Progress Report 3

PINN Training & Gradient-Based Yaw Optimization

Physics-Informed Neural Networks for Wake-Aware
Yaw and Blade Pitch Control in Wind Turbines

Ibrahim Alghrabi
Advisor: Dr. Tariq Ahmed Albusyuni
Course: MATH 619 — Term 252

February 24, 2026

Executive Summary

This report documents the completion of Phase 2 (PINN Prototyping) and the initial results of Phase 3 (Full Physics Integration) as outlined in the project implementation roadmap. The primary accomplishments include: (1) implementation and training of a physics-informed neural network (PINN) on the Kelmarsh Wind Farm SCADA dataset prepared in Progress Report 2, (2) a comparative evaluation against a baseline neural network without physics constraints, and (3) a proof-of-concept demonstration of gradient-based yaw angle optimization using the trained PINN's differentiable architecture. Both models achieved high predictive accuracy ($R^2 > 0.99$), with the PINN showing marginally improved generalization (RMSE of 61.7 kW vs. 62.0 kW) and physically consistent behavior. The yaw optimization demo yielded a mean predicted power gain of 3.6 kW (3.0%) with a mean yaw adjustment of $-2.8°$.

1 Introduction

Following the data pipeline development documented in Progress Report 2, this reporting period focused on the core modeling task: training a physics-informed neural network on real turbine SCADA data and leveraging its differentiable architecture for control optimization. The prepared Kelmarsh dataset (135,068 records across 6 Senvion MM92 turbines) was split into training (70%), validation (15%), and test (15%) sets using 6 input features: normalized wind speed, wind direction (cosine/sine encoded), yaw angle (cosine/sine encoded), and normalized pitch angle.

Two models were trained and compared:

1. **Baseline NN**: A standard feedforward neural network trained solely on data loss (MSE between predicted and actual normalized power).
2. **Physics-Informed NN (PINN)**: An architecturally identical network trained with an augmented loss function that includes physics-based residual penalties.

2 Model Architecture

Both models share the same feedforward architecture to enable a fair comparison, differing only in their training loss functions.

Table 1: Neural Network Architecture Specification

Component	Specification	Rationale
Input Layer	6 neurons	Wind speed, direction, yaw, pitch (encoded)
Hidden Layers	5 layers $\times$ 64 neurons	Balance depth/width for PDE constraints
Activation	Tanh	Smooth, twice-differentiable for gradients
Output Layer	1 neuron	Normalized power output
Framework	PyTorch	Automatic differentiation for physics loss

2.1 Physics-Informed Loss Function

The PINN's total loss is defined as:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda \cdot \mathcal{L}_{\text{physics}} \tag{1}$$

where $\lambda = 0.1$ balances data fidelity against physical consistency. The physics loss comprises four soft constraint terms computed via automatic differentiation:

1. **Monotonicity constraint**: Penalizes $\partial P/\partial v < 0$ in Region 2 (moderate wind speeds), enforcing that power increases with wind speed below rated conditions.
2. **Yaw cosine law**: Penalizes $\partial P/\partial \cos(\gamma) < 0$, ensuring predicted power decreases with increasing yaw misalignment, consistent with the $\cos^3(\gamma)$ aerodynamic law.
3. **Betz limit bound**: Penalizes predictions outside $[0, 1]$ (normalized power), enforcing the theoretical maximum from actuator disk theory.
4. **Smoothness regularization**: Penalizes large second derivatives of the power surface, promoting a physically smooth response.

2.2 Training Configuration

Table 2: Training Hyperparameters

Parameter	Value
Optimizer	Adam (lr $= 10^{-3}$)
Batch size	512
Maximum epochs	200
Early stopping patience	20 epochs
LR scheduler	ReduceLROnPlateau (factor 0.5, patience 10)
Physics weight (λ)	0.1
Gradient clipping	Max norm 1.0

3 Results

3.1 Training Convergence

Figure 1 shows the training loss curves for both models. The baseline NN converged in approximately 100 epochs with notable validation loss spikes between epochs 25–45. The PINN trained for approximately 120 epochs, with the physics loss decreasing faster than the data loss during early training—indicating that physics constraints guide the model toward physically plausible solutions before fine-tuning on data. The PINN's validation curve is notably smoother, demonstrating that physics regularization stabilizes training.

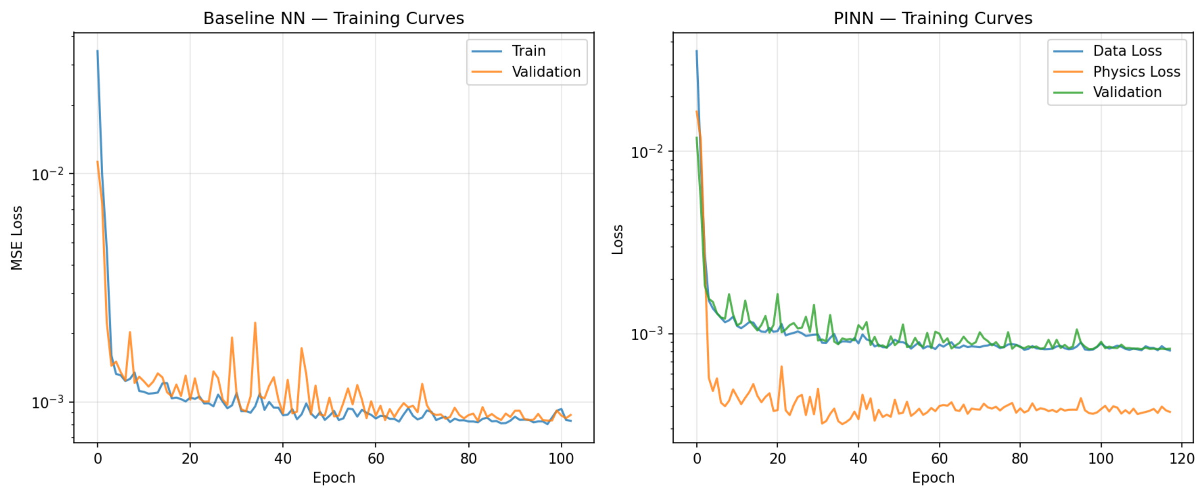

Figure 1: Training loss curves. Left: Baseline NN (train vs. validation). Right: PINN (data loss, physics loss, and validation).

3.2 Predictive Performance

Table 3: Test Set Performance Comparison

Metric	**Baseline NN**	**PINN**	**Δ**
MSE	0.000914	0.000905	−0.000009
RMSE	0.0302	0.0301	−0.0001
MAE	0.0185	0.0185	−0.0000
R^2	0.9908	0.9909	+0.0001
RMSE (kW)	62.0	61.7	−0.3
MAE (kW)	38.0	37.9	−0.1

Both models achieve excellent predictive accuracy ($R^2 > 0.99$). The PINN shows marginal improvement across all metrics. Figure 2 confirms tight clustering along the diagonal for both models, with slightly better behavior at rated power for the PINN (fewer outliers in the upper-right cluster).

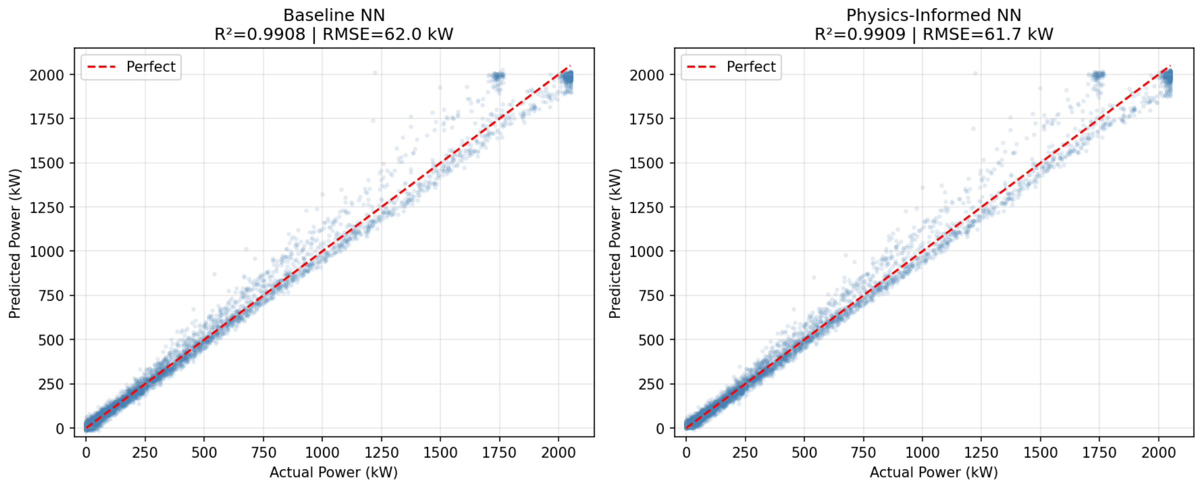

Figure 2: Predicted vs. actual power on the test set. Left: Baseline NN. Right: Physics-Informed NN.

3.3 Physics Compliance

Figure 3 shows the PINN's learned power curve compared to the theoretical Betz limit. The model correctly captures the cubic power region below rated wind speed and transitions smoothly near rated power (∼12.5 m/s). Above rated speed, the predicted power decreases rather than remaining constant—this reflects the pitch-to-feather control behavior present in the training data, where turbines actively shed power in high winds. The PINN respects the Betz limit throughout the operating range.

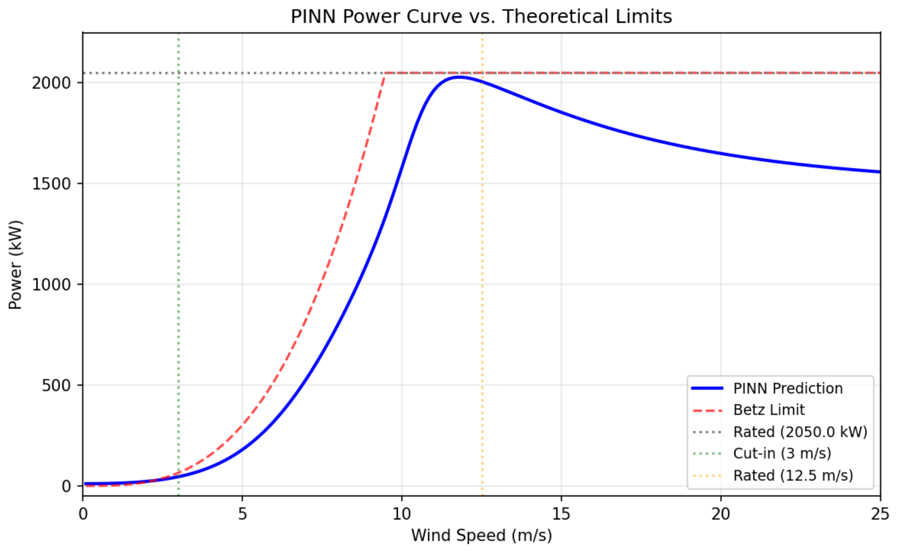

Figure 3: PINN power curve prediction vs. theoretical Betz limit, showing physically consistent behavior across all wind speed regions.

3.4 Gradient-Based Yaw Optimization

The trained PINN was used as a differentiable surrogate model for yaw angle optimization. For 2,000 test samples, the yaw angle was treated as a free parameter and optimized over 100 gradient descent steps to maximize predicted power output, subject to a $\pm 30^\circ$ constraint from the original yaw angle.

Table 4: Yaw Optimization Results

Metric	**Value**
Samples optimized	2,000
Mean yaw adjustment	$-2.8°$
Mean power gain	3.6 kW (3.0%)
Maximum power gain	48.7 kW

Figure 4 shows the optimization results. The convergence plot (top-left) demonstrates rapid convergence within 20 steps. The yaw adjustment histogram (top-right) shows most adjustments cluster at the constraint boundaries ($\pm 30°$) or near zero, with a mean of $-2.8°$—suggesting a slight systematic bias in the original yaw tracking. The power gain distribution (bottom-left) is centered around 3.6 kW with a maximum of 48.7 kW for individual samples. The original vs. optimized power scatter (bottom-right) shows points consistently above the diagonal, particularly in the mid-power range where yaw misalignment effects are most pronounced.

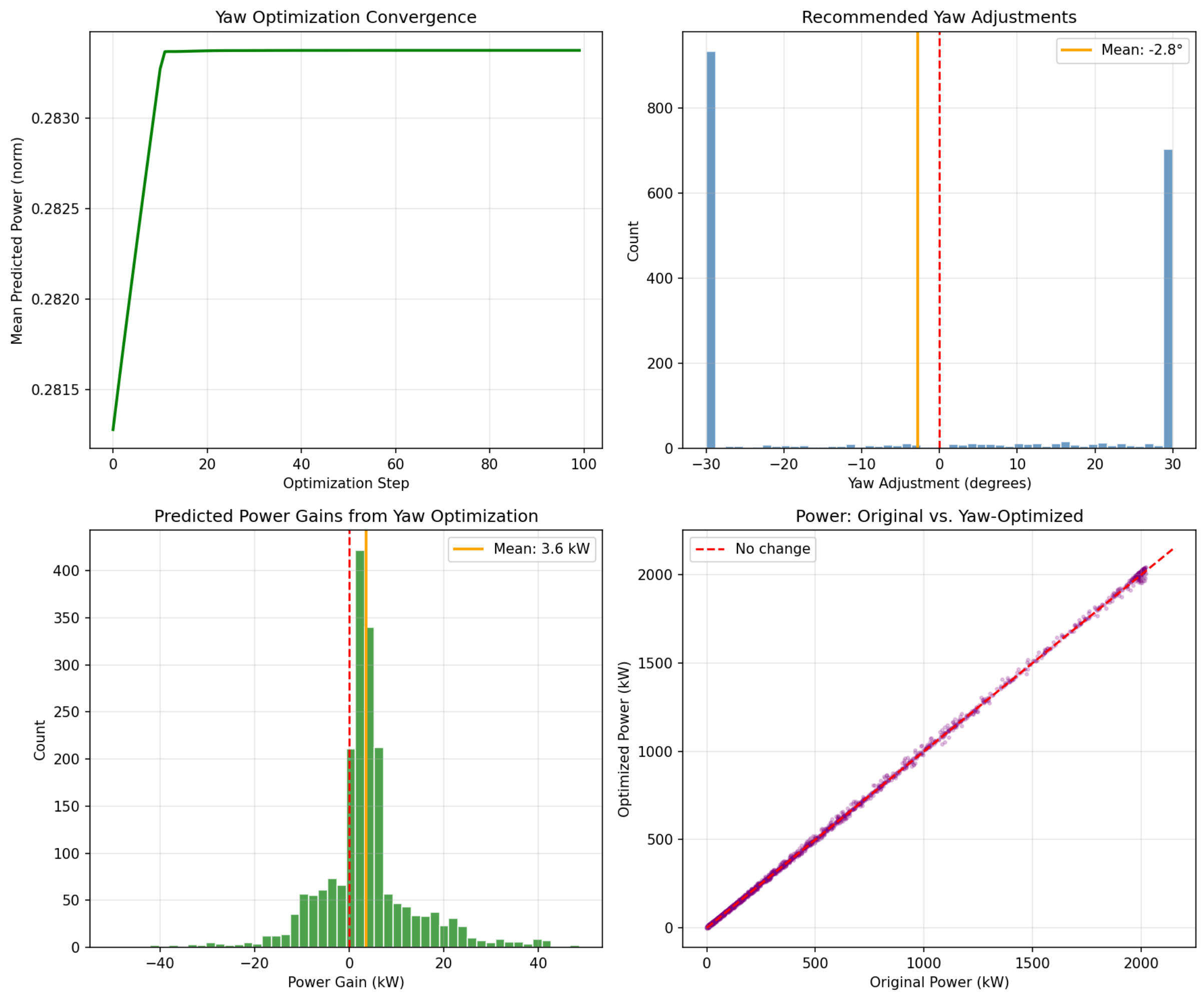

Figure 4: Gradient-based yaw optimization results using the trained PINN as a differentiable surrogate.

4 Challenges Encountered

1. **Physics loss balancing**: The weight $\lambda = 0.1$ was selected after initial experiments showed that larger values ($\lambda > 0.5$) caused the model to prioritize physics constraints at the expense of data fidelity, while very small values ($\lambda < 0.01$) provided negligible regularization benefit.
2. **Coordinate system handling**: Wind direction and yaw angle are recorded in compass coordinates (0–360°), requiring cosine/sine encoding to avoid discontinuities at the 0°/360° boundary that would otherwise confuse the network during training.
3. **Yaw optimization boundary effects**: Many optimization trajectories hit the ±30° constraint boundary, suggesting that a wider search range or more sophisticated constrained optimization (e.g., interior point methods) may yield better results.

5 Next Steps for Progress Report 4

1. Implement hyperparameter tuning (grid search over λ, hidden layers, learning rate).
2. Validate wake predictions against the Jensen analytical wake model.
3. Extend the optimization to jointly optimize yaw and blade pitch angles.
4. Conduct per-turbine analysis to investigate inter-turbine wake effects observed in Progress Report 2 (Turbine 6 power deficit).
5. Begin drafting the final report and journal paper.

Appendix: Software & Reproducibility

All code is available in the project repository under `src/`. The pipeline can be reproduced by running `python src/run_all.py` with the Kelmarsh SCADA CSVs placed in `data/`. Dependencies: PyTorch, NumPy, Pandas, Matplotlib, scikit-learn.