DTU PES Summer School: Future Energy Systems

Rethinking Power System Dynamic Simulations with Machine Learning

Johanna Vorwerk, DTU Wind, Section for Power Systems

May 19 2026

Why do we need to rethink power system dynamics?

Denmark 1980

Source: Energinet

Why do we need to rethink power system dynamics?

Denmark 1980

Source: Energinet

Each generator is a dynamic system described by **Differential Algebraic Equations (DAEs)**:

$$\mathbf{M}\frac{d\mathbf{x}}{dt} = f(\mathbf{x}, \mathbf{u}, t)$$
$$\mathbf{x}(0) = \mathbf{x_0}$$

- $\mathbf{x}$ Differential & algebraic states
- $\mathbf{u}$ Control inputs & disturbances
- $\mathbf{M}$ Mass-Matrix: determines if the row corresponding to state x is algebraic or differential

Characteristics of **SM-dominated power systems**:

- **SM** dynamics **dominate** system behavior
- Loads are predominantly static (or aggregated with simple dynamics)

» **Established analytical and time-domain methods for system-level stability analysis**

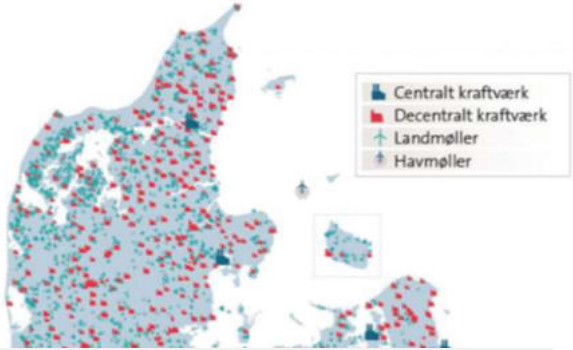
SM: Synchronous Machine

Why do we need to rethink power system dynamics?

Denmark 1980

Source: Energinet

Denmark Today

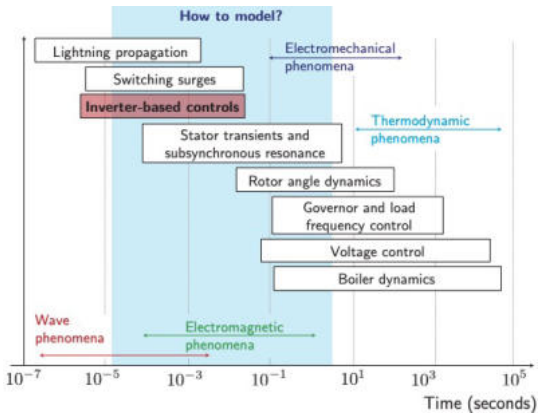


Source: Energinet

Complexity of dynamic system analysis is drastically increasing!

How do we assess the behavior of such a system after a disturbance?
Can we still do this effectively?

There is Another Problem: Time-Scale Separation



Source: N. Hatziargyriou et al, doi: 10.1109/TPWRS.2020.3041774

In SM-dominated systems, **electromechanical** and **thermodynamic** phenomena were key concerns:

- Frequency stability
- Voltage stability
- Rotor-angle stability

Inverters/ Renewable Generators act on the time-scales of **electromagnetic** and **wave** phenomena:

- **resonance stability & converter-driven stability**[1]
- Faster time-scales, and interaction of different dynamics

How do we study these systems?

- Analytical solutions typically do not scale to system-level stability assessment

» Time-domain simulations

[1] N. Hatziargyriou et al., "Definition and Classification of Power System Stability – Revisited & Extended," in IEEE Transactions on Power Systems, vol. 36, no. 4, pp. 3271-3281, July 2021, doi: 10.1109/TPWRS.2020.3041774

How do we perform a classic time-domain simulation?

Time-Domain Simulations require:

- a **system model**: physics & controls
- a **time-stepping, or integration, algorithm**

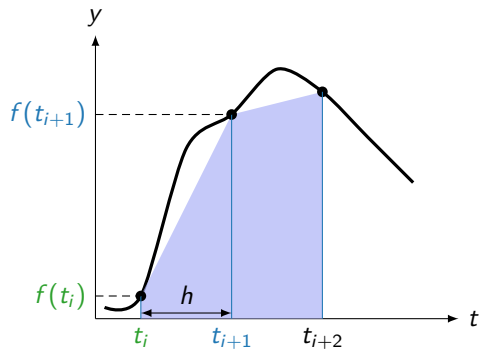
Given the initial condition for the system states $\mathbf{x}_0$, the integration algorithms find the solution for a timeline:

$$\mathbf{x}_{t+h} = \mathbf{x}_t + \int_t^{t+h} f(\mathbf{x}, \tau) d\tau$$

Since we cannot calculate the continuous integral analytically, we discretize it using numerical integration methods. **Example**: Trapezoidal Rule

$$\mathbf{x}_{t+h} = \mathbf{x}_t + \frac{1}{2} h \Big[f(\mathbf{x}, t) + f(\mathbf{x}_{t+h}, t+h) \Big]$$

We use an iterative algorithm to solve the implicit equation for each consecutive timestep.



Computational Complexity for Renewable Grids:

- Step size h must be **small enough** to capture the fastest dynamics
- Simulation time scales with the number of states
- Re-initialization required for discrete events

Classic Time-Domain Simulations are not Error-Free

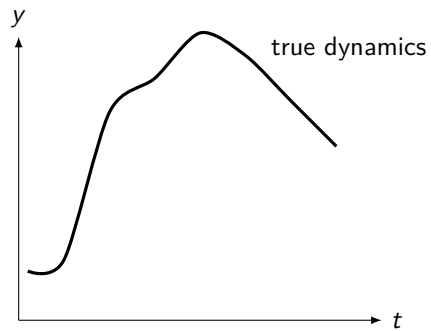
Classic time-domain simulations are subject to several errors. These stem from:

- the **system model**: DAEs are an **approximation** of the true dynamics, called **model error**
- the selected **time-stepping algorithm**

Numerical integration errors include:

- **local truncation error**: error caused by one iteration
- **global truncation error**: cumulative error caused over many time steps

Numerical integration errors depend on the **integration method** and the **time step**.

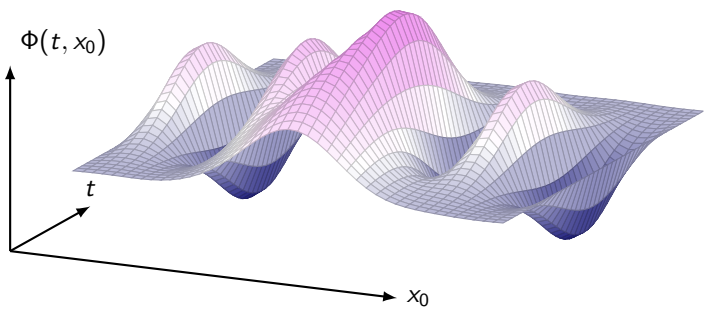


Let us Change The Perspective: Geometrical Interpretation

DAE System

$$\mathbf{M}\frac{d\mathbf{x}}{dt} = f(\mathbf{x}, t)$$
$$\mathbf{x}(0) = \mathbf{x_0}$$

Flowmap of a DAE System



The **flowmap** $\phi(t, x_0)$ contains all trajectories for all initial conditions $\mathbf{x}_0$.

A time-domain simulator computes **only one trajectory on the flowmap**.

What if we knew the flowmap?
A closed-form $\phi(x_0, t + h) \rightarrow x(t + h)$ would let us advance in time in one shot

- no repetitive iterations per step
- no recalculation for a different initial condition

Idea: Can we capture the flowmap with Machine Learning?

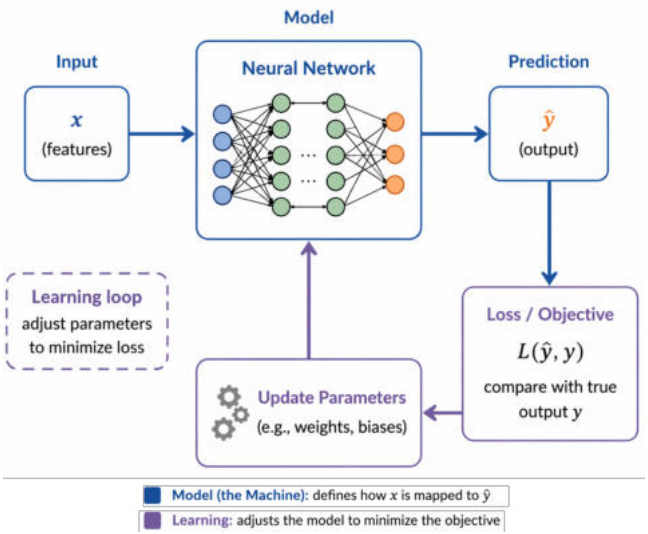
What is Machine Learning?

Machine Learning consists of two parts:

- the **Machine:** a mathematical model, e.g., a Neural Network (NN)
- the **Learning**: the calibration of the mathematical model to fit an objective

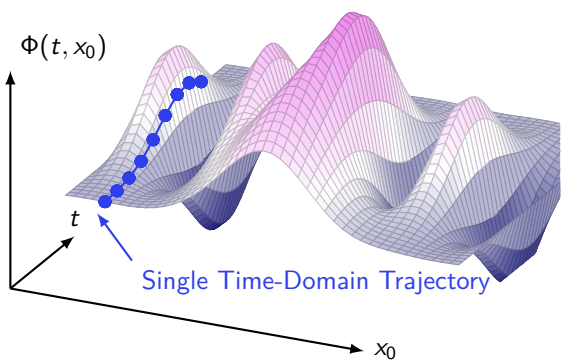
Benefits of using an ML Model:

- We can obtain a continuous function $\widehat{\mathbf{y}}(\mathbf{x})$
- Prediction is a sequence of simple parallel multiplications



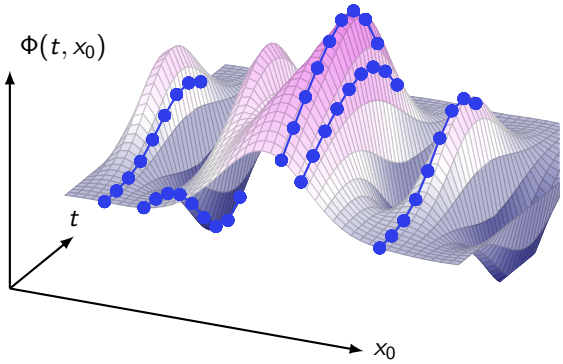
Goal for ML: Learning the Flowmap of the DAE System

Time-Domain Solver solves **iteratively** for **one** initial condition during each simulation



Evaluation per trajectory is computationally expensive. Execution of iterative algorithm per time-step and trajectory.

Goal for ML Algorithm: learn **a continuous representation of the flowmap**



Main computational burden for **learning of the flowmap**, but **fast evaluation of trajectories**.

Motivation to Use Machine Learning for Power System Dynamics

Simulation Speed-Up

Faster inference of the time-domain simulation

- Replace components that slow classic simulation down
- Full ML-based simulator

Accurate Aggregate Models

Dynamic behavior of **multiple/ complex components** in one ML surrogate

- External system
- Complex components
- Accurate dynamics: e.g learn EMT dynamics to represent in RMS domain

Intellectual Property (IP) Protection

ML-models are black-boxes and **provide IP protection inherently**

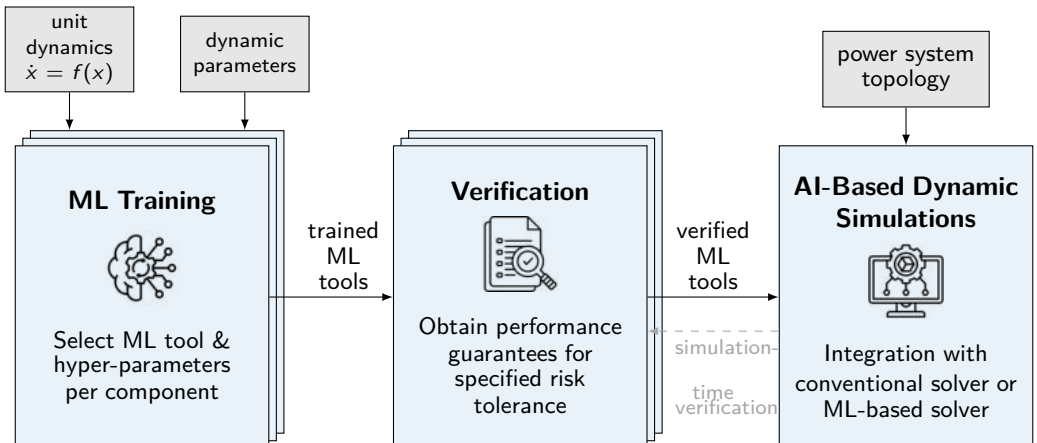
- Protect true dynamic model and parameters
- Potentially reduction of encryption needs
- No known methods for exact reverse engineering

RMS-Simulation: Root-Mean-Square Simulation with Phasor Approximation, EMT: Electromagnetic Transient Simulation

Images on this slide are AI generated (Copilot, 2.20260511.18.0)

How can we make it happen?

Learning the dynamics of an entire system is **infeasible** and extremely complex. Hence, we can learn **components or clusters of components** use them in ML-assisted time-domain simulations.

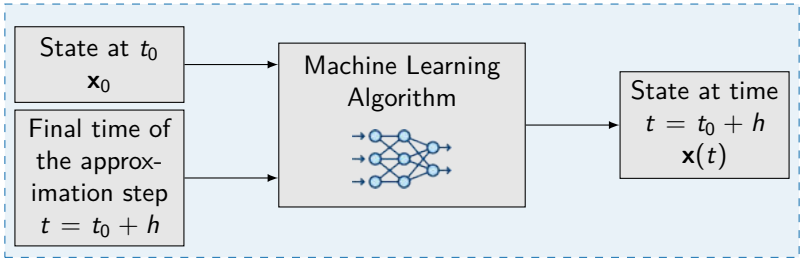


P. Ellinas, I. Karampinis, I. Ventura Nadal, R. Nellikkath, J. Vorwerk, S. Chatzivasileiadis "*Physics-informed machine learning for power system dynamics: A framework incorporating trustworthiness*", SEGAN, Volume 43, 2025, www.doi.org/10.1016/j.segan.2025.101818, presented at IREP 2025

Topic 1:

Overview of ML Tools for Power System Dynamic Applications

Our Goal: Learning an Initial Value Problem



The Initial Value Problem consists of a set of DAEs and an initial condition:

$$\mathbf{M}\frac{d\mathbf{x}(t)}{dt} = f\Big(\mathbf{x}(t), \mathbf{u}(t)\Big)$$
$$\mathbf{x}(t=0) = \mathbf{x}_0$$

Nomenclature from now on:

$\mathbf{x}$	true DAE solution
$\hat{\mathbf{x}}_{NN}$	NN output prediction for input
$\mathbf{x}_0$	initial condition, NN input

Classic Machine Learning Task

Classic machine learning approximates the mapping from features $\mathbf{x}_0$ to predictions $\hat{\mathbf{x}}$

- Supervised learning task
- Minimize a specified loss term during training

Given the true outputs $\mathbf{x}$, we train all model parameters θ, including the weights and biases, by optimizing

$$\min_{\theta} \mathcal{L}((\mathbf{x}_0, \theta), \mathbf{x}(t))$$

For the MSE (Mean-Squared Loss) and N samples:

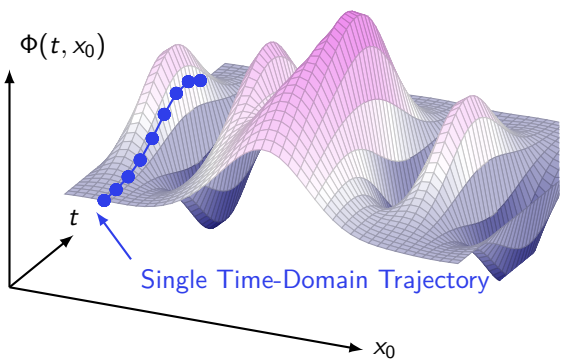
$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i - \hat{\mathbf{x}}_{NN,i})^2$$

The model architecture and activation functions (g, f in our example) are selected during hyper-parameter tuning.

While classic ML fits the data, it may **violate physics**!

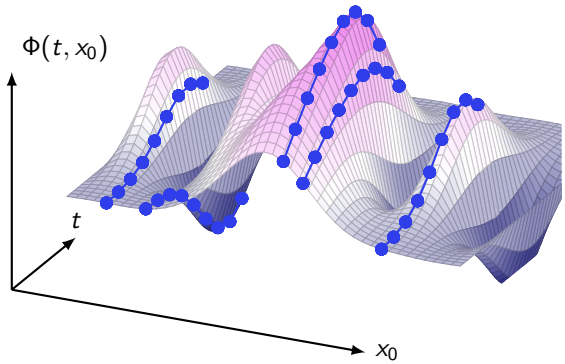
Vanilla Neural Network

$$\hat{\mathbf{x}}_{NN} = f\Big(\mathbf{W}_2 \cdot g(\mathbf{W}_1 \cdot \mathbf{x}_0 + \mathbf{b}_1) + \mathbf{b}_2\Big)$$



Shaping the Loss Function: Integrating Physics

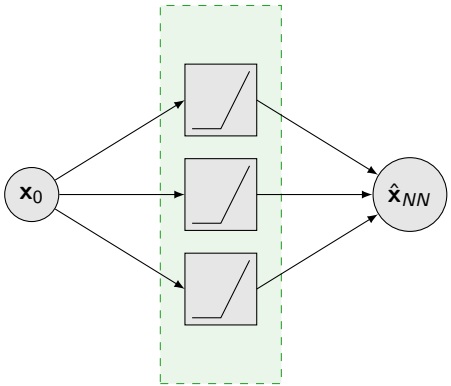
We change the minimization objective during training:



$\hat{x}_{NN}$: NN output, X: true solution, since X is unavailable, we use the NN output $\hat{x}_{NN}$ to evaluate the source term for the residual $\mathcal{R}_p$

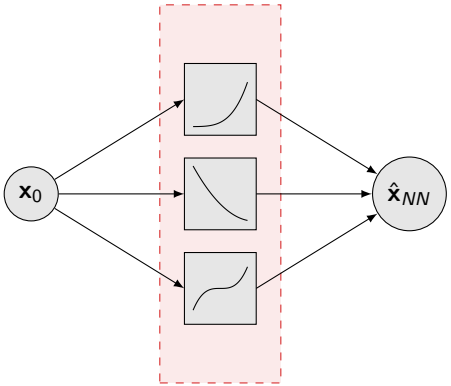
Shaping Activation Functions: Achieving Smaller Neural Networks

Classic (Vanilla) Neural Networks



- Classic activations: ReLU, tanh, linear, sigmoid
- Need large models to capture highly nonlinear behaviors

Learnable Activation Function



- Kalmogorov Arnold Networks (KAN) use B-splines for learnable activation
- Smaller neural network architectures can capture highly nonlinear systems

B-Splines: The Foundation of Kalmogorov Arnold Networks

Key Idea: Any multivariate, continuous function on a bounded domain is a sum of u univariate (1D) functions (**Kalmogorov-Arnold Presentation Theorem**).

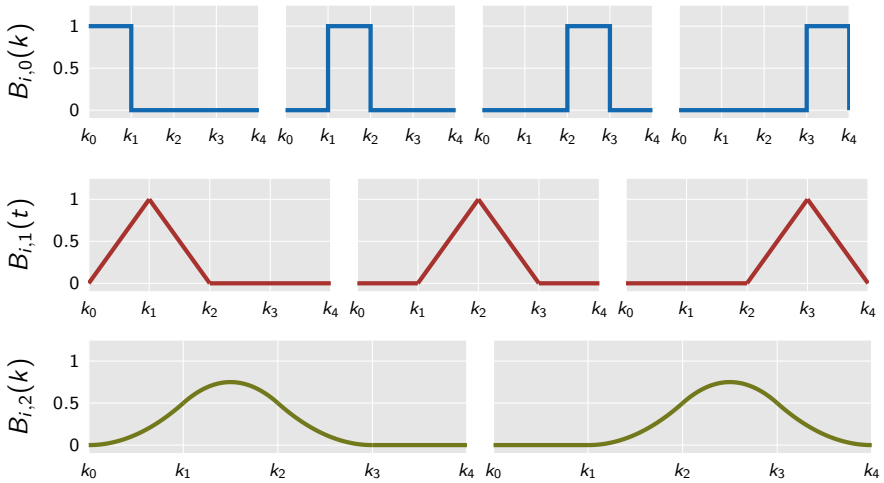
De Boor's Recursive Algorithm for B-Splines: Example with $k = 5$ uniformly distributed control points

0-th order basis function:

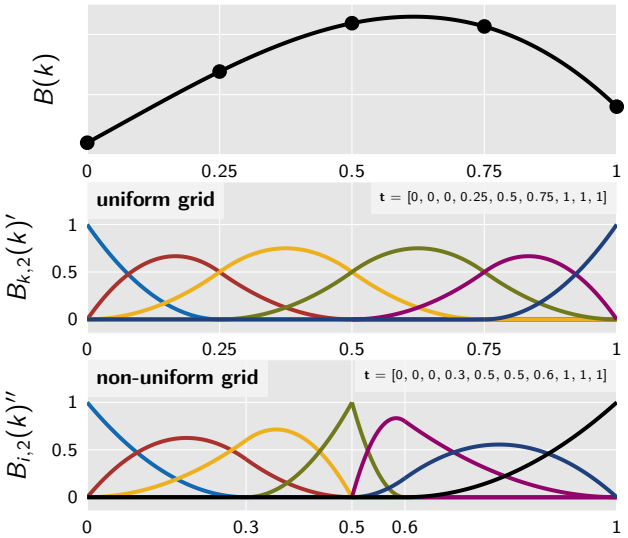
$$B_{i,0}(k) = \begin{cases} 1 & \text{if } k_i \leq k < k_{i+1}, \\ 0 & \text{otherwise.} \end{cases}$$

n-th order basis function:

$$B_{i,n}(k) = \frac{k - k_i}{k_{i+n} - k_i} B_{i,n-1}(k) + \frac{k_{i+n+1} - k}{k_{i+n+1} - k_{i+1}} B_{i+1,n-1}(k)$$



KAN-Layers: Weighted Sum of B-Splines provide Flexible Activation



The **final KAN activation** $\phi(\mathbf{x})$ is:

$$\phi(\mathbf{x}) = \mathbf{w} \cdot \underbrace{\mathbf{c} \cdot \mathbf{B}(x)}_{\text{weighted sum of B-Splines}} + \boldsymbol{\alpha} \cdot \text{base}(\mathbf{x}),$$

where

$\mathbf{w}, \boldsymbol{\alpha}$	weights
$\mathbf{B}(x)$	B-splines
$\text{base}(\mathbf{x})$	pre-defined base activation

KANs are Challenging to Train

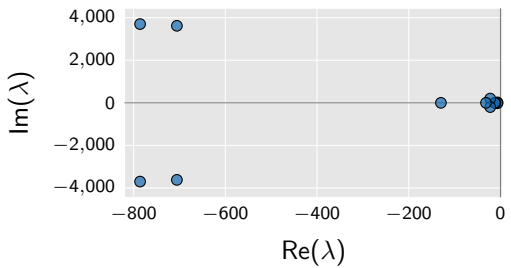
- trainable parameters: $\boldsymbol{\alpha}, \mathbf{c}, \mathbf{w}$
- hyper-parameters: grid (number and location of control points)
- one **recursive evaluation of splines per neuron**, hindering parallelization
- more **prone to overfitting**
- BUT they can provide a **compact closed-form solution**

Supporting Multi-Time Scales: ActNET Layers

Eigenvalue Spectrum of Power System Components

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{A}\,\mathbf{x}(t) + \mathbf{B}\,\mathbf{u}(t)$$

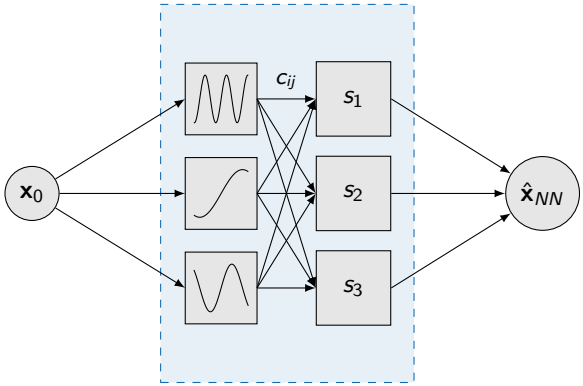
$$x_i \propto e^{-\lambda_{A,i} t} = e^{-(\sigma_i \pm j\omega_i)t}$$



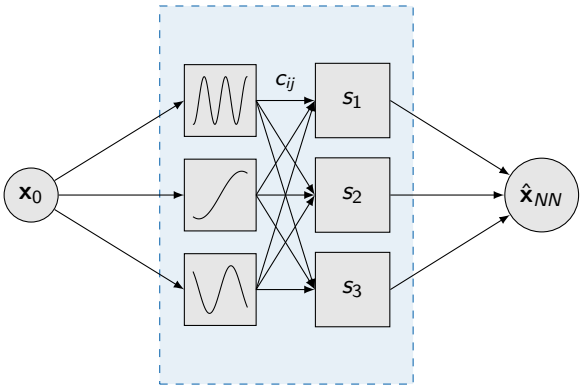
Challenges for ML:

- Dynamic behavior of one state x is superposition of multiple eigenmodes λ on multiple time-scales
- Learning modes across all required time-scales is computationally challenging

ActNET Layer: each input channel is a **sum of learnable sinusoidal components with different frequencies**



ActNET Layers: Decomposition into Multiple Frequency Components



Sinusoidal Activation Function

$$\phi_i(t) = \frac{\sin(\omega_i t + p_i) - \mu_i}{\sigma_i + \varepsilon}$$

- ω_i and p_i are **trainable frequency and phase parameters**
- μ_i, σ_i **normalize** the output for numerical stability

Given input vector $\mathbf{x}$, ActNET layers compute the mixed response of channel/neuron i:

$$\widehat{\mathbf{x}}_{NN} = \sum_{i=1}^{m} s_i(\mathbf{x})), \qquad s_i(\mathbf{x}) = \sum_{j=1}^{n} c_{ij}\, \phi_i(x_j)$$

This ActNET structure aligns the **architecture with the multi-timescale nature of power system dynamics!**

Note: We can learn all these advanced architectures with classic loss terms, or **physics-informed losses**.

Results: Capturing Different Frequency Components

Case Study Setup

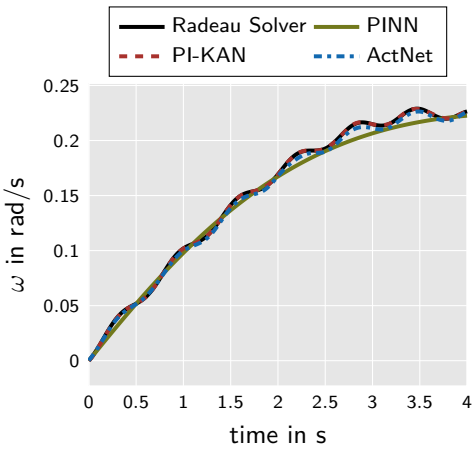
Learning the dynamics of a **single-machine-infinite bus (SMIB) system** with oscillatory voltage inputs:[2]

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & H & 0 \\ 0 & 0 & \tau \end{bmatrix} \frac{d}{dt} \begin{bmatrix} \delta \\ \omega \\ V \end{bmatrix} = \begin{bmatrix} \omega \\ P_m - EV\sin(\delta) + P_L(t) - D\omega \\ V_{\text{ref}} - V + k\sin(\delta) \end{bmatrix}$$

System Parameters and States

H	system inertia
δ, ω, V	dynamic states: angle, frequency and voltage
E	internal voltage of the synchronous machine
P_m, P_L	mechanical and electric load of the synchronous machine
D	damping coefficient
V_{ref}, k	shaping terminal voltage

Dynamic Response of a Vanilla PINN, PI-KAN and ActNET



[2] P. Ellinas, J. Vorwerk, S. Chatzivasileiadis, "*Tools to Explain Neural Networks for Power System Dynamics*"

Results: Accuracy and Runtime Comparison

Case Study Setup

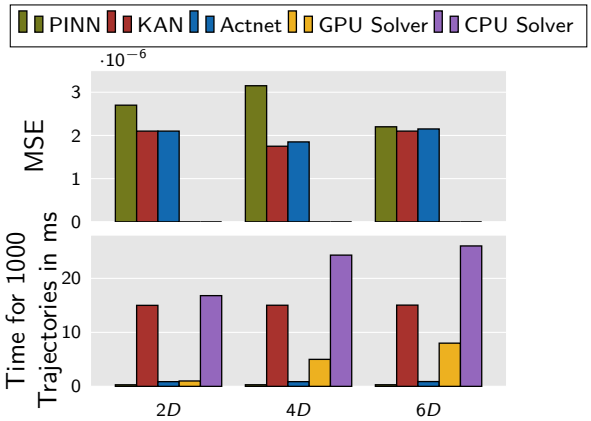
Learning the dynamics of an **SMIB system** of different orders:

- 2nd order SM model → 2D
- 4th order SM model → 4D
- 6th order SM model → 6D

Details in:

P. Ellinas, J. Vorwerk, S. Chatzivasileiadis, "*Tools to Explain Neural Networks for Power System Dynamics*"
www.github.com/elpetros99/tools_explaining_power

I. Karampinis, P. Ellinas, I. Ventura Nadal, R. Nellikkath, S. Chatzivasileiadis, "*Toolbox for Developing Physics Informed Neural Networks for Power Systems Components*," 2025 PowerTech
www.github.com/radiakos/PowerPINN



Key Results

- **Architectures with learnable activation** improve **accuracy** compared to vanilla PINN, while **KANs** significantly reducing architecture size
- Runtime of **ActNET** is comparable to vanilla PINN, since it is more parallelizable than **KANs**
- **KAN** are expensive during inference, **ActNET** provide good trade-off

Neural Operators: Encoding Information into Different Networks

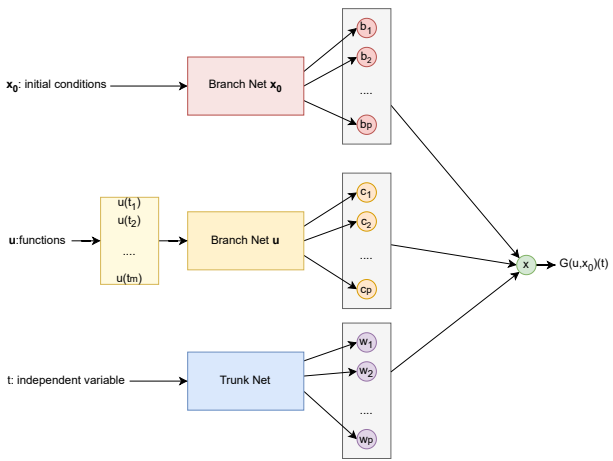
Conceptual Shift of DeepONet (Deep Operator Network):

- **separate encoding of spatial and temporal information** into different neural networks, weighted multiplication to obtain the final result
- **disturbances u(t) are variable in time**, rather than fixed for previously discussed NNs

Stacked DeepONet

- **Branch Nets** encode initial condition and disturbances into one neural network each
- **Trunk Net** encodes temporal information
- **Output**: dot product of spatial and temporal evaluation

Stacked DeepONet



Results: Comparing DeepONets to Vanilla PINNs

Case Study Setup:

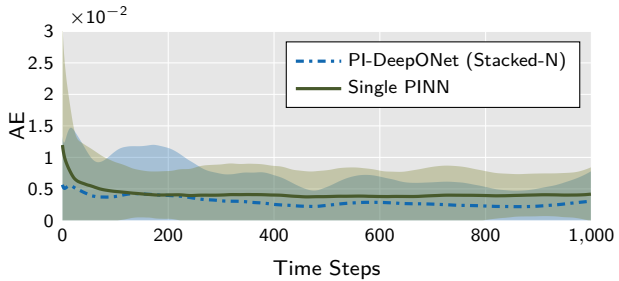
- 4th order SMIB system with varying voltage
- Time-varying input signal $\mathbf{u}(t)$: 1st or 2nd order ODE represents disturbance

Details provided in:

I.Karampinis, P. Ellinas, J. Vorwerk, S. Chatzivasileiadis, "*Neural Operators for Power Systems: A Physics-Informed Framework for Modeling Power System Components*", to be presented at PSCC 2026, www.arxiv.org/abs/2511.05216

www.github.com/radiakos/PowerDeepONet

DeepONet shows more stable errors over the prediction horizon



Inference Times and SpeedUp (in ms)

	Single Trajectory	1000 Trajectories
ODE solver	15.517	15516.834
PINN	0.189 (80×)	6.861 (2260×)
Stacked-N DeepONet	0.485 (30×)	2.415 (6430×)

ODE solver trajectories are evaluated sequentially, DeepONet and PINN simulations are parallelized

Topic 2:

Trustworthiness of ML Tools

How exact is the ML model?
What did the ML model learn?
Can we understand the ML model results?

Verification: How exact is the ML model?

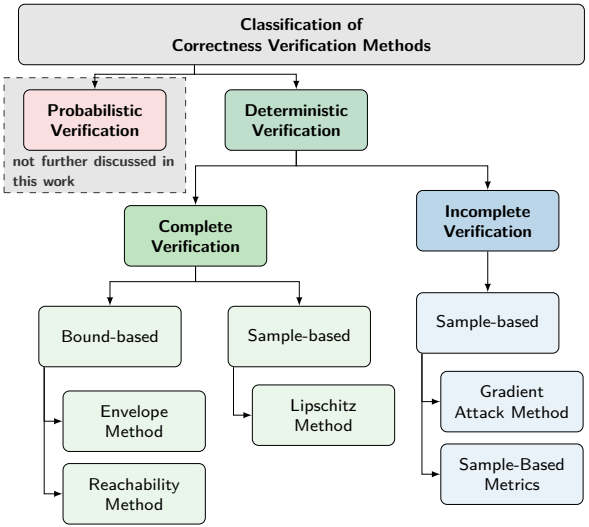
Verification finds the **maximum discrepancy** of the ML model to the **true solution** and checks it agains an error specification ε

$$\max_{(\mathbf{x}_0,t)\in\mathcal{U}} ||\mathbf{x}(\mathbf{x}_0,t) - \widehat{\mathbf{x}}_{NN}(\mathbf{x}_0,t)|| \leq \varepsilon$$

- $\widehat{\mathbf{x}}_{NN}$ is continuous and in close form
- $\mathbf{x}$ is potentially discretized and not in closed form

More details:
P. Ellinas, et al., "*Physics-informed machine learning for power system dynamics: A framework incorporating trustworthiness*", SEGAN, Volume 43, 2025, www.doi.org/10.1016/j.segan.2025.101818, presented at IREP 2025

Verification toolbox:
P. Ellinas, I. Chaudhuri, J. Vorwerk, S. Chatzivasileiadis, "*Verification and Validation of Physics-Informed Surrogate Component Models for Dynamic Power-System Simulation*", in review, www.arxiv.org/abs/2603.17836



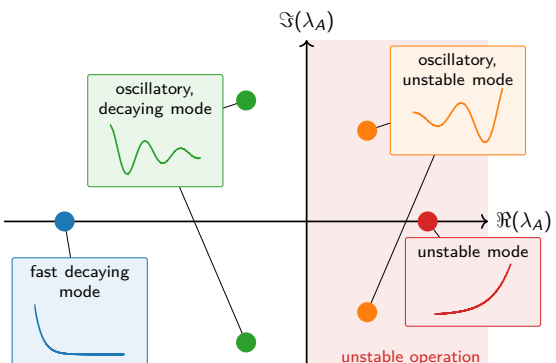
Neural Tangent Kernel (NTK): Insights into Training Dynamics

Small-Signal Stability Analysis

The locations of the eigenvalues $\boldsymbol{\lambda}_A$ of the state-space matrix $\mathbf{A}$ provide insights into the dynamic mode behavior:

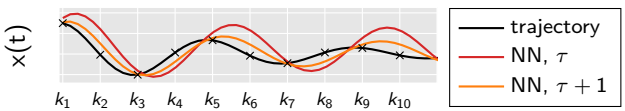
$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{A}\,\mathbf{x}(t) + \mathbf{B}\,\mathbf{u}(t)$$

$$x_i \propto e^{-\lambda_{A,i} t} = e^{-(\sigma_i \pm j\omega_i)t}$$



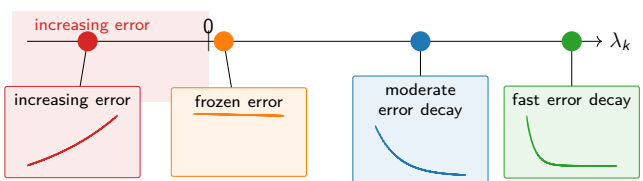
NTK Analysis

NTK provides insights into how fast dynamics are learnt by an ML model. Consider the following trajectory with 10 collocation points:



The NTK eigenvalues $\boldsymbol{\lambda}_k$ indicate how quickly the training error decays per collocation point. The training losses over epochs τ decay with learning rate η:

$$\mathcal{L}_k(\tau) \approx \mathcal{L}_k(0) e^{-\eta \lambda_k \tau}$$



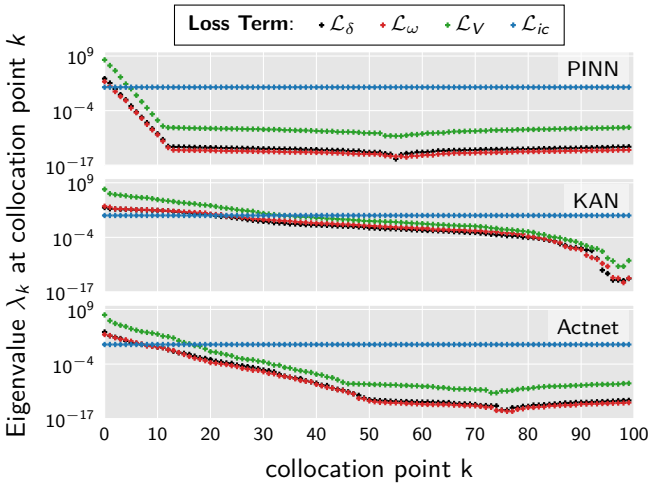
Neural Tangent Kernel (NTK): Insights into Training Dynamics

Case Study: Modified SMIB system from before, 2nd order SM with voltage dynamics
We monitor the different **physics-based loss terms**:

- rotor angle: $\mathcal{L}_\delta$
- rotor speed: $\mathcal{L}_\omega$
- terminal voltage: $\mathcal{L}_V$
- initial condition: $\mathcal{L}_{ic}$

A **flat NTK spectrum** indicates that all collocation points are learnt at an equal rate.

More details: P. Ellinas, J. Vorwerk, S. Chatzivasileiadis, "*Tools to Explain Neural Networks for Power System Dynamics*"



Main Findings:

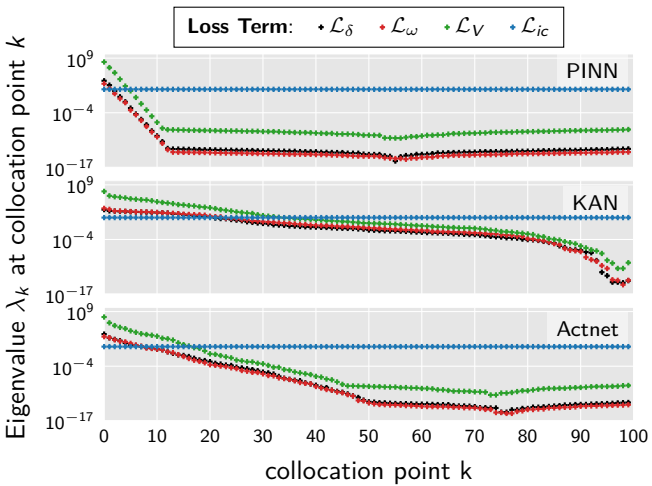
- Losses of fast-changing components (V) are always higher than slow components (δ, ω)
- KAN provides better learning (flatter profile) across the NTK spectrum

Neural Tangent Kernel (NTK): Updated Learning Weights

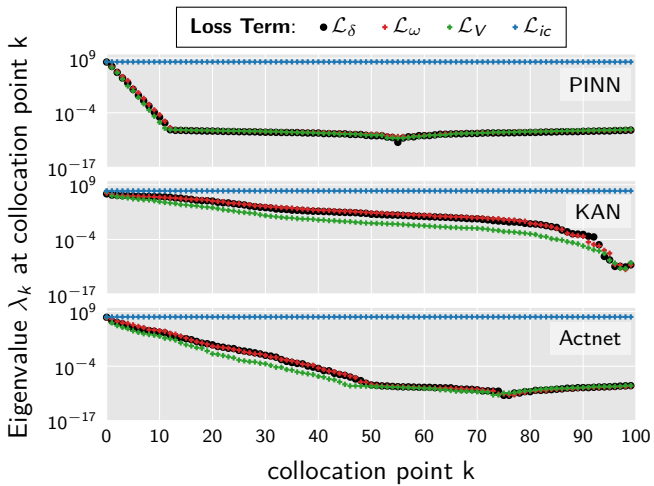
To promote **uniform convergence across loss components**, we integrate an **adaptive weighting scheme** during training:

$$\mathcal{L} = \mathbf{w}_\delta \mathcal{L}_\delta + \mathbf{w}_\omega \mathcal{L}_\omega + \mathbf{w}_V \mathcal{L}_V + \mathbf{w}_{ic} \mathcal{L}_{ic}$$

Classic Physics-Informed Training



Training with Adaptive Weights



Neural Tangent Kernel: Insights into Learning Inverter Dynamics

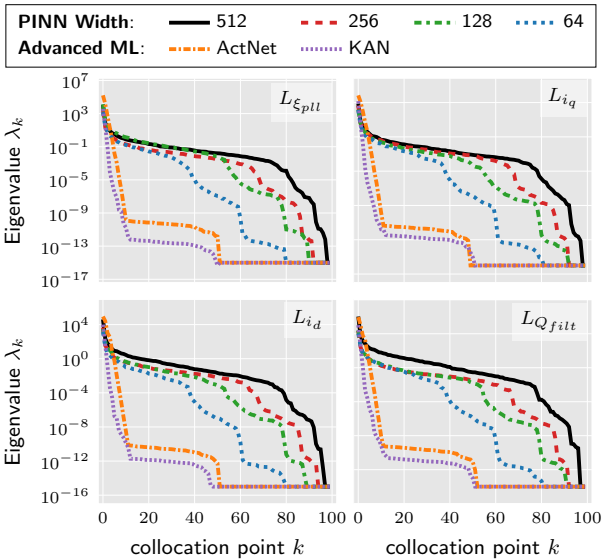
Case Study: **11th order inverter model** with nested control loops
We monitor the different **physics-based loss terms** for each of the 11 dynamics states.

- rotor angle: $\mathcal{L}_\delta$
- rotor speed: $\mathcal{L}_\omega$
- terminal voltage: $\mathcal{L}_V$
- initial condition: $\mathcal{L}_{ic}$

More details: P. Ellinas, J. Vorwerk, S. Chatzivasileiadis, "*Tools to Explain Neural Networks for Power System Dynamics*"

Main Findings:

- Spectra of some inverter states are very uneven: span a large range of magnitude across the entire spectrum
- No balanced learning across the different inverter states observed



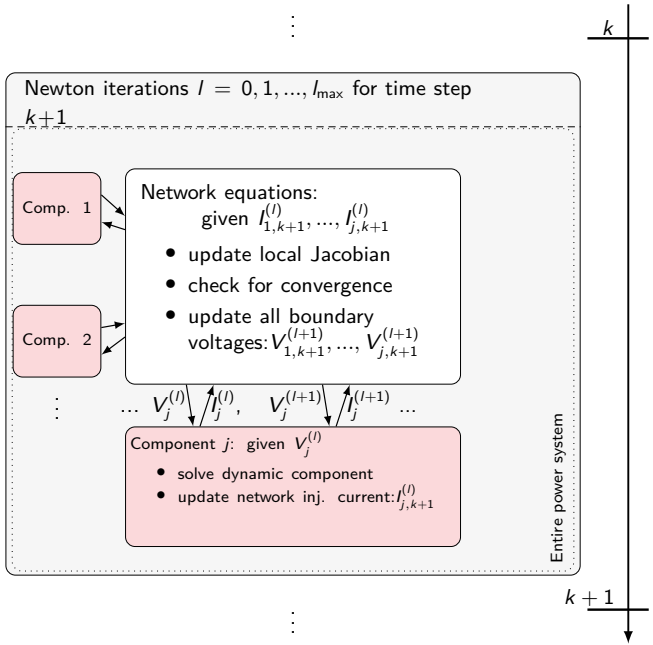
Topic 3:

System-Level Time-Domain Simulations with ML Support

How can we integrate the trained models into existing time-domain simulators?

Classic Time-Domain Simulations

- ODEs describe component dynamics: e.g. synchronous machine, converter, etc.
- Algebraic equations describe the network
- During one time-step, Newton iterations are solved until the network and dynamic equations match
- **Dynamic components** can be computed and updated in **parallel**, convert the terminal voltage to currents
- **Network equations** take current injections from components, solve the network, and update terminal voltages



Integration of Neural Surrogates into Time-Domain Solvers

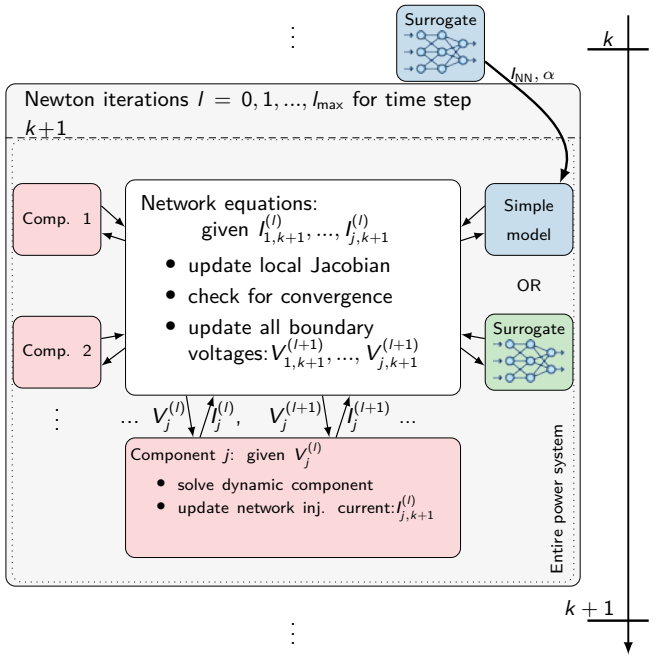
"Once-a-Newton-Iteration" Approach

- The ML surrogate is solved instead of the DAEs in the component
- Requires high accuracy ML surrogate for the Newton iteration to work

"Once-a-Time-Step" Approach

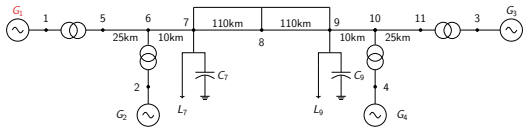
- The ML surrogate is solved **once per time-step**
- We use a simple model that represents the information retrieved from the ML surrogate in the Newton iterations, e.g. a constant

For both approaches, we need to **convert the terminal voltages into the input required** by our neural surrogate.



Example: Integration into Ramses/ STEPSS

Case Study: Replace G1 in the 11-bus Kundur system with a neural surrogate

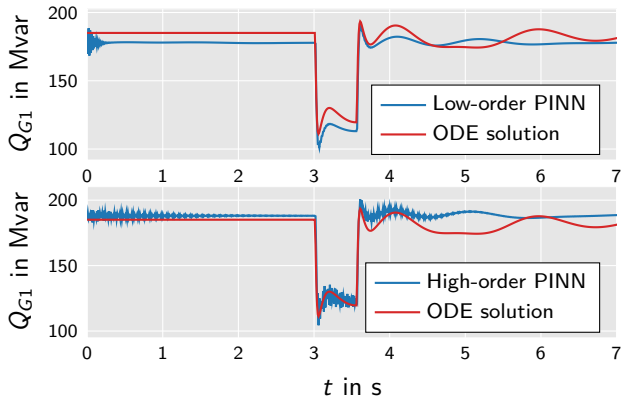


Details:

B. Gelfort, I. Karampinis, M. Desai, P. Aristidou, G. Hug, J. Vorwerk, "*Embedding Neural Surrogates into Established Dynamic Power System Simulators*", to be presented at PowerUp 2026, Boulder

www.github.com/BrunoGelfort/RamsesNN

Results: Once-a-Time-Step Approach



Main findings:

- "Once-a-Newton" Approach not successful: NN errors need to remain below the Newton tolerances
- "Once-a-Time-step" Approach successful, but **drift** in steady-state and oscillations observed

» enhance training procedures to reduce errors

Key Takeaways

ML can aid time-domain simulations through **speed-up**, **representation of complex dynamics** (within one or across simulation domains), and **improved IP**

Engineering activation functions together with **physics-informed learning** are interesting pathways for learning DAEs

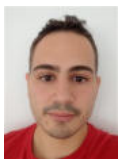
Verification and explainability tools like the **Neural Tangent Kernel (NTK)** analysis are key to enhance **trustworthiness and validity** of ML Tools

Neural surrogates can be **integrated into standard time-domain tools** through once-a-time-step evaluations (advancements needed for Newton iterations)

- We need to **consider downstream tasks when designing ML Tools**, e.g. accuracy requirements for time-domain simulations restrict acceptable training error
- **Improve training error**: integrate (real-world) data into the learning process to reduce overall modeling error, learning complex, high-order models
- Improve **neural surrogate representation in time-domain simulations** and conduct assessment integrated into final application
- Study **steady-state and stability properties** of neural surrogates

Thank you for joining us!

The main body of the research presented in this presentation was conducted by:



Petros Ellinas

Ioannis Karampinis

Bruno Gelfort

and supported through discussions within my other PhDs:

Bastien Giraud

Freeman Martin

Ioannis Papadopoulos

Additional collaborators for the presented work and related papers:

- Spyros Chatzivasileiadis, DTU
- Ignasi Ventura Nadal, DTU
- Rahul Nellikkath, DTU
- Maitraya Desai, ETH Zürich
- Gabriela Hug, ETH Zürich
- Petros Aristidou, Cyprus University of Technology

You are more than welcome reach out for discussions and follow-up!

Johanna Vorwerk
DTU, B325, R152
vorjo@dtu.dk
www.linkedin.com/in/johanna-vorwerk/
Google Scholar