Integrating Physics-Informed Neural Networks and Power Flow for Scalable Online Transient Analysis

Zhangrong Gu*, Yue Zhao†, Meng Yue‡, Tianqiao Zhao‡, Jiaming Li§
*Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY 11794, USA
*Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794, USA
‡Interdisciplinary Science Department, Brookhaven National Laboratory, Upton, NY 11973, USA
§Ads, Meta, New York, NY 10003, USA
Emails: {zhangrong.gu, yue.zhao.2}@stonybrook.edu, {yuemeng, tzhao1}@bnl.gov, nhlijiaming@gmail.com

***Abstract*—Transient dynamic analysis is crucial for power system stability, particularly with increasing renewable integration, yet traditional numerical integration methods are computationally expensive for large-scale online applications. We propose a novel framework integrating Physics-Informed Neural Networks (PINNs) with an AC Power Flow (AC-PF) solver to efficiently simulate dynamic trajectories in power systems in a scalable fashion. A separate PINN is trained for each generator to accurately capture the differential equations of the generator dynamic model. The trained PINNs and an AC-PF solver iteratively update the dynamic and algebraic variables to simulate dynamic trajectories for the entire system. The PINN training process consists of an unsupervised stage to enforce physical laws, followed by a supervised stage leveraging simulation data for enhanced accuracy. As the predictor training is performed separately for individual generators, the method's computational complexity scales linearly with the number of generators in both training and testing. Experiments on a 3-generator 9-bus system demonstrate the very high accuracy of the developed method in simulating full system dynamic trajectories.**

***Index Terms*—Transient analysis, PINN, power flow, iterative algorithm, scalability.**

I. Introduction

Power grids are the backbone of modern energy systems. With rising levels of integration of renewable energy sources, more challenges arise for power grid operations in maintaining stability and reliability. Transient dynamic analysis plays a critical role in assessing the behavior of power systems in the presence of disturbances such as faults and power imbalances. These analyses often require solving large-scale Differential-Algebraic Equations (DAEs) including both the differential equations describing the dynamics of generators and the algebraic equations describing instantaneous relations between system variables. The common practice of solving such DAEs relies on numerical integration methods [1]. While such methods provide accurate results, they are however computationally too expensive for online transient dynamic analysis of large-scale power systems. An alternative approach for stability analysis is the direct method [2] which however does not provide full dynamic trajectories of the system.

This work is supported in part by ONR under Award N00014-22-1-2001 and Award N00014-23-1-2124, and in part by the National Science Foundation under Grant ECCS-2025152.

Recently, machine learning (ML) techniques have been explored as promising alternatives for solving DAEs in power systems. These include Graph Neural Networks [3], Fourier Neural Operators [4], Recurrent Neural Networks [5], and Physics-Informed Neural Networks (PINNs). Among these, PINNs have gained particular attention due to their ability to incorporate physical laws directly into the predictor model and learning process. The foundational work of [6] introduced PINNs as a deep learning framework for solving nonlinear partial differential equations, laying the groundwork for embedding physical laws directly into neural networks. This framework was then applied to power systems [7], demonstrating the effectiveness of PINNs in generating dynamic states in a Single-Machine Infinite-Bus (SMIB) system. [8] further applied PINNs for dynamic state estimation in low-inertia power grids, focusing on frequency dynamics.

While these works highlighted the potential of PINNs, scalability remains a significant challenge when applied to larger and more complex power systems. To address the scalability issue, [9] proposed a hybrid approach that trains a separate Long Short-Term Memory (LSTM)-based predictor for each generator, and the predictors are then coupled with each other via an AC Power Flow (AC-PF) solver to generate system trajectories iteratively. In addition, [10] introduced PINNSim, a method that uses PINNs to model the individual dynamics of power system components, iteratively adjusting the voltage profile by minimizing mismatches with network requirements via Jacobian-based updates.

In this paper, we propose an efficient method by integrating PINNs with an AC-PF solver for simulating dynamic trajectories in a scalable fashion. We train a separate PINN for each generator to model its dynamic behaviors and use AC-PF to update algebraic variables (e.g., bus voltages) of the entire system. An iterative algorithm alternates between predicting generator states with PINNs and solving AC-PF, eliminating the need for traditional numerical integration. As such, the method's complexity scales *linearly* with the number of generators in both training and testing. A two-stage predictor training procedure is developed: an unsupervised stage based on physical laws (i.e., differential equations), followed by a supervised stage with simulation data to achieve further accuracy. We evaluate the method's performance in

a 3-generator 9-bus power system, demonstrating its great accuracy in simulating full system dynamic trajectories.

II. Problem Description

Consider an m-generator n-bus system governed by DAEs:

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), \mathbf{z}(t)) \text{ (differential equations)} \quad (1)$$

$$0 = g(\mathbf{x}(t), \mathbf{z}(t)) \text{ (algebraic equations)} \quad (2)$$

where $\mathbf{x}(t) = [\mathbf{x}_1(t), \mathbf{x}_2(t), \ldots, \mathbf{x}_m(t)]$ is the vector of state variables for the m generators, while $\mathbf{z}(t) = [\mathbf{z}_0(t), \mathbf{z}_1(t), \mathbf{z}_2(t), \ldots, \mathbf{z}_m(t)]$ is the vector of algebraic variables in the system. Specifically, with the buses in the system partitioned into m generator terminal buses and $n-m$ non-generator buses, (a) $\mathbf{x}_i(t)$ represents the state variables of a generator i, (b) $\mathbf{z}_i(t)$ represents the algebraic variables of the corresponding generator terminal bus, and (c) $\mathbf{z}_0(t)$ represents the remaining algebraic variables of all non-generator buses. For example, with a second-order (i.e., classical) dynamic model of generators characterized by swing equations,

$$\mathbf{x}_i(t) = [\delta_i(t), \omega_i(t)]^T, \quad (a)$$

$$\mathbf{z}_i(t) = [|V_i(t)|, \angle V_i(t)]^T, \quad (b)$$

$$\mathbf{z}_0(t) = [\text{complex voltages of all non-generator buses}] \quad (c)$$

where $\delta_i(t)$ is generator i's rotor angle, $\omega_i(t)$ is its rotor speed, and $|V_i(t)|$ and $\angle V_i(t)$ represent the magnitude and angle of generator terminal bus' complex voltage $V_i(t)$. Given a set of initial values of $\mathbf{x}(t)$ and $\mathbf{z}(t)$, the DAEs (1) and (2) can be solved using various numerical integration methods to determine the trajectories of all quantities over time. We note that, the algebraic equations (2) can experience multiple changes in the presence of changes in the system (e.g., faults).

The goal of this work is to simulate dynamic trajectories in power systems with predictors in a scalable fashion, instead of resorting to numerical integration methods. To achieve this, we will (a) replace the dynamic computation of each generator with a trained predictor, while (b) connect all the predictors by solving AC-PF for the power system. In this approach, our algorithm iteratively alternates between a) calling trained predictors and b) solving AC-PF to dynamically update the external inputs to the predictors (i.e., the boundary conditions of (1)). Furthermore, given closed-form expressions of the differential equations (1) of generator dynamics, PINNs can be trained that effectively capture (1) in the predictor model.

III. Methodology

A. Physics-Informed Neural Networks and Their Limitation

We begin with a discussion of PINNs in power systems and their limitation. PINNs are neural networks designed to solve differential equations by embedding physical laws directly into the predictor model and learning process. Consider a SMIB system described by Ordinary Differential Equations (ODEs):

$$\frac{d\mathbf{x}_i(t)}{dt} = f(\mathbf{x}_i(t), \mathbf{z}_i), \quad t \in [0, T], \quad (3)$$

where $\mathbf{z}_i$ captures the constant boundary conditions, and $f(\mathbf{x}_i(t), \mathbf{z}_i)$ describes the generator dynamics. A PINN aims to approximate the solution $\mathbf{x}_i(t|\mathbf{x}_i(0), \mathbf{z}_i)$ of the ODE given the initial conditions $\mathbf{x}_i(0)$ and boundary conditions $\mathbf{z}_i$. In this context, we let $H(\mathbf{x}_i(0), \mathbf{z}_i, t; \theta)$ represent a neural network (NN), where θ denotes its parameters. By querying the NN, one can generate the (approximated) generator trajectory with $\hat{\mathbf{x}}_i(t|\mathbf{x}_i(0), \mathbf{z}_i) = H(\mathbf{x}_i(0), \mathbf{z}_i, t; \theta), \forall t$. In addition to satisfying the initial and boundary conditions, $H(\mathbf{x}_i(0), \mathbf{z}_i, t; \theta)$ is trained to minimize the residual of the ODE [11], denoted by $r(t)$,

$$r(t) := \frac{dH(\mathbf{x}_i(0), \mathbf{z}_i, t; \theta)}{dt} - f(H(\mathbf{x}_i(0), \mathbf{z}_i, t; \theta), \mathbf{z}_i), \quad (4)$$

where $\frac{dH(\mathbf{x}_i(0), \mathbf{z}_i, t; \theta)}{dt}$ is computed using automatic differentiation, allowing precise and efficient calculation of gradients during backpropagation. Accordingly, the training process primarily involves minimizing a physics-based mean squared error (MSE) loss, denoted as $\text{MSE}_{\text{physics}}$. The physics loss is designed to enforce compliance with the underlying ODE at a set of collocation points $\{t_c\}_{c=1}^{N_c}$ within an interval $[0, T]$:

$$\text{MSE}_{\text{physics}} = \frac{1}{N_c} \sum_{c=1}^{N_c} |r(t_c)|^2 \quad (5)$$

This physics loss serves as a critical component of the training objective, ensuring that the learned NN satisfies the physical laws of the power system. Importantly, minimizing $\text{MSE}_{\text{physics}}$ does *not* require solving the ODEs. Therefore, in principle, PINNs as a solution to ODEs can be learned entirely via *unsupervised* learning by minimizing the residuals of the ODEs, effectively setting the training objective to be the satisfaction of the system's physical law.

Notably, in the SMIB system, the boundary conditions of the ODEs, $\mathbf{z}_i$, are assumed to be *constant* [7]. This is critical for PINNs to be feasible to perform as expected: the neural networks can predict the system state directly based on the queried time given the initial conditions $\mathbf{x}_i(0)$, and then the dynamics of the system can be generated for all future times given the constant boundary conditions.

However, to extend the use case to more complex systems with multiple generators, PINN encounters scalability challenges. Specifically, straightforward applications of PINN would entail modeling the entire system with a single predictor, whose complexity can quickly rise as the system size increases. An alternative is to model each generator with a PINN. However, the assumption of constant boundary conditions no longer holds, as the complex voltages at all the buses continuously vary over time due to the interactions among the generators. This reality limits the use of PINNs which are neural networks trained with constant boundary conditions for querying future trajectory values at any time directly. To overcome this challenge, we will next develop an iterative framework that allows the system to iteratively update boundary conditions as the simulation progresses.

B. A Scalable Iterative Algorithmic Framework

We now introduce an iterative algorithmic framework designed to dynamically update boundary conditions (e.g., complex voltages of generator terminal buses) and describe how it interacts with the PINN-based predictors.

In a multi-generator system, we train a PINN for each generator separately. Different from prior work that queries all future times in the trajectories, our method employs an iterative framework that integrates PINNs with an AC-PF solver to simulate system dynamics. Specifically, during testing with trained PINNs, the procedure alternates iteratively between two steps — predicting generator states using PINNs and solving AC-PF to update the algebraic variables — to generate the full system trajectories. The process proceeds as follows (cf. Algorithm 1): at each time step, for each generator i, its PINN reads its previous states $\mathbf{x}_i(t-1)$ and its previous corresponding algebraic variables $\mathbf{z}_i(t-1)$ (boundary conditions). Based on these inputs, the generator states $\mathbf{x}_i(t)$ are predicted. After obtaining the updated states of all generators, $\mathbf{x}(t)$, these and the algebraic variables $\mathbf{z}(t-1)$ are then input into the AC-PF solver to compute the updated algebraic variables (i.e. complex system voltages) $\mathbf{z}(t)$ across the entire system. This iterative process continues and the states of all generators $\{\mathbf{x}(t)\}_{t=1}^{T}$ and algebraic variables $\{\mathbf{z}(t)\}_{t=1}^{T}$ over the time horizon T are obtained. We note that, during this process, there could be system changes occurring at times (e.g., faults, generation/load changes). Whenever a system change occurs, it would simply update the AC-PF solver and/or generators' boundary conditions for the trajectories thereafter.

Algorithm 1 Iterative Framework of Integrating PINNs and AC-PF

Input: Initial generator states $\mathbf{x}(0)$ and algebraic variables $\mathbf{z}(0)$.

Output: Generator states $\{\mathbf{x}(t)\}_{t=1}^{T}$ at all generators and algebraic variables $\{\mathbf{z}(t)\}_{t=1}^{T}$ at all buses over time T.

1: **for** $t \leftarrow 1$ to T **do**
2: **if** A system change occurred at time t **then**
3: Update admittance matrix $\mathbf{Y}_{\text{bus}}$ accordingly.
4: **end if**
5: **for** $i \leftarrow$ *Generator* **do**
6: $\mathbf{x}_i(t) \leftarrow \text{PINN\_Predict}(\mathbf{x}_i(t-1), \mathbf{z}_i(t-1))$ ▷ Predict next generator states with PINN.
7: **end for**
8: Update $\mathbf{z}(t) \leftarrow \text{AC\_PF}(\mathbf{x}(t), \mathbf{z}(t-1))$ ▷ Solve AC-PF to update algebraic variables.
9: **end for**
10: **return** $\{\mathbf{x}(t)\}_{t=1}^{T}$, $\{\mathbf{z}(t)\}_{t=1}^{T}$

A diagram is depicted in Fig. 1 to illustrate the flow of variables fed into the AC-PF solver and the PINNs. The diagram provides an intuitive sense of how the system dynamics are updated (illustrated from time index t to $t+2$).

In addition to ensuring that the PINNs are dynamically informed by continuous changes in boundary conditions, a key advantage of the proposed algorithm is its *scalability*. Since a predictor is trained for each generator, the method's computational complexity scales *linearly* in the number of generators, i.e., with a computational complexity of $O(m)$, in *both training and testing*. Moreover, it can easily accommodate additional generators without retraining the existing predictors. Similarly, the predictors also need not be retrained under changes to the power system, such as those caused by faults.

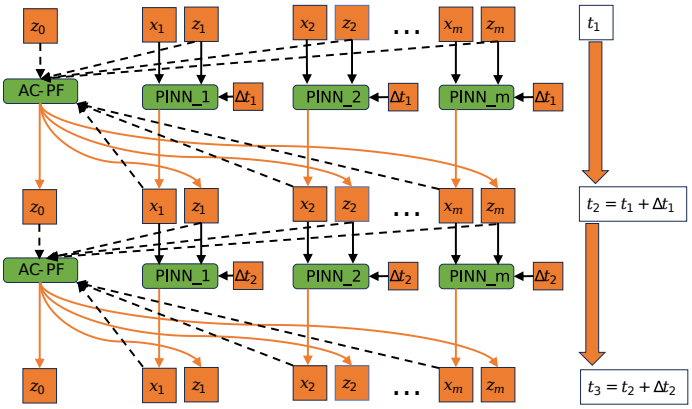

Fig. 1: Diagram of the iterative algorithmic framework in an m-generator system. $\mathbf{x}_{1,2\ldots m}$ and $\mathbf{z}_{1,2\ldots m}$ represent the state variables of m generators and algebraic variables of the corresponding generator terminal buses, respectively. $\mathbf{z}_0$ represents the algebraic variables of the other $n-m$ non-generator buses in the system. Δt_1 and Δt_2 are the step sizes between the consecutive time indices. The black lines indicate the input directions while the orange lines are for the output.

C. Training of PINN-based Predictors

For training high-accuracy PINNs, we propose a two-stage strategy consisting of an unsupervised training stage followed by a supervised training stage. In principle (cf. Section III-A), a PINN-based predictor can be trained via unsupervised learning without any labeled data. Nonetheless, to further improve the predictor's accuracy, we employ a supervised training stage utilizing some simulation data after the unsupervised training phase. Since the PINNs for different generators are trained independently, without loss of generality, we describe the training strategy of a PINN for generator i in the following.

Here, while the training strategy is generally applicable to other generator models, we use a second-order dynamic model as an illustrative example. As such, the inputs and outputs for a PINN are as follows:

a) Inputs: The input variables for the predictor include the time step size between two consecutive time indices, Δt; the relative rotor speed $\Delta\omega_i(t) = \omega_i(t) - \omega_{\text{synchronous}}$; the angle difference $\Delta\delta_i(t) = \delta_i(t) - \angle V_i(t)$; and the product of the winding voltage and generator terminal bus voltage magnitude, $|E_{q(i)}| \cdot |V_i(t)|$, where $E_{q(i)}$ is the complex winding voltage of generator i.

b) Outputs: The output variables of the predictor include the relative rotor speed for the next time step, $\Delta\omega_i(t+1)$; and the predicted angle difference $\Delta\delta_i(t+1) = \delta_i(t+1) - \angle V_i(t)$.

The complex bus voltage $V_i(t)$ (formed by $\angle V_i(t)$ and $|V_i(t)|$) is subsequently updated via AC-PF.

In the unsupervised training stage, an initial condition loss, $\text{MSE}_{\text{initial}}$ is included to ensure that the predicted values match known initial conditions:

$$\begin{aligned}\text{MSE}_{\text{initial}} =& \frac{1}{N_u}\sum_{u=1}^{N_u}|\Delta\hat{\delta}_i(0)-\Delta\delta_i(0)|^2 \\ &+\frac{1}{N_u}\sum_{u=1}^{N_u}|\Delta\hat{\omega}_i(0)-\Delta\omega_i(0)|^2. \end{aligned} \tag{6}$$

where $\Delta\hat{\delta}_i(0)$ and $\Delta\hat{\omega}_i(0)$ represent the predicted outputs when $\Delta\delta_i(0)$ and $\Delta\omega_i(0)$ are fed into the predictor. Here, N_u is the number of input sets $(\Delta\delta_i(0), \Delta\omega_i(0))$ which are randomly sampled within their respective domains. The combined unsupervised loss, $\mathcal{L}_{\text{unsupervised}}$, is then defined as

$$\mathcal{L}_{\text{unsupervised}} = \alpha_{\text{physics}} \cdot \text{MSE}_{\text{physics}} + \alpha_{\text{initial}} \cdot \text{MSE}_{\text{initial}} \tag{7}$$

where α_{physics} and α_{physics} are weighting coefficients, and $\text{MSE}_{\text{physics}}$ is the physics loss (5). The unsupervised training stage enforces the predictor to satisfy both the governing physical laws and any initial conditions.

In the supervised training stage, the loss function is composed of the MSE loss terms for the outputs, specifically for relative rotor speed and angle difference predictions. The supervised loss $\mathcal{L}_{\text{supervised}}$ is defined as

$$\begin{aligned}\mathcal{L}_{\text{supervised}} =& \alpha_{\Delta\omega}\cdot\frac{1}{K\cdot T}\sum_{k=1}^{K}\sum_{i=1}^{T}\left(\Delta\hat{\omega}_i^{(k)}(t)-\Delta\omega_i^{(k)}(t)\right)^2 \\ &+\alpha_{\Delta\delta}\cdot\frac{1}{K\cdot T}\sum_{k=1}^{K}\sum_{i=1}^{T}\left(\Delta\hat{\delta}_i^{(k)}(t)-\Delta\delta_i^{(k)}(t)\right)^2 \end{aligned} \tag{8}$$

where $\alpha_{\Delta\omega}$ and $\alpha_{\Delta\delta}$ are weighting coefficients, K represents the number of simulated cases, while T is the total number of time steps for each trajectory. The terms $\Delta\hat{\omega}_i^{(k)}(t)$ and $\Delta\hat{\delta}_i^{(k)}(t)$ are the predicted values of relative rotor speed and angle difference for generator i at time step t within the k-th trajectory, while $\Delta\omega_i^{(k)}(t)$ and $\Delta\delta_i^{(k)}(t)$ are the ground truths.

Importantly, the unsupervised learning stage ensures consistency with the physical law while reducing the amount of training data needed in the supervised learning stage. It thus improves both model generalizability/robustness and computational efficiency with limited supervised training data.

IV. Case Study and Discussion

A. Data Generation

We perform time-domain simulations to generate system trajectories with an open-source transient stability simulation program PYPOWER-Dynamics [12]. These ground-truth trajectories are generated using the Modified Euler integration method and an AC-PF solver provided by this platform. The simulations are performed on the IEEE 3-Generator 9-Bus Power System (cf. Fig. 2). Each generator is modeled as a synchronous machine using a second-order (i.e., classical) dynamic model, represented by the swing equations as follows:

$$\frac{d\delta_i(t)}{dt} = \Delta\omega_i(t), \tag{9}$$

$$\frac{d\omega_i(t)}{dt} = \frac{1}{2H_i}\left(P_{m,i} - P_{e,i} - D_i\Delta\omega_i(t)\right), \tag{10}$$

where H_i, $P_{m,i}$, and D_i are the inertia constant, mechanical power input, and damping coefficient of generator i, respectively. The electrical power output, $P_{e,i}$, is determined by the network and load as:

$$P_{e,i} = \frac{|E_{q(i)}|\cdot|V_i(t)|\cdot\sin(\delta_i(t)-\angle V_i(t))}{X_{dp}},$$

where X_{dp} is the d-axis transient reactance. The trajectories of generator dynamics (i.e. rotor angles and rotor speeds in our experiment) along with the algebraic variables (i.e., bus voltages) of the system are simulated based on the platform. Each trajectory is simulated with a sampling period of 0.1s for a total length of 10s. We introduce two load changes as disturbances to the system. The first load change happens at the start of the simulation: the load of bus 5 in the system reduces to a random level within [0MW, 75MW] from 125MW. The load is restored at a random time later in the interval [2s, 4s]. 3,800 cases are simulated for the supervised training stage and another 1,000 cases are simulated for testing.

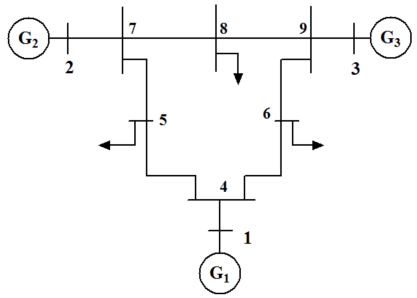

Fig. 2: Diagram of the IEEE 3-generator 9-bus power system.

B. Training and Implementation Details

The Neural network architecture of the predictors is determined via hyperparameter tuning, where we tested configurations ranging from 2 to 5 hidden layers and 50 to 300 neurons per layer. From these, for each predictor, we choose 4 hidden layers with 200 neurons each to balance model expressiveness, computational cost, and overfitting risks. The input features are normalized to the range $[-1, 1]$ using the formula $input = 2\cdot\frac{input-min}{max-min} - 1$. The training is implemented utilizing an NVIDIA RTX 6000 Ada Generation GPU.

As mentioned in III-C, the training process includes both unsupervised and supervised training stages. In the unsupervised training stage, LBFGS optimizer is used with a learning rate of 1.0. The maximum number of iterations is set to 10,000, with a history size of 50. Convergence tolerance is set to -1 for gradients and $1.0\cdot\epsilon$ for parameter changes,

where ϵ is the machine epsilon. The line search function is set to "$strong\_wolfe$", and the optimization process typically stops after approximately 4,000 iterations, which takes around 1,600 seconds in our experiments. This training stage does not rely on any simulated data as labels but instead uses N_u = 100,000 data points for initial conditions, ensuring proper initialization of the system state. Additionally, another N_c = 100,000 data points are used for the physics loss, with these points uniformly distributed over twice the range of the variables encountered in our simulated data. Finally, the loss function (7) is composed of initial condition losses and physics-based losses, with respective weights of $\alpha_{\text{initial}} = 100$, and $\alpha_{\text{physics}} = 0.1$.

In the supervised training stage, the Adam optimizer is utilized with a learning rate of 0.003, an epsilon value of $1\text{e}{-6}$, and the AMSGrad variant enabled. The gradient clip value is set to 1.0 during training. This stage leverages up to K = 3,800 simulated cases that we generated, each containing T = 100 data points, resulting in a total of up to 380,000 data points. The batch size is set to 200. While our default supervised training setting utilizes all 3,800 generated cases with 1,000 training epochs, we also conducted training with fewer generated cases —– specifically, 1,900, 1000, 500, 200, and 100 cases — for comparison. The weighting coefficients $\alpha_{\Delta\omega}$ and $\alpha_{\Delta\delta}$ are determined via hyperparameter tuning for each predictor. Specifically, the weighting ratios for training different generators, G1, G2, and G3, are set as follows: $\alpha_{\Delta\omega} : \alpha_{\Delta\delta} = 1:1$, $1:10$, and $3:2$.

After completion of both training stages, the predictors trained for all generators are ready to be used in dynamic simulations via the iterative testing framework.

C. Numerical Experiment Results

As described in III-B, the iterative algorithm integrating PINNs and AC-PF (see Alg. 1) is implemented to generate each trajectory during testing. Specifically, (a) the states of generators G1-G3 are predicted by three predictors, and (b) the AC-PF is iteratively solved for the 9-bus system connecting these generators to update the algebraic variables of the system.

To evaluate the performance of the trained predictors, we compute the normalized MSE $\mathcal{E}_i$ of the i-th predictor over 1,000 testing cases, given by:

$$\mathcal{E}_i = \frac{1}{2 \cdot K \cdot T} \sum_{k=1}^{K} \sum_{t=1}^{T} \left[\left(\frac{\Delta\hat{\omega}_i^{(k)}(t) - \Delta\omega_i^{(k)}(t)}{\max(\Delta\omega_i^{(k)}) - \min(\Delta\omega_i^{(k)}) + \epsilon} \right)^2 + \left(\frac{\hat{\delta}_i^{(k)}(t) - \delta_i^{(k)}(t)}{\max(\delta_i^{(k)}) - \min(\delta_i^{(k)}) + \epsilon} \right)^2 \right] \tag{11}$$

where $\max(\delta_i^{(k)})$ and $\min(\delta_i^{(k)})$ represent the maximum and minimum values of the ground truths in the k-th trajectory. The term ϵ is a small constant (e.g., 10^{-9}) added for numerical stability to prevent division by zero when the ground-truth trajectory is flat. The same normalization method is applied to the relative rotor speed, $\Delta\omega$. Based on the testing metric in equation (11), we compared the predictor trained with unsupervised learning only and that with the two-stage approach as shown in Table I. We observe that the supervised training stage significantly improves the accuracy of the predictors trained with unsupervised learning.

TABLE I: Normalized MSEs of predictors trained with different training strategies, averaged over 1,000 testing cases.

	Unsupervised Only	**Unsupervised + Supervised**
G1	$1.059 \cdot 10^{-1}$	$2.485 \cdot 10^{-5}$
G2	$9.038 \cdot 10^{-2}$	$1.793 \cdot 10^{-5}$
G3	$1.124 \cdot 10^{-1}$	$2.693 \cdot 10^{-5}$
Overall	$1.029 \cdot 10^{-1}$	$2.324 \cdot 10^{-5}$

To visualize the performance of the trained predictors, we compute the normalized MSE for each testing case k:

$$\mathcal{E}^{(k)} = \frac{1}{6 \cdot T} \sum_{i=1}^{3} \sum_{t=1}^{T} \left[\left(\frac{\Delta\hat{\omega}_i^{(k)}(t) - \Delta\omega_i^{(k)}(t)}{\max(\Delta\omega_i^{(k)}) - \min(\Delta\omega_i^{(k)}) + \epsilon} \right)^2 + \left(\frac{\hat{\delta}_i^{(k)}(t) - \delta_i^{(k)}(t)}{\max(\delta_i^{(k)}) - \min(\delta_i^{(k)}) + \epsilon} \right)^2 \right] \tag{12}$$

We then select the testing case with the *largest* normalized MSE, i.e., the *worst case*, among the 1,000 testing cases. In Fig. 3, we plot this worst case's predicted trajectories of rotor angles and relative rotor speeds of all the generators. We observe that, even for the worst-performing case among the 1,000 randomly generated cases, the state trajectories simulated by the PINN+ACPF framework very closely follow the ground-truth trajectories for all the generators.

Lastly, we present the testing results with different numbers of simulated cases used in the supervised training stage. The results are summarized in Table II. We observe that, even relying only on as few as 100 simulated cases in the supervised training stage, the trained predictors can already achieve remarkable accuracies in testing. This indicates great potential for further reducing the overall computational complexity in the training of predictors. We note that the non-monotonicity observed in the accuracy is due to the randomness of the selected cases.

TABLE II: Normalized MSEs of predictors trained with different training sizes, averaged over 1,000 testing cases.

Training size	G1	G2	G3	Overall
3800	2.485e-5	1.793e-5	2.693e-5	2.324e-5
1900	5.194e-5	4.609e-5	4.620e-5	4.808e-5
1000	4.054e-5	3.282e-5	3.635e-5	3.654e-5
500	2.689e-6	8.773e-6	6.498e-6	5.987e-6
200	1.178e-5	1.801e-5	1.220e-5	1.400e-5
100	8.820e-6	1.294e-5	1.339e-5	1.172e-5

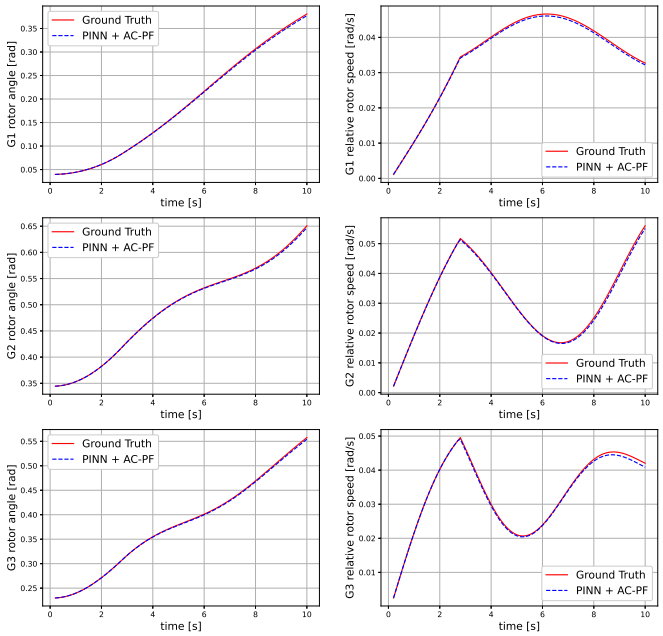

Fig. 3: The worst-performing testing case: comparisons of the trajectories of rotor angles and rotor speeds for the three generators (G1-G3).

D. Discussion

Our PINN-based approach ensures that, as long as any generator's dynamic model can be described by ODEs, a corresponding PINN can be trained for it. As such, when more complex control mechanisms, such as generator Power System Stabilizer (PSS) and Automatic Voltage Regulator (AVR), can be (approximately) modeled by ODEs, the PINN-based approach can continue to be applied. The general implementation of the iterative algorithm during testing remains the same regardless. The method thus has wide applicability to different systems with different kinds of generators. Moreover, as the predictors are trained for generators separately, once a predictor is trained for a generator, if such a generator with the same dynamic model appears in a different system, the already-trained predictor could be applied directly without re-training.

As the training of the PINNs in our method is done for each generator separately, the training complexity scales linearly with the number of generators in the system, and the training for different generators can be performed in parallel. Such linear complexity is very advantageous for training to scale to larger numbers of generators. Notably, all the training and simulation in our method are *offline* computation, which is typically much less resource- and time-constrained. On the other hand, the computational complexity during testing is also linear in the number of generators. This is a key enabler for applying our method in real-time, online transient dynamic analysis.

While we examined transient dynamics under load changes in the case study, the developed method is generally applicable to other types of disturbances as well, including three-phase faults. Regardless of the disturbance types, as long as such disturbances do not affect the governing equations of generators but only the power flow computation, the proposed method with PINNs will continue to work.

V. Conclusion

We have developed a novel framework that integrates PINNs with an AC-PF solver for efficient transient dynamic simulations in power systems. By training a separate PINN-based predictor for each generator and leveraging an iterative algorithm to connect the predictors via AC-PF, the developed approach bypasses traditional numerical integration methods, enabling scalable and accurate simulation of full system dynamics. The two-stage training strategy, consisting of an unsupervised stage to enforce physical laws and a supervised stage to refine predictors with simulation data, ensures both compliance with the governing differential equations and improved accuracy. The effectiveness of the proposed framework was demonstrated on a 3-generator 9-bus system, showcasing its exceptional accuracy in simulating full system dynamic trajectories. As the computational complexity scales linearly with the number of generators, the developed method offers promising scalability for efficiently simulating large-scale power system dynamics.

References

[1] B. Stott, "Power system dynamic response calculations," *Proceedings of the IEEE*, vol. 67, no. 2, pp. 219–241, 1979.

[2] H.-D. Chiang, *Direct methods for stability analysis of electric power systems: theoretical foundation, BCU methodologies, and applications*. John Wiley & Sons, 2011.

[3] T. Zhao, M. Yue, and J. Wang, "Structure-informed graph learning of networked dependencies for online prediction of power system transient dynamics," *IEEE Transactions on Power Systems*, vol. 37, no. 6, pp. 4885–4895, 2022.

[4] W. Cui, W. Yang, and B. Zhang, "A frequency domain approach to predict power system transients," *IEEE Transactions on Power Systems*, vol. 39, no. 1, pp. 465–477, 2023.

[5] J. Li, M. Yue, Y. Zhao, and G. Lin, "Machine-learning-based online transient analysis via iterative computation of generator dynamics," in *2020 IEEE Int. Conf. on Communications, Control, and Computing Tech. for Smart Grids (SmartGridComm)*. IEEE, 2020, pp. 1–6.

[6] 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.

[7] G. S. Misyris, A. Venzke, and S. Chatzivasileiadis, "Physics-informed neural networks for power systems," in *2020 IEEE power & energy society general meeting (PESGM)*. IEEE, 2020, pp. 1–5.

[8] J. Stiasny, G. S. Misyris, and S. Chatzivasileiadis, "Physics-informed neural networks for non-linear system identification for power system dynamics," in *2021 IEEE Madrid PowerTech*. IEEE, 2021, pp. 1–6.

[9] J. Li, Y. Zhao, and M. Yue, "Integrating learning and physics based computation for fast online transient analysis," in *2023 IEEE PES Innovative Smart Grid Tech. Conf. (ISGT)*. IEEE, 2023, pp. 1–5.

[10] 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.

[11] Z. Mao and X. Meng, "Physics-informed neural networks with residual/gradient-based adaptive sampling methods for solving partial differential equations with sharp solutions," *Applied Mathematics and Mechanics*, vol. 44, no. 7, pp. 1069–1084, 2023.

[12] "Pypower-dynamics," Available at https://github.com/susantoj/PYPOWER-Dynamics.