EXPERIMENTAL EVALUATION OF SAMPLING, WEIGHTING, AND COLLOCATION STRATEGIES IN PHYSICS-INFORMED NEURAL NETWORKS FOR POWER SYSTEM MODELS

Jonas Wiendl (s243543)

ABSTRACT

Physics-Informed Neural Networks (PINNs) offer a flexible framework for modeling power-system dynamics, yet their performance depends critically on data sampling, loss balancing, and the distribution of collocation points. This work evaluates these factors for a synchronous machine equipped with automatic voltage regulation and turbine–governor control. First, we compare several evolutionary sampling (Evo) variants and show that both a standard Evo and a denser exploration variant achieve the lowest validation errors. Cross-dataset generalization is also analyzed, revealing that the dataset perturbed with Gaussian noise generalizes best to unseen operating conditions. Second, multiple loss-weighting schemes are assessed, but due to the constraints connected to the LBFGS optimizer used in the framework, none outperform a tuned static baseline. Finally, an online residual-based collocation update yields a consistent 9% reduction in validation error. Overall, the results highlight both the potential and limitations of PINN training for power-system dynamics.

GITHUB REPOSITORY

A complete implementation accompanying this work, including data generation scripts, HPC scripts and notebooks reproducing the results, is available at: `https://github.com/jnasw/02456_DeepLearning`.

1. INTRODUCTION

Physics-Informed Neural Networks (PINNs) combine neural networks with physical modeling by enforcing differential equations directly in the loss function. Instead of learning only from data, a PINN also minimizes the residual of the governing ODE or PDE using automatic differentiation. This makes PINNs attractive for modeling dynamical systems where data may be scarce, expensive, or incomplete, and where incorporating physics can improve generalization. PINNs have been used in fluid mechanics, wave propagation, and nonlinear systems, and can act either as direct solvers of differential equations or data-assisted surrogates [1]. In parallel, the broader literature on neural networks for dynamical systems distinguishes PINNs from time-stepper models and neural ODEs, highlighting their advantages for learning solutions directly in the continuous domain [2].

Despite their promise, PINNs often exhibit unstable or slow convergence [1]. Their loss function consists of several competing terms (data, initial and boundary conditions, and physics residuals) that must all be optimized simultaneously. These terms frequently produce severely unbalanced gradients, with the PDE residual dominating the optimization and causing the network to neglect boundary or data terms [3]. PINN performance is also highly sensitive to how training points are chosen. This involves two distinct aspects: initial sampling, which determines the coverage of the solution domain, and the placement of collocation points, which enforce the physics residual during training. Poor initial sampling can cause the model to miss important dynamics, motivating more structured strategies such as Latin Hypercube or residual-informed sampling [4]. Fixed collocation sets, however, may miss regions that only become challenging later in training, prompting dynamic refinement strategies that relocate or regenerate points based on the residual [5].

These observations highlight three persistent challenges that strongly influence PINN accuracy and robustness: 1) how training data and collocation points are initially sampled; 2) how the multi-term loss is balanced during optimization, and 3) how collocation points are refined or adapted as the model learns. Addressing these challenges is essential for making PINNs reliable in practical scientific and engineering applications and motivates the investigations carried out in this work.

2. BACKGROUND

2.1. PowerPINN Framework

The PowerPINN framework provides a modular and automated pipeline for developing Physics-Informed Neural Networks tailored to power system components. Its architecture separates the modeling equations, dataset generation, neural network configuration, and training routines into distinct modules, enabling reproducible PINN workflows across different dynamic models [6].

The modeling module contains the ODEs describing the

component dynamics, with system parameters and inputs stored independently to allow rapid reconfiguration. The dataset pipeline generates both simulated trajectories using a numerical solver and unlabeled collocation points that are used to enforce physics residuals. Initial conditions for both datasets are sampled within a user-defined state space using Latin Hypercube sampling. The neural network component is a fully connected feed-forward model that maps initial conditions and time to predicted system states. Training minimizes a weighted combination of data-driven and physics-informed loss terms, using automatic differentiation to compute the ODE residuals. The framework also includes standardized evaluation routines for accuracy and inference time.

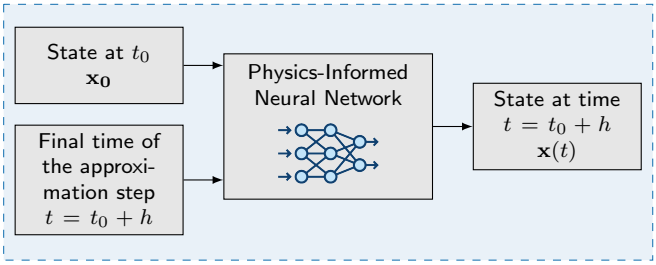

Fig. 1. Schematic of the PINN structure, mapping (x_0, t) to the predicted state $\hat{x}(t)$.

2.2. PINN Physics Dynamics and Use Case

As a representative test system, we use the *synchronous machine equipped with an automatic voltage regulator (AVR) and a governor*, yielding a 9th-order nonlinear ODE system. This model captures electromechanical swing dynamics, excitation control, and turbine-governor response. A complete description of the governing equations, algebraic relationships, and parameter values is provided in [6]. In this work, we rely directly on the implementation distributed with the PowerPINN framework. (GitHub: `https://github.com/radiakos/PowerPINN`)

Within the PINN formulation, the continuous-time dynamics follow

$$\frac{dx(t)}{dt} = f\big(t,\, x(t),\, u,\, \mu\big), \tag{1}$$

where $x(t)$ contains the machine, AVR, and governor states; u represents system inputs; and μ denotes the physical parameters. A neural network surrogate $\hat{x}(t)$ is trained to satisfy both data-based constraints and the physics residual

$$r(t) = \frac{d\hat{x}(t)}{dt} - f\big(t,\, \hat{x}(t),\, u,\, \mu\big), \tag{2}$$

computed via automatic differentiation. At inference time, the PINN receives an initial condition x_0 and a continuous time value t, and outputs the predicted state $\hat{x}(t)$ directly, enabling continuous-time trajectory reconstruction without numerical integration. A schematic overview of this PINN mapping is shown in Fig. 1 [6].

3. METHODOLOGY

3.1. Neural Network Architecture

All experiments use the `DynamicNN` model from the PowerPINN framework, implemented in PyTorch. The PINN is a fully connected feed-forward network mapping the concatenated input (x_0, t) to the predicted state $x(t)$. The architecture consists of four hidden layers with 64 neurons each, using $\tanh$ activations and Xavier-normal initialization. The input dimension is 10 (nine states plus time), and the output dimension is the nine-dimensional system state. Training minimizes a weighted MSE objective over data, derivative, physics-residual, and initial-condition terms. Optimization is performed using full-batch LBFGS and no additional regularization is applied.

3.2. Dataset and Simulation Setup

The model is simulated over 100 time steps, yielding 80,000 training samples, 10,000 validation samples, and 10,000 test samples. Initial conditions for (θ, ω) are drawn from $[-2, 2] \times [-1, 1]$, while the remaining states are fixed at nominal values. Inputs and outputs are normalized to $[-1, 1]$. Physics constraints use 100,000 collocation points and 100 initial-condition points. For adaptive collocation, a pool of 10,000 candidate points is evaluated every K epochs. This candidates set thereby increases density in the (θ, ω) space, while it reduces dimensionality in the remaining state and time space for computational efficiency. The "mutated" dataset adds Gaussian noise with $\sigma = 0.01$ to the trajectories. Validation and test performance are reported as mean absolute error (MAE) over all nine states and all time steps, while training minimizes a composite MSE objective.

3.3. Sampling Strategies

Sampling determines which regions of the state–time space the PINN learns to approximate accurately. Static sampling methods (e.g., grid or Latin Hypercube Sampling) provide predefined coverage, whereas adaptive approaches update points based on the evolving residual. In this work, we employ an evolutionary (Evo) sampling strategy inspired by [7], which combines exploitation of high-residual regions with exploration using space-filling samples. Five Evo-derived datasets are constructed: *dense* (increased coverage within a restricted IC range), *mixed* (combining high-residual and LHS samples), *mutated* (Gaussian-perturbed trajectories), *sparse* (reduced density), and *wide* (broader IC domain). These variants allow us to analyze how coverage density,

domain width, and noise influence PINN training. All sampling strategies are evaluated using final validation MAE and cross-dataset generalization.

3.4. Loss Balancing and Weighting Schemes

PINN training minimizes a weighted sum of data, derivative, physics-residual, and initial-condition losses. These terms naturally produce highly unbalanced gradients, often causing the physics residual to dominate and slowing or destabilizing training [3]. Weighting schemes aim to rebalance these objectives. Three principal families exist: (1) *static* weights with fixed coefficients [8]; (2) *gradient-based* dynamic weighting such as MA-PINN and DN-PINN, which use gradient magnitudes to adjust loss contributions [3, 8]; and (3) *uncertainty-based* weighting, where each loss is treated as a likelihood with learnable variance [9].

Framework Constraint: The PowerPINN framework relies on full-batch LBFGS, a line-search-based optimizer that recomputes gradients internally. Per-step gradient norms are therefore inaccessible, preventing direct use of MA-PINN, DN-PINN, or similar adaptive schemes.

Practical Workarounds: To enable comparative evaluation, we implement three LBFGS-compatible workarounds: (i) Gradient reconstruction, where approximate gradient magnitudes are inferred from loss differences to emulate MA-PINN; (ii) Split-phase static weighting, where training begins with a data-only phase followed by fixed physics weights; and (iii) Two-stage optimization, in which an Adam phase is introduced after LBFGS to access explicit gradients, enabling DN-PINN updates after a short warm-up period.

3.5. Adaptive Collocation

Fixed collocation sets may miss dynamically challenging regions where the PINN develops large physics residuals. To address this, we employ a residual-based adaptive collocation strategy inspired by RAR [10] and related refinement methods [5]. A fixed pool of 10,000 candidate points is generated before training. Every $K = 50$ epochs, the physics residual is evaluated on this pool, and a fraction of the highest residual points is used to replace existing collocation points. This residual-informed update keeps the total number of collocation points fixed while focusing training on regions where the model underperforms.

4. RESULTS AND DISCUSSION

4.1. Impact of Sampling Strategies

Table 1 compares the final validation errors of all sampling variants relative to the grid-sampling baseline. The regular evolutionary *mixed* and the *dense* dataset variant yield the best performance, improving validation loss by 12%, and 24%, respectively. In contrast, datasets with injected noise (*mutated*), reduced coverage (*sparse*), or widened parameter ranges (*wide*) perform worse than the baseline, with the *wide* dataset degrading performance most strongly. These results show that evolutionary sampling can improve performance when it increases coverage in relevant regions of the state space (dense, mixed), but may degrade accuracy when the IC domain becomes too broad or the sampling density too low (wide, sparse).

Table 1. Relative validation loss change compared to the baseline model trained on grid sampling. Negative values indicate improved performance.

Dataset	dense	mixed	mutated	sparse	wide
Change (%)	-24.2	-11.8	$+19.6$	$+14.4$	$+86.1$

Cross-evaluation results (Fig. 2) provide further insight: The model trained on the *mutated* dataset generalizes best overall, suggesting that small amounts of Gaussian noise act as a regularizer. The *dense* dataset also generalizes well except on the noisy dataset, indicating that increasing overall data coverage has a strong influence on performance. Across all models, performance degrades notably on the *wide* and *mutated* datasets, highlighting the inherent difficulty of PINNs to generalize to out-of-distribution regions or noisy, perturbed trajectories.

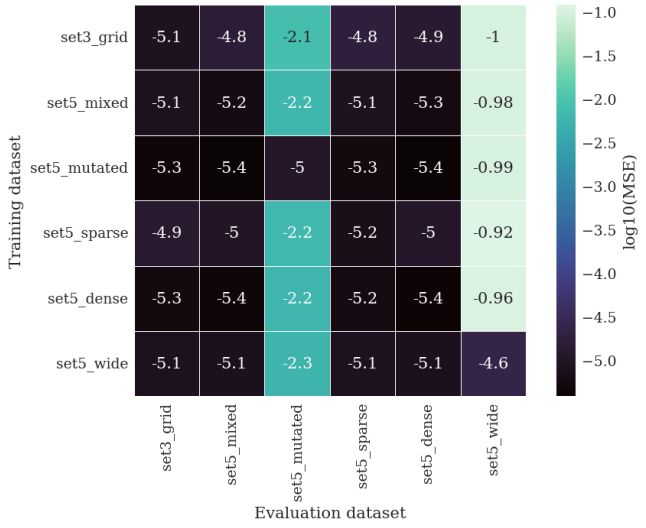

Fig. 2. Heatmap of $\log_{10}$(MSE) for models trained on different sampling sets and evaluated across all datasets.

4.2. Impact of Weighting Strategies

Table 2 summarizes the performance of all tested weighting and optimization strategies. Across all scenarios, the tuned

static weighting baseline (S0) achieved the lowest validation loss (2.53×10^{-3}), and no dynamic or hybrid method was able to surpass it.

The MA weighting applied during LBFGS (S1) resulted in significantly higher validation errors, questioning the effectiveness of rebuilding gradient information. A split-phase strategy, where training begins with a data-only period and introduces weights after 200 epochs, performed inconsistently: the static variant (S2a) degraded performance, indicating that the chosen weights were too large for this model configuration, while an adaptive variant (S2b) achieved the second-best result.

Next, two-stage optimization schemes using LBFGS for initialization followed by Adam were applied: The static version (S3) and the DN-weighted variant (S4) both produced higher validation errors, indicating that switching optimizers alone does not address the sensitivity of the loss landscape. For S4 this is most likely due to the mismatch with the preceding LBFGS phase. Introducing an Adam warm-up phase (S5) stabilized early training but did not improve accuracy. Finally, applying MA weighting only after 1500 epochs (500 LBFGS, 1000 Adam - S6) again resulted in worse performance than the baseline.

Overall, these results show that dynamic weighting techniques are difficult to integrate within an LBFGS-centered training pipeline. The lack of explicit gradient information, together with the sensitivity of the PINN loss structure, limits the effectiveness of reconstructed MA and staged weighting approaches. The manually tuned static scheme remains the most reliable option under the current PowerPINN training setup.

Table 2. Comparison of Training Scenarios

Sc.	Description	Best Epoch	Best L_{val}
S0	LBFGS, static weights (baseline)	519	2.53×10^{-3}
S1	LBFGS, MA weighting	1701	1.07×10^{-2}
S2a	Split-phase (static after 200 ep)	457	1.38×10^{-2}
S2b	Split-phase (adaptive after 200 ep)	466	3.63×10^{-3}
S3	LBFGS $\rightarrow$ Adam @ 500 ep (static)	3531	1.27×10^{-2}
S4	LBFGS $\rightarrow$ Adam (DN)	4231	2.04×10^{-2}
S5	LBFGS $\rightarrow$ Adam (DN + warm-up)	1499	5.93×10^{-2}
S6	LBFGS $\rightarrow$ Adam (MA after 1500 ep)	1502	2.17×10^{-2}

4.3. Impact of Adaptive Collocation

To evaluate whether residual-informed collocation improves training, a baseline PINN with a fixed collocation set is compared against a model using the online refinement strategy described in Section 3.5. Figure 3 shows the residual field at early and late training stages. While part of the residual reduction is due to general optimization progress, the adaptive model repeatedly targets high-error regions during refinement, directing collocation points toward dynamically difficult areas. Quantitatively, the adaptive model achieves

$$L_{\text{val}}^{\text{baseline}} \approx 2.53 \times 10^{-3}, \qquad L_{\text{val}}^{\text{adaptive}} \approx 2.30 \times 10^{-3},$$

corresponding to an improvement of approximately 9%. Although modest, this gain demonstrates that even a simple residual-based refinement step can provide measurable accuracy improvements without modifying the optimizer or loss-weighting configuration.

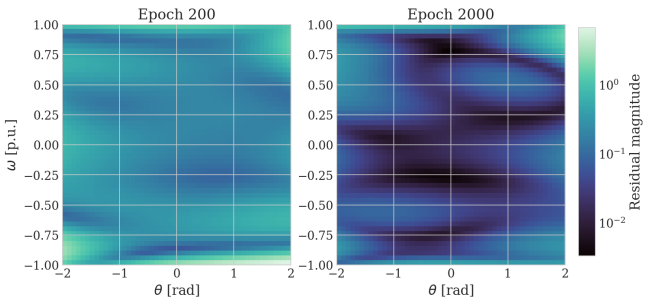

Fig. 3. Residual fields over the (θ, ω) domain at an early (Epoch 200) and late (Epoch 2000) training stage. High-residual regions shrink over time, showing how adaptive collocation focuses training on difficult areas.

5. CONCLUSION

This work investigated how sampling, loss weighting, and adaptive collocation affect PINN performance for the investigated synchronous machine system and its ninth-order ODE.

Among the sampling strategies, regular and dense evolutionary sampling variants improved validation accuracy, whereas sparse, noisy, or widened domains reduced performance. This confirms that the choice of initial-condition distribution strongly shapes the surrogate, and that generalization requires sampling that reflects the intended operational regime.

All tested weighting strategies, including MA and DN, performed worse than a tuned static baseline. The primary limitation is the use of LBFGS, which does not expose stable per-loss gradients and thus prevents reliable application of gradient-based dynamic weighting. Two-stage LBFGS$\rightarrow$Adam schemes also did not improve accuracy.

A lightweight residual-based collocation update improved the final validation error by roughly 9%, highlighting that residual-guided sampling provides measurable benefit even without dynamic weighting.

Outlook. Future work should integrate optimizers that expose explicit per-loss gradients, enabling the full use of adaptive weighting schemes. Multi-stage PINNs or curriculum refinement, where successive models are trained on residual landscapes of previous stages, also show promise for further improving stability and accuracy in power-system PINNs.

6. REFERENCES

[1] 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, 2 2019.

[2] Christian Legaard, Thomas Schranz, Gerald Schweiger, Ján DrgoÅa, Basak Falay, Cláudio Gomes, Alexandros Iosifidis, Mahdi Abkar, and Peter Larsen, "Constructing Neural Network Based Models for Simulating Dynamical Systems," *ACM Computing Surveys*, vol. 55, no. 11, 11 2023.

[3] Sifan Wang, Yujun Teng, and Paris Perdikaris, "Understanding and mitigating gradient pathologies in physics-informed neural networks," 1 2020.

[4] Arka Daw, Virginia Tech, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne, "Rethinking the Importance of Sampling in Physics-informed Neural Networks," .

[5] Jie Hou, Ying Li, and Shihui Ying, "Enhancing PINNs for solving PDEs via adaptive collocation point movement and adaptive loss weighting," *Nonlinear Dynamics*, vol. 111, no. 16, pp. 15233–15261, 8 2023.

[6] Ioannis Karampinis, Petros Ellinas, Ignasi Ventura Nadal, Rahul Nellikkath, and Spyros Chatzivasileiadis, "Toolbox for Developing Physics Informed Neural Networks for Power Systems Components," 2 2025.

[7] Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, and Michael W. Mahoney, "Characterizing possible failure modes in physics-informed neural networks," 11 2021.

[8] Shota Deguchi and Mitsuteru Asai, "Dynamic & norm-based weights to normalize imbalance in back-propagated gradients of physics-informed neural networks," *Journal of Physics Communications*, vol. 7, no. 7, 7 2023.

[9] Fujun Cao, Xiaobin Guo, Xinzheng Dong, and Dongfang Yuan, "wbPINN: Weight balanced physics-informed neural networks for multi-objective learning," *Applied Soft Computing*, vol. 170, 2 2025.

[10] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis, "DeepXDE: A deep learning library for solving differential equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.

DECLARATION OF USE OF GENERATIVE AI

This declaration **must** be filled out and included as the **final page** of the document. The questions apply to all parts of the work, including research, project writing, and coding.

- I/we have used generative AI tools: **yes**

If you answered *yes*, please complete the following sections.

List the generative AI tools you have used:

- ChatGPT
- GitHub Copilot in VSCode

Describe how the tools were used:

What did you use the tool(s) for? The tools were used to generate small code snippets, clarify programming library functionalities, summarize academic papers during the literature review, refine and improve wording in selected sections of the report, and assist with LaTeX formatting issues.

At what stage(s) of the process did you use the tool(s)? Generative AI tools were used during the coding phase, the literature review phase, and the report-writing phase.

How did you use or incorporate the generated output? Code suggestions were reviewed and manually adapted before inclusion. Summaries were used only to support understanding and not copied verbatim. Text refinements provided by the tools were edited and integrated into the final report. Latex assistance was used to resolve formatting issues, while final formatting decisions were made manually.