Big Data Management & Analytics Master

DIFFERENTIAL EQUATIONS SOLVER USING NFTM FOR BURGER EQUATIONS

Samuel CHAPUIS
samuel.chapuis@student-cs.fr

Lucia Victoria FERNANDEZ SANCHEZ
lucia-victoria.fernandez@student-cs.fr

Alexandra PERRUCHOT-TRIBOULET RODRIGUEZ
alexandra.perruchot-triboulet-rodriguez@student-cs.fr

Github Link

Advisor: Nacéra SEGHOUANI - Nacera.Seghouani@centralesupelec.fr
Advisor: Akash MALHOTRA - akash.malhotra@centralesupelec.fr

Contents

Chapter 1

Introduction

1.1 Context and motivation

1.1.1 Project goals

Our objective is to build a generic neural architecture that can internalize the governing rules of physical systems and then advance their fields over time. The model is trained on challenging PDEs with a focus on fluid mechanics (Navier–Stokes) to stress-test its ability to learn complex, multi-scale, nonlinear dynamics. Beyond matching the training horizon, the same architecture should simulate and extrapolate trajectories past the seen time window, remaining stable and accurate while generalizing across PDE families. Comparing these predictions to reference simulations allows us to identify which ingredients are essential for robustness and transfer.

1.1.2 Minimal mathematical frame

Many PDEs of interest can be written in a (semi-)invariant form:

$$\partial_t u(x,t) = \mathcal{F}\big(u(x,t), \nabla u(x,t), \nabla^2 u(x,t); \theta\big), \quad x \in \Omega,\ t > 0, \tag{1.1}$$

with initial/boundary conditions. Here u is a scalar or vector field; $\mathcal{F}$ encodes advection, diffusion, reaction, constraints; and θ collects physical parameters (viscosity ν, conductivity α, permeability $a(x)$, etc.). Our network seeks to learn a local time-advance operator that approximates $\mathcal{F}$ across multiple PDE families.

1.1.3 Historical milestones: AI and PDEs

- **1990s–2000s**: early universal approximators for simple PDEs; occasional RBF/MLP surrogates.
- **2017–2019**: *Physics-Informed Neural Networks* (PINNs) enforce PDE residuals in the loss; good on simple geometries, sensitive to stiffness and turbulent regimes.
- **2020**: *Fourier Neural Operator* (FNO) and *Neural Operators* learn the operator between function spaces; reference benchmarks on Burgers, incompressible Navier–Stokes, Darcy.
- **2021–2024**: rise of spatio-temporal architectures (Transformers, ConvNeXt) for continuous fields; work on numerical stability, spectral regularization, and conservation of physical quantities.

- **NFTM (2025)**: Neural Field Turing Machine (Malhotra & Seghouani) combines **continuous spatial memory + read/write heads + a neural controller** (CNN/RNN/Transformer). It resembles an explicit scheme: local read, compute an update, local write. Our work builds on this idea to generalize across PDEs.

1.2 Fluid Mechanics Overview

1.2.1 Why Navier–Stokes?

Navier–Stokes models is a canonical set of nonlinear PDEs governing fluid flow, encompassing a wide range of phenomena from laminar to turbulent regimes. Their complexity and multi-scale nature make them an ideal testbed for evaluating the capabilities of neural architectures in learning physical dynamics. Successfully modeling Navier–Stokes would demonstrate the potential of AI-driven approaches in computational fluid dynamics (CFD) and beyond.

1.2.2 Governing equations

The Navier–Stokes equations describe the motion of fluid substances and are derived from fundamental conservation laws: mass, momentum, and energy. They can be expressed as follows:

$$\text{Mass Conervation}: \quad \frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{u}) = 0 \tag{1.2}$$

$$\text{Momentum Conervation}: \quad \frac{\partial(\rho\mathbf{u})}{\partial t} + \nabla\cdot(\rho\mathbf{u}\otimes\mathbf{u}) = -\nabla p + \nabla\cdot\boldsymbol{\Sigma} + \rho\mathbf{g} \tag{1.3}$$

$$\text{Energy Conervation}: \quad \frac{\partial E}{\partial t} + \nabla\cdot((E+p)\mathbf{u}) = \nabla\cdot(\boldsymbol{\tau}\cdot\mathbf{u}) - \nabla\cdot\mathbf{q} + \rho\mathbf{u}\cdot\mathbf{g} \tag{1.4}$$

Where all the variables are defined as:

- ρ: fluid density [kg·m^{-3}]
- $\mathbf{u} = (u, v, w)$: velocity field [m·s^{-1}]
- $\nabla\cdot(\rho\mathbf{u})$: divergence of mass flux
- $\partial\rho/\partial t$: local time rate of change of density
- $\rho\mathbf{u}$: momentum density [kg·m^{-2}·s^{-1}]
- $\nabla\cdot(\rho\mathbf{u}\otimes\mathbf{u})$: convective momentum transport
- $-\nabla p$: pressure gradient force per unit volume

- $\boldsymbol{\Sigma}$: viscous stress tensor [Pa]
- $\rho\mathbf{g}$: body force density (e.g. gravity) [N·m^{-3}]
- $E = \rho\big(e + \frac{1}{2}|\mathbf{u}|^2\big)$: total energy density (internal + kinetic)
- p: pressure [Pa]
- $\boldsymbol{\tau}$: stress tensor (viscous + pressure)
- $\mathbf{q}$: heat flux vector [W·m^{-2}]
- $\rho\mathbf{u}\cdot\mathbf{g}$: work done by body forces

These equations require a viscosity model (e.g., Newtonian fluid) and an equation of state (e.g., ideal gas law or van der Waals equation) to close the system. They form the foundation for simulating fluid dynamics in various applications, from aerodynamics to weather forecasting.

$$\text{Newtonian fluid model :} \quad \begin{cases} \boldsymbol{\Sigma} = \mu\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right) + \lambda(\nabla\cdot\mathbf{u})\mathbf{I}, \\ \boldsymbol{\tau} = \boldsymbol{\Sigma} - p\mathbf{I} \end{cases} \tag{1.5}$$

$$\text{Ideal gas :} \quad p = \rho R T \tag{1.6}$$

$$\text{van der Waals :} \quad \left(p + a\left(\frac{n}{V}\right)^2\right)(V - nb) = nRT \tag{1.7}$$

1.2.3 Burger's equation as a starting point

As the project starts, we focus the goal was to use a model simple enough to be learned with limited data and computational resources, yet rich enough to exhibit complex dynamics. Thus, we begin with the burger s equation, a simplified 1D version of the Navier–Stokes equations that captures some features of fluid dynamics, such as shock formation and nonlinear advection. The viscous Burger's equation in 1D is given by:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \quad x \in [0, L],\ t > 0, \tag{1.8}$$

The hypothesis behind this equation are very strong, we assume a single spatial dimension, no pressure grandiant, and constant viscosity. This can be written in a non-dimensional form as:

$$\text{1D spatial domain :} \quad \begin{cases} \mathbf{u} = u(x,t), \\ x \in [0, L] \end{cases}$$

$$\text{No pressure gradient :} \quad \nabla p = \mathbf{0}$$

Chapter 2

Related Work

It is expected to find:

- The research articles related to your defined problem/objective
- Try to present them by category, the main ideas behind and their limitations
- Justify your directions/choices. This part should make a link with the next chapter.

Chapter 3

Background

The objective here is to detail the main concepts/definitions existing algorithms needed to understand your work to be detailed later and to introduce the notations to be used.

Use examples

Don't forget to cite again these existing approaches

Chapter 4

Approaches for Neural PDE Simulators

In this chapter, we present two distinct Neural Field Turing Machine (NFTM) model architectures, developed as data-driven simulators for the 1D Burger's equation. Currently, these are defined for the simulation of Burger's dynamics and both approaches utilize the dataset described in Table 4.1, which consits of 724 training trajectories across 13 different viscosity values (from 0.001 to 0.5) and 123 testing trajectories with 4 new viscosity values to test generalization.
The first model .... **EXPLAIN TRANSFORMER MODEL ....**
The second approach, corresponds to a **Causal Temporal Convolutional Attention Network (TCAN)** that combines attention with convolutional feature extraction, by restricting attention to past timesteps only and using 1D convolutions for spatial processing. Both approaches enforce physics-informed constraints. We detail their architectures, training procedures, and comprehensive evaluation metrics, highlighting respective strengths and limitations.

4.1 Approach 1: Transformer based Model

4.1.1 Model Architecture

4.1.2

4.2 Approach 2: TCAN based Model

4.2.1 Model Architecture

The Temporal Convolutional Attention Network (TCAN) is a feed-forward autoregressive model that predicts next state of a PDE given a sliding window of past states. The TCAN serves as the *controller* in the Neural Field Turing Machine model framework, where it learns a discrete-time update rule for continuous spatiotemporal fields.
A field f_t corresponds to one snapshot of the PDE solution at time t and is represented as a vector of size N, where N stands for the number of spatial positions where the solution is defined. Therefore, the continuous field at time t is given by:

$$f_t = [u(x_1, t), u(x_2, t), ..., u(x_N, t)] \in \mathbb{R}^{1\times N}.$$

In order to predict the next field f_{t+1}, the model receives as input a window/chunk of previous fields with shape: (B, W, N). This is defined as:

$$\text{window} = [f_{t-W+1}, f_{t-W+2}, ..., f_t],$$

Table 4.1: Burger's Equation Dataset: Number of Trajectories by Viscosity.

Training (724 samples)		
Viscosity ν	**Count**	**Percentage**
0.0010	60	8.3%
0.0020	40	5.5%
0.0065	40	5.5%
0.0080	40	5.5%
0.0400	40	5.5%
0.0500	60	8.3%
0.0700	60	8.3%
0.1000	60	8.3%
0.1500	60	8.3%
0.2500	60	8.3%
0.3500	60	8.3%
0.4000	60	8.3%
0.5000	84	11.6%
Testing (123 samples)		
0.0100	50	40.7%
0.0269	13	10.6%
0.2000	30	24.4%
0.3000	30	24.4%

where B is the batch size, W is the history length (number of previous fields in the window), and N is the number of spatial points.
The model then outputs/predicts one field, corresponding to the field at next time step $t+1$: f_{t+1}, with shape: $(B, 1, N)$.

Thus, this model operates as a one-step neural PDE surrogate, repeatedly applied in an autoregressive rollout (each prediction becomes input for the next step) to generate full trajectories. Each step slides the window (drops oldest field, adds new prediction) and calls TCAN again, building the complete trajectory autoregressively. Figure 4.1 illustrates this process.

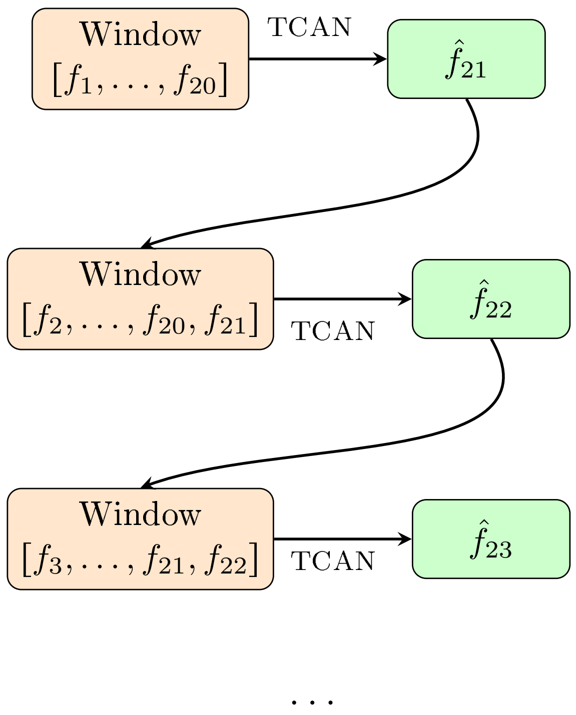

Figure 4.1: Autoregressive rollout process: each TCAN call maps a window of previous fields to one predicted field. The prediction is inserted into the next window, replacing the oldest field, to generate the full trajectory. In this example, the model first predicts f_{21} from the window $[f_1, \dots, f_{20}]$, then uses the updated window $[f_2, \dots, f_{20}, f_{21}]$ to predict f_{22}, and so on.

The architecture of this TCAN model consists of **three main components**:
1. A temporal attention encoder that aggregates information across the history window.
2. A convolutional decoder that produces a bounded correction.
3. A residual update that adds the correction to the last frame.

The full data flow for a single prediction step is illustrated in Figure 4.2.
Figure 4.2 illustrates the complete data flow through the Causal Temporal Convolutional Attention Network during a single prediction step. Each node performs a specific trans-

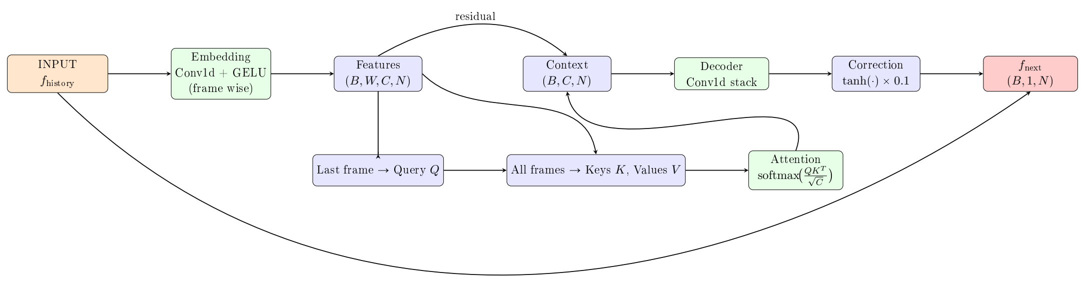
residual
INPUT f_{history}
Embedding Conv1d + GELU (frame wise)
Features (B, W, C, N)
Context (B, C, N)
Decoder Conv1d stack
Correction $\tanh(\cdot) \times 0.1$
f_{next} $(B, 1, N)$
Last frame → Query Q
All frames → Keys K, Values V
Attention $\text{softmax}(\frac{QK^T}{\sqrt{C}})$

Figure 4.2: Data flow through the Causal Temporal Attention Network during one prediction step.

formation:

- $f_{\textbf{history}}$ **(Input)**: Window of $W = 20$ previous fields of shape (B, W, N), containing the most recent spatiotemporal snapshots $f_{t-W+1}, \ldots, f_t \in \mathbb{R}^N$.
- **Conv1d + GELU (frame-wise)**: Applies 1D convolution (kernel size 3) to each of the W fields individually, expanding from 1 to $C = 32$ feature channels per field. Output shape: $(B \cdot W, 32, N)$.
- **Features** (B, W, C, N): Reshaped embedded representation exposing the temporal dimension, with each of the W fields now containing 32 spatial feature maps.
- **Last frame → Query** Q: Extracts features from the most recent field f_t and applies 1×1 convolution to generate query vectors $Q \in \mathbb{R}^{B \times 32 \times N}$.
- **All frames → Keys** K**, Values** V: Projects all W embedded fields through separate 1×1 convolutions to produce keys K and values V, both of shape $(B, W, 32, N)$.
- **Attention** $\text{softmax}(\frac{QK^T}{\sqrt{C}})$: Computes causal attention scores between the query (last frame) and all previous frames, producing temporal attention weights $\alpha_{w,p} = \text{softmax}_w(QK^T/\sqrt{32})$ for each spatial position p.
- **Context** (B, C, N): Aggregates temporal context via weighted sum $\sum_w \alpha_{w,p} V_w$, adds residual connection from last-frame features, and applies GroupNorm. Shape: $(B, 32, N)$.
- **Decoder Conv1d stack**: Two-layer convolutional decoder (32→32→1 channels, kernel size 5) that maps rich temporal-spatial context to a scalar correction field. Final layer zero-initialized for training stability.
- **Correction** $\tanh(\cdot) \times 0.1$: Applies hyperbolic tangent nonlinearity scaled by 0.1 to bound corrections $|\Delta u| \leqslant 0.1$, ensuring numerical stability during long rollouts.
- $u_{\textbf{next}}$ **(Output)**: Residual update combining the last input field $u_{\text{history}}[:, -1, :]$ with the predicted correction: $f_{t+1} = f_t + \Delta u$. Shape: $(B, 1, N)$.

The two residual connections—from Features to Context (within attention encoder) and from input last frame to output—preserve critical spatiotemporal information while enabling stable incremental updates.

4.2.2 Temporal Attention Encoder

The `CausalTemporalAttention` module processes the history window through the following steps:

1. **Frame-wise embedding**: Reshape u_{history} to $(B \cdot W, 1, P)$ and apply a 1D convolution with kernel size 3 followed by GELU activation, producing features of shape $(B \cdot W, C, P)$ where $C = 32$ is the embedding dimension.
2. **Temporal restructuring**: Reshape features back to (B, W, C, P) to expose the temporal dimension.
3. **Query computation**: Extract features of the last frame features$[:, -1, :, :] \in \mathbb{R}^{B \times C \times P}$ and apply a 1×1 convolution to obtain queries $Q \in \mathbb{R}^{B \times C \times P}$.
4. **Key and value projection**: Apply 1×1 convolutions to all W frames (flattened to $(B \cdot W, C, P)$ then reshaped) to obtain keys K and values V, both of shape (B, W, C, P).
5. **Causal attention scores**: Compute scaled dot-product attention over the temporal dimension:
$$\text{scores}_{b,w,p} = \frac{\langle Q_b, K_{b,w} \rangle}{\sqrt{C}}, \quad \alpha_{b,w,p} = \text{softmax}_w(\text{scores}_{b,w,p})$$
6. **Context aggregation**: Compute the attended context:
$$\text{context}_b = \sum_{w=1}^{W} \alpha_{b,w} \cdot V_{b,w} \in \mathbb{R}^{B \times C \times P}$$
7. **Residual projection**: Apply a 1×1 convolution to the context, add the residual connection to the last frame features, and normalize with GroupNorm.

This mechanism allows the last frame to dynamically query relevant past frames at each spatial location, producing spatially-aware temporal context features.

4.2.3 Decoder and Residual Correction

The `ImprovedBurgersNet` decoder processes the attention output context $\in \mathbb{R}^{B \times C \times P}$ as follows:

1. **Convolutional decoder stack**:
   - Conv1d: $C \to 32$ channels, kernel size 5, BatchNorm1d, GELU.
   - Conv1d: $32 \to 1$ channel, kernel size 5, producing raw_correction $\in \mathbb{R}^{B \times 1 \times P}$.

The final layer is zero-initialized to ensure near-zero corrections during early training.

2. **Bounded correction**: Apply hyperbolic tangent clipping scaled by corr_clip = 0.1:

$$\Delta u = \tanh(\text{raw\_correction}) \cdot 0.1$$

3. **Residual update**: Add the correction to the last input frame:

$$u_{\text{next}} = u_{\text{history}}[:, -1:, :] + \Delta u$$

This design enforces incremental updates, preventing instability during long autoregressive rollouts.

4.2.4 Autoregressive Training Procedure

Training employs curriculum learning with multi-step rollouts:

1. Initialize current_window = $u_{\text{gt}}[:, :W, :]$ from ground truth trajectories.
2. For each rollout step $k \in \{0, \dots, D-1\}$ where D is the rollout depth (curriculum: 8→16→64):
   - Predict $\hat{u}_{W+k} = \text{TCAN}(\text{current\_window})$.
   - Compute composite loss:

     $$\mathcal{L} = \text{MSE}(\hat{u}, u_{\text{gt}}) + 0.1 \cdot \|\nabla_x \hat{u} - \nabla_x u_{\text{gt}}\|_2^2 + 0.05 \cdot \mathbb{E}[\max(0, E(\hat{u}) - E(u_t))]$$

     where $E(u) = \frac{1}{2}\int u^2\,dx$ enforces energy dissipation.
   - Update window: current_window ← [current_window$[:, 1:, :], \hat{u}$] (with occasional teacher forcing).
3. Average loss over rollout, backpropagate through unrolled computation graph, apply gradient clipping, and optimize with AdamW + cosine annealing.

4.2.5 Evaluation Metrics

Full-trajectory predictions are evaluated using:

- Standard: MSE, relative L^2, PSNR, SSIM, temporal correlation.
- Physics-informed: mass conservation error, energy monotonicity fraction, mean PDE residual, spectral error, max gradient error.

This comprehensive evaluation ensures both accuracy and physical fidelity across long rollouts.

Chapter 5

Next Steps

Chapter 6

Conclusion and Perspectives

A summary of your work. More focused on the results

The limitations of the work -> which perspectives/clues to deal with limitations, to improve your work

the last paragraph must be dedicated to the work in team

6.1 Gl remarks

GENERAL :

Each table, figure must be cited and explained in the text.

The references must be complete

Each chapter must start with a paragraph to introduce its content (no need to have a separated for that), except the introduction and the conclusion. In the same manner each chapter must finish with a paragraph to conclude and to make a link with the next one, except the introduction and the conclusion.

6.2 Gl remarks about the presentation

The slides must be numbered

The presentation follows more and less the structure of the report

No too much blabla about the the gl context you need to define the objectives of the project (with examples if possible) ...

Then how your work fits into existing works (some main related works), the overall pipeline, your main contributions in this pipeline

also your main contributions in terms of implementation

the main results

Conclusions and next ...

Then the overall

Appendix A

Appendix

The progress draft must be included in the appendix

Bibliography

[1] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural PDE solvers. In Proceedings of the International Conference on Learning Representations (ICLR), 2022. (Not cited.)

[2] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (PINNs) for fluid mechanics: A review. Acta Mechanica Sinica, 37:1727–1738, 2021. (Not cited.)

[3] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. In Advances in Neural Information Processing Systems (NeurIPS), volume 31, pages 6571–6583, 2018. (Not cited.)

[4] Julian D. Cole. On a quasi-linear parabolic equation occurring in aerodynamics. Quarterly of Applied Mathematics, 9:225–236, 1951. (Not cited.)

[5] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing machines. arXiv preprint arXiv:1410.5401, 2014. (Not cited.)

[6] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. Nature, 538:471–476, 2016. (Not cited.)

[7] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In Advances in Neural Information Processing Systems (NeurIPS), volume 32, pages 15379–15389, 2019. (Not cited.)

[8] Eberhard Hopf. The partial differential equation $u_t + uu_x = \mu u_{xx}$. Communications on Pure and Applied Mathematics, 3:201–230, 1950. (Not cited.)

[9] Nikola B. Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. Journal of Machine Learning Research, 24(89):1–97, 2023. (Not cited.)

[10] Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, and Michael W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. In Advances in Neural Information Processing Systems (NeurIPS), volume 34, pages 26548–26560, 2021. (Not cited.)

[11] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In Proceedings of the International Conference on Learning Representations (ICLR), 2021. (Not cited.)

[12] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. PDE-Net: Learning PDEs from data. In Proceedings of the 35th International Conference on Machine Learning (ICML), volume 80, pages 3208–3216, 2018. (Not cited.)

[13] 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, 3:218–229, 2021. (Not cited.)

[14] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In Proceedings of the 36th International Conference on Machine Learning (ICML), volume 97, pages 5301–5310, 2019. (Not cited.)

[15] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378:686–707, 2019. (Not cited.)

[16] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Science, 367(6481):1026–1030, 2020. (Not cited.)

[17] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In Advances in Neural Information Processing Systems (NeurIPS), volume 28, pages 802–810, 2015. (Not cited.)

[18] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM Journal on Scientific Computing, 43(5):A3055–A3081, 2021. (Not cited.)

Abstract:

Keywords: