural Laplace* [14] address this limitation by numerically computing the inverse Laplace transform, thereby

allowing for a wider range of dynamical phenomena beyond the pole–residue framework.

Bridging Gaps via Decoupling and Generalized Forcing. As highlighted in Table 1, a recurring shortfall in existing methods is the lack of an *explicit* factorization of the system's internal transfer characteristics, initial conditions, and arbitrary forcing signals — especially in a solver-free framework. Some methods, such as *Neural IM* [12] and *ODE-RNN* [25], partially decouple interventions or control inputs, but they often rely on iterative updates over time or fail to handle continuous forcing seamlessly. Similarly, *Neural Laplace Control* [13] extends Holt et al. [14] by incorporating past actions into a latent state representation, yet it assumes a homogeneous response for predictions and does not account for future forcing inputs, limiting its applicability in scenarios requiring forward-looking control.

3 Preliminaries

We consider a dynamical system $\mathcal{S}$ with input $\mathbf{x}(t) \in \mathbb{R}^{D_x}$ and response $\mathbf{y}(t) \in \mathbb{R}^{D_y}$. Let $\mathbf{t} = (t_1, \ldots, t_{N+M}) \subset [0, T]$ be discrete times $(t_1 < \cdots < t_{N+M})$, where the first N indices partition the *historical* segment $\mathbf{t}_{\text{hist}}$ and the final $(N+1, \ldots, N+M)$ indices partition the *forecast* segment $\mathbf{t}_{\text{fore}}$. We collect samples into $\mathbf{X} \in \mathbb{R}^{(N+M)\times D_x}$ and $\mathbf{Y} \in \mathbb{R}^{(N+M)\times D_y}$ with the partitions $\mathbf{X}_{\text{hist}} \in \mathbb{R}^{N\times D_x}$, $\mathbf{X}_{\text{fore}} \in \mathbb{R}^{M\times p}$, $\mathbf{Y}_{\text{hist}} \in \mathbb{R}^{N\times q}$, and $\mathbf{Y}_{\text{fore}} \in \mathbb{R}^{M\times q}$ defined analogously. Our objective is to predict $\mathbf{Y}_{\text{fore}}$ given $\mathbf{X}_{\text{hist}}, \mathbf{Y}_{\text{hist}}$, and $\mathbf{X}_{\text{fore}}$. Formally, we learn

$$f : \big(\mathbf{X}_{\text{hist}},\, \mathbf{Y}_{\text{hist}},\, \mathbf{X}_{\text{fore}}\big) \;\mapsto\; \mathbf{Y}_{\text{fore}}. \tag{1}$$

Although the above forecasting objective is stated in the time domain, a powerful way to analyse and solve differential equations is via the Laplace transform. We therefore briefly recall the key properties of this transform and its inverse as they form the foundation of our solution.

The Laplace transform maps time-domain signals into the complex s-domain, where derivatives become algebraic factors. For a function $\mathbf{y}(t)$, the Laplace transform is:

$$\boldsymbol{\mathcal{Y}}(s) \;=\; \mathcal{L}\{\mathbf{y}(t)\} \;=\; \int_0^\infty e^{-st}\,\mathbf{y}(t)\,\mathrm{d}t, \quad s \in \mathbb{C}. \tag{2}$$

The complex variable s is typically written as $s = \sigma + i\omega$, where $\sigma \in \mathbb{R}$ corresponds to exponential growth/decay and $\omega \in \mathbb{R}$ to oscillatory behaviour. Applied to the n-th derivative, the Laplace transform yields:

$$\mathcal{L}\Big\{\frac{d^n\mathbf{y}(t)}{dt^n}\Big\} \;=\; s^n\,\boldsymbol{\mathcal{Y}}(s) \;-\; \sum_{k=0}^{n-1} s^{\,n-1-k}\;\frac{d^k\mathbf{y}(0)}{dt^k}. \tag{3}$$

A pure time delay $\tau \in \mathbb{R}_{>0}$ appears as $e^{-\tau s}$ in the Laplace domain, and fractional derivatives can be treated similarly to Eq. (3)[26,19,15].

Solving an ODE in the Laplace domain calls for the inverse Laplace transform (ILT),

$$\mathbf{y}(t) \;=\; \mathcal{L}^{-1}\{\boldsymbol{\mathcal{Y}}(s)\} \;=\; \frac{1}{2\pi i}\int_{\sigma-i\infty}^{\sigma+i\infty} \boldsymbol{\mathcal{Y}}(s)\, e^{st}\, \mathrm{d}s, \tag{4}$$

which is rarely solvable in closed form.

Since direct evaluation of Eq. (4) is rarely feasible, numerical methods are required. Among them, the Fourier series-based ILT [10] was identified as the most robust for boundary element simulations [17] and proposed for machine learning due to its efficiency and stability [14]. As the name already implies, it employs Fourier transforms at several points in time. It reconstructs smooth, real-valued signals by sampling $\boldsymbol{\mathcal{Y}}(s)$ at a sequence of query points (cf. Eq. (6)) along a vertical line (parallel to the imaginary axis) in the complex plane. The real component of this line is shifted by a time-dependent offset σ. For each time point, this vertical contour is repositioned accordingly. The numerical ILT is given by:

$$\mathbf{y}(t) \approx \frac{1}{\zeta t} e^{\sigma t}\left[\frac{\boldsymbol{\mathcal{Y}}(s_0)}{2} + \sum_{k=1}^{N_{\mathrm{ILT}}} \mathrm{Re}\Big\{\boldsymbol{\mathcal{Y}}(s_k)\, e^{\,i\frac{k\pi t}{\zeta t}}\Big\}\right], \quad t>0. \tag{5}$$

Here, $\zeta \in \mathbb{R}_{>0}$ controls the frequency resolution, N_{ILT} is the number of frequency terms along the imaginary axis. Together ζ and N_{ILT} determine the bandwidth as well as the maximum frequency to be transformed. Please note, that compared to the original literature [10,17] we follow [14] and directly use ζt as a scaling factor.

The query points s_0, s_k define a discrete *grid* of evaluation points for $\boldsymbol{\mathcal{Y}}(s)$:

$$s_k(t) = \sigma + i\frac{k\pi}{\zeta t}, \quad k \in \mathbb{N}_0, \quad k \le N_{\mathrm{ILT}}, \quad \text{with} \quad \sigma = \alpha - \frac{\log(\epsilon)}{\zeta t}. \tag{6}$$

Here, $\alpha \in \mathbb{R}_{>0}$ is chosen so that the contour lies to the right of all singularities of $\boldsymbol{\mathcal{Y}}(s)$, ensuring convergence (typically $\alpha = 10e^{-3}$). $\epsilon \in \mathbb{R}_{>0}$ is used for numerical precision in the computation of σ and typically chosen $\epsilon = 10\alpha$. Intuitively, increasing N_{ILT} increases the density of query points along the imaginary axis, allowing finer resolution of frequency components. The time variable t shifts the contour along the real axis, while ζ scales the overall size of the grid in both directions.

4 Solution

To model Eq. (1), we propose Laplace-Net, a Laplace-based neural network (NN) that removes the solver, decouples initial conditions from inputs, and encapsulates system dynamics. We first decompose the system response in line with classical control theory [23] and then integrate NNs into this framework.

4.1 Decomposition of System Responses

To illustrate the decomposition of $\boldsymbol{\mathcal{Y}}(s)$ into decoupled components, we consider a differential equation of the form:

$$\sum_{i=0}^{n} \mathbf{A}_i \, \frac{d^i \mathbf{y}(t)}{dt^i} \;=\; \mathbf{B}\,\mathbf{x}(t), \tag{7}$$

where $\mathbf{A}_i \in \mathbb{R}^{D_y \times D_y}$ and $\mathbf{B} \in \mathbb{R}^{D_y \times D_x}$ are constant matrices.

Applying the Laplace transform $\mathcal{L}\{\cdot\}$ to both sides and using Eq. (3) yields:

$$\sum_{i=0}^{n} \mathbf{A}_i \left(s^i \, \boldsymbol{\mathcal{Y}}(s) \;-\; \sum_{k=0}^{i-1} s^{i-1-k} \, \frac{d^k \mathbf{y}(0)}{dt^k} \right) = \mathbf{B}\boldsymbol{\mathcal{X}}(s). \tag{8}$$

Rearranging terms and isolating $\boldsymbol{\mathcal{Y}}(s)$, assuming the invertibility of $\left(\sum_{i=0}^{n} \mathbf{A}_i s^i\right)^{-1}$, results in:

$$\boldsymbol{\mathcal{Y}}(s) = \Big(\sum_{i=0}^{n} \mathbf{A}_i s^i\Big)^{-1} \left(\mathbf{B}\boldsymbol{\mathcal{X}}(s) + \sum_{i=0}^{n} \mathbf{A}_i \sum_{k=0}^{i-1} s^{i-1-k} \frac{d^k \mathbf{y}(0)}{dt^k} \right). \tag{9}$$

o simplify the structure of the solution, we introduce the notation $\boldsymbol{\mathcal{H}}(s) \in \mathbb{C}^{D_y \times D_y}$ and $\boldsymbol{\mathcal{P}}(s) \in \mathbb{C}^{D_y}$ as follows:

$$\boldsymbol{\mathcal{H}}(s) := \Big(\sum_{i=0}^{n} \mathbf{A}_i s^i\Big)^{-1}, \quad \boldsymbol{\mathcal{H}}(s) \in \mathbb{C}^{D_y \times D_y}, \tag{10}$$

$$\boldsymbol{\mathcal{P}}(s) := \sum_{i=0}^{n} \mathbf{A}_i \sum_{k=0}^{i-1} s^{i-1-k} \frac{d^k \mathbf{y}(0)}{dt^k}, \quad \boldsymbol{\mathcal{P}}(s) \in \mathbb{C}^{D_y}. \tag{11}$$

Using Eqs. (10) and (11) we can simplify Eq. (9) to:

$$\boldsymbol{\mathcal{Y}}(s) = \boldsymbol{\mathcal{H}}(s)\big(\mathbf{B}\boldsymbol{\mathcal{X}}(s) + \boldsymbol{\mathcal{P}}(s)\big), \tag{12}$$

which represents the decomposition of Eq. 7 into separate components: the system dynamics given by $\boldsymbol{\mathcal{H}}(s)$, the influence of initial conditions captured in $\boldsymbol{\mathcal{P}}(s)$, and the external excitations $\mathbf{B}\boldsymbol{\mathcal{X}}(s)$.

4.2 Neural Network-based Approximation

To generalize the decomposition in Eq. (12), we introduce **Laplace-Net**. Figure 2 provides an overview, and Algorithm 1 details the computational steps. First, historical input-output sequences are encoded into $(\mathbf{P}, \mathbf{z})$, where $\mathbf{P}$ captures initial conditions. Using $\mathbf{P}$ and queries $\mathbf{s}$, $\boldsymbol{\mathcal{P}}(s)$ is formed in the Laplace domain. Second, the external input $\mathbf{X}_{\text{fore}}$ undergoes a numerical Laplace transform and is mapped into the output space via $\mathbf{B}$. Third, a NN approximates the transfer function $\boldsymbol{\mathcal{H}}(s)$, which, together with the other components, is combined using complex-valued operations to compute $\boldsymbol{\mathcal{Y}}(s)$ in the Laplace domain. Finally, an inverse Laplace transform reconstructs the time-domain output $\mathbf{Y}_{\text{fore}}$. To handle long sequences and non-linearities, this process runs recurrently with an adjustable stride δ, resulting in Q steps.

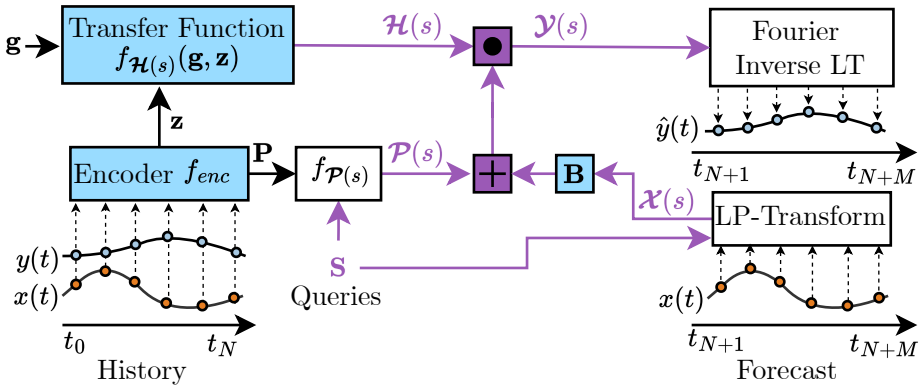

Fig. 2. Overview of the Laplace-Net architecture. Blue elements represent learnable matrices or NNs, including an encoder for historical data and a trainable transfer function. Purple elements denote complex-valued components.

External Input $X(s)$: The Laplace transform of the external input can be computed numerically using either a discrete summation or a Fourier-based approach. The Discrete Laplace Transform (DLT) follows directly from the definition in Equation (2):

$$\mathcal{X}(s) = \sum_{k=0}^{N-1} x(t_k)\, e^{-st_k}\, \Delta t, \tag{13}$$

where $\Delta t = t_{k+1} - t_k$ is the time increment.

If $x(t)$ is assumed periodic, it can be represented as a Fourier series [3]:

$$x(t) = \sum_{k=-K}^{K} a_k e^{i\omega_k t}, \quad 0 \le t < T, \tag{14}$$

where a_k and ω_k are the Fourier coefficients and frequencies, respectively. Using the Fast Fourier Transform (FFT), these coefficients can be efficiently computed. Applying the Laplace transform yields:

$$\mathcal{X}(s) = \sum_{k=-K}^{K} \frac{a_k}{s - i\omega_k}. \tag{15}$$

This approach, referred to as the Fast Fourier Laplace Transform (FFLT), exploits the relationship between Fourier and Laplace transforms.

History Encoding and Initial State $\mathcal{P}(s)$ We represent the initial condition term $\mathcal{P}(s)$ via a combination of an encoder network and an analytic structure, rather than explicitly computing its Laplace transform. Specifically, we introduce a

Algorithm 1 The Laplace-Net algorithm.

Require: $(\mathbf{t}_{\text{hist}}, \mathbf{X}_{\text{hist}}, \mathbf{Y}_{\text{hist}})$, $(\mathbf{t}_{\text{fore}}, \mathbf{X}_{\text{fore}})$, δ, N_{ILT}; **Ensure:** $\mathbf{Y}_{\text{pred}}$

function LAPLACE-NET($\mathbf{t}_{\text{hist}}, \mathbf{X}_{\text{hist}}, \mathbf{Y}_{\text{hist}}, \mathbf{t}^{(z)}_{\text{fore}}, \mathbf{X}^{(z)}_{\text{fore}}, N_{\text{ILT}}$)
  $(\mathbf{P}, \mathbf{z}) \leftarrow f_{\text{enc}}(\mathbf{t}_{\text{hist}}, \mathbf{X}_{\text{hist}}, \mathbf{Y}_{\text{hist}})$
  Compute Queries $\mathbf{s}$ from $\mathbf{t}^{(z)}_{\text{fore}}$ by Eq. (6)
  $\boldsymbol{\mathcal{P}}(s) \leftarrow f_{\boldsymbol{\mathcal{P}}(s)}(\mathbf{P}, \mathbf{s})$ ▷ Eq. (18)
  Compute grid $\mathbf{g}$ from $\mathbf{t}_{\text{fore}}$,N_{ILT} by Eq. (19)
  $\boldsymbol{\mathcal{H}}(s) \leftarrow f_{\boldsymbol{\mathcal{H}}(s)}(\mathbf{g}, \mathbf{z})$ ▷ Eq.(20)
  $\boldsymbol{\mathcal{H}}(s) \leftarrow \frac{\boldsymbol{\mathcal{H}}(s)}{f_{\text{scale}}(\mathbf{t}_{\text{fore}}) \cdot \kappa_{\boldsymbol{\mathcal{H}}(s)}}$ ▷ optional step (Eq. (22))
  $\boldsymbol{\mathcal{X}}(s) \leftarrow \mathcal{L}(\mathbf{t}^{(z)}_{\text{fore}}, \mathbf{X}^{(z)}_{\text{fore}}, \mathbf{s})$ ▷ by either Eq. (13) or Eq. (15)
  $\boldsymbol{\mathcal{Y}}(s) \leftarrow \boldsymbol{\mathcal{H}}(s)(\mathbf{B}\boldsymbol{\mathcal{X}}(s) + \boldsymbol{\mathcal{P}}(s))$ ▷ Eq. (12)
  return Numerical ILT $f_{\mathcal{L}^{-1}}(\boldsymbol{\mathcal{Y}}(s), \mathbf{t}_{\text{fore}})$ ▷ by Eq. (5)
end function
$\mathbf{Y}_{\text{pred}} \leftarrow \emptyset$, $Q \leftarrow \left\lceil \frac{M}{\delta} \right\rceil$
for $q = 0$ to $Q - 1$ **do**
  $(\mathbf{t}^{(z)}_{\text{fore}}, \mathbf{X}^{(z)}_{\text{fore}}) \leftarrow$ Extract window
  $\mathbf{Y}^{(z)}_{\text{pred}} \leftarrow$ Laplace-Net($\mathbf{t}_{\text{hist}}, \mathbf{X}_{\text{hist}}, \mathbf{Y}_{\text{hist}}, \mathbf{t}^{(z)}_{\text{fore}}, \mathbf{X}^{(z)}_{\text{fore}}, N_{\text{ILT}}$)
  $(\mathbf{t}_{\text{hist}}, \mathbf{X}_{\text{hist}}, \mathbf{Y}_{\text{hist}}) \leftarrow$ Update history
  $\mathbf{Y}_{\text{pred}} \leftarrow \mathbf{Y}_{\text{pred}} \cup \mathbf{Y}^{(z)}_{\text{pred}}$
end for
return $\mathbf{Y}_{\text{pred}}$

history-dependent parameter $\mathbf{P} \in \mathbb{R}^{D_y \times P}$ and a latent variable $\mathbf{z}$, both inferred by a history encoder:

$$f_{\text{enc}} \colon \mathbb{R}^1 \times \mathbb{R}^{T_{\text{hist}} \times d_x} \times \mathbb{R}^{T_{\text{hist}} \times d_y} \;\to\; \mathbb{R}^{D_z \times P} \times \mathbb{R}^{D_z}, \tag{16}$$

such that:

$$\mathbf{P},\, \mathbf{z} \;=\; f_{\text{enc}}\big(\mathbf{t}_{\text{hist}}, \mathbf{X}_{\text{hist}},\, \mathbf{Y}_{\text{hist}}\big). \tag{17}$$

Here, $\mathbf{P}$ collects information needed to construct the polynomial representation of the initial state, and $\mathbf{z}$ is a latent state capturing additional system characteristics from historical data.

Comparing $\boldsymbol{\mathcal{P}}(s)$ to the analytic form in (11), we note that $\boldsymbol{\mathcal{P}}(s)$ is a polynomial in s whose coefficients depend on system matrices $\mathbf{A}_i$ and initial states $\frac{d^k \mathbf{y}(0)}{dt^k}$. Thus, the initial state term can be rewritten as:

$$P(s) \;=\; f_{\boldsymbol{\mathcal{P}}(s)}\big(s, \mathbf{P}\big) \;=\; \sum_{i=0}^{P-1} \mathbf{p}_i\, s^i \quad \text{with} \quad \mathbf{P} \;=\; \big(\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{P-1}\big). \tag{18}$$

Here, $P \in \mathbb{N}_{>0}$ is a hyper parameter that reflects the order n of Eq. (8), which usually only can be guessed for real-world datasets. This formulation preserves the analytic dependence on initial conditions while offering a flexible, data-driven adaptation. By encoding $\mathbf{P}$ and $\mathbf{z}$ jointly, the model can capture system properties without direct knowledge of $\mathbf{A}_i$ or $\mathbf{y}(0)$ and its derivates.

Transfer Function $\boldsymbol{\mathcal{H}}(s)$ Next, we learn the overall transfer function $\boldsymbol{\mathcal{H}}(s)$ via a NN $f_{\boldsymbol{\mathcal{H}}(s)}$, which takes as input a 2D-grid $\mathbf{g} \in \mathbb{R}^{N_{\text{ILT}} \times T}$. Here, N_{ILT} is the number of ILT terms (see Eq. (5)), and T is the forecast horizon. Although the ILT method naturally produces the (complex) query points $s_k(t)$ (Eq. (6)), these can grow unbounded for small t, leading to extreme input values for the NN. This results in large weight updates, potentially causing gradient instability and slowing down convergence. To address this, we map the ILT queries onto a normalized grid $\mathbf{g} = [g_{m,n}]$, ensuring numerical stability while preserving spectral and temporal structure. The ILT axis remains uniform, while irregularly sampled time points t_n are scaled to $[-1, 1]$ using:

$$g_{m,n} = 2\frac{t_n - t_{\min}}{t_{\max} - t_{\min}} - 1. \tag{19}$$

This ensures that $t_{\min}$ maps to -1 and $t_{\max}$ to 1, preserving relative spacing.

The transfer function is then modelled by:

$$\boldsymbol{\mathcal{H}}(s) \;=\; f_{\boldsymbol{\mathcal{H}}(s)}(\mathbf{g}, \mathbf{z}), \tag{20}$$

where $\mathbf{z}$ is precisely the latent state inferred by the history encoder (17). Conditioning on $\mathbf{z}$ allows $\boldsymbol{\mathcal{H}}(s)$ to adapt to varying system behaviours and capture locally linear approximations of potentially non-linear processes.

Inverse Laplace Transform and Time-Independent Scaling In Eq. (5), the contour parameter is often set as $\lambda = \zeta t$, making $\boldsymbol{\mathcal{Y}}(s)$ explicitly time-dependent. To remove this dependency, we define the scaling factor[4]:

$$f_{\text{scale}}(t) = \zeta t\, e^{-\sigma t} = \zeta t\, \epsilon^{\frac{1}{\zeta}} e^{-\alpha t}. \tag{21}$$

When scaling is used, we get $\widetilde{\boldsymbol{\mathcal{H}}}(s)$ by Eq. (20) which is transformed into the time dependent transfer function along the t axis by:

$$\boldsymbol{\mathcal{H}}(s) = \frac{\widetilde{\boldsymbol{\mathcal{H}}}(s)}{f_{\text{scale}}(t) \cdot \kappa_{\boldsymbol{\mathcal{H}}(s)}}. \tag{22}$$

Here, the scaling parameter $\kappa_{\boldsymbol{\mathcal{H}}(s)}$ stabilises training by preventing the amplification of small variations in $\widetilde{\boldsymbol{\mathcal{H}}}(s)$ due to division by $f_{\text{scale}}(t)$, ensuring a well-conditioned and robust representation.

5 Evaluation

We evaluate our method by benchmarking it against a sequence-to-sequence LSTM (seq2seq LSTM) and the LNO of [3] on eight univariate dynamical system datasets that differ in complexity and characteristic behaviour. By restricting ourselves to the univariate case, we concentrate on the core dynamics without the complexity of multiple input channels. In this formulation, the input-to-state $\mathbf{B}$ is simply a scalar, set to 1.

[4] A full derivation is provided in Appendix A of the supplementary material.

Experimental Setup. All datasets consist of uniformly sampled time series with 50 historical data points and a forecast horizon of 500 steps. We employ a train-validation-test split: the training set is used to learn model parameters, the validation set for hyperparameter tuning, and the test set for final performance.

For the Spring-Mass-Damper System (SMD) and Mackey-Glass data, we generate our own univariate time series by applying three distinct periodic signals for training (sigmoid), validation (decaying sine), and testing (triangular). In contrast, for the Duffing, Lorenz, and driven pendulum datasets, we adopt the data of [3] directly, which rely on decaying sinusoidal input signals $x(t)$ whose decay coefficient varies across samples. To reduce experimental bias, we determine the learning rate and other model-specific hyperparameters[5] via the Tree-Parzen Estimator (TPE) [24] in `optuna`, using 100 TPE trials per dataset for each method. We then use the best hyper-parameters and train each method six times with different random seeds to assess robustness. All datasets are processed in a single pass (batch size 512) per epoch. Our experiments run on a high-performance cluster with eight GPUs (Nvidia A40 and A100). In total we run 2544 trainings to for our setup.

Test Systems and Results. Table 2 presents the mean and standard deviation of the MSE on the test set over six repeated runs. Below, we provide a brief overview of each system and comment on the observed results. We start with a

Table 2. Mean ± (standard deviation over six seeds) for MSE on test set. **Bold** values indicate the best result per data set.

Dataset/Model	LNO	LSTM	LP-Net
SMD	1.88e-01 (1.48e-01)	**3.56e-04 (4.53e-05)**	8.75e-04 (2.46e-04)
Duffing $c = 0$	7.42e-02 (3.93e-02)	1.21e-01 (2.21e-02)	**1.98e-02 (6.52e-03)**
Duffing $c = 0.5$	1.06e-03 (1.04e-04)	1.33e-03 (8.73e-04)	**5.31e-05 (1.23e-05)**
Lorenz $\rho = 5$	4.05e-02 (1.18e-02)	**1.21e-04 (2.77e-05)**	5.19e-04 (1.09e-04)
Lorenz $\rho = 10$	5.36e+00 (5.97e-01)	2.10e+00 (3.79e-01)	**1.31e+00 (3.27e-01)**
Pendulum $c = 0$	5.81e-01 (9.83e-02)	6.89e-01 (6.43e-02)	**6.61e-03 (1.67e-03)**
Pendulum $c = 0.5$	1.08e-03 (7.37e-04)	6.92e-04 (1.52e-04)	**5.10e-05 (2.00e-05)**
Mackey-Glass	5.90e-01 (2.08e-01)	3.50e-01 (6.99e-02)	**8.83e-03 (3.27e-03)**

simple linear **Spring-Mass-Damper (SMD)** system, also used in [30] as an application example:

$$m\,\ddot{y}(t) + c\,\dot{y}(t) + k\,y(t) = x(t), \quad m, c, k \in \mathbb{R}^+, \tag{23}$$

where m is the mass, c the damping, and k the spring constant. It has an initial displacement $y(0) = y_0$ as well as initial velocity $\dot{y}(0) = v_0$. As the table shows,

[5] Details on the best hyperparameters found by the TPE can be found in Appendix B of the supplementary material.

the seq2seq LSTM achieves the lowest error, yet Laplace-Net still surpasses LNO and provides a stable fit. This indicates that, although SMD is comparatively simpler to model, Laplace-Net remains competitive.

The **Duffing oscillator:**

$$m\,\ddot{y}(t) + c\,\dot{y}(t) + k_1\,y(t) + k_3\,y^3(t) = x(t), \quad m, c, k_1, k_3 \in \mathbb{R}^+, \tag{24}$$

introduces a cubic spring (constant k_3). Following [3], we use damped ($c = 0.5$), which provides transient bahavior as the oscillations decline and undamped ($c = 0$) where oscillatory behaviour dominated. Notably, Laplace-Net achieves the best results for both scenarios, capturing non-linear oscillatory behaviour accurately in the damped case.

The **Lorenz system:**

$$\dot{s}_x = \sigma(y - s_x), \quad \dot{y} = s_x(\rho - s_z) - y, \quad \dot{s}_z = s_x y - \beta s_z - x(t), \quad \sigma, \rho, \beta \in \mathbb{R}^+, \tag{25}$$

is a three-dimensional chaotic model where s_x, s_z are internal states, $\rho = 5$ or $\rho = 10$ sets the degree of chaos. For $\rho = 5$, the table indicates that the LSTM gives the best result. However, at $\rho = 10$, Laplace-Net emerges on top, evidencing superior adaptability under stronger chaotic dynamics.

Another non-linear system is the **Driven pendulum:**

$$\ddot{x}(t) + c\,\dot{x}(t) + \frac{g}{l}\,\sin\big(x(t)\big) = x_{\text{ext}}(t), \quad c, g, l \in \mathbb{R}^+, \tag{26}$$

where g is the gravity and l is the rod length. Again a damped ($c = 0.5$) and undamped ($c = 0$) similar to the Duffing system, a higher damping leads to more transient, decaying behaviour. Here, Laplace-Net clearly outperforms LSTM and LNO in both scenarios, underscoring its strength in modelling non-linear oscillatory phenomena.

The **Mackey-Glass system:**

$$\dot{y}(t) = \beta \frac{y(t-\tau)}{1 + [y(t-\tau)]^n} - \gamma y(t) + x(t), \quad \beta, \gamma, \tau, n \in \mathbb{R}^+, \tag{27}$$

brings explicit time delays τ, which often pose challenges for standard recurrent architectures. Here, β controls the strength of the delayed feedback, while γ represents the dissipation rate. Laplace-Net attains a particularly low error, whereas for the LSTM and LNO we observe that they are not able to capture the behaviour. Because all models share the same training and tuning procedures, this improvement suggests Laplace-Net retains more effective long-range memory, thus handling delayed feedback more robustly.

Taken together, these findings verify that Laplace-Net is accurate and robust across diverse types of dynamical systems. In most cases, it surpasses both seq2seq LSTM and LNO, particularly for non-linear, chaotic, or delay-based dynamics. This ability to handle forced, damping-induced, and chaotic regimes demonstrates the method's versatility for extended forecasting horizons. Table 2 shows that our method, is able to outperform LNO as well as the LSTM for most of the datasets, except for the Lorenz System with $\rho = 5$.

6 Discussion and Limitations

While Laplace-Net offers a flexible, solver-free approach to modelling forced dynamical systems, it has theoretical and practical limitations. A fundamental constraint arises from the Laplace transform: functions growing super-exponentially, i.e., those for which there exist no constants $C, \sigma > 0$ such that $|f(t)| \leq Ce^{\sigma t}$, are not transformable [26]. This limitation is particularly relevant for transforming input signals $\boldsymbol{\mathcal{X}}(s) = \mathcal{L}\{\mathbf{x}(t)\}$. This restriction is mainly of theoretical interest, as, to the best of our knowledge, the current literature does not report typical cases where this limitation is violated. Furthermore, the Laplace transform is widely used and considered broadly applicable in engineering practice, implicitly assuming that real-world signals do not violate these constraints [23].

Although classical error bounds exist for certain numerical ILT algorithms [10,17], deriving comparable stability or approximation guarantees for Laplace-Net mapping remains an open challenge. Our empirical results indicate robust performance, but a theoretical analysis of the error and stability properties is an important direction for future work. Another challenge lies in the numerical approximation of $\boldsymbol{\mathcal{X}}(s)$ and $\boldsymbol{\mathcal{Y}}(s)$. For signals containing high frequencies (e.g. abrupt jumps), a large number of ILT terms (N_{ILT}) is required for accurate reconstruction. While the computation of the proposed Laplace transforms, such as the Discrete Laplace Transform (DLT) in Eq. (13) and the Fast Fourier Laplace Transform (FFLT) in Eq. (15), can be computationally intensive, it can be performed once prior training.

Memory consumption also presents a constraint, as the computational cost grows linearly with both the number of ILT terms N_{ILT} and the number of prediction time points. Particularly for signals with sharp discontinuities, achieving sufficient accuracy requires large N_{ILT} values, which significantly increases memory requirements. Despite these challenges, the design of Laplace-Net is highly parallelisable, with the exception of time-stepping. However, this overhead is negligible compared to the numerous iterations required by ODE solvers.

One major advantage of Laplace-Net is that the Laplace representation of the input $\boldsymbol{\mathcal{X}}(s)$ is handled independently, allowing for pre-validation or even manual refinement before joint training. This makes debugging and refining the forcing component significantly easier. Moreover, if certain components of the system transfer function $\boldsymbol{\mathcal{H}}(s)$ are already known, for instance, from physical principles, they can be directly embedded into the model while learning only the remaining unknown terms. This hybrid approach reduces the learning burden and allows for more effective integration of prior knowledge. Recent work has demonstrated that incorporating such explicit prior knowledge into neural ODE models for electrical circuits can substantially improve performance, consistently outperforming black-box LSTM and standard NODE architectures, particularly in data-limited scenarios [31].

The decoupled structure also enables selective training strategies. When working with controlled experiments where the system is initialized at zero, the initial state term $\boldsymbol{\mathcal{P}}(s)$ can be explicitly set to zero, removing unnecessary degrees of freedom and simplifying optimization. In real-world settings where the initial

conditions vary, the full model can then be fine-tuned, leveraging pre-trained components for efficient adaptation.

Furthermore the decoupled structure of Laplace-Net improves interpretability: In many fields, the structure of the dynamical system, learned with $\boldsymbol{\mathcal{H}}(s)$ is of interest (e.g. when designing the controller C). Learning it as a dedicated component enables domain experts to interpret what was learned.

7 Conclusion and Future Work

In this work, we introduced a decoupled, solver-free neural network (NN) framework for learning forced and delay-aware dynamical systems: the Laplace-based Network (Laplace-Net). Laplace-Net explicitly separates internal system dynamics from external control inputs and initial conditions, addressing key limitations of existing approaches. Unlike many prior approaches such as Laplace Neural Operator (LNO), can learn systems with hard delays or non-local memory, including those appearing in Fractional Differential Equations (FDE). Laplace-Net enhances transferability by enabling fast retraining or fine-tuning for new forcing signals, which is particularly advantageous in data-limited scenarios. Furthermore, it improves interpretability, as its learned components align with classical theoretical concepts familiar to experts (e.g., transfer functions). Finally, Laplace-Net is highly parallelisable and scales linearly with both the number of frequency terms and prediction time steps.

Laplace-Net demonstrates clear advantages across a wide range of dynamical systems. Evaluated on eight datasets covering linear, non-linear, chaotic, and delayed dynamics, it consistently outperforms the LNO in all cases. Compared to an Long Short-Term Memory (LSTM) model, Laplace-Net achieves superior accuracy on nearly all datasets, with the exception of a linear ODE case and one out of seven non-linear scenarios, as shown in Table 2. These results highlight the effectiveness of Laplace-Net in capturing complex system behaviour across various dynamical systems.

While Laplace-Net shows robust performance on synthetic and univariate datasets, this evidence needs to be enhanced for usage in real-world applications. In particular, the evaluation on real-world datasets as well as on multivariate time series is useful. Also, comparisons to more state-of-the-art algorithms such as Fourier NODE or Neural CDEs remain future work. Furthermore, run-time improvements compared to solver-based algorithms as well as resource requirements need to be quantified on a broader basis. Beyond this, investigating the use of more specialised NN architectures for the components of Laplace-Net beyond fully connected networks may enhance performance or increase interpretability. Also, extending to handle two- and three-dimensional use cases, similar to LNO, would increase the applicability of Laplace-Net.

Acknowledgments. We acknowledge ChatGPT-4 for support in writing and for assistance with coding. The authors remain responsible for the ideas, content, and conclusions presented in this work.

C. Coelho would like to thank the KIBIDZ project funded by dtec.bw—Digitalization and Technology Research Center of the Bundeswehr; dtec.bw is funded by the European Union—NextGenerationEU, and project PL24-00057: "Inteligência Artificial na Otimização da Rega para Olivais Resilientes às Alterações Climáticas" financially supported by Fundação "la Caixa"|BPI and Fundação para a Ciência e Tecnologia (Portuguese Foundation for Science and Technology) and the funding by FCT through the CMAT projects UIDB/00013/2020 and UIDP/00013/2020.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Biloš, M., Sommer, J., Rangapuram, S.S., Januschowski, T., Günnemann, S.: Neural Flows: Efficient Alternative to Neural ODEs. In: Neural Information Processing Systems (Jan 2021)
2. Butcher, J.C.: A history of Runge-Kutta methods. Applied Numerical Mathematics **20**(3), 247–260 (Jan 1996). https://doi.org/10.1016/0168-9274(95)00108-5
3. Cao, Q., Goswami, S., Karniadakis, G.E.: Laplace neural operator for solving differential equations. Nature Machine Intelligence **6**(6), 631–640 (Jun 2024). https://doi.org/10.1038/s42256-024-00844-4
4. Chen, T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. In: Neural Information Processing Systems (2018)
5. Coelho, C., P. Costa, M.F., Ferrás, L.: Neural fractional differential equations (2025). https://doi.org/https://doi.org/10.1016/j.apm.2025.116060
6. Coelho, C., Costa, M.F.P., Ferrás, L.L.: Neural Fractional Differential Equations: Optimising the Order of the Fractional Derivative. Fractal and Fractional **8**(9), 529 (Sep 2024). https://doi.org/10.3390/fractalfract8090529
7. Dafilis, M.P., Frascoli, F., McVernon, J., Heffernan, J.M., McCaw, J.M.: The dynamical consequences of seasonal forcing, immune boosting and demographic change in a model of disease transmission. Journal of Theoretical Biology **361**, 124–132 (2014)
8. Diethelm, K.: The Analysis of Fractional Differential Equations: An Application-Oriented Exposition Using Differential Operators of Caputo Type, Lecture Notes in Mathematics, vol. 2004. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14574-2,
9. Diethelm, K., Ford, N.J.: Analysis of Fractional Differential Equations. Journal of Mathematical Analysis and Applications **265**(2), 229–248 (Jan 2002). https://doi.org/10.1006/jmaa.2000.7194
10. Dubner, H., Abate, J.: Numerical Inversion of Laplace Transforms by Relating Them to the Finite Fourier Cosine Transform. Journal of the ACM **15**(1), 115–123 (Jan 1968). https://doi.org/10.1145/321439.321446
11. Dupont, E., Doucet, A., Teh, Y.W.: Augmented Neural ODEs. In: 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada. (Apr 2019)
12. Gwak, D., Sim, G., Poli, M., Massaroli, S., Choo, J., Choi, E.: Neural Ordinary Differential Equations for Intervention Modeling (2020). https://doi.org/10.48550/ARXIV.2010.08304

13. Holt, S., Hüyük, A., Qian, Z., Sun, H., van der Schaar, M.: Neural laplace control for continuous-time delayed systems. In: Proceedings of the 26th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 206, pp. 1747–1778. PMLR (2023)
14. Holt, S.I., Qian, Z., van der Schaar, M.: Neural Laplace: Learning diverse classes of differential equations in the Laplace domain. In: Proceedings of the 39th International Conference on Machine Learning, p. 88118832. Proceedings of Machine Learning Research, PMLR (Jan 2022)
15. Kexue, L., Jigen, P.: Laplace transform and fractional differential equations. Applied Mathematics Letters **24**(12), 2019–2023 (Dec 2011). https://doi.org/10.1016/j.aml.2011.05.035
16. Kidger, P., Morrill, J., Foster, J., Lyons, T.: Neural Controlled Differential Equations for Irregular Time Series. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems, p. 66966707. Curran Associates, Inc (Jan 2020)
17. Kuhlman, K.L.: Review of inverse Laplace transform algorithms for Laplace-space numerical approaches. Numerical Algorithms **63**(2), 339–355 (Jan 2013). https://doi.org/10.1007/s11075-012-9625-3
18. Li, X., Zhang, J., Zhu, Q., Zhao, C., Zhang, X., Duan, X., Lin, W.: From Fourier to neural ODEs: Flow matching for modeling complex systems. In: Proceedings of the 41st International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 235, pp. 29390–29405. PMLR (2024)
19. Liang, S., Wu, R., Chen, L.: Laplace transform of fractional order differential equations. Electronic Journal of Differential Equations **2015** (05 2015)
20. Lu, L., Jin, P., Pang, G., Zhang, Z., Karniadakis, G.E.: Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nature Machine Intelligence **3**(3), 218–229 (Mar 2021). https://doi.org/10.1038/s42256-021-00302-5
21. Ma, J., Cui, Y., Liu, L.: Hopf bifurcation and chaos of financial system on condition of specific combination of parameters. Journal of Systems Science and Complexity **21**(2), 250–259 (2008)
22. Monsel, T., Semeraro, O., Mathelin, L., Charpiat, G.: Time and state dependent neural delay differential equations. In: Proceedings of the 1st ECAI Workshop on "Machine Learning Meets Differential Equations: From Theory to Applications". Proceedings of Machine Learning Research, vol. 255, pp. 1–20. PMLR (2024)
23. Ogata, K.: Modern Control Engineering. Prentice Hall, Upper Saddle River, 5th ed. edn. (2010)
24. Ozaki, Y., Tanigaki, Y., Watanabe, S., Onishi, M.: Multiobjective Tree-Structured Parzen Estimator for Computationally Expensive Optimization Problems. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp. 533–541. GECCO '20, Association for Computing Machinery, New York, NY, USA (Jan 2020). https://doi.org/10.1145/3377930.3389817
25. Rubanova, Y., Chen, R.T.Q., Duvenaud, D.K.: Latent ordinary differential equations for irregularly-sampled time series. In: Wallach, H., Larochelle, H., Beygelzimer, A., dAlché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019)
26. Schiff, J.L.: The Laplace Transform: Theory and Applications. Springer eBook Collection Mathematics and Statistics, Springer, New York, NY (Jan 1999). https://doi.org/10.1007/978-0-387-22757-3

27. Valencia, J., Olivar, G., Franco, C.J., Dyner, I.: Qualitative analysis of climate seasonality effects in a model of national electricity market. In: Analysis, Modelling, Optimization, and Numerical Techniques: ICAMI, San Andres Island, Colombia, November 2013. pp. 349–362. Springer (2015)
28. Zhu, Q., Guo, Y., Lin, W.: Neural Delay Differential Equations. In: Ninth International Conference on Learning Representations. arXiv (2021). https://doi.org/10.48550/arxiv.2102.10801
29. Zimmering, B., Coelho, C., Niggemann, O.: Optimising neural fractional differential equations for performance and efficiency. In: Proceedings of the 1st ECAI Workshop on "Machine Learning Meets Differential Equations: From Theory to Applications". Proceedings of Machine Learning Research, vol. 255, pp. 1–22. PMLR (Oct 2024), https://proceedings.mlr.press/v255/zimmering24a.html
30. Zimmering, B., Niggemann, O.: Integrating continuous-time neural networks in engineering: Bridging machine learning and dynamical system modeling. In: ML4CPS – Machine Learning for Cyber-Physical Systems (March 2024). https://doi.org/10.24405/15313
31. Zimmering, B., Roche, J.P., Niggemann, O.: Enhancing Nonlinear Electrical Circuit Modeling with Prior Knowledge-Infused Neural ODEs. In: 2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA). pp. 01–08. IEEE, Padova, Italy (Sep 2024). https://doi.org/10.1109/ETFA61755.2024.10711112
32. Zolock, J., Greif, R.: Application of time series analysis and neural networks to the modeling and analysis of forced vibrating mechanical systems. In: ASME International Mechanical Engineering Congress and Exposition. vol. 37122, pp. 1157–1164 (2003)