Physics-Informed Machine Learning for Power System Dynamics: A Framework Incorporating Trustworthiness

Petros Ellinas*, Ioannis Karampinis, Ignasi Ventura Nadal, Rahul Nellikkath,
Johanna Vorwerk, and Spyros Chatzivasileiadis
Department of Wind and Energy Systems, Technical University of Denmark, Lyngby, Denmark
*petrel@dtu.dk

***Abstract*—In this paper, we introduce a framework for trustworthy physics-informed machine learning (PIML) applied to power system dynamics, emphasizing transparency, interpretability, and explainability of Neural Network (NN) models for safety-critical applications. With the massive deployment of converter-interfaced resources and the growing uncertainty in both generation and demand, there is an urgent need for computationally efficient models that can reliably approximate complex dynamic behaviors. By embedding physical laws within machine learning models, physics-informed NNs (PINNs) offer the potential for faster, more accurate dynamic simulations. We present model reduction and validation methods, introducing novel correctness verification techniques and an interpretability-driven architecture, Kolmogorov-Arnold Networks (KANs), to enhance trust in NN approximations. Further, we put forward PINNSim, an integrated simulation tool leveraging PINNs for large time-step dynamic simulations, demonstrating its performance on reduced-order synchronous machine (SM) models. This work contributes to the design of transparent, interpretable, and rigorously validated PIML frameworks to accelerate the deployment of reliable AI tools in power systems.**

***Index Terms*—Physics-informed machine learning, power system dynamics, Neural Networks, trustworthiness, transparency, interpretability, correctness verification, Kolmogorov-Arnold Networks (KANs), PINNSim, safety-critical applications**

I. Introduction

Current advancements in computational science have established Neural Networks (NN) as an integral tool to approximate complex mathematical functions and handle large datasets. For high-dimensional problems NNs drastically enhance calculation speed compared to classical methods without a substantial loss of accuracy [1]. Applications with massive amounts of repetitive calculations particularly benefit from these advancements. Thus, NNs enable real-time analysis in many technological, scientific, and other applications where it was previously impossible while providing insights into formerly intractable problems [2, 3].

Such fast computation capabilities might solve several challenges in the power systems domain. The increasing amount of renewable generation and the simultaneously ongoing electrification of the heating and transport sectors currently put power systems worldwide at risk. The volatility, intermittency, and limited predictability of these resources raise the need for real-time operational decision making including stability assessment and fast simulation capabilities. Such tools, based on Neural Networks (NNs), would permit real-time simulation and control of massive amounts of flexible resources, thereby fostering secure system operation in the future. Classic dynamic simulation tools often face extended simulation times, consume substantial computational resources, and are particularly unfit for the simulation of large multi-layer power systems with millions of controllable units [4]. Thus, in alignment with [5], new approaches are needed to improve dynamic simulation capabilities and computational efficiency.

Recent research presents different machine-learning (ML) tools as promising alternatives to address various of the current challenges faced in power systems. Literature reviews are provided in [6, 7]. While ML is widely used in load and generation forecasting, another class of learning methods appears useful in the power system domain: Physics-informed methods integrate fundamental physical laws and constraints directly into the training of ML tools such that their output aligns with the established physical principles. Their effectiveness has been demonstrated across many domains in power systems, including parameter and state estimation [8], transient stability analysis [9], and optimal power flow [10]. The potential advantages of employing physics-informed machine learning (PIML) for time-domain simulations of power systems are established in [11, 12] by employing physics-informed neural networks (PINNs) for approximating Ordinary Differential Equations (ODEs). In particular, [11, 12] demonstrate that PINNs are capable of solving ODEs two to three orders of magnitude faster than conventional methods. Furthermore, [13–15] demonstrate PINNs' superiority in approximating systems of Differential Algebraic Equations (DAEs) [13, 15]. However, the dimension of the presented studies is limited to low-order models and single machines. For complete time-domain simulations of realistic power systems, capturing high-order models, i.e. for renewable generation, and the handling of system-wide simulations are essential.

Even though different approaches to conducting ML-supported system-level time-domain simulations of power systems have been introduced in the literature, it is a growing research field. Generally, the usability of NNs for time-domain simulations of an entire system is limited since they need to capture changing power system topologies, varying

This work was supported by the ERC Starting Grant VeriPhIED, funded by the European Research Council, Grant Agreement 949899.

operating set-points, and different disturbance types accurately. Attempting to include all possible conditions and faults during the learning phase of a *single* ML model leads to an intractable problem, as the required dataset becomes prohibitively high-dimensional [15].

Thus, a more promising option is to decompose the process by breaking down the learning task into smaller, more manageable ones. Ref. [14,16,17] enable computational feasibility by capturing the dynamics of individual components in smaller-scale NNs and training them independently. While [16, 17] present approaches for developing dynamic equivalents of individual power system components using neural ODEs and integrating these ML-based surrogates into traditional time-domain simulation tools and neural differential-algebraic equation solvers, their reliance on neural ODEs may necessitate multiple forward passes to predict over larger time steps. In contrast, [14] proposes a *complete* PINN-based time-domain simulation tool, called PINNSim, by employing a root-finding algorithm to account for the interaction of individual dynamic components and the grid. Speed-ups are achieved in both approaches by capturing complex dynamics in computationally more efficient ML-based surrogates and, in the case of the PINNSim in [14], by increasing the time step of the simulation due to smaller errors per time step. However, the scalability of both approaches remains to be tested.

Despite the notable achievements and potential benefits of ML, one important aspect that hinders its adoption in safety-critical domains remains: such domains, including the power system community and industry, are skeptical of trusting AI solutions. Enforced by recent stipulations of the AI Act [18] and other movements toward regulating AI development, the research community is urged to establish principles to assess transparency, exactness, interpretability, and explainability [19]. Currently, most of the terms are not fully or only generally defined and thus may not fully align with the specifics of the power systems domain. Therefore, suitable definitions, requirements, and evaluation metrics must be established.

To enhance the trustworthiness of ML in power systems and assess their performance, there has been a substantial effort in the literature to verify their behavior by obtaining worst-case performance guarantees. For example, [20] probably reports the first attempt to determine the worst-case violations of a NN trained to solve the DC Optimal Power Flow problem; it did that by converting the RELU-based NN to a Mixed Integer Linear Problem (MILP). [21] improved substantially the scalability of this algorithm by accelerating the verification process through the general-purpose NN verification algorithm $\alpha\beta$-CROWN [22]. Ref. [23] moved a step further and achieved remarkable speedups by tailoring the algorithm and using the GPUs instead of the CPUs. Nonetheless, all these efforts focus on providing guarantees for NNs that approximate the solution of an optimization problem. To the best of our knowledge, approaches to enhance the trustworthiness of PINNs for time-domain simulations is still a research gap.

Addressing the need for fast time-domain simulations and the aforementioned research gaps, we present a comprehensive framework for ML-assisted dynamic simulations that also provides performance guarantees. While parts of the framework have already appeared in the literature, our goal with this work is to present for the first time a *complete* framework. Our key purpose is not to conduct a literature review but to provide a comprehensive introduction of all steps that are required, in our opinion, to step towards ML-assisted time-domain simulations of real power systems. As such, in this work we aim to include all details necessary for readers with a power system background to delve into the field of ML-assisted time-domain simulations. The framework comprises three main components: *Training of PIML tools*, their *Verification*, and *PIML-Based Dynamic Simulations*. The contributions of the presented work are as follows:

- We demonstrate the training procedure of two different NN types. First, we train (i) We demonstrate that Physics-Informed Neural Networks (PINNs) can model the dynamic behavior of Synchronous Machines (SMs) in two ways: using simplified (reduced-order) models and detailed (full-order) models. This work provides the first evidence that PINNs are capable of accurately capturing the complex dynamics of full-order SM models. To address the interpretability challenges associated with vanilla PINNs, we also design and implement the recently emerging (ii) Kolmogorov-Arnold Networks (KANs) [24, 25] that permit smaller architectures without losing in accuracy. All presented PIML tools are benchmarked against classic ODE solutions.
- We introduce and demonstrate novel correctness verification techniques to rigorously validate the trained PIML tools, ensuring their reliability and robustness before deploying them in a safety-critical environment.
- We demonstrate, assess, and compare the integration of PIML tools into existing numerical ODE solvers, and present the performance of PINNSim, the PINN-based solver developed in [14].

The remainder of this paper is structured as follows: Section II introduces the suggested PIML-based framework for fast dynamic simulations of power systems. Then, Section III introduces and demonstrates the training of appropriate PIML tools for different order SM models to arrive at suitable dynamic equivalents. Section IV presents the *Verification* process, and mathematically assesses the trained PIML tools to ensure they meet specified risk tolerances. In Section Section V, we integrate the verified PIML tools into *AI-Based Dynamic Simulations*, demonstrating their usability for both conventional and ML-based solvers. Finally, Section VI provides insights into the remaining research challenges to enable the adoption of the framework for large-scale time-domain simulations, and Section VII concludes the work.

II. Introduction to the PIML-based Dynamic Simulation Framework

Due to the increasing computational complexity, classic time-domain tools for power system dynamic simulations face significant challenges. While the climate crisis mandates the electrification of the heating and transport sector through increasing renewable generation assets, the number of units required for power system dynamic analyses drastically expands. Furthermore, many of these units connect in the

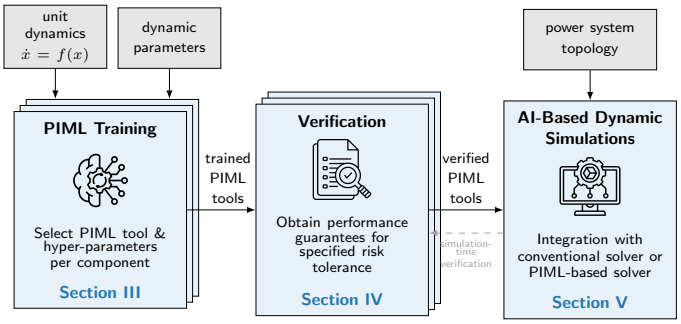

Fig. 1. Overview of the presented framework for AI-assisted dynamic simulations.

distribution layers that were previously neglected in stability studies. In classic time-domain simulations, each of these units is represented by a set of differential-algebraic equations (DAEs). Since no analytical solution exists for these large DAE systems, numerical integration methods are employed to iteratively compute the time-domain trajectory of the system response when subjected to a pre-defined contingency. Parallelization of the solution of each single system component and decoupling it from the grid interactions can result in significant fasten of these methods [26]. However, these speed-ups are limited, and the complexity of the simulation remains.

Physics-informed machine learning (PIML) could offer an alternative by enabling the development of high-performance dynamic models that incorporate the governing physics in the learning process. As such, the loss terms minimized during PIML training consider the system's DAEs. Two different potential pathways are possible to replace and support classic time-domain simulation tools with PIML: (i) training a *single* PIML tool that represents the entire system, or (ii) developing *multiple* PIML equivalents that each present one unit or region of the physical system. While in the first approach, the PIML tool directly outputs the system response, the second method is similar to traditional time-domain simulators that require connecting the various PIML models to obtain a system response.

Developing PIML tools capable of accurately capturing the DAE representation of the *entire* system presents significant challenges. Ensuring consistent model performance across varying grid configurations is particularly difficult. Furthermore, ML models often struggle to generalize to unforeseen conditions or faults, and training on every possible scenario would require an impractically large dataset, leading to excessive complexity. In addition, [14] shows that capturing the whole system with a *single* dynamic equivalent is not scalable.

To address these challenges, we suggest a modular approach allowing for a Plug'n'Play manner that learns separate PIML equivalents for each system component and considers the network structure and component interaction in a subsequent step. With this, the learning task is parallelized. Figure 1 provides an overview of the suggested framework. First, a PIML tool, e.g. a physics-informed neural network (PINN), is selected per system component and trained using the unit dynamics through its DAE representation and the dynamic parameters. These PIML-based dynamic equivalents may represent a single generator in the system. Still, they could also capture more complex dynamics of wind farms or active distribution systems with flexible resources. Since power systems are safety-critical, the second step of the framework uses verification algorithms to obtain rigorous performance guarantees for each trained PIML-based dynamic equivalent. Through verification, mathematical guarantees are established that compute the maximum error of the PIML output compared to the DAE solution. Once each system component's specified risk and error tolerance is calculated, the final step is the implementation of AI-supported dynamic simulations. This step must consider the grid architecture and provide the component interaction.

Since a modular approach is selected, and PIML-based dynamic equivalents are obtained for each system component, two distinct solutions for performing time-domain simulations are possible. On the one hand, classic dynamic simulators could integrate the derived PIML tools. When the PIML represents a unit with complex dynamics, e.g. a wind park, simulation times of the classic DAE simulation can significantly improve. On the other hand, each component in the system could be replaced by a PIML-based dynamic equivalent, resulting in a PIML-based dynamic simulator like the one suggested in [14].

III. Training of PIML-Based Dynamic Equivalents

The first essential step of the presented framework includes the training of NNs that capture the complex dynamics of each system component individually. As such, this Section demonstrates the training of ML-based dynamic equivalents for the Synchronous Machines (SMs), which are probably the most well-studied power system component. We first provide an overview of the classic NN training, and then the introduce the recently appearing Kolmogorov-Arnold Networks (KANs) which are able to achieve higher interpretability and smaller Network architectures. Finally we present the mathematical foundation of Physics-Informed NN training for dynamical systems. We test the different training procedures and architectures on SM models of varying complexity, starting from a 2nd-order model up to a 6th order.

At this point, let us first define notation which applies for the remainder of the paper: Scalars are represented by ordinary small and capital letters x. In physical models, small letters denote a quantity in per unit and capital letters denote physical units. Vectors are represented with boldface small letters $\mathbf{x}$, while matrices are denoted by bold capital letters $\mathbf{X}$. The sets are denoted as $\mathcal{X}$. Time-derivatives are represented through the dot notation, where $\dot{x}$ denotes the first-order time derivative of x. Predictions of ML-tools are denoted with the hat notation $\hat{\mathbf{y}}$.

A. Neural Networks

NNs are computational models consisting of multiple interconnected layers filled with neurons (also called nodes) that process input data to generate an output. The primary objective of a NN is to learn through data a mapping function f, which transforms input vectors $\mathbf{x}$ into desired output vectors $\mathbf{y}$. In the context of regression tasks, NNs predict continuous values. Given an input vector $\boldsymbol{x}$, the network produces a continuous output $\hat{\mathbf{y}}$ through a series of matrix multiplications. For a

simple NN with one hidden layer and an output layer, this mapping is:

$$\hat{\mathbf{y}} = f(\mathbf{W}_2 \cdot g(\mathbf{W}_1 \cdot \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) \tag{1}$$

Here, $\mathbf{W}_1$ and $\mathbf{W}_2$ represent the weight matrices; $\mathbf{W}_1$ connects the input layer to the hidden layer and $\mathbf{W}_2$ connects the hidden layer to the output layer. The vectors $\mathbf{b}_1$ and $\mathbf{b}_2$ are the bias terms for the hidden and output layers. The function g denotes the activation function of the hidden layer, which introduces non-linearity into the model, enabling it to capture complex patterns in the data. Common activation functions, among others, include the rectified linear unit (ReLU): $g(x) = \max(0, x)$; and the hyperbolic tangent (tanh): $g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

The training process of a NN for regression involves four steps: forward propagation, loss computation, back-propagation, and weight updates. During forward propagation, the input $\mathbf{x}$ is passed through the network to obtain the predicted output $\hat{\mathbf{y}}$. The difference between the predicted output and the true output $\hat{\mathbf{y}}$ is quantified by a loss function $\mathcal{L}$. For regression tasks, the Mean Squared Error (MSE) is commonly used.

The goal is to minimize this loss by adjusting the network's weights and biases through backpropagation and gradient descent. As such, backpropagation computes the gradients of the loss function with respect to each weight $\theta \in \mathbf{W}$ and bias $b \in \mathbf{B}$, which are then updated through the gradient descent algorithm according to the learning rate η:

$$\mathbf{W} = \mathbf{W} - \eta \cdot \nabla_{\mathbf{W}} \mathcal{L} \tag{2}$$

$$\mathbf{b} = \mathbf{b} - \eta \cdot \nabla_{\mathbf{b}} \mathcal{L} \tag{3}$$

where $\nabla_{\mathbf{W}} \mathcal{L}$ and $\nabla_{\mathbf{b}} \mathcal{L}$ are the gradients of the loss $\mathcal{L}$ with respect to the weights and biases, respectively. By iterating over many epochs, with the objective to minimize the loss function, the NN learns to predict continuous outcomes with a high degree of accuracy.

The NN architecture and several other parameters have to be selected by the user during NN training. This has led to the development of standard processes and pipelines to help define the best-suited architecture for a specific task while minimizing computational efforts. As such, the goal is to find the smallest architecture that delivers the desired accuracy. Besides the architecture design, hyper-parameter tuning includes the comparison and the selection of the most appropriate optimizer, its optimization parameters, e.g. the learning rate η, and the selection of activation functions. This, in combination with the search for the most suitable loss functions, makes the learning problem very difficult.

B. Kolmogorov-Arnold Networks

The more recently developed Kolmogorov-Arnold Networks (KANs), introduced in [24], offer a promising alternative for learning ML-based dynamic equivalents. Unlike vanilla NNs we described in the previous Section, which rely on matrix-based weights and nonlinear, pre-defined activation functions, KANs feature learnable, flexible activation functions, enhancing precision and interpretability while using smaller network structures. Figure 2 highlights the differences

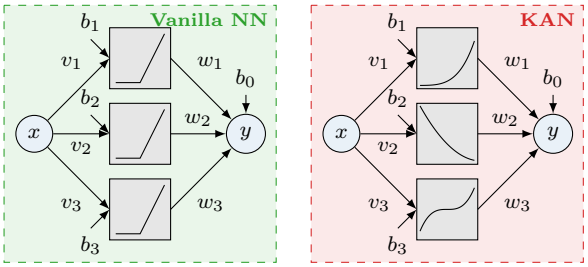

Fig. 2. Architecture of a classic vanilla NN (left) and a KAN (right).

in the architectures. While activation functions in vanilla NNs are the same predefined function in an entire layer, KANs feature learnable activation functions, which results in different activation shapes for each neuron.

KANs use flexible activation functions, so-called B-splines, where each B-spline is parametrized by learnable control points. The result is a network with a more compact and interpretable structure and a flexible way to approximate complex mappings. This is particularly beneficial for tasks requiring high interpretability, and fine control over the shape of the activation functions.

In this section, we will establish the mathematical foundation for KANs before providing insights into KAN architecture.

1) Kolmogorov-Arnold Representation Theorem: The foundation of KANs is a set of unique equations, represented by the Kolmogorov-Arnold Representation Theorem, that defines how these NNs map input data. The theorem states that any multivariate, continuous function f on a bounded domain can be represented as a finite composition of continuous univariate functions and simple addition. As such, the function $f : [0, 1]^n \to \mathbb{R}$ with the input vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ can be reformulated as:

$$f(\boldsymbol{x}) = \sum_{q=1}^{2n+1} \left[\boldsymbol{\Phi}_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right) \right], \tag{4}$$

where $\phi_{q,p} : [0, 1] \to \mathbb{R}$ are the univariate functions, and $\boldsymbol{\Phi}_q : \mathbb{R} \to \mathbb{R}$ represents the final assembly step combining them.

Suppose a supervised learning task with input-output pairs $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}$ and the aim to find a mapping such that $y_i = f(\boldsymbol{x}_i)$. Eq. Eq. (4) states that it is sufficient to find appropriate univariate functions $\boldsymbol{\Phi}_q$ and $\phi_{q,p}$. Since all these functions are univariate, they can be parameterized by unidimensional piecewise polynomials, or B-splines, with learnable coefficients.

2) B-Splines: KANs employ linear combinations of B-splines, or basis splines, to feature learnable activation functions. A B-spline $B(t)$ of order $k + 1$ is a collection of piecewise polynomial basis functions $B_{i,k}(t)$ of degree k in a variable t. The values in t where the polynomials meet are the G control points of the spline and are sorted in non-decreasing order in the knot vector $[t_0, t_1, \ldots, t_G]$.

The B-splines can be constructed by applying the De Boor's recursive algorithm. First, the B-splines of degree $k = 0$ are

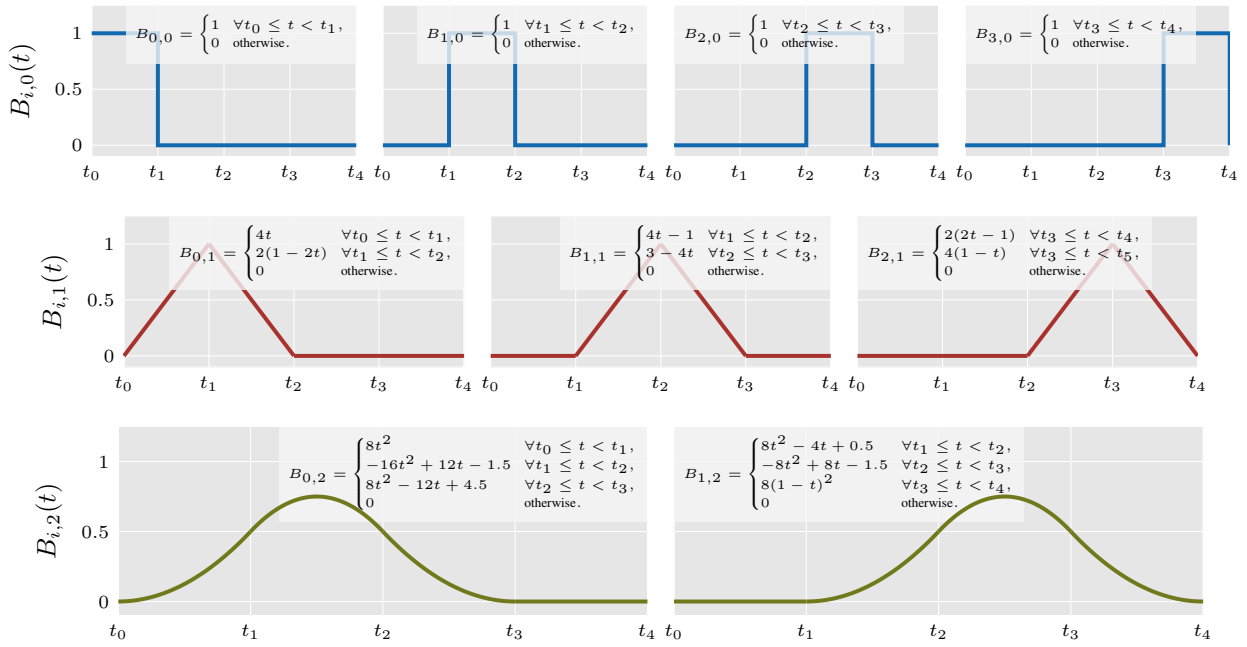

Fig. 3. Example of different order basis functions with a uniform knot vector $\mathbf{t} = [0, 0.25, 0.5, 0.75, 1]$: While 0 order basis functions (top) are constant and non-zero between two knots, first order basis functions (middle) are linear and non-zero across three knots, while third order basis functions (bottom) are quadratic and non-zero across four knots.

piecewise constants that are unity on the interval $[t_i,t_{i+1}]$:

$$B_{i,0}(t) = \begin{cases} 1 & \text{if } t_i \le t < t_{i+1}, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

The higher degree B-splines are defined by recursively evaluating

$$B_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} B_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(t). \tag{6}$$

As such, the B-spline of k-th order is a piecewise linear combination of the B-splines of order $k-1$. Figure 3 illustrates the recursive construction of second-order basis functions for a uniform knot vector with five knots $\mathbf{t} = [0, 0.25, 0.5, 0.75, 1]$. Applying Eq. (5), the first-order splines shown in the top row are unity between two knots. Hence, there are four zero-order splines for a knot vector with five elements. The middle row depicts the first-order splines that are obtained from applying Eq. (6). For example, when constructing $B_{0,1}(t)$, $i = 0$, $k = 1$ and thus it is obtained as

$$\begin{aligned} B_{0,1}(t) &= \frac{t - t_0}{t_1 - t_0} B_{0,0}(t) + \frac{t_2 - t}{t_2 - t_1} B_{1,0}(t) \\ &= 4t\, B_{0,0}(t) + (2 - 4t)\, B_{1,0}(t). \end{aligned} \tag{7}$$

Since $B_{0,0}(t)$ is only non-zero between $[t_0, t_1]$ and $B_{1,0}(t)$ is only non-zero between $[t_1, t_2]$, the resulting first-order spline consists of two linear segments.

Finally, the second-order splines are obtained by applying the same recursive scheme from Eq. (6). When obtaining $B_{0,2}$, $i = 0$ and $k = 2$, thus:

$$\begin{aligned} B_{0,2}(t) &= \frac{t - t_0}{t_2 - t_0} B_{0,1}(t) + \frac{t_3 - t}{t_3 - t_1} B_{1,1}(t) \\ &= 2t\, B_{0,1}(t) + (1.5 - 2t)\, B_{1,1}(t). \end{aligned} \tag{8}$$

Since the first-order splines are linear across two consecutive segments each, the second-order spline is quadratic across three segments.

The flexible activation functions $B(t)$ in KANs exploit this recursive scheme and utilize a weighted sum of the highest-order B-splines:

$$B(t) = \sum_{i=1}^{G-k-1} c_i B_{i,k} \tag{9}$$

where c_i represent the weights and is described by $G - k - 1$ piecewise polynomial functions of order $k - 1$.

The number and spacing of knots and the order of B-splines employed significantly impact the flexibility of KANs and need to be considered during hyper-parameter tuning. Figure 4 provides the final spline (top) and second-order basis splines for two different knot vectors. In the middle, a uniform grid with multiplied edge points is selected, so that it contains $G = 5$ elements. Hence, when applying the De Boor algorithm, $G - k - 1 = 6$ second-order B-splines are obtained. In contrast, the bottom case, employs a non-uniform grid with $G = 10$ elements and multiplied control points at the edges and in the middle of the interval. It results in $G - k - 1 = 7$ second-order B-splines. Consequently, the final spline is constructed with a different number of B-splines for the two examples, where they have different shapes due to the differently selected control point locations. For the smaller grid with $G = 6$, six B-splines are linearly combined, while for the larger grid, the spline is represented by the weighted linear combination of seven B-splines.

3) KAN Architecture: To obtain multi-layer KANs, a KAN layer with input dimension n_{in} and output dimension n_{out} is

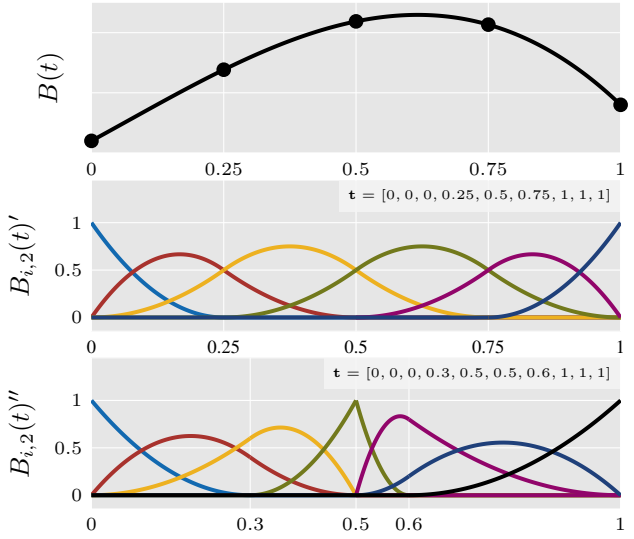

Fig. 4. Example of a B-spline (top) constructed with the weighted sum of second-order basis functions: The middle example uses uniformly spread knots with multiplied control points at the edges, while the bottom example uses a non-uniform grid with several multiplied control points.

defined as a matrix of unidimensional functions:

$$\mathbf{\Phi} = \{\phi_{q,p}\}, \quad p = 1, \ldots, n_{\text{in}}, \quad q = 1, \ldots, n_{\text{out}}, \tag{10}$$

with Φ consisting of $n_{\text{in}} \cdot n_{\text{out}}$ functions. The pre-activation value $x_{l+1,j}$ of the $(l+1, j)$-th neuron computed as:

$$x_{l+1,j} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}) \tag{11}$$

where l and j is the layer and neuron index, respectively. Each activation function $\phi(x)$ is represented by a unique B-spline curve that is learned during training:

$$\phi(x) = \boldsymbol{w}\, b(\boldsymbol{x}) + \boldsymbol{\alpha}\, \text{base}(\boldsymbol{x}), \tag{12}$$

where base(x) is a pre-defined base activation function, similar to the ones from classic NN training, and $b(\boldsymbol{x})$ is the weighted sum of B-splines:

$$b(\boldsymbol{x}) = \boldsymbol{c}\, \boldsymbol{B}(\boldsymbol{x}). \tag{13}$$

Throughout the training process, the parameters $\boldsymbol{c}$, $\boldsymbol{\alpha}$, and $\boldsymbol{w}$ are optimized. The entirety of trainable parameters within KANs, consisting of L layers each containing N neurons, amounts to N^2GL.

4) Training KANs: Training KANs can be more challenging than training vanilla NNs. This is primarily due to the enhanced ability of KANs to capture high-frequency functions [27]. While this property makes KANs more expressive and capable of achieving better performance with fewer parameters compared to vanilla NNs, they are more prone to overfitting. As a result, the choice of hyperparameters, such as learning rate, regularization, and network architecture, become critical to ensure a balanced and robust model performance.

KANs train a separate spline for each neuron. As such, KAN training does not just adjust the weights for the B-splines but also modifies the configuration of the activation function within each neuron. This leads to drastically more compact architectures. Equally importantly, KANs can determine approximate closed-form solutions of the function they are trained to map. Considering that, in the general case, this function is unknown (mapping an unknown function from available input/output data is exactly one of the key motivations to use NNs in the first place), obtaining a tractable closed-form solution is one of the key strengths of KANs, which has not been possible before. This approximate closed-form relationship between the network's inputs and outputs is obtained by fitting the activation functions of KANs to known mathematical functions.

C. Mathematical Foundation of Physics-Informed Learning

Since NN training requires large datasets, obtaining time-domain trajectories to train vanilla NNs or KANs is computationally expensive, as also shown in [28]. This can often lead to a lack of sufficient training data, and, as a result, the obtained ML models may lack generalization capabilities as they fail to learn the actual underlying physical function effectively [29]. These are obstacles that Physics-Informed learning can overcome by embedding physical laws directly into the training process.

The work in [29, 30] demonstrates the effectiveness of integrating structural information of the problem, such as physical laws, into ML algorithms. This is accomplished by merging the governing differential equations into the loss function, which ensures that the final ML model satisfies the underlying physics [31]. Thus, by transforming the problem of solving differential equations into an optimization task, ML models minimize the discrepancy between their predictions and the physical laws.

PIML models are capable of approximating multi-dimensional partial differential equations (PDEs) [31]. However, in the power systems domain, dynamic simulations require the solution of ODE systems, formed as Initial Value Problems. Therefore, in this work, we present a tailored physics-informed learning formulation for approximating the flow of initial value problems, for Power System Problems.

1) Initial Value Problem: An Initial Value Problem (IVP) is a type of ODE that seeks a function $u(t)$ satisfying both a differential equation and an initial condition. Formally, it is defined as:

$$\dot{\mathbf{u}} = f(\mathbf{u}, t), \quad \mathbf{u}(t_0) = \mathbf{u}_0, \tag{14}$$

where $f(\mathbf{u}, t)$ is a given function, t_0 is the initial time, and $\mathbf{u}_0$ is the initial value of the function at t_0. The goal is to find a function $\mathbf{u}(t)$ that not only solves the differential equation but also fulfills the initial condition $(t_0, \mathbf{u}_0)$, and thus provides the time-domain response of the system.

The *flow* refers to the evolution of the system's state over time for various initial conditions, as highlighted in Fig. 5. The flow, represented by the colored lines, maps an initial state $\mathbf{u}_0$, denoted by the dots in the initial set, to its state at a future time t, denoted by the x in the solution space. This mapping encapsulates how the system transitions from its initial state to subsequent states over time.

The Picard-Lindelöf theorem [32] provides a fundamental proof for IVPs, stating that a unique solution $\mathbf{u}(t)$ exists around t_0, if the function $f(\mathbf{u}, t)$ is continuous in t and

Lipschitz continuous in $\mathbf{u}$ within a neighborhood around the initial condition.

$$\mathbf{u}(t) = \mathbf{u}_0 + \int_{t_0}^{t} f(\mathbf{u}(\tau), \tau)\, d\tau. \tag{15}$$

The conventional ODE solvers approximate the solution of the ODEs, based on this theorem. Specifically, they approximate $\mathbf{u}(t)$ by discretizing the time interval $[t_0, t]$ and evaluating the integral in a stepwise iterative manner. In the next section, we introduce the formulation of the Physics-Informed learning process, with which NNs can learn the flow of the IVP, providing a faster method to calculate the solution of the IVP.

2) Physics-Informed Loss Terms: Given that the flow, and consequently the solution of an IVP, is unique, additional loss terms may be considered during PINN training. In the general PDE case, such additional terms might introduce significant bias in the learning process. However, that is not the case for well-posed IVPs.

Therefore, we can use simulated data to guide the optimization process. Adhering to the work published in [11], the complete loss function for well-posed IVPs includes two additional loss components:

$$\mathcal{L} = \lambda_{\mathrm{p}}\mathcal{L}_{\text{physics}} + \lambda_{\mathrm{b}}\mathcal{L}_{\text{boundary}} + \lambda_{\mathrm{d}}\mathcal{L}_{\text{data}} + \lambda_{\text{rhs}}\mathcal{L}_{\text{rhs}}, \tag{16}$$

where $\mathcal{L}$ is the overall loss function, $\mathcal{L}_{\text{physics}}$ enforces compliance with the governing differential equations and $\mathcal{L}_{\text{boundary}}$ ensures that boundary and initial conditions are satisfied. These loss terms are evaluated for specified collocation points and transform the continuous differential equations into a discrete set of residuals that are minimized during PIML model training. Moreover, $\mathcal{L}_{\text{data}}$ permits the inclusion of data points from measurements or time-domain simulations, $\mathcal{L}_{\text{rhs}}$ is a loss that acts as an intermediate step between data loss and physics loss and it helps improve the generalization performance [11]. The physics and boundary loss are included according to Eq. (17) and Eq. (14). Each loss component is considered with a respective weight, denoted by λ, that is adjusted during the hyper-parameter tuning and permits to optimize performance.

Consequently, we define the physics-informed loss as the summation of the residuals of the ODE at a selected set of collocation points $\{(\mathbf{x}_f^{(i)}, t_f^{(i)})\}_{i=1}^{N_f}$:

$$\mathcal{L}_{\text{physics}} = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|\mathcal{R}_\theta(\mathbf{x}_f^{(i)}, t_f^{(i)})\right|^2, \tag{17}$$

where N_f is the number of collocation points. The residual $\mathcal{R}_\theta$ is the difference of the derivative of the NN output and the evaluation of the source term. Since the true solution $\mathbf{u}(\mathbf{x}, t)$ is not known, the NN output $\hat{\mathbf{y}}(\mathbf{x}, t)$ is used to evaluate the source term at the collocation point:

$$\mathcal{R}_\theta(\mathbf{x}, t) = \underbrace{\frac{\partial \hat{\mathbf{y}}(\mathbf{x}, \mathbf{t})}{\partial t}}_{\text{Derivative of NN output}} - \underbrace{f(\hat{\mathbf{y}}(\mathbf{x}, t), t)}_{\text{Source term evaluation}}. \tag{18}$$

and the boundary loss obtained from Eq. (14):

$$\mathcal{L}_{\text{boundary}} = \frac{1}{N_b}\sum_{i=1}^{N_b}\left|\hat{\mathbf{y}}(\mathbf{u}_0, 0) - \mathbf{u}_0\right|^2, \tag{19}$$

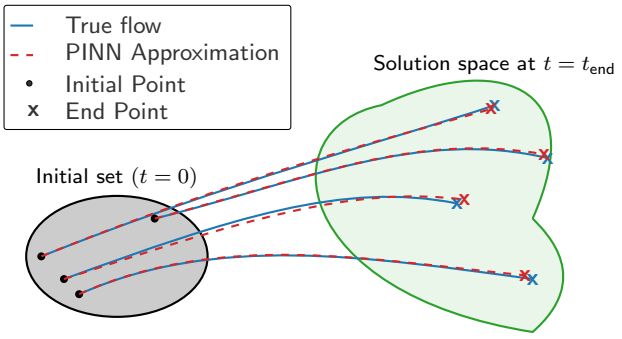

Fig. 5. The flow of initial conditions ($t = 0$) to a solution space ($t = t_{end}$) of an ODE system for different initial conditions. The transparent shapes denote the initial (gray) and reachable sets (green).

where N_b is the number of distinct initial conditions of the collocation points considered.

By minimizing the loss function, PIML models are capable of learning physics-driven behavior while satisfying the underlying ODE and adhering to the ODE boundary conditions without the need for any dynamic trajectories or measurements during training. As such, PIML models are promising candidates to accurately learn and capture the dynamic behavior of power system components by effectively approximating the underlying ODEs. We highlight, that a key strength of PIML models is their ability to capture the entire *flow* of a dynamical system rather than just a single solution trajectory.

By providing $\mathbf{u}_0$ as an input, the NN effectively learns a family of solutions corresponding to a set of initial conditions. Thus, the PIML model can serve as a surrogate model for the flow, providing approximate solutions of the system's evolution over time for any initial state $\mathbf{u}_0$, as illustrated in Fig. 5.

For clarity, we define the set $\mathcal{U}$ as the collection of all reachable states $(\mathbf{u}, t)$ of the system. This set includes all pairs $(\mathbf{u}, t)$, where $\mathbf{u}$ represents the state variables at time t. The evolution of the system starts from the initial set of states $\mathcal{U}_0$, which contains the initial conditions $\mathbf{u}_0 \in \mathcal{U}_0$, and progresses according to the system dynamics.

Besides the different loss functions, the PIML models' training process is quite similar to that of vanilla NNs. Generally, PIML models with differentiable activation functions, e.g. $\tanh$, have been reported to achieve higher accuracy than PIML models with partially differentiable activation functions like ReLU activation [33, 34].

The data loss is obtained for a number of data points N_d:

$$\mathcal{L}_{\text{data}} = \frac{1}{N_d}\sum_{i=1}^{N_d}\left|\hat{\mathbf{y}}(\mathbf{u}_0^{(i)}, t_d^{(i)}) - \mathbf{u}^{(i)}\right|^2, \tag{20}$$

where $\{(\mathbf{u}_0^{(i)}, t_d^{(i)}), \mathbf{u}^{(i)}\}_{i=1}^{N_d}$ are the data points, created by a simulator or a measurement. Note that $N_d = N_t N_{d_{ic}}$, with $N_{d_{ic}}$ as the number of initial conditions per trajectory, and N_t as the uniformly sampled time points for each. The fourth loss component, namely $\mathcal{L}_{\text{rhs}}$, was established in [11] to enhance the NN's capability in approximating the underlying solution. This component acts as an intermediate stage bridging the

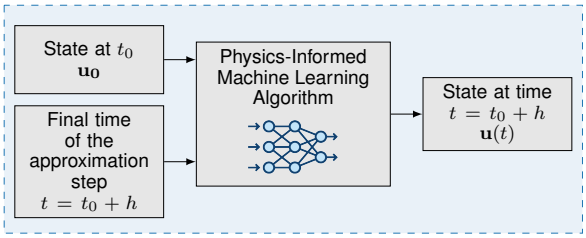

Fig. 6. Schematic of a physics-informed machine learning algorithm for approximating the state $\mathbf{u}(t)$ at time $t = t_0 + h$ based on the initial state $\mathbf{u}_0$ and the time t.

vanilla NN training and the training of physics-informed NNs. Thus, we define it as:

$$\mathcal{L}_{\text{rhs}} = \frac{1}{N_d} \sum_{i=1}^{N_d} \left| \frac{\partial \hat{\mathbf{y}}^{(i)}(\mathbf{u}_0^{(i)}, t_d^{(i)})}{\partial t} - f(\mathbf{u}^{(i)}, t) \right|^2, \tag{21}$$

where $\mathbf{u}^{(i)}$ comes from the dataset $\{(\mathbf{u}_0^{(i)}, t_d^{(i)}), \mathbf{u}^{(i)}\}_{i=1}^{N_d}$ of simulated trajectories.

It is worth mentioning that the points of physical loss and data loss evaluation do not necessarily coincide, where PINN training actually does not require any observable data input [35]. Therefore, a comparative study must be made to clarify the best approach to go.

The number of collocation points needed, depends on the stiffeness and smoothness of the solution.

D. Comparative Performance Analysis

This Section illustrates the effectiveness of the previously introduced Physics-based learning methods for deriving dynamic equivalents of power system components. To demonstrate the efficacy, this Section compares PINNs and PI-KANs trained to represent Synchronous Machine (SM) models of different complexity (i.e. ranging from 2nd order to 6th order models) and benchmarks them against the ODE solutions. SMs are selected as they are fundamental in analyzing the dynamic behavior of power systems, and are simpler and more well understood than converter-interfaced generation. As such, they allow for a more straightforward and insightful comparison. This Section first lists the SM models considered for the analysis and discusses the implementation and training of PINNs and PI-KANs. Afterwards, we define evaluation metrics and benchmark the performance of the trained ML-based equivalents against ODE solutions.

1) Synchronous Machine Models: For the comparative performance analysis, PINNs and PI-KANs are trained to represent the dynamics of different classic SM models to assess the usability of the different ML tools for various model complexities. To study the accuracy and the potential benefits of using PINNs instead of classic ODE solvers, we examine several test cases that simulate the behavior of SMs under transient conditions. While higher-order models capture intricate dynamics, they pose challenges in terms of computational complexity and analytical tractability.

Each selected scenario permits the performance assessment of PINNs across different levels of model complexity, providing insights into their scalability and effectiveness in solving higher-order ODEs. All SM models are described in detail in [36] but are repeated here for transparency. Specifically, second-, fourth-, and sixth-order SM representations are considered in the analysis. In all scenarios, the SM connects to an infinite bus. The SM parameters are provided in Appendix A in Table III.

2D Scenario: The second-order model, referred to as the $2D$ scenario for the remainder of the work, captures essential rotor dynamics that are governed by the power balance at the SM rotor. As such, it captures the rotor angle δ and frequency dynamics ω through:

$$\dot{\delta} = \omega, \tag{22a}$$

$$\dot{\omega} = \frac{\omega_b}{2H} \left(P_m - P_e - D\omega \right), \tag{22b}$$

where ω_b denotes the base frequency, H is the machine inertia, P_m and P_e are the mechanical and electrical power, respectively, and D denotes the damping coefficient. Assuming that the SM connects to an infinite bus, the electric power equals $P_e = V_1 V_2 \sin(\delta)$, where V_1 is the voltage at the machine terminal and V_2 the voltage magnitude at the infinite bus.

4D Scenario: The fourth-order model, referred to as the $4D$ scenario for the remainder of this work, expands the $2D$ scenario by including the transient internal voltage dynamics of the SM in the representations. As such the transient internal voltage dynamics (e'_d,e'_q) are added to the representation in Eq. (22) to arrive at the fourth-order representation:

$$\dot{e'_d} = \frac{1}{T'_{q0}} \left(-e'_d + i_q (x_q - x'_q) \right), \tag{23a}$$

$$\dot{e'_q} = \frac{1}{T'_{d0}} \left(-e'_q + i_d (x_d - x'_d) + e_{fd} \right), \tag{23b}$$

$$P_e = e'_d i_d + e'_q i_q + i_d i_q (x'_q - x'_d) \tag{23c}$$

where T'_{dq0} denotes the transient time constant, $x'_{d,q}$ denotes the transient reactances, i_{dq} represents the stator current, and e_{fd} is the field voltage. The electric power in this case is determined by the internal voltages and the machine current.

6D Scenario: The final use case is the sixth-order representation of an SM, referred to as the $6D$ scenario in the remainder of this work. It captures subtransient dynamics in the simulation by including the subtransient internal voltages (e''_d, e''_q). The following two ODEs are added to the fourth order model in Eq. (23) to arrive at the full 6th order representation:

$$\dot{e''_d} = \frac{1}{T''_{q0}} \left(e'_d - e''_d - i_q (x'_q - x''_q) \right) \tag{24a}$$

$$\dot{e''_q} = \frac{1}{T''_{d0}} \left(e'_q - e''_q - i_d (x'_d - x''_d) \right) \tag{24b}$$

2) NN Training and Implementation: First, two datasets for the training and testing phase of the NNs are obtained:

$$\mathcal{T}_1 = \{(\mathbf{u}_0^{(i)}, t^{(i)}), \mathbf{u}^{(i)}\}_{i=1}^{N_d=500.000} \tag{25a}$$

$$\mathcal{T}_2 = \{(\mathbf{u}_0^{(i)}, t^{(i)})\}_{i=1}^{N_f=500.000} \tag{25b}$$

Obtaining state variable trajectories via an ODE solver is the first step to create the dataset $\mathcal{T}_1$ with the labeled data. The

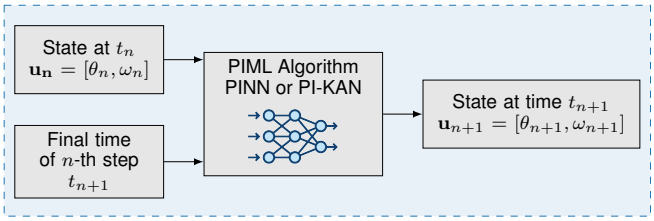

Fig. 7. Input and output of PIML tools for the second order SM presentation

solver was operated with a $1\,\mathrm{ms}$ timestep and a simulation time of $t \in [0\,\mathrm{s}, 1\,\mathrm{s}]$. The $N_{d_{ic}}$ = 500 different initial conditions $\mathbf{u}_0$ that initialized ODE solver solutions were selected through Latin hypercube sampling on the initial condition set $\mathcal{U}_0$. The output is a set of 500 distinct trajectories with N_t=1000 time points. The subsequent processing of that output results in the set $\mathcal{T}_1$ represented in the input-output form $(\mathbf{u}_0, t), \mathbf{u}$, which contains N_d = 500.000 data points.

Set $\mathcal{T}_2$ collects the collocation points, new $N_{f_{ic}}$ = 500 initial conditions $\mathbf{u}_0$ were sampled with Latin hypercube sampling from the set $\mathcal{U}_0$. These initial conditions were then combined with the same N_t= 1000 time points used in $\mathcal{T}_1$ resulting in inputs of the form $(\mathbf{u}_0, t)$, which is also the final form of $\mathcal{T}_2$. However, for this set we did not have to compute the trajectories.

During training, data points within both sets were selectively neglected within the same trajectory. This contributes to preventing overfitting and helps the NN to generalize to unseen data, rather than closely approximating the given trajectories.

The loss function $\mathcal{L}$ is a weighted sum of four objectives, as established in Eq. (16). Hence, appropriate weights need to be selected for each component. Failing to do so may trap the training process in local minima, consequently leading to suboptimal performance. To achieve an optimal balance among the loss components, we carefully adjusted their contributions, initially assigning a contribution of $\lambda_{data} = 1$ to the data loss, as error in this component could propagate and negatively impact the physics loss. Finally, the empirically selected weights are $\lambda_{boundary} = \lambda_{rhs} = 10^{-3}$ for the boundary and rhs loss, and $\lambda_{physics} = 10^{-4}$ for the physics loss. These values were selected empirically to prioritize the data loss while ensuring balanced contributions from the other terms, leading to robust optimization and improved performance.

In addition to the aforementioned method, we also experimented with two dynamic weighting methods, which adjust the weights of each loss component based on the convergence behavior of each individual term. These methods are grounded in gradient-based attention techniques and Neural Tangent Theory [37]. Our tests suggested that the scaling of the weights was consistent across all three methods, resulting in comparable weights for the different loss components. Therefore, considering that the results are similar across all three weighting methods, for the sake of readability, in the remainder of this Section, we report results obtained only with the first weighting method.

All NNs are implemented in Python with PyTorch [38] using the second-order optimizer LBFGS [39]. Standard PINNs were trained for 600 epochs with tanh activation and an architecture consisting of three hidden layers with 64 neurons each for all SM cases. More information on the software used can be found in [40] and the code used can be found in [41].

In contrast, KANs were trained for 300 epochs to prevent overfitting. The architecture for KANs comprised two hidden layers with 12 neurons each across all scenarios, a grid size $G = 10$ and degree $k = 3$, for all SM cases. We maintained consistent NN and KAN architectures for all test cases to evaluate their performance across the three SM models with rising complexity. This approach allows for a fair comparison and demonstrates the architecture's generalization capability.

The experiments were conducted in the DTU High Performance Computing (HPC) cluster, using an NVIDIA V100 GPU equipped with 16GB of memory. The computational tasks were executed on an 8-core Intel Xeon Processor 2660v3, supported by a system having 12GB of RAM.

3) Evaluation Metrics: We compare the performance of the implemented PINNs and PI-KANs measuring their Mean Squared Error (MSE), their Mean Absolute Error (MAE), and the Maximum Absolute Error (Max AE) to identify the largest absolute error highlighting the worst-case prediction scenario. A subset of the 500 distinct trajectories, comprising the dataset $\mathcal{T}_1$, was used exclusively for the testing phase.

Three additional metrics beyond the mentioned classic error metrics are evaluated to enhance the comparison between the different PINN methods. These include:

- The *number of trainable parameters* as it is reflects model complexity. A limited number may hinder the model's ability to capture underlying dynamics, while too many parameters may hinder interpretability and introduce optimization difficulties,
- the *inference time for a single trajectory* measures the time required to generate a prediction for one trajectory, indicating the model's suitability for real-time applications, and
- the *inference time for 50 trajectories* to assess the scalability and efficiency of the model when processing multiple trajectories simultaneously.

The inference time to obtain time-domain trajectories is measured for both the trained PINNs/PI-KANs and the classic ODE solvers, and are evaluated on the same computer to ensure fairness.

4) Results: This section evaluates the different PIML-based dynamic equivalents for SMs, including PINNs, PI-KANs, and traditional ODE Solvers, revealing distinct strengths and trade-offs among the methods.

Figure 8 presents two randomly selected time-domain trajectories for the $2D$ test case and indicates that both the PINN and the PI-KAN successfully learned the underlying physics. Similar observations were made for the higher-dimensional test cases, and example trajectories are included in the appendix.

Figure 9 depicts the time-domain trajectory of the MSE for the 2nd order, 4th order and 6th order SM models. Note that the ODE solver provided the benchmark or true solution for the error evaluation. Generally, the MSE is highest in the initial phase directly after initialization at $t = 0$, and significantly drops thereafter. While the PI-KAN depicts the lowest error

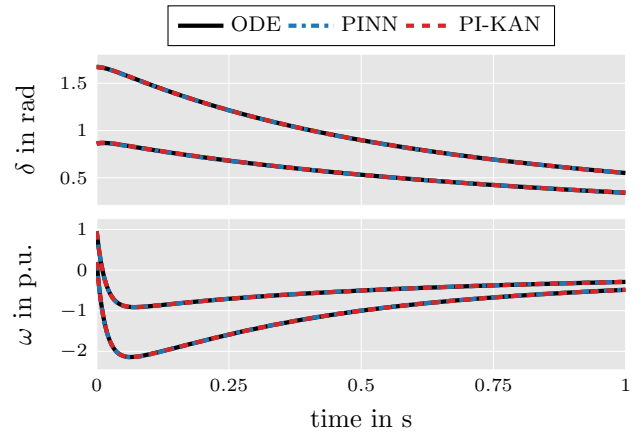

Fig. 8. Two selected time-domain trajectories for the 2D SMIB test case: Results for the ODE-solver, PI-KAN and PINN predictions when trained with the provided collocation points.

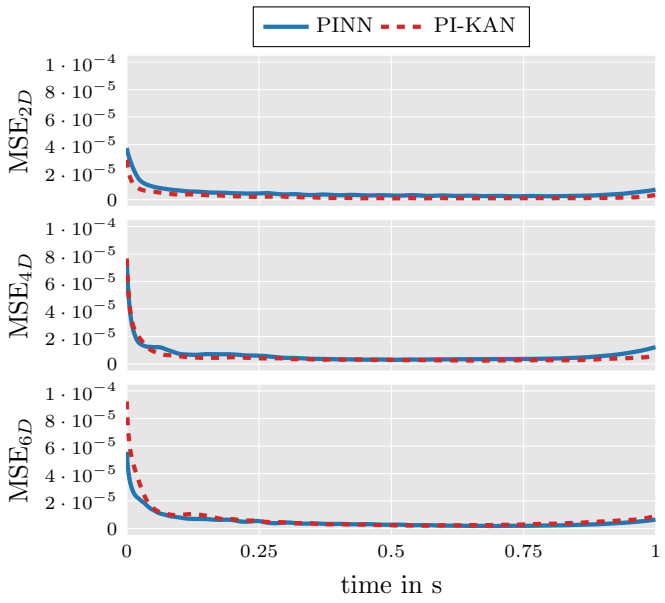

Fig. 9. Performance comparisons of PINN-based NNs and KANs for the three presented SM scenarios.

across the entire trajectory in the $2D$ case, it is subject to slightly larger MSE during the initial transient of the $6D$ case.

Insights into all evaluation metrics are provided in Fig. 10. Again, the errors are evaluated against the ODE solution, hence, no error metrics are included for the ODE solver. The results suggest that KANs consistently achieve lower or comparable MSE and MAE compared to PINNs irrespective of the complexity of the model ($2D$, $4D$, and $6D$). Specifically, the MSE values of KANs range from $3.5\,10^{-6}\mathrm{p\,u}$ to $4.7\,10^{-6}\mathrm{p\,u}$ and MAE scores between $1.3\,10^{-3}\mathrm{p\,u}$ to $1.5\,10^{-3}\mathrm{p\,u}$, outperforming PINNs, which typically exhibit higher error metrics.

In terms of model complexity, KANs are advantageous due to their significantly lower number of trainable parameters, approximately 4800 compared to more than $13\,000$ parameters that are required for ordinary PINNs. This reduction in parameters not only simplifies the training process but also enhances the interpretability of the model.

For single trajectory simulations, KANs experience the longest execution times compared to ordinary PINNs and the ODE solver. However, when scaling to multiple trajectories, PINNs and KANs both significantly outperform the classic ODE solver. While the ODE solver's execution time scales linearly with the number of simulated trajectories, the KANs and PINNs achieve sublinear (and PINNs almost logarithmic) execution. For instance, computing 50 trajectories with a classical ODE solver takes 128-236ms, depending on the model complexity. In contrast, PI-KANs are 4-7 times faster, takes approximately 31 to 35 milliseconds, and PINNs are 68-124 times faster, completing the task in under $2\,\mathrm{ms}$.

Despite the slower execution of PI-KANs compared to PINNs, the balance between model simplicity, interpretability, and accuracy can make KANs a compelling choice for scenarios where understanding the model's internal workings is crucial. The ability to interpret less compact architectures may lead to more informed decisions and facilitate the integration of domain knowledge into the modeling process. Overall, PI-KANs and PINNs present a promising alternative to traditional ODE solvers.

IV. Trustworthiness of PIML-based Dynamic Equivalents

Recent efforts have started to constrain and regulate the uncontrolled use of AI, especially in domains that could lead to significant harm, such as healthcare, transportation, and energy. As an example, the EU AI Act [18] represents a pioneering regulatory framework aimed at governing the development, deployment, and use of AI, particularly in safety-critical systems that are inherently risk avoiding. By setting out strict guidelines for transparency, safety, and human oversight, these regulatory frameworks seek to mitigate the risks associated with complex AI systems.

Consequently, it is crucial to establish the trustworthiness of AI Tools [42], [43] to facilitate their integration into practical applications. Nonetheless, a common understanding for the definition and assessment of these criteria is currently lacking, not only in the power systems domain. Therefore, this Section briefly introduces key concepts, definitions, and evaluation methods in the field of trustworthiness before assessing the PINN- and PI-KAN-based dynamic equivalents developed in the previous section. As such, this Section first breaks trustworthiness down into its essential elements, providing definitions for each of them. Subsequently. available methods for accuracy verification are introduced, classified, and applied to assess the ability of the previously trained NNs in solving ODE systems. Finally, the Section concludes with a discussion on the interpretability of PINNs and PI-KANs.

A. Definition of Trustworthiness

Trustworthiness is essential to bridge the gap between the technical capabilities of AI and the safety expectations from industry and society. In the power systems domain, we shall consider at least the following four key dimensions: transparency, interpretability, explainability, and the verification of its performance.

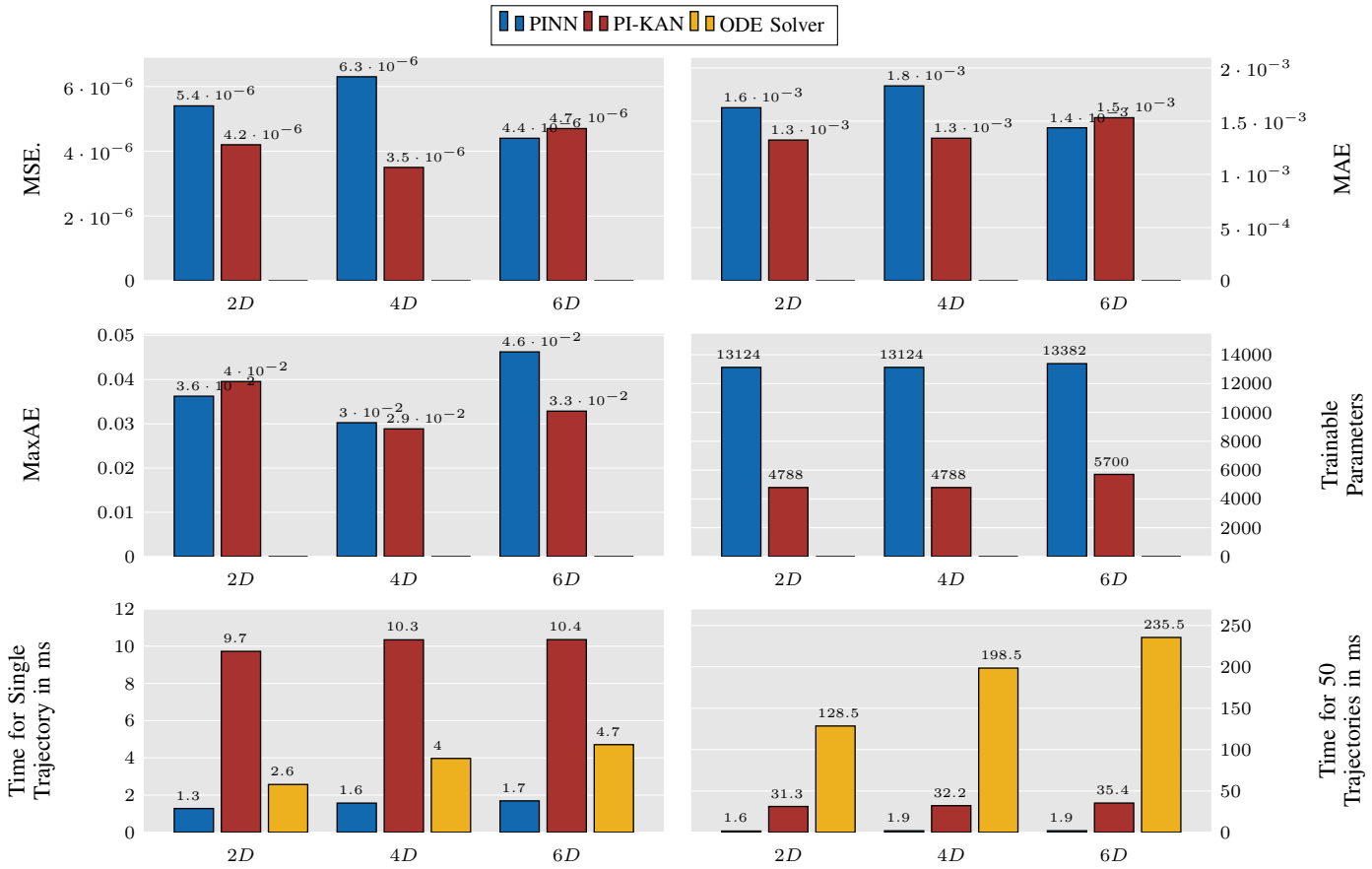

Fig. 10. Performance comparison of ordinary PINNs and PI-KANs for all test cases. The error metrics are benchmarked against the ODE solution.

Generally, transparency, interpretability, and explainability involve understanding how an AI tool processes data and how results are obtained. Transparency can be assessed through three key components: (i) model transparency, (ii) design transparency, (iii) algorithmic transparency. Model transparency refers to how clear the structure of an AI tool is. Design transparency focuses on the rationale behind model choices and the training process. Algorithmic transparency requires clarity and insights into the training procedure. Interpretability, on the other hand, entails the understandability of the internal workings of AI tools and examines why the model behaves the way it does. In terms of metrics, interpretability is often assessed through feature analysis or other proxies. In contrast, explainability considers human interaction and assesses if the AI output and decisions can be understood by humans using model insights and domain knowledge [44]. Incorporating domain knowledge and scientific consistency enhances interpretability and explainability and ensures that results align with known principles. These efforts are particularly crucial in safety-sensitive contexts.

The fourth dimension of trustworthiness considers the accuracy of the AI solutions and decisions compared to classical tools and is unique for several scientific applications, including the one presented in this work. Generally, verification assesses whether a model's implementation precisely aligns with the developer's conceptual descriptions and specifications; in our case, the mathematical description of ODEs. The verification process evaluates to which extent the AI model operates according to its original conception, specification, and design. In conducting this process, the output of the AI model is compared with the conceptual descriptions, specifications, or definitions utilized during its development [45]. It is essential to undertake this verification to ascertain that the AI model has correctly captured the physics articulated by the mathematical model. In this work's context, verification is the assessment of how well the AI solution overlaps with the underlying ODE systems that are represented through time-domain trajectories obtained through classic ODE solvers.

Balancing transparency, interpretability, and explainability with performance is essential to creating trustworthy, ethical AI systems that meet the rigorous demands of safety-critical applications, promoting both public trust and effective deployment [44]. In this work, we primarily focus on the verification of the correctness and the interpretability of the previously trained PIML-based dynamic equivalents, since the fact that underlying physical representations, in the form of ODEs, exist for the trained models permits to rigorously test and assess their performance.

B. The Core Concept of Correctness Verification

Correctness verification is a research field that aims to provide *guarantees* on how well an AI model captures the

function it was trained for.

Correctness verification for a NN that approximates the solution $\mathbf{u}$ of an IVP with the approximated solution $\hat{\mathbf{y}}$ aims to find the maximum absolute distance between the approximated and the true solution across the entire domain. Mathematically, this maximum distance $E(\mathbf{u}, \hat{\mathbf{y}}, \mathcal{X})$, which we refer to as the approximation error in the remainder of the work, is expressed as:

$$E(\mathbf{u}, \hat{\mathbf{y}}, \mathcal{X}) = \max_{(\mathbf{u}_0, t) \in \mathcal{U}} |\mathbf{u}(\mathbf{u}_0, t) - \hat{\mathbf{y}}(\mathbf{u}_0, t)|. \tag{26}$$

In most cases, including power system dynamic simulations, an analytical closed-form solution of the true underlying solution does not exist or cannot be obtained. As a result, despite the recent achievements in providing mathematical error guarantees for some specific problem types in [46], or specific forms of PDEs in [47, 48], there is a significant gap in the understanding the approximation error of ODE solvers and AI tools.

Moreover, it is crucial to understand that the components of the loss function used for PINN training, specifically $\mathcal{L}_{physics}$ and $\mathcal{L}_{boundary}$, do not directly translate to the approximation error $E(\mathbf{u}, \hat{\mathbf{y}}, \mathcal{U})$. Instead, they only indirectly capture the NN approximation error. Consequently, constraining these loss terms does neither ensure the NN performance nor does it provide performance guarantees for NN obtained from training.

C. Formulation of IVP Correctness Verification Methods

Since correctness verification methods are a very young research field, especially for power systems, in this paper we attempt a first systematic classification of tools that provide some kind of guarantee for the verification problem formulated in Eq. (26), before providing the mathematical formulation for a selection of these methods. The goal of each method included in this study is to solve the optimization problem in Eq. (26). Specifically, the aim is to estimate the approximation error between the true flow of the initial value problem (IVP) and the flow generated by a ML Model, even in the absence of a closed-form solution for the true flow.

Figure 11 presents a summary of this classification. First, we distinguish between probabilistic and deterministic verification methods. Their key difference lies in the output provided by the verifier. While probabilistic methods provide upper and lower bounds on the precision of the verification problem, deterministic tools return one exact or approximate bound on the correctness problem. Since probabilistic verification requires stringent assumptions on the data distribution, this work focuses on deterministic methods.

Deterministic verification methods are further divided into complete and incomplete tools. Complete verification provides an exact solution to the optimization problem in Eq. (26). Thus, such methods deliver rigorous guarantees on the approximation error for a given interval of time and initial conditions $\mathcal{U}$. In contrast, incomplete verification methods involve a heuristic approach to solve the aforementioned optimization. Consequently, incomplete verification methods only provide an estimated solution for the optimization problem in Eq. (26). Complete verification methods are generally much more computationally demanding. On the other hand, incomplete methods can only provide an estimate of the maximum error, but not a formal guarantee.

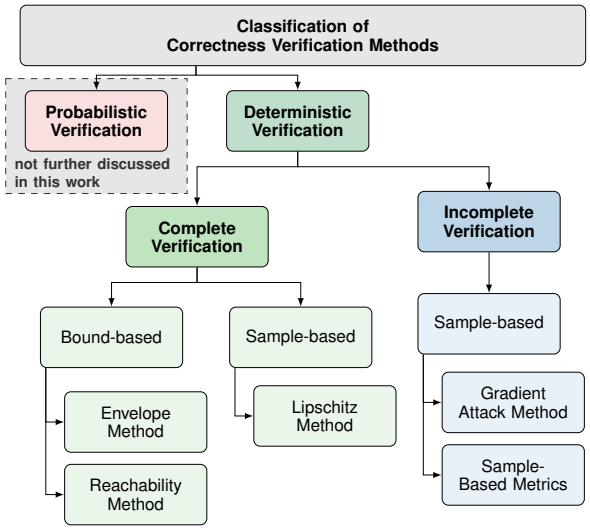

Fig. 11. Classification of correctness verification methods.

Complete verification methods are further separated into bound- and sample-based methods. Bound-based tools employ bound propagation algorithms to determine the maximum and minimum limits of the NN output by performing step-by-step estimation of the bound of each NN layer. The CROWN toolbox, published in [49], and detailed in Appendix C-A, enables such methods.

In contrast, sample-based methods discretize the continuous set of initial conditions and then either use a sampling or bounding method to approximate the bounds of these discrete sets. These methods can either provide an estimated bound (incomplete verification) or an exact bound (complete verification) for the approximation error. The Lipschitz-Method is the only sample-based method which provides an exact bound, and, therefore, falls under the complete verification category. In contrast, the gradient attack method and sample-based metrics can only provide a bound estimate, and, therefore, not a formal guarantee, but instead they do deliver a faster estimation of the approximation error.

The remainder of this Section will provide insights into the mathematical formulation and implementation for each of the deterministic methods.

1) Envelope Method: produces an exponential envelope encompassing the approximation error of NNs relative to the true solution. As such, it is a bound-based method that provides complete verification.

Theorem [32, Ch. I, Variant of Thm. 10.2] provides a reformulation of Eq. (26) that is valid only for specific conditions. To be able to obtain the NN output approximation error, the

following conditions must be fulfilled:

$$\max_{(x,0)\in\mathcal{U}} \overbrace{|\hat{\mathbf{y}}(0) - \boldsymbol{u}_0|}^{\text{Boundary loss}} \leq \alpha, \tag{27}$$

$$\max_{(x,t)\in\mathcal{U}} \underbrace{\left|\frac{\partial \hat{\mathbf{y}}(t)}{\partial t} - f\left(\hat{\mathbf{y}}(t), t\right)\right|}_{\text{Residuals}\mathcal{R}_\theta^{\text{ODE}}} \leq \zeta(t) \tag{28}$$

These two equations bound the maximum error for the $\mathcal{L}_{\text{physics}}, \mathcal{L}_{\text{boundary}}$ that are minimized during training. These conditions, in addition to the enforcement of the Picard-Lindelof Theorem, which requires the function f to be Lipschitz continuous with a Lipschitz constant C, determine that the approximation error lies inside the time-varying envelope:

$$E(\mathbf{u}, \hat{\mathbf{y}}, \mathcal{U})(t) \leq e^{Ct}\alpha + \int_0^\tau e^{C(\tau - t)}\zeta(t)dt, \tag{29}$$

where parameters α and $\zeta(t)$ correspond to the maximum bounds of the properties formulated in Eq. (27) and Eq. (28). Thus, Eq. (29) is a closed-form equation of the envelope that encloses the approximation error $E(\mathbf{u}, \hat{\mathbf{y}}, \mathcal{U})$. It is essential to highlight, that the envelope method links terms of the loss function $\mathcal{L}_{\text{physics}}$ and $\mathcal{L}_{\text{boundary}}$, that are indirect proxies of the approximation error, with the actual NN's approximation error.

The largest distance between the approximated and true solution of an IVP can be found by solving Eq. (27)-Eq. (28), and applying α and $\zeta(t)$ to Eq. (29). The result is a measure of how "good" the NN learned the flow of the ODEs. Attempting to solve these optimization problems still presents a significant challenge since it is a non-convex, nonlinear problem. Moreover, the sub-problem in Eq. (28) requires optimization over the derivative of the NN with respect to its inputs. Given that bound propagation is conducted by advancing the bounds in a forward mode, it is necessary to identify a method for representing the derivative.

The CROWN tool formulated in [49, 50] and applied in [51] can help to efficiently obtain an overapproximation of NN output functions based on a given input interval. Since the tool uses interval arithmetic, wide input intervals lead to overly conservative overapproximations. To address this, a custom branch and bound algorithm to pinpoint input areas yielding the tightest overapproximations and worst function bounds is used. Lacking a pruning criterion due to unknown final values of α and ζ, the heuristic approach, called gradient attack and described in Section IV-C4, is used to underestimate α and ζ. Thereby, we avoid the need to explore the entire domain. A description of the algorithm can be found in Appendix C-C. We call this the Worst Envelope Method.

Another challenge lies in representing and bounding the derivatives of the NN. Unlike using finite differences to represent the NN time derivatives as presented in [51], we use the Clarke-Jacobian method [52] during bound propagation in Eq. (27) and Eq. (28). The Clarke-Jacobian bounding approach provides a method to compute tight Lipschitz bounds by leveraging the convex hull of gradients and a backward computational graph for bound propagation. This method refines the representation of time derivatives by avoiding the pitfalls of finite differences, thereby ensuring tighter and more computationally efficient optimization. Further insights to the method are provided in Appendix C-D.

Obtaining worst-case guarantees with the envelope method poses significant challenges due to the substantial computational effort required. However, due to the link of conditions Eq. (27) and Eq. (28) to the loss function terms minimized during training, it is possible to deliver instantaneous and rigorous error assurances by calculating the residual errors and the envelope during inference. This is done by calculating the residual errors $\mathcal{L}_{\text{physics}}, \mathcal{L}_{\text{boundary}}$, during inference and then using Eq. (29), to calculate the error envelope. We refer to this method as Instant-Envelope Method.

2) Reachability Method: The reachability method is a junction of reachability analysis, validated integration, and the envelope method introduced in the previous Section IV-C1. Thus, it is a complete, bound-based verification tool. Before describing the reachability method mathematically, this Section briefly introduces reachability analysis and validated integration, which are fundamental techniques in the safety assurance of dynamic systems [53]. The method described in this work is an adaptation of the approach previously documented in [54]. To the best of our knowledge, this is the first time it has been introduced to the Power Systems literature.

Forward reachability analysis determines the set of all possible states a system can reach from a given set of initial conditions and constraints. As such, reachability analysis is capable of identifying potential safety violations and ensuring that the system operates within the desired range.

Validated integration complements reachability analysis by providing numerical methods that accurately emulate the system's trajectories while accounting for uncertainties and minimizing computational errors that could otherwise lead to incorrect conclusions on system safety. By combining these approaches, a comprehensive understanding of the system's behavior and an assurance that the numerical simulations faithfully represent the theoretical models can be obtained.

An exemplary tool that incorporates this integration is Flow* [55]. Flow* leverages reachability analysis to compute reachable sets using advanced set representations e.g. intervals or zonotopes. It further employs validated integration techniques, such as Taylor model-based methods, to ensure precise and reliable trajectory computations. Thus, Flow* effectively simulates complex nonlinear dynamics. Thereby, it provides robust safety guarantees and enables early detection and mitigation of potential hazards in dynamical systems.

An intriguing property of these methods is that the solution flow that we acquire from the reachability analysis tool is guaranteed to enclose the true ODE solution. Since the pretrained NN learned the flow of the equation, we can use bound propagation tools, such as CROWN, to bound the learned flow with respect to a set of initial conditions and time $\mathcal{U}$, as happens in the forward reachability analysis of the system, bridging reachability analysis and validated integration with the envelope method introduced in Section IV-C1.

Similar to the envelope method, the reachability method can bound the largest approximation error of the NN output. Given

a subset of the entire input domain $\mathcal{X} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \subset \mathcal{U}$, the ODE solution $\mathbf{u} : \mathbb{R}^n \to \mathbb{R}^m$ under with an over-approximation of its interval output range in the form of $[\underline{\mathbf{u}}(\underline{\mathbf{x}}, \overline{\mathbf{x}}), \overline{\mathbf{u}}(\underline{\mathbf{x}}, \overline{\mathbf{x}})]$, the so called reach tube is obtained, and its NN approximation $\hat{\mathbf{y}} : \mathbb{R}^n \to \mathbb{R}^m$, then the error between $\mathbf{u}(\mathbf{x})$ and $\hat{\mathbf{y}}(x)$ can be estimated as follows:

$$E(\mathbf{u}, \hat{\mathbf{y}}, \mathcal{X}) \leq \sqrt{\sum_{j=1}^{m} d_{j,\max}^2}, \tag{30}$$

where

$$d_{j,\max} = \max\Big(\big|\overline{\mathbf{u}}_j(\underline{\mathbf{x}}, \overline{\mathbf{x}}) - \underline{\hat{\mathbf{y}}}_j(\underline{\mathbf{x}}, \overline{\mathbf{x}})\big|, \big|\overline{\hat{\mathbf{y}}}_j(\underline{\mathbf{x}}, \overline{\mathbf{x}}) - \underline{\mathbf{u}}_j(\underline{\mathbf{x}}, \overline{\mathbf{x}})\big|\Big),$$

where the first term in $d_{j,\max}$ is the distance between the maximum point of the flow obtained from the validated integration technique and the minimum point of the flow obtained from the NN bound propagation technique on the provided input domain, and the second term is the reverse. Consequently, Eq. (30) computes the maximum distance between the validated integration technique and the NN output across the selected domain.

Evaluating Eq. (30) involves three consecutive steps. Symbolic bound propagation tools such as CROWN are used to compute the upper and lower bounds of the NN outputs, denoted by $\overline{\hat{\mathbf{y}}}(\underline{\mathbf{x}}, \overline{\mathbf{x}})$ and $\underline{\hat{\mathbf{y}}}(\underline{\mathbf{x}}, \overline{\mathbf{x}})$. Additionally, using the reachability analysis tool Flow*, provides the reachable sets $\overline{\mathbf{u}}_j(\underline{\mathbf{x}}, \overline{\mathbf{x}})$ and $\underline{\mathbf{u}}_j(\underline{\mathbf{x}}, \overline{\mathbf{x}})$. In summary, this method first computes the NN output bounds with CROWN, before determining the reachable tube through safe integration and reachability analysis using Flow*, and finally applies using the results in Eq. (30) delivers the approximation error bound.

The resulting bounds on the approximation error depend on the spread of the input interval $[\underline{\mathbf{x}}, \overline{\mathbf{x}}]$. Therefore, branch and bounds algorithms might achieve a tighter estimation of the approximation error. Such a branch and bound algorithm [51] recursively divides the problem into smaller, more manageable sub-problems. As such, this process systematically explores the solution space while eliminating regions that do not contain the largest bound for $E(\mathbf{u}, \hat{\mathbf{y}}, \mathcal{X})$.

Finally, we establish that the proposed solution can be effectively utilized while partitioning $\mathcal{X}$ into lattices, thereby enabling efficient exploration and tighter bounds on the error. The compact set of inputs $\mathcal{X}$ is divided into N_p pieces, each represented as $\mathcal{X}_i = [\overline{\mathbf{x}}_i, \underline{\mathbf{x}}_i]$, ensuring that $\mathcal{X} \subseteq \bigcup_{i=1}^{N_p} \mathcal{X}_i$. After splitting the compact input set $\mathcal{X}$, the guaranteed error between $\mathbf{u}(\mathbf{x})$ and $\hat{\mathbf{y}}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{X}$ becomes:

$$E(\mathbf{u}, \hat{\mathbf{y}}, \mathcal{X}) \leq \max_i \sqrt{\sum_{j=1}^{m} d_{ij,\max}^2}, \tag{31}$$

with

$$d_{ij,\max} = \max_{i=1,\ldots,N_p} \Big\{\|\overline{\mathbf{u}}_j(\underline{\mathbf{x}}^{(i)}, \overline{\mathbf{x}}^{(i)}) - \underline{\hat{\mathbf{y}}}_j(\underline{\mathbf{x}}^{(i)}, \overline{\mathbf{x}}^{(i)})\|, \|\overline{\hat{\mathbf{y}}}_j(\underline{\mathbf{x}}^{(i)}, \overline{\mathbf{x}}^{(i)}) - \underline{f}_j(\underline{\mathbf{x}}^{(i)}, \overline{\mathbf{x}}^{(i)})\|\Big\}$$

Notably, both components of the presented reachability method, the reachability analysis for nonlinear IVPs and tight NN bound propagation algorithms, exhibit exponentially increasing computation times with the number of inputs and variables. Therefore, these methods are computationally expensive for large-scale systems of ODEs, despite the partitioned reformulation of Eq. (30) to Eq. (31).

3) Lipschitz Method: The Lipschitz method is the last complete verification method introduced in this work. In contrast to the envelope and reachability methods, the Lipschtz method is a sample-based one. As such, it provides a theoretical rigorous upper bound on the worst-case approximation error. This study builds upon a method previously detailed in [54], introducing it for the first time to the Power Systems field.

Let $\mathcal{X} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \subset \mathbb{R}^n$ represent a bounded input domain. If the true ODE solution $u : \mathbb{R}^n \to \mathbb{R}^m$ has a Lipschitz constant L_f and if its NN approximation $\hat{\mathbf{y}}$ has a Lipschitz constant $L_{\hat{\mathbf{y}}}$, then the maximum error is bounded by:

$$E(\mathbf{u}, \hat{\mathbf{y}}, \mathcal{X}) \leq \|\mathbf{u}(\underline{\mathbf{x}}) - \hat{\mathbf{y}}(\underline{\mathbf{x}})\| + L_E\|\overline{\mathbf{x}} - \underline{\mathbf{x}}\| \tag{32}$$

where $L_E = L_f + L_{\hat{\mathbf{y}}}$.

Similar as before, the input domain $\mathcal{X}$ can be partitioned into smaller intervals $\mathcal{X}^{(i)} = [\underline{\mathbf{x}}^{(i)}, \overline{\mathbf{x}}^{(i)}]$, $i = 1, \ldots, N_l$ to reduce the effect of the second term in Eq. (32). An upper bound on the approximation error is then provided by:

$$E(\mathbf{u}, \hat{\mathbf{y}}, \mathcal{X}) \leq \max_{i=1,\ldots,N_l} \Big\{\|\mathbf{u}(\underline{\mathbf{x}}^{(i)}) - \hat{\mathbf{y}}(\underline{\mathbf{x}}^{(i)})\| + L_E\|\overline{\mathbf{x}}^{(i)} - \underline{\mathbf{x}}^{(i)}\|\Big\}. \tag{33}$$

While this method provides theoretical guarantees, calculating $L_{\hat{\mathbf{y}}}$ often relies on bound-propagation techniques, such as CROWN [52]. Overly large input domains lead to conservative estimates; thus, partitioning the domain reduces this conservativeness.

4) Gradient Attack Method: The gradient attack method is a sample-based incomplete verification tool, hence only providing an estimate of the approximation error in Eq. (26). It optimizes the input of the NN to identify the output with the maximum residual error using gradient ascent. Starting from a set of initial conditions, the input is iteratively refined using:

$$\mathbf{x}_{t+1} = \text{Proj}_{[\underline{\mathbf{x}}, \overline{\mathbf{x}}]}\Big(\mathbf{x}_t + \eta \cdot \text{sign}\big(\nabla_{\mathbf{x}} \hat{\mathbf{y}}(\mathbf{x}_t)\big)\Big), \tag{34}$$

where $\mathbf{x}_t$ is the input at iteration t, η is the learning rate, and $\text{Proj}_{[\underline{\mathbf{x}}, \overline{\mathbf{x}}]}$ ensures that $\mathbf{x}_t$ remains within the specified input range $[\underline{\mathbf{x}}, \overline{\mathbf{x}}]$.

This method leverages the ADAM optimizer to refine the inputs and parallelizes the optimization across multiple initial conditions. Although effective for identifying local maxima of the residuals, this method does not guarantee convergence to the global maximum and is therefore classified as an incomplete Verification approach. Despite this limitation, it provides a scalable heuristic for high-dimensional input spaces.

5) Sample-Based Metrics: Similar to the gradient attack method, sample-based metrics provide a fast estimate of the approximation error. For this method, the NN is evaluated over a predefined set of sample points. This approach identifies the largest residuals, $\mathcal{L}_{\text{boundary}}$ and $\mathcal{L}_{\text{physics}}$, within the sample set

and hence estimates the bounds α and ζ from the envelope method formulated in Eq. (27) and Eq. (28), respectively. Then, the final approximation error is estimated using Eq. (29). Alternatively, the error could be determined directly through comparison of the NN outputs and trajectories generated from simulations.

While effective for low-dimensional input spaces, this method becomes computationally restrictive in high-dimensional domains, as exhaustive sampling may require billions of points. To address this, smarter sampling strategies, such as gradient attack, are employed.

D. Demonstration of PINN Verification

After having established five different PINN verification methods that apply for the domain power system simulation, this Section demonstrates the usability of the presented verification tools. As such, this Section first presents the test cases, before depicting the analysis.

1) Test Cases: For the proof-of-concept experiments provided in this Section, the $2D$ model formulated in Section III-D1 is used. For the following analysis, PINN structures with the two input parameters are trained. The initial angle δ and the simulation time are variable. In contrast, the initial rotor speed deviation is kept constant at $\Delta\omega = 0.6$p.u. The input ranges used during the NN training were $\delta \in [-1, 1]$, over the time window $t \in [0, 2]$s. The remaining parameters are: $P_m = 0.26$p.u., $V_1 = V_2 = 0.92$ p.u., $\frac{D\Omega_b}{2H} = 0.375$. The obtained PINN is evaluated for all the provided five verification methods.

2) Implementation: In our case study, we train a traditional PINN using an LBFGS optimizer to minimize a composite loss function incorporating boundary and physics-informed terms $\mathcal{L}_{boundary}$ and $\mathcal{L}_{physics}$. The PINN model is initialized with two hidden layers of 32 neurons each. Boundary losses enforce consistency with known boundary conditions, while physics-informed losses apply differential constraints using automatic differentiation on the model outputs. It is worth noting that this architecture is not trained with simulated data. The setup is implemented in PyTorch.

For the purpose of NN bound propagation and the determination of Lipschitz constant bounds, we utilized the Auto-Lirpa toolbox [50]. All experimental procedures were conducted on a computer equipped with 16GB of RAM and an Intel i7 1355U processor.

3) Computing the Lipschitz Constant: The Lipschitz constant is often dominated by the term with the largest derivative in the system. Thus, for the given system with the provided parameters:

$$\dot{\delta} = \omega, \quad \dot{\omega} = -0.375\omega - 0.92\sin(\delta) + 0.26 \tag{35}$$

the system Jacobian becomes:

$$\mathbf{J}(x) = \begin{bmatrix} 0 & 1 \\ -0.92\cos(u) & -0.375 \end{bmatrix}. \tag{36}$$

Comparing the different entries in the Jacobian, the highest magnitude of the derivatives is the maximum of $-0.92\cos(u)$, and hence, the Lipschitz constant of the system is $L = 0.92$.

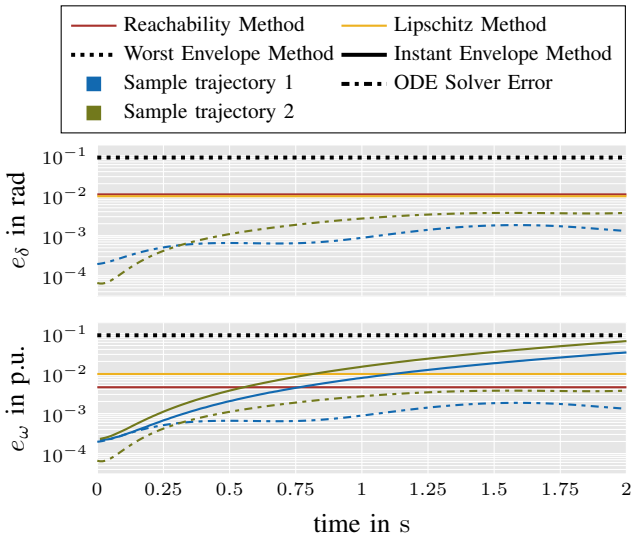

Fig. 12. Comparison of Complete verification methods. Note that the ordinate is plotted in log scale. Solid lines represent the results of the Envelope, Reachability and Lipschitz method, while the Absolute Error between the ODE solver solution and PINN solution, for random trajectories, is depicted by dashed lines. Blue and green correspond to two different time-domain trajectories.

4) Results: This Section compares the proposed verification methods in their effectiveness to bound the approximation error $E(\mathbf{u}, \hat{\mathbf{y}}, \mathcal{X})$.

Figure 12 presents the bounds on the approximation error for the reachability, Lipschitz and envelope method for two randomly selected trajectories. For comparison, the ODE solver error is obtained as $|\mathbf{u}(x_i) - \hat{\mathbf{y}}(x_i)|$ for sample trajectories $x_i \in \mathcal{X}$ and depicted by the dashed lines in Fig. 12.

Before studying the results, note that the depicted results for the reachability method focus on regions within $[\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ where the Lipschitz method exhibited the largest errors, allowing to obtain the most accurate bounds. Currently, we are unable to implement a branch and bound algorithm due to the lack of an integrated tool for combining CROWN and Flow*. Future work will involve developing such an algorithm to leverage both CROWN's optimization capabilities and Flow*'s reachability computations, thereby enhancing the precision and efficiency of our analysis.

Among the evaluated methods, Fig. 12 suggests that the reachability approach consistently provides tight bounds throughout the simulation period, effectively enclosing the sample trajectories. Notably, for the ϵ_ω case, it yields the tightest bounds compared to the other methods. In contrast, the Lipschitz method offers tighter bounds than the reachability method when applied to ϵ_δ, but less tight bounds for ϵ_ω. The reachability and Lipschitz methods show larger gaps from the sample trajectories and ODE error, especially early in the simulation.

As expected, all bounding methods are more conservative than the ODE error for the randomly picked trajectories, illustrated by the dashed-dotted lines. This is because bounding methods are designed to account for system uncertainties in addition to numerical errors. However, tighter upper bounds

TABLE I
VERIFICATION BOUNDS PROVIDED BY DIFFERENT METHODS (IN P.U.).

Conditions	Sample-Based Metrics	Gradient Attack	Envelope Method
$Eq.$ (27)	2.62×10^{-4}	2.06×10^{-4}	2.75×10^{-4}
$Eq.$ (28)	2.99×10^{-3}	3.49×10^{-3}	3.49×10^{-3}
$E(\mathbf{u}, \hat{\mathbf{y}}, \mathcal{X})$	7.70×10^{-2}	8.80×10^{-2}	9.00×10^{-2}

are preferable since they provide a more accurate estimate of the maximum possible error.

The envelope method, i.e. blue and green solid lines in Fig. 12, produces the most conservative bounds across all trajectories, especially as the simulation time increases. This conservativeness ensures that the sampled error trajectories remain well within the established bounds, thereby validating the effectiveness of these methods in containing the maximum error.

Table I compares the approximation error bounds generated by different methods: the sample-based method, the gradient attack method, and the envelope method. These methods utilize Eq. (29) to determine the maximum approximation error.

Table I suggests that the bounds produced by the pure sample-based method are more tight compared to those from the gradient attack and envelope methods. However, as a heuristic approach, the sample-based method may sometimes yield invalid bounds. In contrast, the envelope method, being a complete verification technique, provides bounds that are slightly wider than the sample-based bounds, but due to the mathematical guarantees provided by the method, they maintain validity. The gradient attack method is less tight than the sample-based approach, as it captures input points with larger residual errors, yet it remains incomplete.

Furthermore, Fig. 13 compares the computational times of the six different analytical methods applied for bounding the approximation error of PINNs. Since many of the inner processes of the methods can be done in parallel, we included the computational time of the most time-consuming inner process for each method. The methods evaluated are the Worst Envelope Method, Reachability Method, Lipschitz Method, Gradient Descent Method, Sample-Based Method, and the Instant Envelope Method. The results indicate that the Worst-case Envelope Method requires the most computation time at 20 seconds, closely followed by the Reachability Method and the Lipschitz Method, which take 16 seconds and 15 seconds, respectively. The Gradient Descent Method shows a moderate improvement with a computation time of 7.2 seconds. Notably, the Sample-Based Metrics Method and the Instant Envelope Method demonstrate significantly lower computation times of 0.1 seconds and 0.002 seconds, respectively. This stark contrast highlights the superior efficiency of the Instant Envelope Method, making it the most computationally efficient option among those tested for 2D analyses. In addition, we have to highlight that the computational time grows significantly considering bigger NN architectures and bigger systems of Differential Equations, with more state variables.

In summary, the reachability method consistently offers tight bounds on the approximation error in PINNs, effectively enclosing sample trajectories and performing particularly well in specific scenarios. Although more conservative, the envelope method ensures all error trajectories remain within established bounds, confirming its reliability over heuristic sample-based methods that may yield invalid results. Regarding computational efficiency, while methods like the reachability and worst-case envelope are time-intensive, the Instant Envelope Method demonstrates significant efficiency gains, highlighting a trade-off between computational cost and the tightness of error bounds.

5) Assumptions and Limitations of the Verification Methods: This section examines the limitations of our verification methods, which offer explicit error guarantees as a form of explanation for the machine learning model. Similar to conventional approaches—where users are informed about the approximate magnitude of the error—our methods provide a comparable measure by quantifying the potential error within a defined range.

Some of the methods, relied on assumption on what is learned but the NNs. Specifically, n the Envelope methods, we make the assumption that the approximated solution has the semigroup property, meaning that if $S(t)$ denotes the solution operator over time t, then for any non-negative times t_1 and t_2, we have $f(t_1 + t_2) = f(t_1) \circ f(t_2)$. This assumption ensures temporal consistency in the dynamic evolution, enabling the use of the theorems.

Reachability analysis has scalability issues since, with the increase in dimensions, the computational complexity is increasing exponentially [56]. In the gradient descent method, no additional assumptions about the model are required since we can perform zero-th-order differentiation on every black-box model, although in this work, we are using automatic differentiation. The gradient descent method can give us a good notion of the maximum local error of the ML model. Using different versions of gradient descent or second-order methods can improve the accuracy of the result.

In contrast, the Lipschitz method assumes that the model is Lipschitz continuous, with a known Lipschitz constant. This is not a restrictive assumption since there are methods available to approximate the Lipschitz constant of black-box models [57]. Therefore, if we have the ability to sample the black-box model, they can be used to validate the ML models against black-box models [58], [59]. However, two main challenges arise when applying these methods using a meshed uniform grid. First, the grid size grows exponentially with the number of dimensions, which can lead to evaluating a significantly high number of points. Second, these methods require the ability to sample from the model; they cannot be applied if only a fixed dataset is available for validation. Despite these challenges, it is important to note that this approach is general and can be used to validate any black-box model. Future work on this method should focus on enhancing its scalability and incorporating non-uniform sampling capabilities.

E. Discussion on Interpretability

To complete our analysis of the trustworthiness of the presented PINN-based dynamic equivalents, the trustworthiness section closes with a discussion on interpretability. Generally,

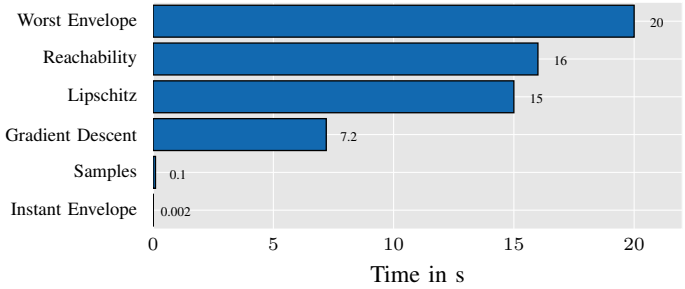

Fig. 13. Comparison of time needed for all Error Bounding Methods

interpretability in the ML context refers to the extent to which a human can understand the reasoning behind a model's decisions. This concept can be divided into two main categories: model and post-hoc explanations. Model explanations involve an inherent understanding of how the model operates internally, considering two distinct aspects (i)Simulatability implies that a person can grasp the entire model at once, enabling them to manually follow every calculation leading to a prediction, which is often feasible with simple or small models. (ii)Decomposability entails that each component of the model—inputs, parameters, and calculations—is intuitively understandable, requiring that the features themselves are interpretable so one can see how each part contributes to the overall prediction [60].

Post-hoc explanations, on the other hand, involve techniques applied after the model is trained to extract insights without necessarily understanding its internal mechanics. These include generating natural language explanations that justify the model's decisions in human-readable form and creating visualizations of the model's learned features or decision boundaries to gain qualitative insights. These tools offer local explanations by examining the model's behavior around a specific input and justifying decisions with similar training data instances. By examining interpretability through these distinct definitions, we achieve a clearer understanding of how to make models more transparent and explain their decisions, thereby increasing trust and usability in ML applications.

Interpretability is very important for applications of NNs to safety critical systems such as power systems. In contrast with vanilla NNs, KANs offer higher interpretability by design when applying the definitions of simulatability and decomposability) and at the same time provide post-hoc explanations (e.g. feature importance and visualizations). The relatively small size of KANs permits the provision of post-hoc explanations, including visualizations and feature importance assessment, and thus can enhance the intuitive understanding of the user. Moreover, visualization can be used to highlight which connections of neurons have contributed the most for each neuron.

To demonstrate the usability of the aforementioned interpretability tools, we train a KAN to approximate the solution of the Test Case, described in Section IV-D1. The KAN had 2 inputs, with 1 hidden layer of 8 neurons and 2 outputs. Then we inputted to the KAN various kinds of batches of inputs to see.

Figure 15-18 showcase the post-hoc interpretability of KANs. By sampling activation functions on input data, the visualization highlights feature importance, with lighter connections indicating low weights and darker ones showing higher weights.

Figure 15 visualizes the network's response to all training points, while Fig. 16 focuses on boundary points. In the latter, the time feature is less significant due to the constant time input in boundary data. In contrast, varying initial conditions, as shown in Fig. 15, reveal a more dynamic contribution from the time feature. This ability to distinguish feature importance enhances transparency during inference.

Figure 17 and Fig. 18 show the KAN's outputs at various time steps for two distinct trajectories with different initial conditions. Here, the initial conditions are consistent across all inputs and do not affect the outcome. Additionally, the sampled activation functions exhibit shape variations.

KANs, despite longer inference times, provide compact, interpretable architectures. Their activation functions, B-Splines, enable meaningful visualizations and approximate closed-form expressions relating inputs to outputs, fostering user trust in the decision-making process.

V. Applying PIML-Based Equivalents in Dynamic Simulations of Power Systems

Physics-informed machine learning (PIML) enables the development of high-performance models without the need for massive datasets. In the previous sections, we have made significant strides in applying these techniques to power system components and demonstrated methods to verify their performance, fostering trustworthy machine-learning algorithms. To leverage the benefits of these high-performing models in practice, they need to provide an advantage to existing power system dynamic simulation methods in an easy and straightforward way.

This Section completes our framework by presenting how the developed NN-based surrogate models can be integrated in a Plug'n'Play manner into existing power system simulation tools or into redefined simulation algorithms. As such, this section consists of two parts. First, we revisit the classic time-domain simulation problem and the state-of-the-art solution algorithm. Then, two different approaches that leverage PIML-based dynamic equivalents to accelerate the simulation's computing time are presented: (i) The first method integrates PIML tools into a classic time-domain solver, thus paving the way for the integration of PIML dynamic equivalents in standard power system simulation tools. (ii) The second method is a complete ML-based solver solely relying on PINNs, the so-called PINNSim; here, every dynamic component must be represented by a PINN. Note that all methods presented in this section assume phasor approximation for now, i.e. electromagnetic transients are not considered in this work [26]. We demonstrate both methods on the classic IEEE 9-bus test system shown in Fig. 19. This system consists of 9 buses with 3 loads and 3 dynamic components modeled by the 2D synchronous machines models explained in Eq. (22). Although the following integration approaches are modular and can scale to systems with thousands of buses, we select a small test case

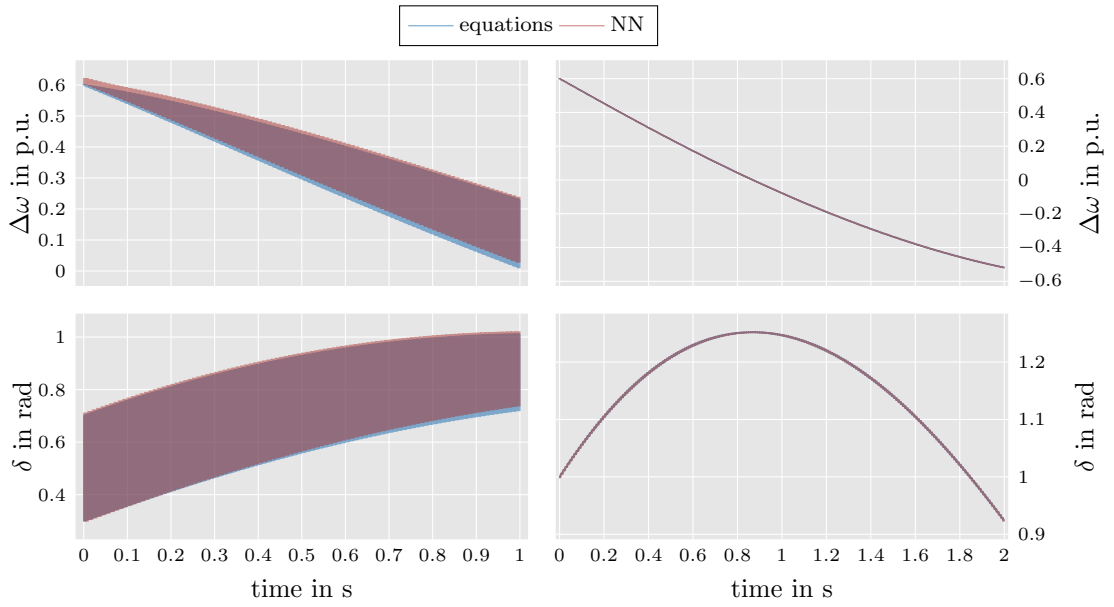

Fig. 14. Comparison of solution bounds from ODE Reachability analysis and CROWN on Physics Informed Dynamic Equivalent, with identical initial conditions

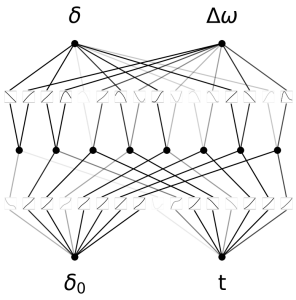

Fig. 15. KAN yielded by sampling the activation function space over the whole input domain.

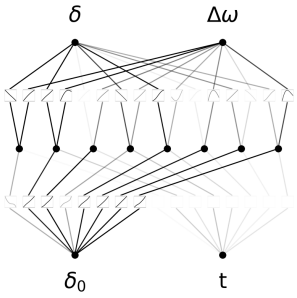

Fig. 16. KAN yielded by sampling the activation function space over the boundary conditions input domain.

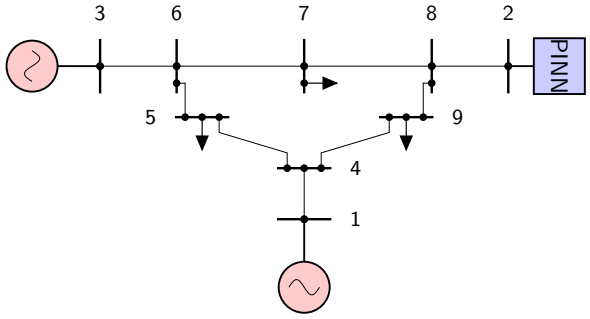

Fig. 19. Single line diagram of the implemented 9-bus test case. The SMs at bus 1 and bus 3 are represented by ODEs, whereas the SM at bus 2 is replaced by a PINN.

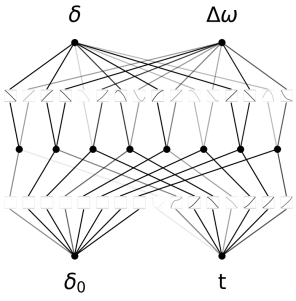

Fig. 17. KAN yielded by sampling the activation function space over a single trajectory.

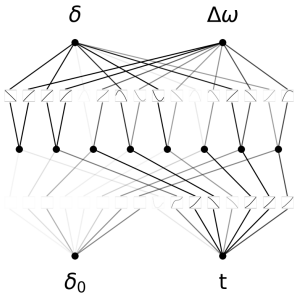

Fig. 18. KAN yielded by sampling the activation function space over a different single trajectory.

with two dynamic components to help us better explain the algorithm to the reader. For the integration approach, we will replace the machine in bus 2 with a PIML model to accelerate the simulation time. In contrast, in the PINNSim simulator, all synchronous machines will be captured by a PIML model.

Table II presents the parameters used in all simulations done using the 9-bus system.

TABLE II
MACHINE PARAMETERS OF THE STUDIED SYSTEM TAKEN FROM [61].

Machine	H in s	D in p.u.	$\boldsymbol{X_d}$ in p.u.	$\boldsymbol{X'_d}$ in p.u.	$\boldsymbol{R_s}$ in p.u.	$\boldsymbol{P_m}$ in p.u.
1	6.4	1.28	0.8958	0.1969	0.0	1.612
2	3.01	0.903	1.3125	0.1813	0.0	0.859
3	23.64	2.364	0.146	0.0608	0.0	0.71

The remainder of the Section first revisits the mathematical foundation of classic time-domain simulation tools in power systems before introducing the ML-assisted and ML-based solutions for time-domain simulations. The demonstration and results for each method are provided in the corresponding subsections.

A. Classic Time Domain Simulations in Power Systems

Power system time-domain simulations are typically described by a system of Differential-Algebraic Equations (DAE), where the initial condition is known from a load flow. Such DAE system is mathematically described as

$$\begin{aligned} \dot{\mathbf{u}} &= f(\mathbf{u}, \mathbf{z}) \\ 0 &= g(\mathbf{u}, \mathbf{z}), \end{aligned} \tag{37}$$

where $\mathbf{u}(t)$ are the differential states and $\mathbf{z}(t)$ the algebraic variables, both in vector form. These vectors contain the state and algebraic variables of the models of every component in the system, such as the one defined in Eq. (22). We consider the semi-explicit form of DAEs to be of index 1, which is a characteristic met for almost all power system transient stability simulations except when the system is close to voltage collapse [62, 63]. Since all the components in a system are interconnected, they must collectively satisfy the conditions of the load flow balance

$$0 = \mathbf{Y}\mathbf{v} - \mathbf{i}, \tag{38}$$

where $\mathbf{Y}$ represents the bus admittance matrix, $\mathbf{v}$ is the bus voltages, and $\mathbf{i}$ are bus currents.

The DAE system, defined in Eq. (37), cannot be solved analytically, as the differential states are defined by ODEs. Thus, numerical integration methods are typically employed to calculate the evolutions of $\mathbf{u}(t)$ and, indirectly $\mathbf{z}(t)$. The evolution of the differential states is then defined by

$$\mathbf{u}(t) = \mathbf{u}_0 + \int_{t_0}^{t} f(\mathbf{u}(\tau), \mathbf{z}(\tau)) d\tau, \tag{39}$$

where the goal is the computation of the solution $\mathbf{u}(t)$ at time t, considering the known conditions $\mathbf{u}_0$ at time t_0.

There are many types of numerical integration methods capable of evaluating the integral in Eq. (39), where the most common ones are Runge-Kutta methods. Since integrals cannot be solved analytically, these methods use temporal discretization to iteratively approximate the integrals solution. As such, they initialize with the initial condition of the DAE $\mathbf{u}_0$, and iteratively obtain the trajectory. We show the DAE system with one of the most well-known and straightforward integration methods, the trapezoidal rule. This method approximates the integral with an algebraic relationship, thereby transforming the integration problem into a determined system that can be solved by root-finding algorithms such as the Newton-Raphson method. Applying the trapezoidal rule to Eq. (39), results in the following system of equations:

$$\mathbf{u}(t) = \mathbf{u}_o + \frac{t}{2}\left(f(\mathbf{u}_0, \mathbf{z}_0) + f(\mathbf{u}(t), \mathbf{z}(t))\right) \tag{40a}$$

$$0 = g(\mathbf{u}(t), \mathbf{z}(t)), \tag{40b}$$

where t is called the step size of the iterative method.

The selection of the time step size used in numerical integration schemes plays a critical role in determining the simulation's accuracy and speed. Generally, smaller time steps result in smaller approximation errors. However, with a very small time step, the algorithm requires many iterations to compute a trajectory, hence the simulation becomes very slow. Consequently, balancing between the selection of the time step size and computational efficiency and accuracy is crucial. Variable time step algorithms improve computational efficiency by iteratively adjusting the time step size of the numerical integration method and automatically selecting smaller time steps during system transients and larger time steps when the system trajectory has stabilized.

Integration methods are classified into two categories: implicit and explicit. The trapezoidal method is implicit, meaning that it solves the next time step considering the current *and* next evaluation time. In contrast, explicit methods do not consider the next evaluation time to approximate the solution and thus are a fast alternative to implicit methods. The most well-known explicit methods are the Euler method and the Runge-Kutta 4. Even though they are more expensive to evaluate, implicit methods are the most used in the power systems domain since they are more suitable for stiff problems. Nonetheless, the algorithm's workflow of an implicit and explicit method is similar.

B. Modular Integration into Classic Time-Domain Solvers

Integrating ML-based dynamic equivalents directly into classic time-domain solvers enables a straightforward synergy between the speed and accuracy of the ML-based methods with the established simulation workflow, as thoroughly described in [64]. As discussed in Section III, PINNs provide accurate approximations even for large timesteps. To integrate NN-based methods with transient simulation framework, these methods must learn the dynamics considering the system's interaction with them. This formulation enables PIMLs to accurately model power system dynamics over extended time steps, surpassing simpler approximations. By dividing the learning task into modular units, we can achieve more precise approximations over larger time steps, shorter training times, and scalable performance since each component can be trained separately. In the future, this technique could replace the dynamics of complex components, like low-damping synchronous generators or nonlinear inverter-based generators and loads, which would require small time steps to accurately capture their fast dynamics. Replacing those units with PINN-based dynamic equivalents that are accurate for larger time steps permits the simulator to operate with larger time steps, hence accelerating the simulation's computing time.

The remainder of this Section first revisits the PINN formulation for direct integration into conventional solvers. Secondly, it compares the PINN integration approximation to a well-known Runge-Kutta scheme. Lastly, it demonstrates the PINN integration using the 9-bus system previously introduced.

1) PINN formulation: As previously discussed, we aim to provide an alternative method to the integration shown in Eq. (39), using a PINN. Thus, the PINN's output $\hat{y}$ has to approximate the integral

$$\text{PIML}(t, \mathbf{u}, \mathbf{z}) \approx \int_{t_0}^{t} f(\mathbf{u}(\tau), \mathbf{z}(\tau)) d\tau, \tag{41}$$

which is typically approximated by conventional Runge-Kutta schemes.

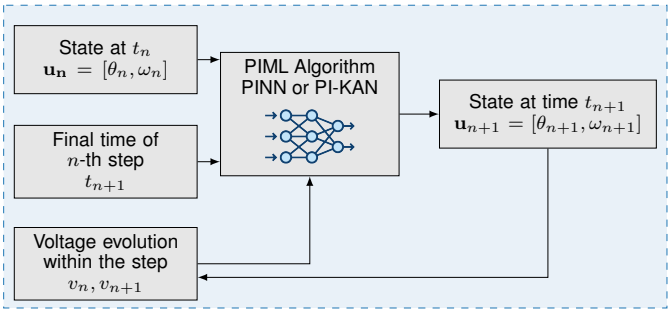

Fig. 20. Depicted are the PINNs inputs and outputs in every simulation time step.

The critical step in defining a formulation that correctly captures the trajectory of $\mathbf{u}(t)$ is the design of the inputs and the choice of the input domain. Firstly, the range in which the PINN will approximate $t \in [0, t_{max}]$ is selected. Then, the initial conditions of the DAEs must be accurately captured. Finally, the evolution of the algebraic variables $\mathbf{z}(t)$ throughout the time step t must be assumed; here, we consider a linear trajectory from the beginning to the end of the time step, represented by $\mathbf{z}_0$ and $\mathbf{z}_t$, respectively. Hence, the evolution of the algebraic states is denoted by

$$\hat{\mathbf{z}} = \mathbf{z}_0 + \frac{(\mathbf{z}_t - \mathbf{z}_0)}{(t - t_0)} t. \tag{42}$$

By approximating the $\mathbf{z}(t)$ evolution with a linear profile, the PINN becomes completely consistent with the established transient simulation framework, such as the trapezoidal rule, and can modularly replace Runge-Kutta schemes with the following definition:

$$\hat{\mathbf{y}}_t = \mathbf{u}_0 + \text{PIML}(t, \mathbf{u}_0, \mathbf{z}_0, \mathbf{z}_t). \tag{43}$$

To represent this as an ML model, we used the Residual Neural Network architecture that has been proven more precise than conventional architectures in the literature [65]. Figure 20 illustrates what are the required inputs of the PIML equivalent and what are the outputs of such a model. As can be seen in Eq. Eq. (43), the inputs can be divided into three categories: (i) the initial states of the dynamic model, which are iteratively being updated, (ii) the time step size taken in each iteration, and (iii) the algebraic variables evolution. This algebraic variable evolution, which in our study case is represented by the complex voltage, requires the initial value and the variable's value at the end of the time step, constantly being updated by Newton's method until it converges to the true value. The outputs of the PIML are then the states $\mathbf{u}(t)$ at t.

2) Integration Comparison: While the trapezoidal rule is highly flexible and offers sufficiently accurate approximations, it requires small time steps due to its reliance on a linear function evolution. In contrast, PINNs allow us to capitalize on the initial training effort to incorporate the flow of the system and hence contain more information about the underlying dynamics than conventional Runge-Kutta schemes, which assume a linear evolution of the profile between time steps (see Fig. 21). Consequently, as long as we train PINNs for sufficiently large time steps, they yield more accurate

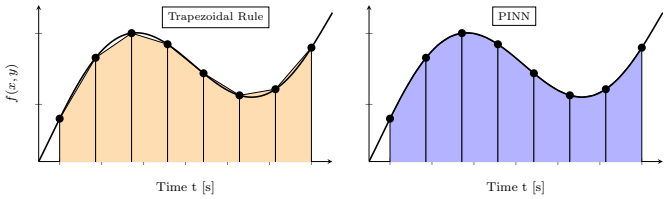

Fig. 21. Comparison of the integration using the trapezoidal rule (left) and the PINN approximation (right).

approximations, thus enabling the use of larger time steps in simulations. Figure 21 depicts that PINNs become more accurate approximators than established techniques such as the trapezoidal rule.

Similar to the trapezoidal rule, the PINN approximation formula introduced in Eq. (43) has an implicit nature. This implicit formulation enables the method to integrate in a plug'n'play manner into simultaneous-implicit solvers. Eqs. Eq. (44) show how the PINN approximation formula uses the same variables as the trapezoidal rule, making both methods compatible. The only differences between both methods are the evaluated functions and the functions' structure. A similar concept can be replicated for an explicit method that is to be integrated into a partitioned-explicit solver [61].

$$y_t = u_0 + \frac{t}{2}\Big(\mathrm{f}(u_0, V_0) + \mathrm{f}(y_t, V_t)\Big) \tag{44a}$$

$$\hat{y}_t = u_0 + \text{PIML}\Big(u_0, t, V_0, V_t\Big) \tag{44b}$$

3) Proof of concept: Demonstration of PINN Integration: For training details we refer to Section III. The only difference included by this PINN formulation used here comes from the algebraic variables $y(t)$, that are now included in the PINN training within the operating bounds of each variable, as previously shown in Eq. (44). The code to reproduce the results can be found in [66].

Figure 23 depicts the time-domain voltage trajectories of the terminal buses of each synchronous machine. The black dashed lines are obtained with a very small time step size and are used as the true trajectory. The colored lines show the performance of using a PINN to capture the dynamics of Machine 2 while the other two are modeled with a conventional trapezoidal rule. The solid lines use a time step size of 10 ms while the dash-dotted lines use a time step size of 40 ms. Integrating PIML models into time-domain simulations not only unlocks opportunities for better component modeling but also can accelerate simulations and provide more accurate results, as discussed below.

To compare the performance improvements of including a PIML in conventional simulators, we distinguish between a pure and a hybrid solver. On the one hand, the pure solver is based on the standard transient simulation algorithms, using the trapezoidal rule to approximate the integration of all components' states in the system. On the other hand, the hybrid solver uses exactly the same algorithm of the pure solver but replaces the integral approximation of Machine 2 by a trained PINN.

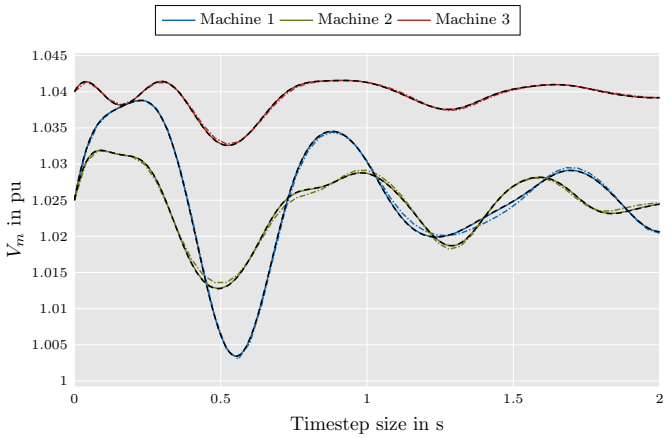

Fig. 22. Voltage trajectories after perturbing the state variable 1.1ω. The black dashed line depicts the true trajectories with a time step size of 0.1 ms, the colored solid lines the integration prediction with a time step size of 10 ms, and the colored dash-dotted lines the integration prediction with a time step size of 40 ms.

Figure 23 shows the error distributions after one timestep for 150 different initial conditions randomly sampled from the operating domain of the 9-bus system. The considered error metric is the absolute error for each variable to the true trajectory. For time steps $\Delta t \in [5, 40]$ ms, we compute several statistical metrics of the error distributions, such as the median, interquartile range (IQR) and the upper whisker, calculated as Upper Whisker $=$ Q3 $+ \frac{3}{2}$IQR. The upper whisker represents the maximum value of the data that is not considered an outlier. It typically encapsulates around 99 % of the data. We plot these results for the pure and hybrid solver for two simulation variables, a differential and an algebraic one. The hybrid solver clearly outperforms the pure one, especially for Machine 2, which is the one modelled with a PINN in the simulation. The next Section presents PINNSim, where instead of incorporating individual PINNs in existing time-domain simulators, we discuss the option of creating a new modular power system dynamic simulator that uses primarily ML-based dynamic models.

C. PINNSim: A Purely PINN-Based Simulator

PINNSim is a PINN-based simulator explicitly designed for power system dynamics and facilitates large timesteps while solving the set of DAEs. Similar to the modular integration method described in Section V-B, PINNSim employs a decomposition approach to handle the dynamics of individual components efficiently, where each component's dynamics are modeled by separate PINNs, reducing the dimensionality and complexity of each learning task.

However, unlike the method in the previous Section V-B, the interactions across the network here are resolved using a heuristic gradient descent-based iterative root-finding algorithm. PINNSim leverages a polynomial approximation for voltage evaluation at each timestep, as we see in the next paragraph, which enables more accurate time-domain simulations over large timesteps, thus minimizing the mismatch in the load flow balance in the system.

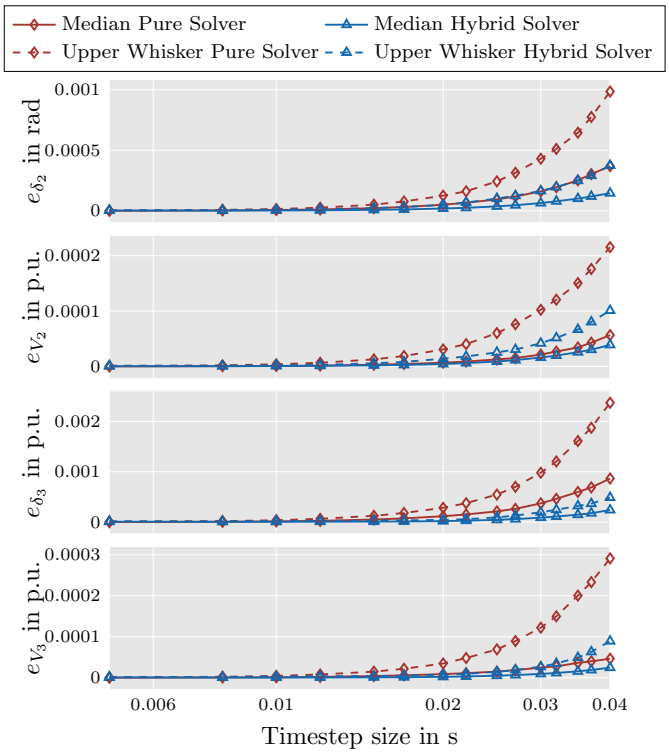

Fig. 23. Error distributions of the pure and hybrid solvers for one hundred fifty random initial conditions. The solid and dashed lines show the median and the upper whisker of the distributions for the rotor angle δ and terminal voltage magnitude for bus 2 and 3.

1) Parametrization of Voltage Profiles: To approximate voltage profiles across time steps, bus voltages are expressed in polar form and parameterized as a truncated polynomial function of time:

$$\hat{v}_i(t) = \left(\sum_{k=0}^{r} V_{k,i} (t - t_0)^k \right) e^{j\left(\sum_{k=0}^{r} \theta_{k,i} (t-t_0)^k\right)}, \qquad (45)$$

where r is the degree of the polynomial. The parameter set $\mathbf{\Xi}_i = [V_{0,i}, \theta_{0,i}, \ldots, V_{r,i}, \theta_{r,i}] \in \mathbb{R}^{2(r+1)}$ defines the temporal evolution of voltage at bus i. These parameters are iteratively refined during the simulation to ensure the balance of currents in the system.

2) Solving Component Dynamics: In PINNSim, dedicated PINNs are trained for each component to approximate the solution of its differential equations (Eq. (39)). These PINNs take as inputs the time step size (Δt), initial states (u_O, i), and voltage profiles defined by $\mathbf{\Xi}$ to predict the final state of the system as follows:

$$\hat{y}_i(t) = \mathrm{PINN}(u_{0,i}, \Delta t, z_i, \mathbf{\Xi}) \qquad (46)$$

Pre-trained on datasets containing likely operating conditions, the PINNs deliver robust predictions, even under larger time step sizes.

3) Updating Voltage Profiles: Each grid component is modeled as a current source. Thus, the component current injection $\hat{\mathbf{i}}^C$ will be a function voltage evolution, parameterized by Ξ and the state of the component u_t. To manage

the interactions between different power system components and to ensure current balance across the network, PINNSim iteratively adjusts the voltage profile parameters $\boldsymbol{\Xi}$ to minimize the mismatch between network currents $\hat{\mathbf{i}}^N$ and component currents $\hat{\mathbf{i}}^C$. This problem is formulated as a nonlinear least-squares optimization:

$$\min_{\boldsymbol{\Xi}} \sum_{i=1}^{n} \|\hat{\mathbf{i}}_i^C - \hat{\mathbf{i}}_i^N\|^2, \tag{47}$$

where $\|\cdot\|$ denotes the norm induced by the inner product over the interval $[t_0, t_0+\Delta t]$. Automatic differentiation is used to compute the Jacobian, enabling efficient updates within a sparse matrix framework for scalability.

4) Algorithm: PINNSim Time Stepping: The complete PINNSim algorithm integrates these steps to simulate each time step:

i) Initialize Ξ and component states.
ii) Use PINNs to simulate component dynamics and compute $\hat{i}^C$.
iii) Calculate $\hat{i}^N$ based on the current voltage profile.
iv) Solve the optimization problem to update Ξ, iterating until convergence.
v) Proceed to the next time step with updated Ξ and states.

This iterative procedure combines the flexibility of PINNs with conventional simulation techniques to achieve scalable and efficient time-domain simulations for large power systems.

5) Experimental Setup: This section demonstrates a proof of concept for PINNSim through numerical experiments conducted on the IEEE 9-bus system shown in Fig. 19, which includes three generators. The code used to reproduce the results can be found in [67].

The synchronous machine parameters can be found in Table II. The generator dynamics are modeled using the second-order Swing Equation Eq. (22), with the rotor angle (δ) and frequency deviation ($\Delta\omega$) forming the state vector. The voltage magnitude (V) and angle (θ) at the terminals, along with the mechanical power (P_m) and excitation voltage (E_{fd}), are treated as control inputs. Component parameters are detailed in Table I. The approach is scalable and can be extended to include more detailed models, such as higher-order dynamics, governor and exciter dynamics, inverter-based resources, or voltage-dependent loads.

6) Implementation and Training: The simulator is implemented in PyTorch. Each PINN consists of three hidden layers with 32 neurons per layer and employs the tanh activation function. Additional implementation details can be found in [14].

7) Accuracy of PINNSim: The accuracy of PINNSim is benchmarked against a Differential-Algebraic Equation (DAE) solver employing the trapezoidal rule, as described in Section V-B. Both methods are compared to the exact solution using a time step size of 0.1 ms. For this evaluation, 100 different initial states were analyzed for Generators 1–3 of the IEEE 9-bus system. Additionally, the accuracy of the predicted solutions is evaluated against the exact solution for varying time step sizes. The results, presented in Fig. 24, show the maximum and median absolute errors for four key variables:

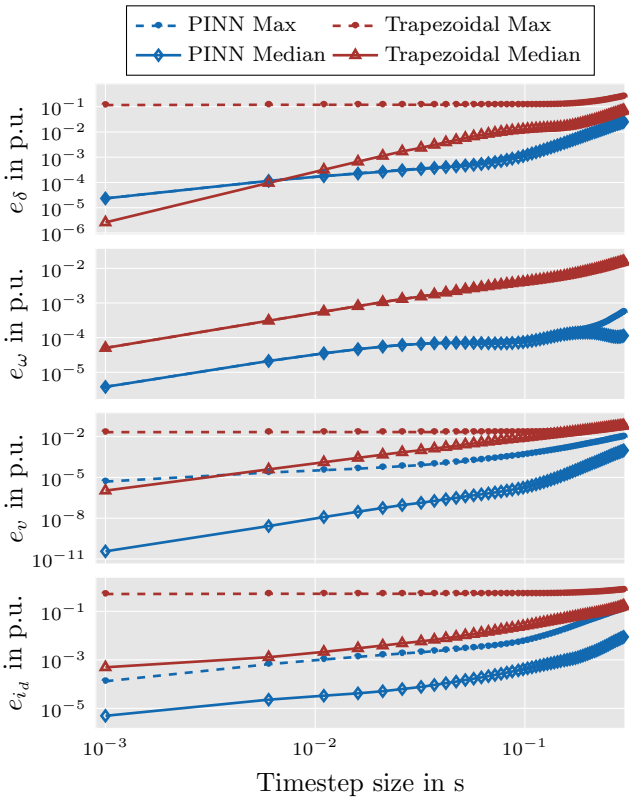

Fig. 24. Error distributions as a function of time step size are presented for both the pure trapezoidal method and the PINNSim simulation, evaluated across 100 random initial conditions. The dashed line represents the maximum absolute error, while the solid line illustrates the mean absolute error.

rotor angle error (e_δ) for δ, frequency deviation error (e_ω) for ω, terminal voltage magnitude error (e_v) for V, and d-axis current error (e_{id}) for i_d in the dq domain across different time step sizes.

PINNSim consistently outperforms the trapezoidal rule, particularly for larger timesteps ($> 10^{-2}$), across all variables analyzed. While the trapezoidal rule exhibits significant error escalation with increasing timestep size, PINNSim maintains low and stable errors, demonstrating robustness and reliability. This consistent accuracy, even at larger timesteps, makes PINNSim a highly suitable option for power system applications that require precise and efficient predictions.

VI. Challenges for ML-Based Dynamic Simulations

This section provides a discussion on the challenges of the methods and results presented in each of the aforementioned sections.

A. PIML-based dynamic equivalents

A comparative study is essential to evaluate the generalization ability of PIML-based dynamic equivalents trained on a specific input domain $\mathcal{U}$ when applied to a larger domain. This investigation would help determine the amount of data required to achieve better generalization or to facilitate training over extended input domains.

To enhance the efficiency of PIML-based dynamic equivalents training and reduce reliance on extensive simulated data, new techniques should be developed or adapted. For instance, employing adaptive collocation points [68] can improve the generalization ability of PINNs while requiring less data.

Furthermore, there is a pressing need for a standardized toolbox for PINN training across various components. Standardizing the pipeline not only holds significant research interest but also facilitates a systematic exploration of PINNs. Such a toolbox would integrate different techniques, including adaptive collocation point methods and loss weight adaptation strategies, to improve training processes. This approach would enable researchers to implement and utilize components in their work more effectively.

The envisioned toolbox should support plug-and-play functionality for component equations, including inverters, synchronous machines, time-varying loads, condensers, and other relevant elements.

B. Trustworthiness

Standardized test datasets are indispensable for the consistent benchmarking of NNs and techniques pertaining to specific components. Such datasets will be devised through comprehensive sampling of responses, thereby furnishing an equitable metric for the evaluation of novel methods and providing a uniform foundation for assessing the efficacy of training and model performance.

Current complete verification methods that provide rigorous guarantees are neither fast enough nor scalable, often being too conservative. Therefore, research into NN verification and reachability analysis is essential to develop faster and more scalable tools. Existing NN verification tools are suited for robustness verification with small input domains and known objective functions. Robustness verification pertains to the assurances we provide that a model's output will remain largely unchanged when subjected to a minor perturbation in the input. In contrast, our verification problem in Eq. (26) involves large input domains and unknown objective functions [51]. By unknown objective functions, we mean that any closed-form functions cannot describe the part of the ground truth solution of the ODE, and an approximation of it can be only described by the

Verification times are directly linked to interpretability; smaller architectures typically require less time for verification. Identifying activation functions that enable compact, interpretable architectures while being sufficiently expressive is a promising direction. These activation functions should possess mathematical properties that facilitate methods like linear bound propagation tools (e.g., CROWN) to provide fast, accurate bounds. For instance, NNs with $\tanh$ activation functions are more challenging to bound than those with sinusoidal activation functions [22].

The established connection between reachability analysis and NN verification opens avenues to employ techniques like branch and bound for approximate reachability analysis using PIML-based dynamic equivalents while ensuring error guarantees on the approximation error.

Despite demonstrating some of the benefits of using NNs over conventional ordinary differential equations (ODEs), NNs are still not perceived as sufficiently trustworthy. The research community and industry need to establish the requirements for trustworthy NN surrogates. The concept of trustworthiness remains vague, and questions like "NN is a block box; how can I trust it?" persist, even though worst-case approximation errors can be determined, as illustrated in this paper.

C. PIML-Based Equivalents in Dynamic Simulations

Although we have demonstrated the feasibility of decomposing the power system and integrating ML-based equivalents into existing time-domain simulators in a Plug'n'Play manner or creating a completely new ML-dominated power system dynamic simulator (PINNSim), several open questions remain. First, at the component level, we still need detailed NN models capable of accurately capturing the fast transient behaviors of power system components, such as converters. Although a few works have looked into developing enhanced PINN architectures to predict converter-controlled resources, this is particularly challenging in scenarios where the underlying physics could be more complex or poorly understood, making the development of physics-informed NNs (PINNs) exceptionally difficult. Second, at the simulator level, our focus has been on RMS-based simulations (i.e., using the phasor approximation), which neglects electromagnetic transient phenomena. Future efforts shall address electromagnetic transient (EMT) simulations to capture dynamic phenomena that become prevalent with large shares of converter-interfaced resources. Additionally, in the context of the PINNSim, we still need to refine the algorithm to enable efficient parallelization of simulations and fully unlock the potential of PINNSim.

VII. Conclusion

The proposed framework for trustworthy physics-informed machine learning (PIML) in power system dynamics represents a significant advancement in integrating advanced ML models into dynamic simulations. By embedding domain knowledge directly into neural network (NN) architectures, the PIML algorithm ensures that machine learning predictions align closely with established physical principles, enhancing both accuracy and reliability. We have demonstrated that the proposed PINN and PI-KAN architectures are highly effective in capturing the dynamics of power system components, such as the sixth-order representation of synchronous machines, at a fraction of the computational time required to solve the ODEs. The emphasis on interpretability, transparency, and verification further addresses barriers to industrial adoption, reassuring stakeholders that these methods are trustworthy for safety-critical applications. Additionally, the framework's modular approach—training ML models for individual components and integrating them in a Plug'n'Play manner—offers flexibility and scalability, making it well-suited for increasingly complex power systems. We have also shown improved accuracy over larger time step sizes when simulating by replacing components with their PINN approximations in a traditional time-domain simulation tool or by fully transitioning to an ML-based simulation tool. This research lays the foundation

for faster, more efficient computational analyses, enhancing decision-making for grid operators and fostering confidence in the adoption of AI-driven tools within the energy sector.

REFERENCES

[1] J. Han, A. Jentzen, and W. E, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505–8510, 2018.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[4] F. Milano, *Power System Modelling and Scripting*. Springer Science & Business Media, 2010.

[5] 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 Trans. Power Syst.*, vol. 39, no. 2, pp. 2421–2437, 2024.

[6] M. Marković, M. Bossart, and B.-M. Hodge, "Machine learning for modern power distribution systems: Progress and perspectives," *Journal of Renewable and Sustainable Energy*, vol. 15, no. 3, p. 032301, 06 2023.

[7] M. S. Ibrahim, W. Dong, and Q. Yang, "Machine learning driven smart electric power systems: Current trends and new perspectives," *Applied Energy*, vol. 272, p. 115237, 2020.

[8] L. Pagnier and M. Chertkov, "Physics-informed graphical neural network for parameter & state estimations in power systems," 2021. [Online]. Available: www.arxiv.org/abs/2102.06349

[9] J. Stiasny, G. S. Misyris, and S. Chatzivasileiadis, "Transient stability analysis with physics-informed neural networks," 2023. [Online]. Available: www.arxiv.org/abs/2106.13638

[10] X. Lei, Z. Yang, J. Yu, J. Zhao, Q. Gao, and H. Yu, "Data-driven optimal power flow: A physics-informed machine learning approach," *IEEE Trans. Power Syst.*, vol. 36, no. 1, pp. 346–354, 2021.

[11] J. Stiasny and S. Chatzivasileiadis, "Physics-informed neural networks for time-domain simulations: Accuracy, computational cost, and flexibility," *Electric Power Systems Research*, vol. 224, p. 109748, 2023.

[12] J. Stiasny, S. Chevalier, and S. Chatzivasileiadis, "Learning without data: Physics-informed neural networks for fast time-domain simulation," in *2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2021, pp. 438–443.

[13] Y. Huang, C. Zou, Y. Li, and T. Wik, "Minn: Learning the dynamics of differential-algebraic equations and application to battery modeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2024.

[14] J. Stiasny, B. Zhang, and S. Chatzivasileiadis, "Pinnsim: A simulator for power system dynamics based on physics-informed neural networks," *Electric Power Systems Research*, vol. 235, p. 110796, 2024.

[15] C. Moya and G. Lin, "Dae-pinn: a physics-informed neural network model for simulating differential algebraic equations with application to power networks," *Neural Computing and Applications*, vol. 35, no. 5, pp. 3789–3804, Feb 2023.

[16] M. Bossart, J. D. Lara, C. Roberts, R. Henriquez-Auba, D. Callaway, and B.-M. Hodge, "Acceleration of power system dynamic simulations using a deep equilibrium layer and neural ode surrogate," *arXiv e-prints*, 2024.

[17] T. Xiao, Y. Chen, S. Huang, T. He, and H. Guan, "Feasibility study of neural ode and dae modules for power system dynamic component modeling," *IEEE Transactions on Power Systems*, vol. 38, no. 3, pp. 2666–2678, 2022.

[18] "EU Artificial Intelligence Act (Regulation (EU) 2024/1689) - Updates, Training, Compliance — artificial-intelligence-act.com," www.www.artificial-intelligence-act.com/, [Accessed 21-11-2024].

[19] J. Kelly, S. A. Zafar, L. Heidemann, J.-V. Zacchi, D. Espinoza, and N. Mata, "Navigating the eu ai act: A methodological approach to compliance for safety-critical products," in *2024 IEEE Conference on Artificial Intelligence (CAI)*. IEEE, Jun. 2024, p. 979–984. [Online]. Available: http://dx.doi.org/10.1109/CAI59869.2024.00179

[20] A. Venzke, G. Qu, S. Low, and S. Chatzivasileiadis, "Learning optimal power flow: Worst-case guarantees for neural networks," in *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2020, pp. 1–7.

[21] R. Nellikkath, M. Tanneau, P. V. Hentenryck, and S. Chatzivasileiadis, "Scalable exact verification of optimization proxies for large-scale optimal power flow," 2024. [Online]. Available: www.arxiv.org/abs/2405.06109

[22] Z. Shi, Q. Jin, J. Z. Kolter, S. Jana, C.-J. Hsieh, and H. Zhang, "Formal verification for neural networks with general nonlinearities via branch-and-bound," 2024. [Online]. Available: https://openreview.net/forum?id=ivokwVKY4o

[23] S. Chevalier, I. Murzakhanov, and S. Chatzivasileiadis, "Gpu-accelerated verification of machine learning models for power systems," 2023. [Online]. Available: www.arxiv.org/abs/2306.10617

[24] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, and M. Tegmark, "Kan: Kolmogorov-arnold networks," 2024. [Online]. Available: www.arxiv.org/abs/2404.19756

[25] Z. Liu, P. Ma, Y. Wang, W. Matusik, and M. Tegmark, "Kan 2.0: Kolmogorov-arnold networks meet science," 2024. [Online]. Available: https://arxiv.org/abs/2408.10205

[26] P. Aristidou, "Time-domain simulation of large electric power systems using domain-decomposition and parallel processing methods," Ph.D. dissertation, ULiège - Université de Liège, 26 June 2015.

[27] Y. Wang, J. Sun, J. Bai, C. Anitescu, M. S. Eshaghi, X. Zhuang, T. Rabczuk, and Y. Liu, "Kolmogorov arnold informed neural network: A physics-informed deep learning framework for solving forward and inverse problems based on kolmogorov arnold networks," 2024. [Online]. Available: www.arxiv.org/abs/2406.11045

[28] J. Vorwerk, T. Zufferey, P. Aristidou, and G. Hug, "Using quantile forecasts for dynamic equivalents of active distribution grids under uncertainty," in *2022 IREP Bulk Power System Dynamics and Control Symposium*, 2022.

[29] R. Tipireddy, P. Perdikaris, P. Stinis, and A. Tartakovsky, "A comparative study of physics-informed neural network models for learning unknown dynamics and constitutive relations," 2019. [Online]. Available: www.arxiv.org/abs/1904.04058

[30] F. Djeumou, C. Neary, E. Goubault, S. Putot, and U. Topcu, "Neural networks with physics-informed architectures and constraints for dynamical systems modeling," 2022. [Online]. Available: www.arxiv.org/abs/2109.06407

[31] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[32] E. Hairer, S. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I Nonstiff problems*, 2nd ed. Berlin: Springer, 2000.

[33] K. Leng and J. Thiyagalingam, "On the compatibility between neural networks and partial differential equations for physics-informed learning," 2023. [Online]. Available: www.arxiv.org/abs/2212.00270

[34] A. Bonfanti, R. Santana, M. Ellero, and B. Gholami, "On the generalization of pinns outside the training domain and the hyperparameters influencing it," *Neural Computing and Applications*, Sep 2024. [Online]. Available: www.doi.org/10.1007/s00521-024-10178-2

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

[36] P. Kundur, *Power System Stability and Control*, 2nd ed. McGraw-Hill, 1994.

[37] S. Wang, S. Sankaran, H. Wang, and P. Perdikaris, "An expert's guide to training physics-informed neural networks," 2023. [Online]. Available: www.arxiv.org/abs/2308.08468

[38] P. Foundation, "Pytorch," last accessed on 23-11-2024. [Online]. Available: www.pytorch.org/

[39] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, pp. 503–528, Aug 1989. [Online]. Available: https://doi.org/10.1007/BF01589116

[40] I. Karampinis, P. Ellinas, I. V. Nadal, R. Nellikkath, and S. Chatzivasileiadis, "Toolbox for developing physics informed neural networks for power systems components," 2025. [Online]. Available: https://arxiv.org/abs/2502.06412

[41] "GitHub - radiakos/PowerPINN — github.com," https://github.com/radiakos/PowerPINN, [Accessed 25-03-2025].

[42] E. Commission, "Ethics guidelines for trustworthy ai," www.ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai, 2019, accessed: 2024-04-27.

[43] IEEE, "Ethically aligned design: A vision for prioritizing human well-being with autonomous and intelligent systems," www.ethicsinaction.ieee.org/, 2019, version 2, Accessed: 2024-04-27.

[44] R. Roscher, B. Bohn, M. F. Duarte, and J. Garcke, "Explainable machine learning for scientific insights and discoveries," *IEEE Access*, vol. 8, pp. 42 200–42 216, 2020.

[45] D. Caughlin, "Verification, validation, and accreditation (vv&a) of models and simulations through reduced order metamodels," in

Proceedings of the 27th Conference on Winter Simulation, ser. WSC '95. USA: IEEE Computer Society, 1995, p. 1405–1412. [Online]. Available: https://doi.org/10.1145/224401.224832

[46] T. De Ryck and S. Mishra, "Generic bounds on the approximation error for physics-informed (and) operator learning," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 10 945–10 958. [Online]. Available: www.proceedings.neurips.cc/paper_files/paper/2022/file/46f0114c06524debc60ef2a72769f7a9-Paper-Conference.pdf

[47] T. D. Ryck and S. Mishra, "Error analysis for deep neural network approximations of parametric hyperbolic conservation laws," 2022. [Online]. Available: www.arxiv.org/abs/2207.07362

[48] T. De Ryck, A. D. Jagtap, and S. Mishra, "Error estimates for physics-informed neural networks approximating the navier–stokes equations," *IMA Journal of Numerical Analysis*, vol. 44, no. 1, pp. 83–119, 01 2023. [Online]. Available: www.doi.org/10.1093/imanum/drac085

[49] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," 2018. [Online]. Available: www.arxiv.org/abs/1811.00866

[50] K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh, "Automatic perturbation analysis for scalable certified robustness and beyond," 2020. [Online]. Available: www.arxiv.org/abs/2002.12920

[51] P. Ellinas, R. Nellikath, I. Ventura, J. Stiasny, and S. Chatzivasileiadis, "Correctness verification of neural networks approximating differential equations," 2024. [Online]. Available: www.arxiv.org/abs/2402.07621

[52] Z. Shi, Y. Wang, H. Zhang, Z. Kolter, and C.-J. Hsieh, "Efficiently computing local lipschitz constants of neural networks via bound propagation," 2022. [Online]. Available: www.arxiv.org/abs/2210.07394

[53] M. Althoff, G. Frehse, and A. Girard, "Set propagation techniques for reachability analysis," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 369–395, 2021.

[54] Y. Yang, T. Wang, J. P. Woolard, and W. Xiang, "Guaranteed approximation error estimation of neural networks and model modification," *Neural Networks*, vol. 151, pp. 61–69, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608022001010

[55] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *Computer Aided Verification*, N. Sharygina and H. Veith, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 258–263.

[56] X. Chen and S. Sankaranarayanan, "Decomposed reachability analysis for nonlinear systems," in *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2016, pp. 13–24.

[57] Z. B. Zabinsky, R. L. Smith, and B. P. Kristinsdottir, "Optimal estimation of univariate black-box lipschitz functions with upper and lower error bounds," *Computers Operations Research*, vol. 30, no. 10, pp. 1539–1553, 2003, part Special Issue: Analytic Hierarchy Process. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054802000825

[58] F. Fourati, S. Kharrat, V. Aggarwal, and M.-S. Alouini, "Every call is precious: Global optimization of black-box functions with unknown lipschitz constants," 2025. [Online]. Available: https://arxiv.org/abs/2502.04290

[59] G. Serré, P. Beja-Battais, S. Chirrane, A. Kalogeratos, and N. Vayatis, "Lipo+: Frugal global optimization for lipschitz functions," 2024. [Online]. Available: https://arxiv.org/abs/2406.19723

[60] Z. C. Lipton, "The mythos of model interpretability," *Queue*, vol. 16, no. 3, pp. 31–57, 2016. [Online]. Available: www.arxiv.org/abs/1606.08346

[61] P. W. Sauer and M. A. Pai, *Power system dynamics and stability*, 1st ed. Upper Saddle River, N.J: Prentice Hall,, 1998.

[62] K. Brenan, S. Campbell, and L. Petzold, *Numerical Solution of Initial-value Problems in Differential-algebraic Equations*, ser. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1996. [Online]. Available: www.books.google.es/books?id=Yxppk7ryf_8C

[63] M. La Scala and A. Bose, "Relaxation/newton methods for concurrent time step solution of differential-algebraic equations in power system dynamic simulations," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 40, no. 5, pp. 317–330, 1993.

[64] I. V. Nadal, J. Stiasny, and S. Chatzivasileiadis, "Integrating physics-informed neural networks into power system dynamic simulations," 2024. [Online]. Available: https://arxiv.org/abs/2404.13325

[65] R. Yu and R. Wang, "Learning dynamical systems from data: An introduction to physics-guided deep learning," *Proceedings of the National Academy of Sciences*, vol. 121, no. 27, p. e2311808121, 2024.

[66] "GitHub - ignvenad/PINNs-for-IEEE-systems — github.com," https://github.com/ignvenad/PINNs-for-IEEE-systems, [Accessed 25-03-2025].

[67] "GitHub - jbesty/PINNSim — github.com," https://github.com/jbesty/PINNSim, [Accessed 25-03-2025].

[68] G. K. R. Lau, A. Hemachandra, S.-K. Ng, and B. K. H. Low, "PINNACLE: PINN adaptive collocation and experimental points selection," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=GzNaCp6Vcg

APPENDIX A
SYNCHRONOUS GENERATOR PARAMETERS

TABLE III
SM PARAMETERS

Symbol	Unit	Parameter	Value
D	p.u.	Damping factor	2
E_{fd}	p.u.	Field excitation voltage	1
H	s	Inertia constant	5.06
P_m	p.u.	Mechanical power input	0.7
R_s	p.u.	Stator resistance	0
T'_d	s	Direct-axis transient time constant	4.75
T'_q	s	Quadrature-axis transient time constant	1.6
X_d	p.u.	Direct-axis synchronous reactance	1.25
X'_d	p.u.	Direct-axis transient reactance	0.232
X_q	p.u.	Quadrature-axis synchronous reactance	1.22
X'_q	p.u.	Quadrature-axis transient reactance	0.715
X''_d	p.u.	Direct-axis subtransient reactance (from Kundur)	0.2
T''_d	s	Direct-axis subtransient time constant	0.05
X''_q	p.u.	Quadrature-axis subtransient reactance	0.25
T''_q	s	Quadrature-axis subtransient time constant	0.04

TABLE IV
LINE AND SYSTEM PARAMETERS

Symbol	Unit	Parameter	Value
X_{ep}	p.u.	Reactance of the line between SM and Bus	0.1
R_e	p.u.	Resistance of the line between SM and Bus	0
ω_b	rad/s	Angular frequency of the system	314.159
V_s	p.u.	System voltage magnitude	1
θ_{vs}	rad	Voltage phase angle	0

APPENDIX B
DEMONSTATIVE RESULTS FOR PINNS

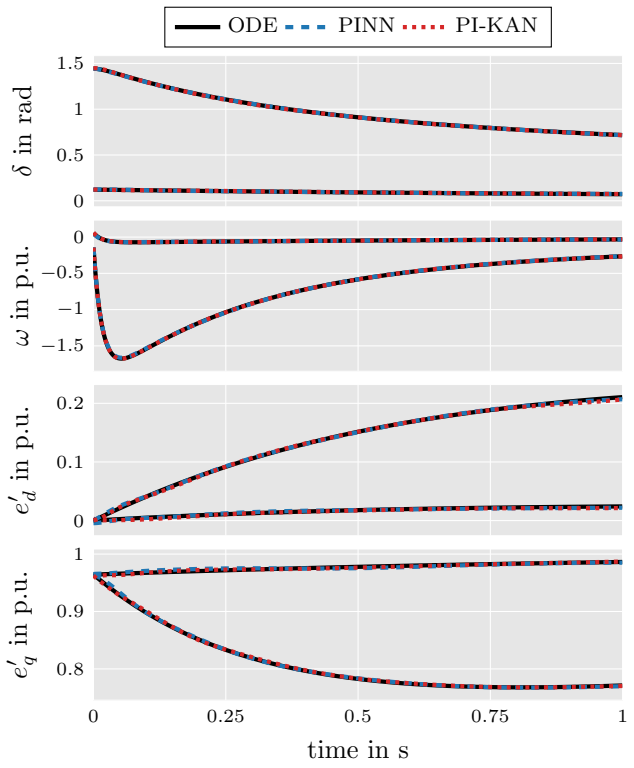

Fig. 25. Performance comparisons of PINN-based MLPs and KANs for the $4D$ case

TABLE V
THE INITIAL CONDITION SET $\mathcal{U}_0$

Variable	Unit	Range	No. samples
θ	rad	$[-2, 2]$	10
ω	rad/s	$[-1, 1]$	10
E'_d	p.u.	[0]	1
E'_q	p.u.	[0.9, 1.1]	5
E''_d	p.u.	[0]	1
E''_q	p.u.	[1]	1

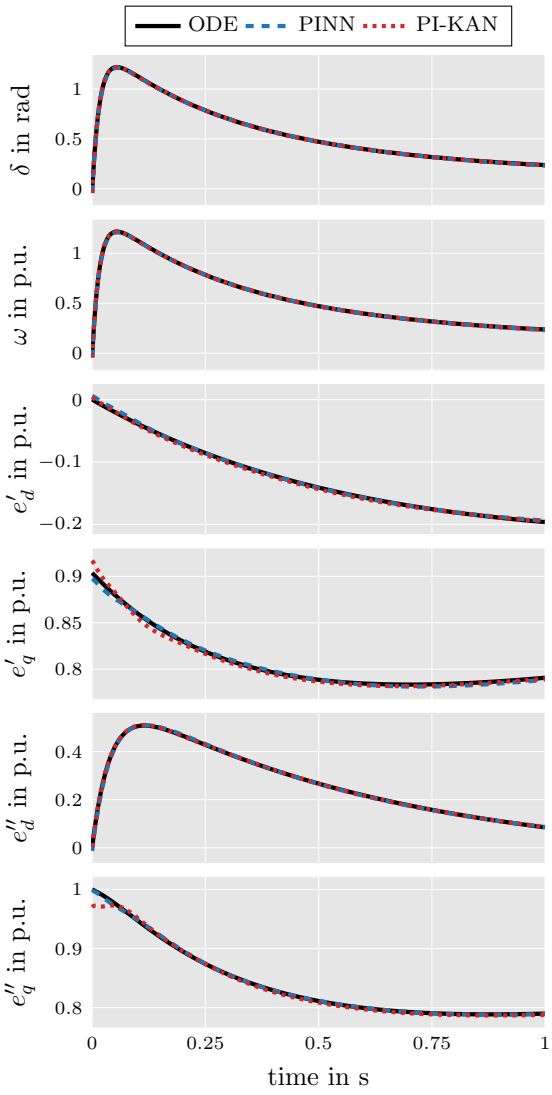

Fig. 26. Performance comparisons of PINN-based MLPs and KANs for the $6D$ case

APPENDIX C
ALGORITHMS FOR ASSESSMENT OF TRUSTWORTHINESS

A. CROWN

CROWN [49] is a framework designed to certify the robustness of NNs with general activation functions against adversarial perturbations. Unlike methods limited to ReLU activations, CROWN extends certification to functions such as tanh, sigmoid, and arctan.

The core idea is to bound each activation function $\sigma(\mathbf{y})$ using linear or quadratic functions:

$$h_L(\mathbf{y}) \leq \sigma(\mathbf{y}) \leq h_U(\mathbf{y}),$$

where $h_L(\mathbf{y})$ and $h_U(\mathbf{y})$ are adaptively chosen based on the input range of each neuron, reducing approximation errors and leading to tighter certified bounds.

For a NN function $f(x)$ with input x perturbed within an ℓ_p-norm ball around x_0, CROWN provides explicit upper and lower bounds:

$$f_L(x) = \Omega^{(0)}x + \sum_{k=1}^{m} \Omega^{(k)} \left(b^{(k)} + \Theta^{(k)} \right),$$

$$f_U(x) = \Lambda^{(0)}x + \sum_{k=1}^{m} \Lambda^{(k)} \left(b^{(k)} + \Delta^{(k)} \right),$$

where $\Omega^{(k)}$, $\Lambda^{(k)}$, $\Theta^{(k)}$, and $\Delta^{(k)}$ are parameters computed from the network's weights, biases, and the bounds of activation functions at each layer.

By recursively applying these bounds through the network layers, CROWN estimates the minimal perturbation required to change the network's output, thus providing a certified lower bound on adversarial distortion.

CROWN is computationally efficient with polynomial time complexity relative to network size, making it scalable to large networks. Experimental results show that CROWN achieves tight certified bounds.

In conclusion, CROWN advances robustness certification by accommodating general activation functions and improving the tightness of certified bounds through adaptive bounding strategies, all while maintaining computational efficiency.

B. Lipschitz method

Proof. Define the error function $d(\mathbf{x}) = \mathbf{u}(\mathbf{x}) - \hat{\mathbf{y}}(\mathbf{x})$. For any $\mathbf{x}, \mathbf{y} \in \mathcal{X}$:

$$\begin{aligned} \|d(\mathbf{x}) - d(\mathbf{y})\| &= \|[\mathbf{u}(\mathbf{x}) - \hat{\mathbf{y}}(\mathbf{x})] - [\mathbf{u}(\mathbf{y}) - \hat{\mathbf{y}}(y)]\| \\ &= \|\mathbf{u}(\mathbf{x}) - \mathbf{u}(\mathbf{y}) - [\hat{\mathbf{y}}(\mathbf{x}) - \hat{\mathbf{y}}(y)]\| \\ &\leq \|\mathbf{u}(\mathbf{x}) - \mathbf{u}(\mathbf{y})\| + \|\hat{\mathbf{y}}(\mathbf{x}) - \hat{\mathbf{y}}(y)\| \\ &\leq L_u\|\mathbf{x} - \mathbf{y}\| + L_{\hat{\mathbf{y}}}\|\mathbf{x} - \mathbf{y}\| = L_E\|\mathbf{x} - \mathbf{y}\| \end{aligned}$$

Thus, $d(\mathbf{x})$ is Lipschitz continuous with constant L_E. We know that:

$$\|d(\mathbf{x})\| - \|d(\underline{\mathbf{x}})\| \leq \|d(\mathbf{x}) - d(\underline{\mathbf{x}})\|$$

For any $\mathbf{x} \in \mathcal{X}$:

$$\begin{aligned} \|d(\mathbf{x})\| &= \|d(\underline{\mathbf{x}}) + [d(\mathbf{x}) - d(\underline{\mathbf{x}})]\| \\ &\leq \|d(\underline{\mathbf{x}})\| + \|d(\mathbf{x}) - d(\underline{\mathbf{x}})\| \\ &\leq \|u(\underline{\mathbf{x}}) - \hat{\mathbf{y}}(\underline{\mathbf{x}})\| + L_E\|\mathbf{x} - \underline{\mathbf{x}}\| \\ &\leq \|u(\underline{\mathbf{x}}) - \hat{\mathbf{y}}(\underline{\mathbf{x}})\| + L_E\|\overline{\mathbf{x}} - \underline{\mathbf{x}}\| \end{aligned}$$

Taking the supremum over $x \in \mathcal{X}$ yields:

$$E(\mathbf{u}, \hat{\mathbf{y}}, \mathcal{X}) \leq \|\mathbf{u}(\underline{\mathbf{x}}) - \hat{\mathbf{y}}(\underline{\mathbf{x}})\| + L_E\|\overline{\mathbf{x}} - \underline{\mathbf{x}}\|.$$

□

The Lipschitz method bounds the approximation error by combining the initial error at $\underline{\mathbf{x}}$ with the product of the combined Lipschitz constant L_E and the size of the input interval $\|\overline{\mathbf{x}} - \underline{\mathbf{x}}\|$.

C. Envelope Method Algorithm

The algorithm presented systematically, in Fig. 27, analyzes the NN $\hat{\mathbf{y}}$ by iteratively refining input intervals to compute certified bounds or identify potential adversarial examples. It begins by initializing with the network's initial output $\hat{\mathbf{y}}$, an input interval $[\underline{\mathbf{u}}, \overline{\mathbf{u}}]$ within the input space $\mathcal{U}$, a precision threshold Θ, and the number of sub-intervals P for domain splitting. An initial value of κ is computed using a gradient-based attack, and a heap data structure is initialized to manage and prioritize intervals based on their computed β_i values. The algorithm enters a loop where it continues to refine the input intervals as long as the current interval width exceeds the precision threshold or the computed κ remains at its maximum value.

Within each iteration of the loop, the algorithm recomputes κ to incorporate new information, then splits the current input domain into P smaller intervals for a more granular analysis. For each sub-interval, it computes a corresponding β_i value using the CROWN method, which provides certified bounds on the network's outputs to ensure robustness against adversarial perturbations. Intervals with β_i values exceeding the current maximum are added to the heap, and the interval with the highest β_i is extracted for further processing. This iterative refinement continues until the desired precision is achieved, ensuring that the NN's robustness is thoroughly evaluated across the specified input ranges.

D. Clarke Jacobian

This appendix describes the methodology to compute upper bounds for the Clarke Jacobian using the CROWN framework.

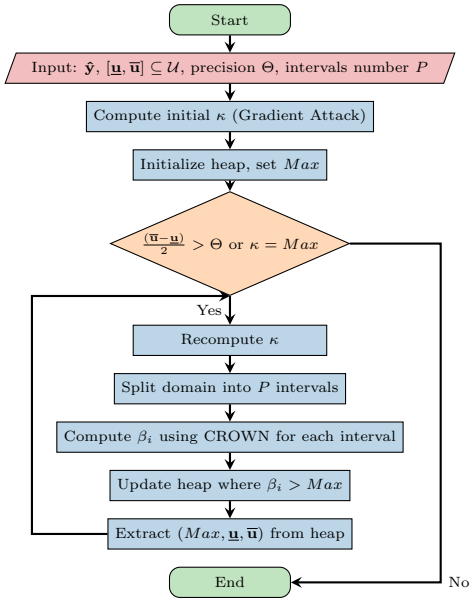

Fig. 27. Envelope method algorithm to solve optimization problem Eq. (27) and Eq. (28).

The focus is on estimating ℓ_∞ local Lipschitz constants in NNs with non-differentiable or differentiable activation functions.

The Clarke Jacobian, $\partial f(x)$, generalizes the Jacobian matrix for non-smooth functions such as NNs with ReLU activations. It is defined as the convex hull of all Jacobians at differentiable points near x. The aim is to bound the ℓ_∞ norm of $\partial f(x)$, which provides an estimate of the local Lipschitz constant.

To achieve this, the computation leverages a computational graph. A computational graph is a directed acyclic graph that represents the sequence of operations performed by a NN. Each node corresponds to an operation (e.g., matrix multiplication, addition, or activation functions), and edges represent the data flow between operations. By augmenting the computational graph with additional nodes and edges for backward propagation, it becomes possible to calculate the norm of the Clarke Jacobian efficiently.

The methodology involves the following steps:

- Formulating the augmented computational graph: This graph extends the standard forward pass graph with a backward graph for computing the Clarke Jacobian. The backward graph relies on intermediate pre-activation values ($z_i(x)$) from the forward pass to compute the Clarke gradient of activations.
- Linear bound propagation: Nonlinear functions in the backward graph (e.g., ReLU and $\|\cdot\|_\infty$) are relaxed using tight linear bounds. For example:
  - For a ReLU activation $\sigma(x) = \max(x, 0)$, if $l \leq x \leq u$, the upper bound is a line passing through $(l, \sigma(l))$ and $(u, \sigma(u))$, while the lower bound is a horizontal line at zero.
  - For differentiable activations (e.g., sigmoid or tanh), bounds can be propagated directly using their derivatives or approximations.
- Bounding the Clarke gradient: For each layer, the Clarke gradient is relaxed using linear functions. For example, for a neuron j in layer i, given bounds $[l, u]$ on z_j:

$$\min(0, l) \leq \Delta_i(x)_{jj} \leq \max(0, u),$$

  where $\Delta_i(x)_{jj}$ is the Clarke gradient for neuron j.
- Backward bound propagation: Using the chain rule, bounds are propagated backward from the output layer to the input. Each layer's weights and the Clarke gradient relaxations contribute to the overall bounds.
- Computing the final norm: After propagating the bounds to the input, the ℓ_∞ norm of the Clarke Jacobian is computed as:

$$\|J(x)\|_\infty = \max_{1 \leq k \leq K} \sum_{j=1}^{d} |J_{kj}(x)|,$$

  where $J(x)$ is the Jacobian matrix, K is the output dimension, and d is the input dimension.

To further refine the bounds, a branch-and-bound approach can be employed. The input domain is divided into smaller subdomains, and the linear bound propagation is applied to each subdomain. The results are then combined to obtain tighter overall bounds.

This method is applicable not only to ReLU networks but also to networks with differentiable activation functions such as sigmoid, tanh, and softplus. For such functions, the Clarke Jacobian coincides with the standard Jacobian, and bounds can be derived directly using their derivatives.

Notice that we can bound the Lipschitz constant of the equation, by applying a norm on the Jacobian of the final NN layer.

E. Reachability Method

Proof. Let $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are intervals in $\mathbb{R}^n$ defined by their component-wise bounds:

$$\underline{\mathbf{x}}_i \leq \mathbf{x}_i \leq \overline{\mathbf{x}}_i, \quad \underline{\mathbf{y}}_i \leq y_i \leq \overline{\mathbf{y}}_i, \quad \text{for } i = 1, \ldots, n.$$

Our goal is to bound the Euclidean distance $\|\mathbf{x}-\mathbf{y}\|$ between any $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$.

First, for each coordinate i, the maximum possible difference between x_i and y_i is given by:

$$|x_i - y_i| \leq d_{i,\max}, \quad \text{where} \quad d_{i,\max} = \max\left\{|\underline{\mathbf{x}}_i - \overline{\mathbf{y}}_i|,\ |\overline{\mathbf{x}}_i - \underline{\mathbf{y}}_i|\right\}.$$

This accounts for the largest distance between the bounds of $\mathcal{X}$ and $\mathcal{Y}$ in each dimension.

Now, since $|x_i - y_i| \leq d_{i,\max}$ for all i, we have:

$$(x_i - y_i)^2 \leq (d_{i,\max})^2.$$

Using the definition of the Euclidean norm:

$$\|\mathbf{x} - \mathbf{y}\| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \leq \sqrt{\sum_{i=1}^{n} (d_{i,\max})^2}.$$

Therefore, the maximum possible Euclidean distance between any $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$ is bounded by:

$$\|\mathbf{x} - \mathbf{y}\| \leq D_{\max}, \quad \text{where} \quad D_{\max} = \sqrt{\sum_{i=1}^{n} (d_{i,\max})^2}.$$

This provides an explicit upper bound based on the intervals defining $\mathcal{X}$ and $\mathcal{Y}$. □