INTEGRATING CONTINUOUS-TIME NEURAL NETWORKS IN ENGINEERING: BRIDGING MACHINE LEARNING AND DYNAMICAL SYSTEM MODELING

Bernd Zimmering
Helmut-Schmidt-University
Holstenhofweg 85, 22043 Hamburg
bernd.zimmering@hsu-hh.de

Oliver Niggemann
Helmut-Schmidt-University
Holstenhofweg 85, 22043 Hamburg
oliver.niggemann@hsu-hh.de

24-01-2024

ABSTRACT

This paper examines the integration of Continuous-Time Neural Networks (CTNNs), including Neural ODEs, CDEs, Neural Laplace, and Neural Flows, into engineering practices, particularly in dynamical system modeling. We provide a detailed introduction to CTNNs, highlighting their underutilization in engineering despite similarities with traditional Ordinary Differential Equation (ODE) models. Through a comparative analysis with conventional engineering approaches, using a spring-mass-damper system as an example, we demonstrate both theoretical and practical aspects of CTNNs in engineering contexts. Our work underscores the potential of CTNNs to harmonize with traditional engineering methods, exploring their applications in Cyber-Physical Systems (CPS). Additionally, we review key open-source software tools for implementing CTNNs, aiming to facilitate their broader integration into engineering practices.

Keywords Cyber-Physical Systems · Dynamical Systems Modeling · Neural Ordinary Differential Equations

1 Introduction

In recent years, Machine Learning (ML) has emerged as a hyped area in scientific research as well as general public, demonstrating its prowess in fields like image and voice recognition, and text processing. The latest iterations of Chat GPT exemplify these advances. Despite its disruptive influence in these areas, ML's potential in the engineering domain has been comparatively untapped.

In the landscape of ML, recent popular models have been developed to address the unpredictable and varied nature of data in fields like image, text, and speech processing,

contrasting sharply with the structured and often deterministic data encountered in engineering disciplines, particularly those dealing with dynamical systems. ML Models such as the Variational Autoencoder (VAE) [7], Generative Adversarial Network (GAN) [9], and Normalizing Flow (NF) [23] are primarily designed for scenarios where the underlying data generation process is non-deterministic. VAEs leverage stochastic latent spaces, GANs rely on adversarial dynamics to generate data, and NFs use probabilistic transformations to model complex distributions. These characteristics make them particularly suited for domains like image and text processing, where data generation is inherently uncertain and diverse. Similarly, recent advancements in stable diffusion models [25], which iteratively refine a random noise input towards generating realistic images, have marked a significant breakthrough in the field of image synthesis.

However, machine learning in Cyber-Physical Systems (CPS) faces unique challenges such as integrating time dynamics, uncertainty estimation, and learning beneficial representations [20]. CPS, often modeled as *dynamical systems* (first defined by George D. Birkhoff [26]), evolve deterministically over time, typically represented by Ordinary Differential Equations (ODEs). This deterministic and continuous nature contrasts with the stochastic foundations of many ML models and is addressed in control engineering using models like ODEs and frequency domain representations [17].

In response to this, the past five years have seen the development of ML models that align more closely with the paradigms used in engineering for dynamical systems. One such breakthrough is the realization that the ResNet [14], a leading model for image classification, closely resembles the numerical approximation of an ODE using an Euler solver [27]. This insight has given rise to Neural Ordinary Differential Equations (Neural ODEs or NODE) [24], which are able to learn the underlying ODE of a sequence or time series.

While the ML research community explored the realms ODEs, concurrently, the integration of physical principles—captured by Partial Differential Equations (PDEs)—into machine learning frameworks was revolutionized by the introduction of Physics-Informed Neural Networks (PINNs) [22]. These networks adeptly embed known physical laws, such as the Navier-Stokes equations for fluid dynamics, into the predictive machinery of neural networks, offering a robust approach especially in contexts where data may be sparse. Nonetheless, our research primarily focuses on Neural ODEs, given their unique advantage of not necessitating a-priori physical knowledge for modeling time series of dynamical systems.

Although Neural ODEs closely mirror control engineering modeling practices, Recurrent Neural Networks (RNNs) such as Long Short-Term Memory (LSTM) [11] and the Gated Recurrent Unit (GRU) [6] remain predominant learning from CPS that contain dynamical systems [13,19,30]. This preference may stem from their analogy to nonlinear Delayed Difference Equations (DDEs) [29], a concept deeply rooted in engineering, coupled with their established presence (extending over 18 years as of 2024). Despite the extensive research and recognition of Neural ODEs as well as related models [15] in the ML domain – as highlighted in the Workshop "The Symbiosis of Deep Learning and Differential Equations III" at NeurIPS 2023[1] – their integration into ML applications within Cyber-Physical Systems (CPS) remains limited. This paper seeks to elevate the visibility of these models, collectively referred to as Continuous Time Neural Networks (CTNNs), and aims to broaden their application beyond conventional ML domains. The key contributions of this work are threefold:

[1] https://dlde-2023.github.io

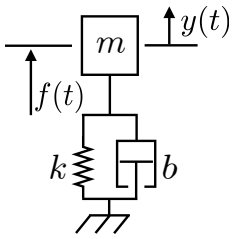

Figure 1: Spring-Mass-Damper System.

1. **Concise Review of Advanced Continuous ML Models:** This paper provides a detailed review of the latest continuous ML models, particularly those related to ODEs often employed in control engineering. We illustrate these concepts using a Spring-Mass-Damper system, which we model symbolically as an ODE, Transfer Function, and in closed form. This approach highlights the parallels with CTNNs.
2. **Comparison of CTNN and Engineering Models:** We present a concise comparison between CTNN approaches like Neural ODEs and modeling of dynamical systems engineering, focusing on their theoretical foundations.
3. **Overview of Open-Source Implementations:** We offer overview and guide of open-source tools for implementing these ML models in engineering applications.

The paper is organized as follows: We start with an analysis of dynamical system modeling, using a spring-mass-damper system as a case study. This is followed by an introduction to the mechanics of four CTNN models. Next, we explore the specifics of these ML models and compare their methodologies with conventional engineering approaches. Additionally, a detailed overview of essential open-source software tools for CTNNs is provided. The paper concludes with a summary and final reflections.

2 State of the Art and Background

2.1 Modeling in Controls Engineering

Modeling in control engineering often revolves around dynamical systems that adhere to conservation laws like energy conservation of sum of forces. In the case of a mechanical system the internal energy remains constant unless external forces are applied or dissipative forces, such as friction, are present. These dynamic interactions, are commonly described by sets of ODEs. Alongside energy conservation principles, which are typically well-understood, a single ODE can be formulated to describe the movement of a dynamical system under specific conditions. In engineering, modeling of dynamical systems is traditionally grounded in first principles (physical laws). Utilizing a Spring-Mass-Damper (SMD) system, depicted in Figure 1, serves as a practical example to illustrate the derivation of symbolic models across various mathematical domains, including ODEs, the Laplace domain, and the time domain. Please note that the SMD system is chosen for its illustrative value, simplifying the explanation of symbolic model derivation. Alternative examples, such as RLC circuits, could similarly demonstrate these concepts. In subsequent chapters, we will see how CTNNs model systems in a similar fashion, but relying on data rather than prior knowledge.

In the Spring-Mass-Damper (SMD) system, the mass experiences several forces. The force exerted by the spring is $-ky(t)$, where k is the spring constant and $y(t)$ is the displacement from the equilibrium position. The damping force, due to the damper, is $-b\dot{y}(t)$, with b being the damping coefficient and $\dot{y}(t)$ the velocity. Additionally, the system might be subject to an external force, represented by $f(t)$.

To derive the motion equation for the SMD system, we apply Newton's second law, which is $f = m \cdot \ddot{y}$. In this system, the forces include the spring force, the damping force, and any external force. By summing these forces, the equation of motion is obtained as follows:

$$m\ddot{y}(t) + b\dot{y}(t) + ky(t) = f(t) \tag{1}$$

Here, $\ddot{y}(t)$ is the acceleration of the mass. This equation describes the dynamic behavior of the SMD system, incorporating the effects of spring force, damping force, and external force.

While this form of a system can be challenging to analyze in-depth, the **State Space Model (SSM)** approach is often used, which describes a system as a combination of first-order ODEs. This is analogous to the latent space $\mathbf{z} \in \mathbb{R}^M$ used in Machine Learning, such as in an Autoencoder. However, in many cases, the state space is not necessarily lower-dimensional than the observation space. To apply SSM to our SMD system, we define the state variables as $z_1(t) = y(t)$ and $z_2(t) = \dot{y}(t)$. This allows us to rewrite Equation (1) as follows:

$$\dot{z}_1(t) = z_2(t) \tag{2a}$$

$$\dot{z}_2(t) = \frac{1}{m}\left[f(t) - bz_2(t) - kz_1(t)\right] \tag{2b}$$

Here, $\dot{z}_1(t)$ corresponds to velocity, and $\dot{z}_2(t)$ to acceleration. The SSM effectively encapsulates the system's dynamics in a two-dimensional state space, representing the primary energy storage components: the kinetic energy of the mass (in z_2) and the potential energy in the spring (in z_1).

Another method to analyze Equation (1) is the Laplace transform [28]. It enables the analysis of system properties and behavior without the necessity for time-domain simulations. This powerful tool transforms differential equations into algebraic equations, simplifying the analysis significantly. Applying the Laplace transform to Equation (1), we denote the Laplace transform of a function $g(t)$ as $\mathcal{L}\{g(t)\} = G(s)$, where $s \in \mathbb{C}$ is the complex frequency variable. For the SMD system, the transformed equations become:

$$\begin{aligned}\mathcal{L}\{m\ddot{y}(t)\} + \mathcal{L}\{b\dot{y}(t)\} + \mathcal{L}\{ky(t)\} &= \mathcal{L}\{f(t)\} \\ ms^2Y(s) - msy(0) - m\dot{y}(0) & \\ +bsY(s) - by(0) + kY(s) &= F(s)\end{aligned} \tag{3}$$

Here, $Y(s)$ and $F(s)$ are the Laplace transforms of the displacement $y(t)$ and the external force $f(t)$, respectively. The terms $sy(0)$ and $\dot{y}(0)$ represent the initial conditions in the Laplace domain. We can further rearrange Equation (3) to get insights representing the system's response in the frequency domain:

$$s^2Y(s) + \frac{b}{m}sY(s) + \frac{k}{m}Y(s) = \frac{1}{m}F(s) + sy(0) + \dot{y}(0) + \frac{b}{m}y(0) \tag{4}$$

$$Y(s) = \frac{F(s) + msy(0) + m\dot{y}(0) + by(0)}{ms^2 + bs + k} \tag{5}$$

In case the initial states $y(0)$ and $\dot{y}(0)$ are set to zero, we get the Transfer Function (TF):

$$H(s) = \frac{Y(s)}{F(s)} = \frac{1}{ms^2 + bs + k} \tag{6}$$

The system's dynamic behavior is largely determined by the nature of its poles, derived from the quadratic formula:

$$s = \frac{-\frac{b}{m} \pm \sqrt{\left(\frac{b}{m}\right)^2 - 4\frac{k}{m}}}{2}. \tag{7}$$

These poles, particularly in the underdamped case ($(\frac{b}{m})^2 - 4\frac{k}{m} < 0$), lead to complex conjugate roots and oscillatory system behavior. The solution for an underdamped system without external force ($f(t) = 0$) is:

$$y(t) = e^{-\frac{b}{2m}t}\left(y_0 \cos(\omega t) + \frac{v_0 + \frac{b}{2m}y_0}{\omega}\sin(\omega t)\right), \tag{8}$$

where $\omega = \sqrt{\frac{k}{m} - \left(\frac{b}{2m}\right)^2}$ represents the damped natural frequency. This formulation succinctly captures the characteristic damped oscillatory motion of the system.

2.2 Continuous Time Neural Networks

Following our exploration of the SMD system, which showcased equation derivation in the ODE, Time, and Laplace domains, we now delve into CTNN models. These models mirror the mathematical frameworks used to derive the symbolic models for the SMD system.

Neural Ordinary Differential Equations (NODE), as introduced by Chen et al. [24], merge the realms of machine learning and differential equations in a novel way. The essence of NODE lies in solving an Initial Value Problem (IVP) through the use of neural networks. Here, the IVP is characterized by finding a function that concurrently satisfies a *learned* differential equation and an initial condition.

In the NODE framework, system dynamics are encapsulated within a latent space $\mathbf{z} \in \mathbb{R}^M$ (similar to the SSM). This space is a transformation of the input sequence $\{\mathbf{x}_0, \ldots, \mathbf{x}_T\}$ where $\mathbf{x} \in \mathbb{R}^N$. The evolution of this system through time is governed by a differential equation defined by a neural network:

$$\dot{\mathbf{z}}(t) = f_{\text{ODE}}(\mathbf{z}(t), t; \theta_{\text{ODE}}). \tag{9}$$

Here, $\dot{\mathbf{z}}(t)$ signifies $\frac{d\mathbf{z}}{dt}$, the rate of change of the latent state, while $f_{\text{ODE}} : \mathbb{R}^M \to \mathbb{R}^M$ is a neural network function parameterized by θ_{ODE}. An interesting aspect of NODE is its use of a VAE approach to sample the initial state $\mathbf{z}_0$. This approach ensures the latent space remains efficiently dimensioned, employing just the necessary states for representation, and additionally permits the use of the trained model as a generative tool in inference phases when needed. The VAE-like step involves an RNN running backwards through the observations $\mathbf{x} \in \mathbb{R}^N$, which outputs the parameters of a normal distribution $\mathcal{N}(\mu_{\mathbf{z}_{t_0}}, \sigma_{\mathbf{z}_0})$. The initial state $\mathbf{z}_0$ is then sampled from this distribution:

$$\mu_{\mathbf{z}_0}, \sigma_{\mathbf{z}_0} = \text{RNN-Encoder}(\{\mathbf{x}_T, \ldots, \mathbf{x}_0\}), \tag{10}$$

$$\mathbf{z}_0 \sim \mathcal{N}(\mu_{\mathbf{z}_0}, \sigma_{\mathbf{z}_0}). \tag{11}$$

The development of $\mathbf{z}(t)$ from this initial state is computed over K discrete time intervals, where K represents the number of desired steps in a forecast scenario:

$$\mathbf{z}(t_{i+1}) = \mathbf{z}(t_i) + \int_{t_i}^{t_{i+1}} f_{\text{ODE}}(\mathbf{z}(\tau), \tau; \theta_{\text{ODE}}) d\tau,$$
$$\text{for } i = 0, 1, \ldots, T + K - 1. \tag{12}$$

Each latent state in the sequence $\{z_0, \ldots, z_T\}$ is decoded pointwise by a feedforward neural network to generate the output $\hat{\mathbf{y}}$:

$$\hat{\mathbf{y}}_i = \text{Decoder}(\mathbf{z}_i). \tag{13}$$

Chen et al.'s incorporation of ODE solvers (e.g. adaptive Runge-Kutta [3]) into the PyTorch framework was key in enabling backpropagation through these solvers, a crucial aspect for training NODE models. The primary challenge here was the substantial memory cost associated with backpropagation in complex or extended dynamical systems. The *adjoint method* proposed by Chen et al. provides a solution to this problem, improving memory efficiency while maintaining gradient accuracy.

Neural Controlled Differential Equations (CDE), as proposed by Kidger et al. [16], extend the NODE concept to handle irregularly observed time series by directly integrating observations into the differential equations. The CDE framework modifies the NODE model by introducing a multiplicative term to the differential equation:

$$\dot{\mathbf{z}}(t) = f_{\text{CDE}}(\mathbf{z}(t); \theta_{\text{CDE}}) \cdot \tilde{\dot{\mathbf{x}}}(t), \tag{14}$$

where $f_{\text{CDE}} : \mathbb{R}^M \to \mathbb{R}^{M \times N}$ is a neural network. To handle the typically unknown derivatives of observations, natural cubic splines are used to smoothly interpolate the data $\mathbf{x}$, allowing the computation of $\tilde{\dot{\mathbf{x}}}(t)$.

CDEs are particularly adept at processing time series with irregular observations but are not designed for forecasting scenarios where future inputs are unknown. Instead, they are more suitably employed as a replacement for the RNN encoder in Equation (10), enhancing the processing of past observed data.

Neural Flows by Biloš et al. [2] innovatively learn the flow of states, which is essentially the solution of an IVP, without the need to solve differential equations. Among various models, they introduce the ResNet-Flow, which satisfies key properties of an ODE solution: I) $F(0, x_0) = x_0$, ensuring that the flow aligns with the initial state at $t = 0$, and II) $F(t, \cdot)$ is invertible for all t, guaranteeing the uniqueness of the solution with respect to the initial condition.

The Resnet-Flow as used in their continous-time latent variable model (Encoder-Decoder approach similar to NODE) is defined as:

$$F(t, \mathbf{z}) = \mathbf{z}_0 + \varphi(t) g(t, \mathbf{z}_0; \theta_{\text{flow}}), \tag{15}$$

where $\mathbf{z}_0$ denotes the initial state. In this model, $\varphi : \mathbb{R} \to \mathbb{R}^N$ is a predetermined function (i.e. a hyperparameter) and $g : \mathbb{R}^{N+1} \to \mathbb{R}^N$ is a neural network designed to learn the flow dynamics, parameterized by θ_{flow}. To satisfy condition I) and ensure that the flow aligns with the initial state at $t = 0$, the function $\varphi(t)$ is chosen as $\varphi(t) = \tanh(t)$, which evaluates to zero when $t = 0$. For addressing condition II), which guarantees the invertibility of $F(t, \cdot)$ for all t, the network g is designed as a contractive neural network. This design ensures that the Lipschitz constant of g is less than 1, facilitating the maintenance of invertibility in the flow dynamics.

Neural Laplace, by Holt et al. [12], leverages a neural network for approximating a TF in the Laplace domain to predict time series dynamics. This approach parallels the

capabilities of Neural ODEs but does not require an ODE solver, thereby simplifying the process and enhancing computational efficiency.

Central to the Neural Laplace model is the learning of the transfer function $H(s)$ for $\mathbf{s} \in \mathbb{C}^D$. This is achieved using a neural network $g(\mathbf{p}, \mathbf{s}; \theta_L)$, where $\mathbf{p} \in \mathbb{R}^D$ represents the initial conditions similar to those in Equation (6).

The key components of the Neural Laplace model include:

- An **Encoder** $f(\mathbf{x}_0, \ldots, \mathbf{x}_T - k; \theta_{\text{enc}})$ that encodes time series data into initial conditions $\mathbf{p}$ for the Laplace function $H(\mathbf{s})$.
- A **Neural Network** $g(\mathbf{p}, \mathbf{s}; \theta_L)$ designed to model the Laplace function $H(\mathbf{s})$.
- The **Inverse Laplace Transform (ILT)** algorithm that uses $g(\mathbf{p}, \mathbf{s}; \theta_L)$ to numerically construct a time series for given timesteps $\{t_0, \ldots, t_T + K\}$ where K is the number of forecast time steps. Please note that similar to ODE solvers there exist a few methods, that perform task depended. [12] uses Fourier Series Inverse which uses bi-linear approximation to solve the ILT integral [10]

The Neural Laplace model's notable advantage is its ability to operate without the need for an ODE solver. This feature, along with its domain-centric approach, allows it to efficiently handle a diverse range of differential equations, including forced ODEs and ODEs with deadbands, making it a robust tool for complex system analysis and modeling.

3 Comparative Analysis of Continuous-Time Neural Networks and Engineering Modeling Approaches

As demonstrated in the previous section, CTNN bear similarities to the engineering modeling paradigms for dynamical systems, particularly in their shared approach to solving an IVP. The NODE model, utilizing a system of first-order ODEs, closely mirrors the SSM in engineering. This similarity is especially evident in the concept of modeling system dynamics within a latent space governed by first-order ODEs, akin to the SSM approach. However, notable differences exist:

1. The initial state in NODE, and consequently the properties of the latent space, are determined by a VAE approach. This involves sampling and regularizing the initial values to adhere closely to the predefined prior $\mathcal{N}(\mathbf{0}, \mathbf{I})$. This method contrasts with SSM, where the initial state typically corresponds to a tangible physical state, like velocities or movements.
2. NODE models typically represent the dynamics of a homogeneous ODE, lacking an explicit input function $f(t)$ that is commonly present in SSM. This absence means that NODE's dynamics are derived from a homogeneous system, differing from the SSM approach, where external inputs or forces (represented by $f(t)$) are often integral to the system's behavior and analysis.

Considering the second difference, Neural CDEs align more closely with SSM in engineering, as they allow the incorporation of excitations beyond just observations $\mathbf{x}$ in the machine learning context. However, the definition of f_{CDE} in CDEs, transforming a vector to a matrix, diverges from the typical SSM model. Additionally, while the excitation force in the SSM model, as per Equation (2), is additive, in CDEs (Equation (14)) it is multiplicative, a derivation from the field of rough analysis in mathematics. Despite the nomenclature similarity to controlled differential equations in control theory,

Table 1: Comparison of Continuous-Time Neural Network Models Applied to the Spring-Mass-Damper System

CTNN Model	**Algorithmic Approach**	**Learned Equation for SMD System**
NODE	Utilizes a neural network to parameterize an ODE.	SSM of Eq (2) with $f(t) = 0$ including $z_1(0) = y(0)$; $z_2(0) = \dot{y}(0)$
CDE	Enhances NODE by incorporating observations into the ODE.	SSM similar to Eq. (2), but with multiplicative input of $\dot{f}(t)$
Neural Laplace	Uses the Laplace domain for solving the ODEs including initial conditions.	System response in laplace domain $Y(s)$ of Eq. (5) including initial conditions $\mathbf{p} = [y(0), \dot{y}(0)]^T$
Neural Flows	Learns the flow of states without directly solving differential equations.	Solution of IVP for $f(t) = 0$ like Eq.(8)

which is $\dot{x}(t) = f(x(t), u(t), t)$, Neural CDEs are based on a different theoretical foundation, limiting their direct applicability to model forced dynamics in the same manner as engineering models.

Both Neural CDEs and NODEs utilize numerical ODE solvers frequently used in engineering, such as adaptive Runge-Kutta methods [3]. However, these solvers are often avoided in engineering due to their computational intensity.

In engineering, direct modeling of system responses to specific signals is uncommon, as it its usefullness for unknown excitation signals is limited. However, Neural Flows follow this paradigm, as they use a neural net that is satisfying the conditions of an IVP. Regarding the relevance and generalizability of Neural Flows to other types of signal remain a question in engineering contexts, where it is not always possible to assume the presence of such excitations in the dataset.

Neural Laplace models are closely related to transfer functions used in control engineering. Their capability to extend ODE models by easily modeling transport delay times — a common feature in control engineering—positions them as superior to SSM, particularly when the system's ODE is assumed to be linear. Unlike typical engineering practices where initial conditions are often neglected or assumed to be zero, Neural Laplace models utilize initial conditions determined by an encoder. Moreover, the use of the ILT algorithm is atypical for control engineering, where solutions to IVPs are usually obtained through partial fraction decomposition and laplace correspondence tables. Numerical ILT is not widely known or commonly used in control engineering. Nonetheless, as the neural network in Neural Laplace models learns a transfer function, it opens up possibilities for deriving system properties like poles and zeros from data.

In the context of applying these CTNN models to a specific engineering case, such as the SMD system, their distinct algorithmic approaches and the resulting learned equations offer a clear comparative perspective. Table 1 summarizes how each CTNN model would approach the SMD system, highlighting their unique methodologies and outcomes.

For instance, the NODE model would parameterize the ODE of the SMD system and operate under the assumption of a homogeneous system, omitting explicit input functions. In contrast, CDEs would modify this approach by incorporating observations in a multiplicative manner. This is different from the additive nature of excitations commonly seen in engineering SSMs. Neural Laplace models, aligning with control engineering's transfer functions, bring a novel perspective to modeling SMD systems by focusing on the system's Laplace domain properties. Lastly, Neural Flows, deviating from the traditional direct modeling of system responses in engineering, offer a unique approach by learning the state flow without directly solving differential equations.

4 Software Libraries

The convergence of CTNNs with engineering paradigms necessitates quantitative evaluations of their performance and limitations. As these networks rely on complex theories and algorithms, existing software libraries are indispensable for practical applications. Table 2 presents a summary of prominent open-source software for CTNN studies. It's important to note that 'GitHub Stars' should be seen merely as an indicator of a library's popularity and recognition within the developer community, rather than a definitive metric for its usage or quality.

Table 2: Summary of Software Libraries for Neural ODEs, Neural Laplace, and Neural Flows

Library	**Reference**	**Functionality**	**Language& ML Framework**	**GitHub Stars**
`torchdiffeq`	[5]	ODE Solvers	Python Pytorch	4900
`torchdyn`	[21]	ODE Solvers	Python Pytorch	1200
`torchode`	[18]	ODE Solvers	Python Pytorch	<200
`neurodiffeq`	[4]	ODE Solvers PDE Solvers	Python Pytorch	<600
`diffrax`	[15]	CDE Solvers, ODE Solvers	Python Jax	1100
`torchlaplace`	[12]	Neural Laplace, ILT Solvers	Python Pytorch	<100
Neural Flows	[2]	Neural Flows	Python Pytorch	<100

`torchdiffeq`, `torchdyn`, and `torchode`, all Python libraries utilizing the PyTorch framework [1], offer similar functionalities in solving ODEs. `torchdiffeq` is notable for its integration with Chen et al.'s work [5], while `torchdyn` focuses on applying NODEs in neural network depths akin to ResNet. `torchode` enhances solver efficiency through JIT-Compilation and parallel processing.

`neurodiffeq` caters to both ODE and Partial Differential Equation (PDE) solutions, emphasizing boundary conditions integration. `diffrax`, leveraging the JAX framework

[8], offers speed-optimized solutions for CDEs and ODEs and is maintained by Kidger [15].

`torchlaplace` specializes in the Neural Laplace algorithm, offering advanced ILT solvers [12]. *Neural Flows*, though not a formal package, provides practical code for implementing Neural Flow models [2].

5 Conclusion

In this paper, we have effectively bridged the divide between CTNNs and traditional engineering methodologies for modeling dynamical systems. By examining the mechanics of CTNNs, especially Neural ODEs, CDEs, Neural Laplace, and Neural Flows, against systems like the spring-mass-damper model, we have underscored the parallels between these ML models and advanced dynamical system modeling in engineering. CTNNs offer significant potential in scenarios where symbolic modeling of dynamical systems is intricate, providing a robust avenue for developing reliable ML-based models. These models, grounded in theoretical principles akin to those in engineering, enhance trust in their modeling processes. In addressing the limited utilization of CTNNs in engineering, our comprehensive guide and recommendations on open-source tools aim to encourage the engineering community to explore and confirm the practicality of these ML algorithms in real-world engineering tasks.

Acknowledgments

We would like to acknowledge the assistance of AI tools, including ChatGPT, for their support in refining the readability of this manuscript. Their contributions complemented our efforts, helping to make the paper more accessible.

References

[1] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[2] M. Biloš, J. Sommer, S. S. Rangapuram, T. Januschowski, and S. Günnemann. Neural flows: Efficient alternative to neural odes. In *Neural Information Processing Systems*, 2021.

[3] J. C. Butcher. A history of runge-kutta methods. *Applied Numerical Mathematics*, 20(3):247–260, 1996.

[4] F. Chen, D. Sondak, P. Protopapas, M. Mattheakis, S. Liu, D. Agarwal, and M. Di Giovanni. Neurodiffeq: A python package for solving differential equations with neural networks. *Journal of Open Source Software*, 5(46):1931, 2020.

[5] R. T. Q. Chen. torchdiffeq, 2018.

[6] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. 2014.

[7] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[8] R. Frostig, M. Johnson, and C. Leary. Compiling machine learning programs via high-level tracing. 2018.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc, 2014.

[10] Harvey Dubner and Joseph Abate. Numerical inversion of laplace transforms by relating them to the finite fourier cosine transform. *J. ACM*, 15:115–123, 1968.

[11] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] S. I. Holt, Z. Qian, and M. van der Schaar. Neural laplace: Learning diverse classes of differential equations in the laplace domain. In K. Chaudhuri, S. Jegelka, Le Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 8811–8832. PMLR, 2022.

[13] M. Jafari, A. Kavousi-Fard, M. Dabbaghjamanesh, and M. Karimi. A survey on deep learning role in distribution automation system: A new collaborative learning-to-learning (l2l) concept. *IEEE Access*, 10:81220–81238, 2022.

[14] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[15] P. Kidger. *On Neural Differential Equations*. PhD thesis, University of Oxford, 2021.

[16] P. Kidger, J. Morrill, J. Foster, and T. Lyons. Neural controlled differential equations for irregular time series. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6696–6707. Curran Associates, Inc, 2020.

[17] J. Lunze. *Regelungstechnik 1: Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen*. Springer Vieweg, Berlin, Heidelberg, 12th ed. 2020 edition, 2020.

[18] Marten Lienen and Stephan Günnemann. torchode: A parallel ode solver for pytorch.

[19] B. C. Mateus, M. Mendes, J. T. Farinha, R. Assis, and A. M. Cardoso. Comparing lstm and gru models to predict the condition of a pulp paper press. *Energies*, 14(21):6958, 2021.

[20] O. Niggemann, B. Zimmering, H. Steude, J. L. Augustin, A. Windmann, and S. Multaheb. Machine learning for cyber-physical systems. In B. Vogel-Heuser and M. Wimmer, editors, *Digital Transformation: Core Technologies and Emerging Topics from a Computer Science Perspective*, pages 415–446. Springer Berlin Heidelberg, Berlin, Heidelberg, 2023.

[21] M. Poli, S. Massaroli, A. Yamashita, H. Asama, J. Park, and S. Ermon. Torchdyn: Implicit models and neural numerical methods in pytorch.

[22] 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, 2019.

[23] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1530–1538, Lille, France, 2015. PMLR.

[24] C. Ricky T. Q., R. Yulia, B. Jesse, and D. David Kristjanson. Neural ordinary differential equations. In *NeurIPS*, 2018.

[25] R. Robin, B. Andreas, L. Dominik, E. Patrick, and O. Björn, editors. *High-Resolution Image Synthesis with Latent Diffusion Models*, 2022.

[26] T. Roque. Stability of trajectories from poincaré to birkhoff: approaching a qualitative definition. *Archive for History of Exact Sciences*, 65(3):295–342, 2011.

[27] L. Ruthotto and E. Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, 62(3):352–364, 2020.

[28] J. L. Schiff. *The Laplace Transform: Theory and Applications*. Springer eBook Collection Mathematics and Statistics. Springer, New York, NY, 1999.

[29] A. Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404(8):132306, 2020.

[30] K. Zarzycki and M. Ławryńczuk. Lstm and gru neural networks as models of dynamical processes used in predictive control: A comparison of models developed for two chemical reactors. *Sensors (Basel, Switzerland)*, 21(16), 2021.