Embedding Neural Surrogates into Established Dynamic Power System Simulators

Bruno Gelfort[1], Gabriela Hug[2], Maitraya Avadhut Desai[2], Petros Aristidou[3], Ioannis Karampinis[1], Johanna Vorwerk[1]

[1] *Technical University of Denmark* [2] *Power Systems Laboratory, ETH Zurich* [3] *Cyprus University of Technology*

Suggested Citation

B. Gelfort, G. Hug, M. A. Desai, P. Aristidou, I. Karampinis, and J. Vorwerk, "Embedding Neural Surrogates into Established Dynamic Power System Simulators," in *Proceedings of the PowerUp Conference*, 2026.

BibTeX Entry

```
@inproceedings{gelfort2026embeddingneural,
  author = {Gelfort, Bruno and Hug, Gabriela and Desai, Maitraya Avadhut and
      Aristidou, Petros and Karampinis, Ioannis and Vorwerk, Johanna},
  title = {Embedding Neural Surrogates into Established Dynamic Power System
      Simulators},
  booktitle = {Proceedings of the PowerUp Conference},
  year = {2026},
}
```

Embedding Neural Surrogates into Established Dynamic Power System Simulators

Bruno Gelfort[§], Ioannis Karampinis[†], Maitraya Desai[‡], Petros Aristidou[*], Gabriela Hug[‡], Johanna Vorwerk[†]

***Abstract*—The evolving landscape of power systems is leading to larger and more complex systems. Future power systems comprise more diverse components, exhibit faster dynamics, and are, in general, less predictable. As a result, time-domain simulations are becoming increasingly necessary for both offline studies and real-time stability assessment. This evolution renders dynamic simulations computationally more demanding, calling for improved simulation speed. Although classical solvers are reliable, their computational performance is limited by the need for iterative solution schemes. We present a modular method that allows Neural Network (NN) based surrogates of any power system component to be embedded within the decoupled Newton iterations of a time-domain solver. A case study integrates Physics-Informed Neural Networks (PINNs) into a solver and demonstrates the general applicability of the proposed approach. The results reveal specific challenges regarding PINN accuracy but highlight the promise of the framework and motivate further exploration using a wider variety of surrogate models.**

***Index Terms*—Newton method, power system stability, power system simulation, machine learning**

I. Motivation & Contribution

Time-domain simulations are becoming essential tools for power system planners and operators to assess system stability. The notion of system stability is evolving with the energy system's transition [1]. The transition from a power system dominated by few Synchronous Machines (SMs) towards a system dominated by Converter-Interfaced Units (CIUs) is accelerating. This transition brings many challenges, one of which is the computational complexity of the power systems' dynamic simulations [2]. The increasing complexity originates from a larger number of components, higher order components, and their faster time-scale dynamics. At the same time, the reduced physical inertia, faster control dynamics, and new stability phenomena introduced by CIUs require more detailed and frequent dynamic studies to ensure reliable system operation. Thus, time-domain simulations are becoming increasingly important for the assessment of system stability.

PINNs have been shown to effectively learn the solutions of Ordinary Differential Equations (ODEs) [3]. In short, a PINN is a NN with an augmented loss term that includes the standard data loss and loss terms based on the underlying physical equations of the ODE. As demonstrated in [4], PINNs represent a fast alternative to classical iterative numerical methods. However, the works on PINNs for power systems mostly represent and analyse a PINN as a stand-alone system.

Some works have investigated the integration of NNs into time-domain simulations. In [5], the authors manually construct a solver using the trapezoidal method and replace the solution process of some SMs with a PINN. The work in [6] further develops the idea by presenting promising results through the substitution of low-order SM models. Both studies are constructed using home-made simulators, leaving the question of compatibility with established simulators unanswered.

Other works have explored the acceleration of time-domain simulations with the help of Machine Learning (ML) tools. In [7], the simulation speed is increased by a factor of 4 by using data-driven surrogates to replace physics-based aggregation models of a part of the power system. The focus lies on a seamless replacement of the physics-based aggregation model by enforcing boundary conditions.

This work discusses the integration of neural surrogates into established power system time-domain simulators. The presented methods can work for any NN based surrogate that represents one single component, or a larger part of the power system and focuses on the integration rather than the component or components it replaces. As such, the contributions of the presented work are as follows:

1) We suggest two different methods on how to integrate neural surrogates into time-domain simulators. The suggested methods are applicable to any neural surrogate.
2) We demonstrate the integration approach through a case study that integrates PINNs into the time-domain simulator RAMSES [8]. As such, one SM in the two-area Kundur system is replaced by a PINN. Different model complexities of the connected synchronous machine unit such as the inclusion of the Automatic Voltage Regulator (AVR) and governor are considered.
3) The test case highlights how model accuracy impacts performance in time-domain simulations. Several requirements on how to enhance performance are discussed.

First, we present the internals of time-domain solvers, and propose two methods of embedding a surrogate model into the workflow. Then, in a case study, we train PINNs and integrate them into a solver using the two developed methods. We discuss the results and outline potential directions for future work.

II. Dynamic Simulations – Solution Through Decoupled Newton Iterations

Since the presented work focuses on the integration of neural surrogates into time-domain simulations, this section first provides an overview of classic time-domain simulation

§ Author is with Distributed Energy Systems section, Technical University of Denmark DTU. This work was conducted during his master's thesis, a collaboration between ETH Zürich, the Technical University of Denmark DTU and Cyprus University of Technology CUT.

‡Authors are with the Power Systems Laboratory, ETH Zürich, Zürich, Switzerland.

†Authors are with the Power Systems section in the department of Wind and Energy Systems, Technical University of Denmark, Kgs. Lyngby, Denmark.

∗ Author is with the Sustainable Power Systems Laboratory, CUT, Limassol, Cyprus.

processes for power systems through Newton iterations. Then, the section discusses how neural surrogates can be integrated in the presented classic, numerical time-domain simulation scheme. This overview links directly to the contributions by showing where and how surrogates fit into the solver.

A. Mathematical Notation

The following notation is used throughout the manuscript:

- Uppercase and bold lowercase symbols are real or complex vectors or matrices.
- Lowercase symbols are real or complex numbers.
- $\dot{\mathbf{x}} = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$ is the time derivative, and $\Psi_V = \frac{\partial \Psi}{\partial V}$ is the matrix of partial, element-wise derivatives of Ψ with respect to V.

B. How Decoupled Newton Iterations Work

This work is conducted on the widely used quasi-static phasor approximation, also called RMS-simulation. In short, the transmission system's circuit dynamics are represented as discrete changes between steady-state operating points. It is an important simulation method for transient stability assessment. The reader is referred to [2] for an in-depth review of the validity and limitations of this approximation.

The derivation of the method follows the approach in [9]. The following initial value problem describes the dynamics of a power system:

$$\Gamma_\Phi \, \dot{\mathbf{x}} = \Phi(\mathbf{x}, V, t), \tag{1a}$$

$$0 = \Psi(\mathbf{x}, V, t), \tag{1b}$$

$$\mathbf{x}(t_0) = \mathbf{x}_0, \quad V(t_0) = V_0, \tag{1c}$$

where the state vector is defined as $\mathbf{x} = [\omega, I, ...]^T$ and contains all system states except the voltages V. The initial operating point $(\mathbf{x}_0, V_0)$ at time t_0 is determined with a load flow calculation. Equations (1a) correspond to the ODEs Φ of the components, where the binary matrix Γ_Φ filters the unnecessary variables out of $\dot{\mathbf{x}}$. The algebraic equations Ψ correspond to the network equations. Using the current-injection model, (1b) corresponds to

$$\Psi(\mathbf{x}, V, t) = \boldsymbol{g}(\mathbf{x}, V) = \Gamma_I \mathbf{x} - YV = I - YV = 0. \tag{2}$$

Γ_I is binary such that $\Gamma_I \mathbf{x} = I$, the complex vector of currents. Y is the constant and independent admittance matrix. Generally, the equations in (1) do not have an explicit solution and must be solved numerically, which requires discretisation: Following the approach in [9] that uses a general discretisation formula, the differential equations (1a) are discretised along the independent variable t and reformulated as

$$\Phi(\mathbf{x}_{k+1}, V_{k+1}) - \frac{1}{h\beta_0}\Gamma_\Phi(\mathbf{x}_{k+1} - \boldsymbol{\beta}_k) = 0,$$

where the choice of $\boldsymbol{\beta}_k$ determines the discretisation method. The initial equations can now be written as purely algebraic:

$$\begin{aligned} &(\mathbf{x}_{k=0}, V_{k=0}) = (\mathbf{x}_0, V_0), \\ &\boldsymbol{f}(\mathbf{x}_{k+1}, V_{k+1}) := \Phi(\mathbf{x}_{k+1}, V_{k+1}) - \frac{1}{h\beta_0}\Gamma_\Phi(\mathbf{x}_{k+1} - \boldsymbol{\beta}_k) = 0, \\ &\boldsymbol{g}(\mathbf{x}_{k+1}, V_{k+1}) = \Psi(\mathbf{x}_{k+1}, V_{k+1}) = 0, \end{aligned} \tag{3}$$

which can be interpreted as a root-finding problem for both functions $\boldsymbol{f}$ and $\boldsymbol{g}$ at each time step. This calls for Newton iterations.

Newton iterations are an attempt to compute the values of the system's states for the next time step $k+1$, iterating over the Newton step l. For the k-th time step, the l-th Newton iteration is

$$\begin{bmatrix} V_{k+1}^{(l+1)} \\ \mathbf{x}_{k+1}^{(l+1)} \end{bmatrix} = \begin{bmatrix} V_{k+1}^{(l)} \\ \mathbf{x}_{k+1}^{(l)} \end{bmatrix} + \begin{pmatrix} \boldsymbol{\delta} V_{k+1}^{(l)} \\ \boldsymbol{\delta} \mathbf{x}_{k+1}^{(l)} \end{pmatrix}, \tag{4}$$

where the vector $(\boldsymbol{\delta}_{\cdot\, k+1}^{(l)})$ is the solution to the linear problem

$$\begin{bmatrix} \Psi_V & \Psi_{\mathbf{x}} \\ \Phi_V & \Phi_{\mathbf{x}} - \frac{1}{h\beta_0}\Gamma_\Phi \end{bmatrix} \begin{pmatrix} \boldsymbol{\delta} V_{k+1}^{(l)} \\ \boldsymbol{\delta} \mathbf{x}_{k+1}^{(l)} \end{pmatrix} = - \begin{bmatrix} \boldsymbol{g}\left(\mathbf{x}_{k+1}^{(l)}, V_{k+1}^{(l)}\right) \\ \boldsymbol{f}\left(\mathbf{x}_{k+1}^{(l)}, V_{k+1}^{(l)}\right) \end{bmatrix}. \tag{5}$$

The matrix on the left-hand side is sparse.

In short, the domain decomposition method presented in [9] enables each subdomain, i.e. each component in the case of power system simulators, to create its own local Jacobian and solve its own localised subsystem. The synchronisation with the rest of the network is achieved via Schur complement elimination: Every component exchanges an interface variable with the network at every Newton iteration until a global consensus is attained on the interface variables. Figure 1 visualises the numerical routine for a single time step and shows how each component is called multiple times during the solution process of the Newton iterations for the next time step $k+1$. Every component is given a new voltage and responds with an updated current injection.

The global consensus is measured by evaluating the mismatch vectors on the right-hand side of (5) against a convergence tolerance, i.e.

$$\left\| \boldsymbol{f}\left(\mathbf{x}_{k+1}^{(l)}, V_{k+1}^{(l)}\right) \right\|_\infty \leq \varepsilon_f, \quad \left\| \boldsymbol{g}\left(\mathbf{x}_{k+1}^{(l)}, V_{k+1}^{(l)}\right) \right\|_\infty \leq \varepsilon_g, \tag{6}$$

where ε_f and ε_g are the convergence tolerances of the numerical solution of the differential and algebraic equations, respectively.

C. How and Where to Embed a Neural Surrogate?

Figure 1 suggests two possible approaches for integrating a neural surrogate into the numerical scheme. Both methods include the replacement of the implicit solution process of a power system component with the explicit evaluation of a surrogate model.

1) *"Once-a-Newton-Iteration" Approach*: The solution process of one component is directly replaced by the explicit evaluation of the neural surrogate. In Fig. 1, this corresponds to replacing the call of one red component with the call of the neural surrogate.
2) *"Once-a-Time-Step" Approach*: The neural surrogate is called once per time step. This workflow differs from the previous approach and requires an additional model. The neural surrogate is evaluated before the beginning of the Newton iterations, and the information retrieved from the neural surrogate is placed inside the component's evaluation block. This calls for a separate model that

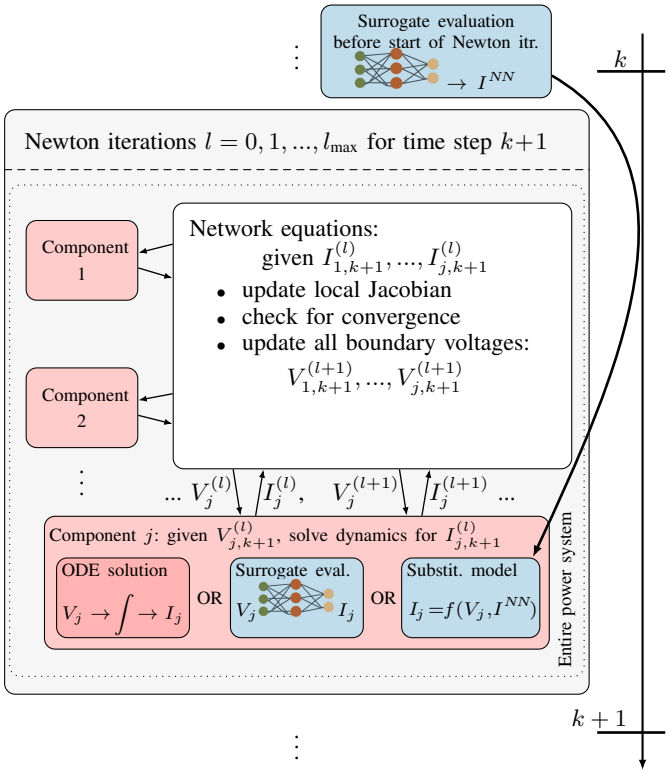

Fig. 1. Scheme explaining decoupled Newton iterations for one time step. The network connects with all power system components through the buses' boundary variables V and I, both physically and computationally. The blue boxes depict the neural surrogates' embedding methods "Once-a-Time-Step", where the surrogate is evaluated outside of the Newton iterations and "Once-a-Newton-Iteration", where the surrogate is directly communicating with the network.

interacts with the network during the Newton iterations. The model can be a simple constant, contain a parameter, or be a more elaborate model that is, for instance, dependent on the network voltage.

The architecture of the decoupled Newton iterations separates the solution of the network equations from the component equations. On the boundary remain the coupling variables V and I, which determines the shape that the neural surrogate block must adopt. It must follow the injector-typical input-output structure of Fig. 2. Given the initial dynamic states, an estimation of the network voltage (v_x, v_y), and the time step, it returns the corresponding injection current (i_x, i_y).

D. Which Neural Surrogates can be Embedded?

Any neural network surrogate can be used for the components for the presented integration methods. In [3], the

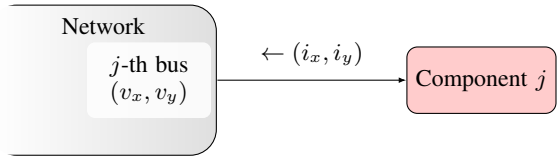

Fig. 2. Exchange of interface variables between the network and the component. At every Newton iteration, the component model reads the bus voltage and returns the corresponding current injection.

advantages of PINNs for ODEs are demonstrated, and [5] has confirmed the suitability of low-order PINNs for power system simulations. As a consequence, in this work, two PINNs are trained as a first substitution attempt using the toolbox presented in [4].

Once the choice of neural surrogate is made, it has to be included in an injector block. Indeed, in the time-domain formulation, a power system component normally takes nodal voltages as inputs and returns the corresponding injector currents. A PINN, however, operates on the dynamic states of the substituted component rather than on voltages directly. This means that the voltages first need to be mapped to the state vector used by the PINN. The PINN then advances these states from $\mathbf{x}_j(k)$ to $\mathbf{x}_j(k+1)$. Finally, the updated states are transformed back into the injector currents $(i_x, i_y)_{k+1}$. The internals of the injector block in Fig. 2 follow exactly this logic and are arranged to accommodate the PINN evaluation.

III. Case Study

A case study of the presented method is conducted on the 11-bus, two-area Kundur test system, at the steady-state operating point given in exercise 12.8 in [10]. The corresponding single-line diagram is given in Fig. 5. The pipeline of this case study consists of isolating the component, training the surrogate, and finally integrating the trained model into the simulator architecture.

1) Isolating the Substituted Component: For training the PINN, generator 1 is disconnected from the system and simulated with a connection to an infinite bus. The line equations connect the terminal voltages and injection currents in Park's reference frame with the infinite bus voltages:

$$\begin{bmatrix} c & s & -R_{\text{eq}}c + X_{\text{eq}}s & R_{\text{eq}}s + X_{\text{eq}}c \\ s & c & -R_{\text{eq}}s - X_{\text{eq}}c & -R_{\text{eq}}c + X_{\text{eq}}s \end{bmatrix} \begin{pmatrix} v_d \\ v_q \\ i_d \\ i_q \end{pmatrix} = \begin{pmatrix} v_{\infty,x} \\ v_{\infty,y} \end{pmatrix},$$

where $c = \cos\delta$, $s = \sin\delta$, with δ being the rotor angle. The parameters $R_{eq} = 0.0522\,\text{p u}$ and $X_{eq} = 0.10\,\text{p u}$ make for the equivalent impedance of the Kundur system seen at bus 1. The equations of the SMs are extracted from [11]. Two different models are implemented:

- The sixth order Sauer-Pai model, model 6.a corresponding to Eqs. (15.2)-(15.8), (15.11), (15.13), (15.15) in [11]. We call this model the "high-order ODE".
- The fourth order Two-Axis model, corresponding to Eqs. (15.2)-(15.8), (15.11), (15.29), (15.30) in [11]. We call this model the "low-order ODE".

Further, a constant governor with $P = 700\,\text{MW}$ and a simple integrator AVR $\frac{\text{d}E_f}{\text{d}t} = K_{\text{exc}}\,(V_{\text{ref}} - v_t)$ are integrated into the machine equations. The parameters are set as $K_{exc} = 10$, and $V_{ref} = 1.03\,\text{p u}$.

2) Neural Surrogate Training: The neural surrogate is trained on a dataset that is sampled from a truncated normal distribution around the steady-state conditions of the aforementioned operating point. With this dataset, we expect improved learning of the SM dynamics around the operating point at which the simulation is initialised. The exact values are summarised in Table I.

TABLE I
DESCRIPTION OF THE SET OF INITIAL CONDITIONS USED TO SIMULATE THE HIGH-ORDER ODE AND CREATE THE DATASET FOR THE HIGH-ORDER PINN (THE VALUES FOR THE LOW-ORDER ODE AND PINN ARE IN BRACKETS).

SM State	Lower bound	Upper bound	Number of samples	Steady-state values	
δ	-1.1	-0.4	11 (10)	-0.81854	(-0.80834)
ω	0.99	1.01	11 (10)	1.00000	(1.0)
e'_d	0.3	0.8	1 (2)	0.47720	(0.48167)
e'_q	0.7	1.1	1 (2)	0.95161	(0.94973)
ψ''_d	0.7	1.1	1 (0)	0.88543	(-)
ψ''_q	-1	-0.6	1 (0)	-0.62244	(-)
v_∞	0.98	1.04	11 (8)	1.01282	(1.01282)
θ_∞	-0.15	0	11 (8)	-0.07478	(-0.07478)
v_f	1.4	3	11 (8)	1.94418	(0.06438)

The NN architecture used in this work is detailed in Section A of the appendix. In short, it is a fully connected network with 4 hidden layers of width 64 and $\tanh$ activation function.

The two PINNs, one for the higher-order and one for the lower-order SM model, are trained on DTU's High-Performance Cluster on an NVIDIA A100 GPU [12].

For improved accuracy, the formulation of the loss function of the PINN has been augmented. Typically, the loss function of a PINN consists of

- the data loss: the Mean Squared Error (MSE) between the NN's prediction and the ground truth, given by the equation.

$$\mathcal{L}_{\text{data}} = \frac{1}{N_d}\sum_{i=1}^{N_d}\left|\hat{x}^{(i)}_{0+t} - x^{(i)}_{0+t}\right|^2, \tag{7}$$

- the physics informed data loss: the MSE between the derivative of the ODE and the Automatic Differentiation derivative of the NN.

$$\mathcal{L}_{\text{ph. data}} = \frac{1}{N_d}\sum_{i=1}^{N_d}\left|\frac{\mathrm{d}}{\mathrm{d}t}_{AD}(\hat{x}^{(i)}) - F\left(x^{(i)}, t, y_{0+t}\right)\right|^2, \tag{8}$$

- the physics-based loss: same as the physics informed data loss, but the points are randomly sampled within the input domain.

$$\mathcal{L}_{\text{ph. coll}} = \frac{1}{N_c}\sum_{i=1}^{N_c}\left|\frac{\mathrm{d}}{\mathrm{d}t}_{AD}(\hat{x}^{(i)}) - F\left(\hat{x}^{(i)}, t, y_{0+t}\right)\right|^2. \tag{9}$$

The PINN's loss function was further augmented to include a currents loss term, added as part of an effort to improve PINN accuracy. It is given by the MSE between the real and imaginary parts of the ground-truth and NN-predicted injection currents, i and $\hat{i}$, computed from x_{0+t} and $\hat{x}_{0+t}$.

$$\mathcal{L}_{\text{currents}} = \frac{1}{N_d}\sum_{i=1}^{N_d}\left|\hat{i}^{(i)}_{0+t} - i^{(i)}_{0+t}\right|^2. \tag{10}$$

3) Integration of the Trained PINN into the Simulator Architecture: Once the PINN is trained, it is exported in Open Neural Network Exchange (.onnx) format [13], which enables the transfer of any trained NN in a standardised and language-agnostic format. This conversion step is necessary because the power system simulator is implemented in Fortran, whereas the PINN is trained in Python. The ONNX model is then loaded into the Fortran environment using the RoseNNa library [14]. This native Fortran NN-inference implementation offers significant computational gains compared with Python-based versions [15].

After the neural model is imported, a new injector component is defined within the RAMSES architecture, which natively allows the creation of additional power system elements. Once the injector model is added, the RAMSES executable is compiled and a simulation is performed with the explicit neural surrogate replacing the implicit ODE solution process.

With the neural model available inside the simulator, the PINN-block logic can be implemented following the structure shown in Fig. 3. This figure illustrates how the PINN is embedded into a current-injector block that receives the required electrical quantities, evaluates the NN, and outputs the final injection currents into the grid.

The PINN is then tested on both embedding strategies described in Section II-C. For the "Once-a-Time-Step" approach, a constant-current injector, updated at each step, is chosen as the substitution model, with a fixed time step of $0.01\,\text{s}$.

IV. RESULTS

This section evaluates the performance of the solver using the PINNs. The analysis is structured around two elements: the accuracy of the PINN itself, i.e. its final prediction error, and the accuracy of the time domain solver using the surrogate models.

A. PINN Learning and Deployment

The error analysis is presented only for the PINN trained on the lower-order SM model since the analysis of the PINN trained on the higher order model gives very similar results. The training of the PINN yields a Mean Absolute Error (MAE) for the prediction of each of the SM's output states. However, this metric is not of primary interest. As indicated in Fig. 3, the MAE of the machine states does not translate linearly into the MAE of the current injection. Therefore, the error in the injection currents is estimated using a Monte Carlo simulation in which each output state of the PINN is independently perturbed, and the resulting error propagation is recorded and summarised in Table II. The most informative value is the MAE of the PINN in predicting the injection current:

$$\text{MAE}_{\text{PINN}}(|i|) = 0.0036,$$

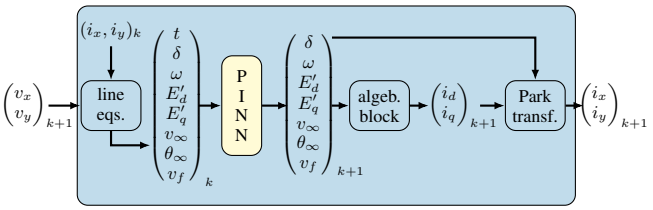

Fig. 3. Injector block that receives the predicted voltages $(v_x, v_y)_{k+1}$ and produces the corresponding injector currents $(i_x, i_y)_{k+1}$. The PINN evaluation is carried out internally within the injector block.

TABLE II
PINN LEARNING ERRORS AND PROPAGATION OF THESE ERRORS ON THE PREDICTION OF THE CURRENTS

Metric	Variable	Value
Individual MAE on test set ($\cdot 10^{-4}$)	$\begin{pmatrix}\delta\\ \omega\\ E'_d\\ E'_q\\ v_\infty\\ \theta_\infty\\ v_f\end{pmatrix}$	$\begin{pmatrix}5.2\\ 9.1\\ 2.0\\ 5.7\\ 8.2\\ 8.3\\ 7.1\end{pmatrix}$
Gain of individual variables on final MAE	MAE($\lvert i\rvert$)/MAE(vbl) for (δ ω ... v_f)	$\begin{pmatrix}2.2\\ 0\\ 1.5\\ 2.5\\ 2.0\\ 2.1\\ 0\end{pmatrix}$
Joint Monte Carlo		
final mean MAE	MAE $\begin{pmatrix}i_x\\ i_y\\ \lvert i\rvert\end{pmatrix}$	$\begin{pmatrix}0.0025\\ 0.0021\\ 0.0036\end{pmatrix}$

which aligns with the results reported in the literature [4]. This corresponds to a prediction accuracy of $(\Delta\, i_{pu} \cdot u_{pu}) \cdot S_{base} = 0.0036 \cdot 1.03 \cdot 900 = 3.3$ MVA. This relatively low prediction accuracy affects the convergence process of the Newton iterations.

B. Performance of the Simulation

The performance is measured for the two approaches described in Section II-C.

1) Performance of the PINNs for the "Once-a-Newton-Iteration" Approach: The PINN injector block predicts current injections with an expected MAE of $0.0036\,\mathrm{p\,u}$ As a consequence, if the convergence criterion for the Newton iterations is set to a value of similar dimension or lower, it has to be expected that no convergence is reached. This is confirmed by simulations with $\varepsilon_g = 10^{-4}$ and $\varepsilon_f = 10^{-4}$.

These results also provide a first indication of the error levels that a neural surrogate must achieve to be compatible with the Newton iterations. The MAE obtained here compares with PINN-based surrogates reported in the literature [4], but it remains too large relative to the reasonably set tolerances imposed by the solver. These simulations provide insights into the gap between the accuracy delivered by a practical PINN and the accuracy required for seamless integration into the Newton iterations.

2) Performance of the PINNs for the "Once-a-Time-Step" Approach: The "Once-a-Time-Step" approach differs from the "Once-a-Newton-Iteration" approach, as it does not rely on the Newton iterations to converge including the PINN injector. Instead, the PINN injector is called before the start of the Newton iterations, and passes the current injection value to the substitution model. The latter interacts with the Newton iterations by being a constant-current injector with a fixed prediction error throughout the Newton iterations. This allows for the convergence of the Newton iterations.

Figure 4 displays the reactive power injection of generator 1, modelled by an ODE, the low-order-trained PINN, and the high-order-trained PINN, respectively. Then, to excite fast dynamics of the system, a reactive power $Q = 150\,\mathrm{Mvar}$ is injected by an external source at bus 1 for $t \in [3\,\mathrm{s}, 3.5\,\mathrm{s}]$. Figure 4 shows, for both PINNs, initial oscillations and steady-state. The PINN introduces a steady-state offset of $3\,\mathrm{Mvar}$ to $6\,\mathrm{Mvar}$, depending on the model. Further, after the disturbance, we observe that the overall shape of the response is correct, but the PINN-modelled generator fails to follow the correct curve exactly. These observations are discussed in the next section.

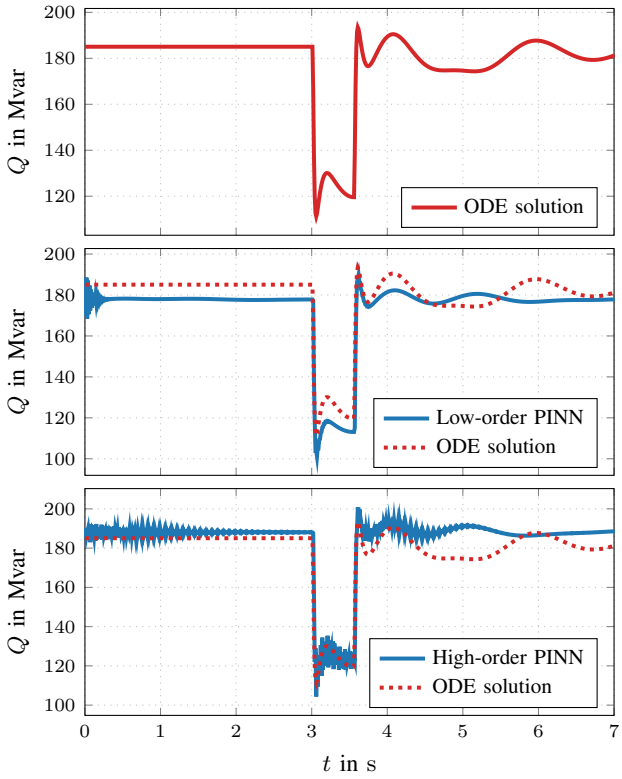

Fig. 4. Reactive power injection of generator 1 obtained from: (top to bottom) the ground-truth ODE, the PINN trained on the low-order ODE, and the PINN trained on the high-order ODE. Both PINN models are embedded via the "Once-a-Time-Step" method.

V. Discussion and Learnings

This section moves beyond the numerical results and considers their broader implications. The following points concern the embedding approach, the chosen surrogate, the training method and identify the limitations that the simulations have revealed.

1) Generality of the Approach: Although a PINN has been chosen in this work, any neural surrogate model can be inserted in the proposed framework. The only requirements are the input-output architecture illustrated in Fig. 2 and the level of accuracy needed to achieve convergence for the given convergence tolerances. Given the present setup, the PINN accuracy (amplified through the algebraic blocks illustrated in Fig. 3) must be improved by two orders of magnitude to respect the convergence tolerance of the Newton iterations.

2) Limitations of the Chosen Surrogate: The presented simulation highlights the challenge of the PINN to maintain the same fixed point, or steady-state as the underlying physical equations. Every trained model shows a drift from the physical steady-state. This presents a major drawback of the current implementation. Training methods need to be improved in order to ensure the steady-state of the ODE, and more generally, a closer match and more accurate representation.

3) How to Learn a Component's Behaviour: The component that shall be substituted by a neural surrogate must be linked to a suitable system to obtain time domain trajectories for training. The goal is to capture and learn the entire operating range of the SM for the PINN to be a perfect substitute. The SMIB approach applied in this work creates a closed electrical system and trajectories can be generated. However, this adds complexity since the line parameters need to be included in the model. These line parameters limit the generalisability of the trained PINN to varying network topologies.

Alternatively, a simpler approach is to impose a voltage evolution on the model's terminal and to learn the response of the component to this strict bound. Nonetheless, this does not come without risk. By imposing the voltage at the terminal rather than behind the impedance of a line, the flow of reactive power is not fixed. The machine freely chooses its reactive power injection. The effect of this on the trained PINN needs to be examined.

4) The Choice of the Surrogate Model: Eventually, the aim of this approach is to replace any component with a surrogate model. Aiming to develop a universal surrogate architecture that fits any model complexity might not be a sensible choice. The surrogate design becomes increasingly important as model complexity increases. For example, discrete converter models may require dedicated architectures, which is still an open research area.

VI. Conclusion

This work develops two methods of embedding any neural surrogate into the decoupled Newton iterations that are used by dynamic power system simulators such as RAMSES. In the presented case study, a PINN replaces a synchronous machine in the two-area Kundur system. This application highlights the modularity of the proposed method and its ability to host any type of neural surrogate. It also makes the accuracy requirement explicit: the MAE on the predicted injection currents must remain below the Newton tolerances. It thus demonstrates the weakness of the trained PINN in terms of digit precision.

The method calls for further experimentation. Future work includes the exploration of alternative neural surrogates with better-defined boundary conditions and higher numerical accuracy, as well as a systematic assessment of the computational speed-up achievable when replacing ODE-based components with neural models. In parallel, a trustworthiness framework with active monitoring and a fallback to the physical equations is needed to ensure robust operation. Once these elements are in place, the approach can be extended to additional system components, or entire system areas, leveraging the modular structure of the method. This line of investigation moves toward fast, ML-based dynamic simulations with potential applications in the real-time control of power systems.

VII. AI Usage Disclosure

AI assistance was used for code development (GitHub Copilot) and for semantic-level editing of this manuscript (Microsoft Copilot's GPT-5.2).

VIII. Acknowledgements

The work resulting in this paper was partially funded by the Independent Research Fund Denmark (DFF) in the DFF-Research Project 1 (thematic research) program under grant ID 10.46540/4307-00134B.

Appendix

A. PINN Training Details

The NN architecture used in this work is summarised in the following points:

- **Framework, precision**: The model is implemented in PyTorch and trained with `float64` precision. The global seed is 37.
- **Layers**: It is a fully connected network. There are 4 hidden layers of width 64. The forward pass uses the following activation functions:
  - Linear(`input_dim`→64)+`tanh`,
  - Linear(64→64)+`tanh`,
  - Linear(64→64)+`tanh`,
  - Linear(64→64)+`tanh`, Linear(64→`output_dim`).
- **Weights initialisation**: Xavier normal.
- **Additions**: No normalisation, dropout, or residual connections are used.

The important parameters for the training are the following:

- **Optimiser**: SOAP, from [16]. Learning rate = $1e^{-3}$, betas=(0.95, 0.95), weight decay=0.01, precondition frequency=10, max precondition dimension = 10000.
- **Criterion**: MSE loss, automatic switch to MAE loss after 300 epochs for improved robustness.
- **Splitting**: 80/10/10 for train/validation/test set. The dataset trajectories are randomly shuffled before splitting.
- **Batch size**: Training uses full batch optimisation.
- **Weights**:
  - supervised data loss: 1
  - physics data loss: $1e^{-3}$
  - physics collocation loss: $1e^{-4}$
  - currents loss: $8e^{-2}$ (this loss is added to the PINN toolbox for improved precision)
- **Training horizon, convergence**: max epochs: 10000, patience: 3000 epochs, tolerance: $1e^{-7}$.

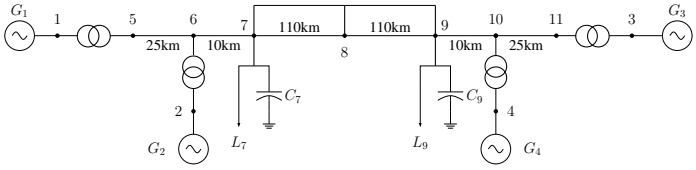

Fig. 5. Single-line diagram of the 11-bus Kundur test system

REFERENCES

[1] D. Vahle, V. Staudt, and C. Sourkounis, "Stabilizing power of grid-connected power systems," in *Conference on Sustainable Energy Supply and Energy Storage Systems*, pp. 33–48, 2024.

[2] J. D. Lara, R. Henriquez-Auba, D. Ramasubramanian, S. Dhople, D. S. Callaway, and S. Sanders, "Revisiting power systems time-domain simulation methods and models," *IEEE Transactions on Power Systems*, vol. 39, p. 2421–2437, Mar. 2024.

[3] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[4] I. Karampinis, P. Ellinas, I. Ventura Nadal, R. Nellikkath, and S. Chatzivasileiadis, "Toolbox for developing physics informed neural networks for power systems components." arXiv preprint arXiv:2502.06412, 2025.

[5] I. Ventura Nadal, J. Stiasny, and S. Chatzivasileiadis, "Physics-informed neural networks: A plug and play integration into power system dynamic simulations," *Electric Power Systems Research*, vol. 248, p. 111885, Nov. 2025.

[6] I. V. Nadal, R. Nellikkath, and S. Chatzivasileiadis, "Physics-informed neural networks in power system dynamics: Improving simulation accuracy," in *IEEE Kiel PowerTech*, June 2025.

[7] M. Bossart, J. D. Lara, C. Roberts, R. Henriquez-Auba, D. S. Callaway, and B.-M. Hodge, "Acceleration of power system dynamic simulations using a deep equilibrium layer and neural ode surrogate," *IEEE Transactions on Energy Conversion*, vol. 40, no. 4, pp. 2710–2722, 2025.

[8] P. Aristidou, S. Lebeau, L. Loud, and T. Van Cutsem, "Prospects of a new dynamic simulation software for real-time applications on the hydro-québec system," *Cigre Science & Engineering*, vol. 4, pp. 88–, February 2016.

[9] P. Aristidou, *Time-Domain Simulation of Large Electric Power Systems Using Domain-Decomposition and Parallel Processing Methods*. PhD thesis, Université de Liège, 2015.

[10] P. S. Kundur and O. P. Malik, *Power System Stability and Control*. New York: McGraw Hill, 2 ed., 2022.

[11] F. Milano, *Power System Modeling and Scripting*. London: Springer, 2010.

[12] DTU Computing Center, "Dtu computing center." Technical University of Denmark, 2024.

[13] "Onnx: Open neural network exchange." https://onnx.ai/. Accessed: 2026-01-29.

[14] A. Bati and S. H. Bryngelson, "Rosenna: A performant, portable library for neural network inference with application to computational fluid dynamics," *Computer Physics Communications*, vol. 296, p. 109052, Mar. 2024.

[15] A. Furlong, X. Zhao, B. Salko, and X. Wu, "Native fortran implementation of tensorflow-trained deep and bayesian neural networks," 2025.

[16] N. Vyas, D. Morwani, R. Zhao, M. Kwun, I. Shapira, D. Brandfonbrener, L. Janson, and S. Kakade, "Soap: Improving and stabilizing shampoo using adam," *arXiv preprint arXiv:2409.11321*, 2025.

I. Review #169A

A. Paper summary

The paper presents a method to integrate neural surrogate models into existing time domain solvers used in power systems (based on Newton iterations). It proposes two different ways to achieve integration and provides numerical results and very interesting discussion points from the application to a classical two-area network with synchronous generators. For the neural surrogate Physics-Informed Neural Networks are chosen although the framework could support other models too. The dynamics represented are of synchronous generators, including a simple AVR representation (based on one differential equation).

B. Comments for authors

The paper is very interesting and builds on existing work on the use of PINNs for speeding up power system time domain simulations, a timely and important issue. Overall, the paper is very well written and offers very interesting insights. It is still early-stages work as the results suggest, due to the persisting relatively high errors that do not allow convergence within Newton iterations. The alternative approach does allow integration in time domain solvers with some errors still persisting, although this might be something that to some extent can be justifiable (?), i.e. a less accurate solution for the sake of speed-up. Some detailed comments follow:

1) One key advantage of using ML surrogate models is the ability to speed-up the solution. It would be nice to have this element quantified also in this paper for the case studies presented.
2) Related to speed, does the need for the additional injector block with algebraic equations introduce any (quantifiable) computational burden?
3) The PINNs are trained on an infinite bus equivalent network which as the authors discuss might introduce limitations. I was particularly interested to see the authors view with respect to interactions between dynamic devices (e.g. inter-area modes or control interactions, etc.). Would the training of the PINN need to be able to reflect any internal dynamics related to such phenomena and what that might entail for training? It would be interesting to further discuss this aspect.
4) I understand that this might fall outside of the scope of this paper, however it would be interesting to discuss any limitations with respect to limiters and/or other discrete dynamics. Since the motivation is for future power systems, such behaviours will be increasingly observed.
5) The training of the PINN is sampled through a normal distribution around an operating point. This would cause the PINN to be valid around that operating point but what would the implications/limitations of this be for large disturbances? Some clarification around this would be useful to have. For example, what would happen for a short circuit fault where large excursions of state variables and nonlinearities play an important role?
6) The error analysis is presented for the low order model. Was it comparable for the high order model also?
7) When calculating the error, each output state of the PINN is independently perturbed for the calculation. Could the authors please comment on potential interactions if more state variables change at the same time? – again something likely to happen when facing large disturbances.
8) In the results presented in fig. 4 for the high order model, there are some high frequency oscillations. Would you happen to know where these are coming from? Could it be for example from not accurate training related to ODE's that reflect these fast dynamics (e.g. stator dynamics), because the training has not properly captured such behaviours? This could link to how a robust training set can be designed.

C. Summarize your recommendation in one to two sentences

Overall, a very interesting paper describing initial work on integrating PINNs in time domain solvers. I would recommend some further clarifications on the limitations as mentioned in the detailed comments.

— Panagiotis Papadopoulos

II. Review #169B

A. Paper summary

This paper provides two different approaches for neural network-based surrogates to be embedded within the decoupled Newton iterations of an established time-domain solver: (1) a once-a Newton-iteration approach, and (2) a once-a-time-step approach. This is demonstrated via using a PINN to replace a synchronous machine within the RAMSES simulator on the Kundur system; exchange between the PINN and the broader system is done via estimation of injector currents. Results show that PINN accuracy is insufficient to enable convergence under approach (1). Accuracy is sufficient to enable convergence under approach (2), but the resultant pseudo-surrogate simulation has some steady-state offset; after a disturbance, the overall response shape is correct, but the exact curve is not exactly followed. The authors discuss implications of these results, as well as paths forward for modularly integrating neural surrogates into time domain simulation.

B. Comments for authors

Strengths:

- The study is well-motivated by the need for faster time domain simulation, the potential of neural network surrogates to aid with this, but the need to understand the limitations of these surrogates.
- The approaches and methodological details are overall clear.
- The analysis of the implications of PINN error for overall time-domain results is interesting and useful.

Weaknesses:

- I think Figure 1 could be restructured to be clearer. While the text description of the two different approaches presented is clear, I was not able to follow the figure (perhaps changing the arrow placement or using boxes to highlight certain parts of the figure could be helpful).

Questions:

- As I do not work in this area, it is difficult for me to situate the contribution with respect to prior work. Notably, the discussion of prior work indicates that embedding neural surrogates into time-domain reduction is not new. As such, is the new contribution specifically embedding it into RAMSES (an established simulator)? Is this then largely a software engineering contribution, or did new methodologies/interfaces need to be developed for integration into RAMSES? Further clarification of the contribution would be helpful for me as a non-expert in this area.

C. Summarize your recommendation in one to two sentences

I believe this paper provides a solid approach and case study for integration of neural network surrogates into time domain simulation, and warrants acceptance.

— Priya Donti

III. Review #169C

A. Paper summary

The paper proposes two general methods to embed neural surrogates in the form of Physics-informed Neural Networks (PINNS) into classic Newton-based time-domain simulators. The paper demonstrates the approach through a case study that replaces a synchronous machine model with a PINN in an established simulator RAMSES.

B. Comments for authors

I must admitt I am not the expert on state of the art power system simulators, but I am faily familiar with time-domain solvers and PINNs. So please take my comments from this perspective.
Strengths:

- the paper is addressing relevant and timely problem of hybrid neural-numerical solvers for modern power systems simulations.
- The paper presents integration-focused contribution: rather than only presenting a surrogate, the paper tackles the practical and necessary problem of embedding NN surrogates into established simulator architectures (FORTRAN-based RAMSES), which is of high practical value for power system experts.
- The paper does not oversell, the authors reports limitations (steady-state offsets, numerical-precision issues, required improvement in surrogate accuracy) and discuss operational safeguards (monitoring, fallbacks).

Weaknesses and comments:

- The PINN results show steady-state offsets and errors that prevent the Newton solver from converging with standard tolerances, but this issue is not sufficiently treated in the current paper. There is a rich body of literature that allows PINNs to train with improved convergence even for stiff systems, e.g., utilizing adaptive sampling of collocation points, adaptive weight tuning, and quasi second order methods.
- 11-bus system is suitable case study for demonstrating new ideas, it will be interesting to see future work with large-scale systems. Discussions on current limitations and potential solution approaches in these realistic settings could strenghten the conceptual and forward looking part of the paper.
- PINNs represent implicit solvers that are typically trained for specific initial and boundary conditions. While possible to train parametric PINNs that generalize over multiple initial conditions this would render the PINN problem extremely high dimensional and not very practical. To tackle this limitation, the scientific machine learning community have developed an idea of physics-informed neural operators (PINO) that allow to train solution operators in infinite dimensional spaces, following similar learning principles to PINNs. I think the future work in this space of neural surrogates for physical systems should focus on the use of PINOs instead of PINNs.

C. Summarize your recommendation in one to two sentences

Accept: I think this is a solid paper with interesting ideas and clearly highly proficient technical implementation from the power systems simulation side.

— Jan Drgona

IV. Brief response to reviewers (optional)

Review #169A

1 - One key advantage of using ML surrogate models is the ability to speed-up the solution. It would be nice to have this element quantified also in this paper for the case studies presented.

We agree with this. The speedup of the NN component compared to the numerical integration is not quantified. However, at this stage, it does not make sense to compare two execution speeds of simulations that do yield the same, or similar results. Individual solves or NN inference would be interesting to measure. We attempted to clock the execution of the blocks in Fortran. However, the execution is too fast to be measured with Fortran's clock.

2 - Related to speed, does the need for the additional injector block with algebraic equations introduce any (quantifiable) computational burden?

These operations are executed faster than Fortran's system clock resolution. This makes it difficult to measure reliably.

3 - The PINNs are trained on an infinite bus equivalent network which as the authors discuss might introduce limitations. I was particularly interested to see the authors view with respect to interactions between dynamic devices (e.g. inter-area modes or control interactions, etc.). Would the training of the PINN need to be able to reflect any internal dynamics related to such phenomena and what that might entail for training? It would be interesting to further discuss this aspect.

The PINN learns the internal dynamics of the synchronous machine by directly learning the ODE's behaviour. The paper mentions the limits of simulating the "trajectories" (and further learning them) of the SM when connected to an infinite bus through a line, with fixed line parameters. The PINN is expected to perform well around this operating point, and this line configuration. The authors acknowledge the limitations of this and propose a different training approach (simulating imposed voltage variations as part of the training). The expected improvements should be visible once the PINN's prediction accuracy permits the integration inside the Newton iterations.

4 - I understand that this might fall outside of the scope of this paper, however it would be interesting to discuss any limitations with respect to limiters and/or other discrete dynamics. Since the motivation is for future power systems, such behaviours will be increasingly observed.

This point needs to be studied in detail and has a strong influence on the choice of surrogate model. We have added an explanation that clarifies this point.

5 - The training of the PINN is sampled through a normal distribution around an operating point. This would cause the PINN to be valid around that operating point but what would the implications/limitations of this be for large disturbances? Some clarification around this would be useful to have. For example, what would happen for a short circuit fault where large excursions of state variables and nonlinearities play an important role?

The training set should include any expected operating point. Since the focus of this work was to obtain a working integration first, this point was not treated. However, it is an important consideration and needs to be included in future work.

6 - The error analysis is presented for the low order model. Was it comparable for the high order model also?

Yes. A sentence was added to the paper regarding this point.

7 - When calculating the error, each output state of the PINN is independently perturbed for the calculation. Could the authors please comment on potential interactions if more state variables change at the same time? – again something likely to happen when facing large disturbances.

The point mentioned in the comment was addressed in the paper by the Monte Carlo simulation, in which the variables are simultaneously perturbed. This results in a 'joint error gain' as stated in Table II.

8 - In the results presented in fig. 4 for the high order model, there are some high frequency oscillations. Would you happen to know where these are coming from? Could it be for example from not accurate training related to ODE's that reflect these fast dynamics (e.g. stator dynamics), because the training has not properly captured such behaviours? This could link to how a robust training set can be designed.

The exact source of these oscillations could not be found. Their origin is being investigated for future work.

Review #169B

1 - "Weaknesses": Figure 1 is hard to understand.

Figure 1 has been reworked and we hope it is clearer now.

2 - "Questions": The contribution of this paper.

Prior work has focused on creating specialised, custom solvers for the integration, and has integrated NNs into a specific integration method (i.e. trapezoidal). The contribution of this paper lies in the successful integration of any surrogate into an established solver's architecture, and a case study with a PINN.

Review #169C

1 — The PINN results show steady-state offsets and errors that prevent the Newton solver from converging with standard tolerances, but this issue is not sufficiently treated in the current paper. There is a rich body of literature that allows PINNs to train with improved convergence even for stiff systems, e.g., utilizing adaptive sampling of collocation points, adaptive weight tuning, and quasi second order methods. The reported accuracy is already the best we observed under the chosen setup. We use a second-order optimiser (SOAP), deliberately concentrate the model to the operating region in order to push accuracy as high as possible, and add an extra loss term that directly penalises the current prediction. Without altering the surrogate's architecture, this is the lowest error we obtained in both MAE and MSE. The adaptive collocation sampling and adaptive weight tuning the reviewer mentions are valuable directions and are noted as future work.

2 — 11-bus system is suitable case study for demonstrating new ideas, it will be interesting to see future work with large-scale systems. Discussions on current limitations and potential solution approaches in these realistic settings could strenghten the conceptual and forward looking part of the paper. Scaling is a secondary goal. As the integration does not yet converge even on this simplified system, large-scale deployment is premature: the current bottleneck is surrogate accuracy, not system size, so the scaling question is secondary at this stage. Once accuracy permits convergence, scaling follows from the modular structure. We treat this as future work.

3 — PINNs represent implicit solvers that are typically trained for specific initial and boundary conditions. While possible to train parametric PINNs that generalize over multiple initial conditions this would render the PINN problem extremely high dimensional and not very practical. To tackle this limitation, the scientific machine learning community have developed an idea of physics-informed neural operators (PINO) that allow to train solution operators in infinite dimensional spaces, following similar learning principles to PINNs. I think the future work in this space of neural surrogates for physical systems should focus on the use of PINOs instead of PINNs. We appreciate this suggestion and agree that operator-learning is a potential alternative to PINNs. We'd add two points of nuance for clarity. First, we view a PINN not as a solver in the algorithmic sense, but as a variational state approximator: a PINN parametrises the solution field of a single problem instance by minimising an ODE residual, without an iterative state-update or integration map at inference. So the "implicit solver" label slightly blurs the line between amortising a solve into an optimisation and the structure of a numerical scheme. Second, on PINOs being "infinite-dimensional": the operator being approximated does map between function spaces, but the learned model itself is finitely parametrised (e.g. a fixed spectral truncation in FNO-based PINO, or a predetermined input-output distribution with DeepONet), with accuracy bounded by the training distribution and resolution band rather than by any infinite-dimensional computation. From our perspective, then, the key distinction is less "solver versus operator" and more single-instance approximation versus learning a map across a family of conditions, which we believe our current framing already captures.

V. Brief summary of changes made to the final paper

Figure 1 was improved for clarity. Further, details about the changes of the NN training process were added. Lastly, minor changes in the wordings were made to address the reviewers' comments.