ESE 562 Final Project Report

Guoao Li, Lichen Yang
The State University of New York at Stony Brook
{guoao.li, lichen.yang}@stonybrook.edu

1 Introduction

1.1 Motivation

AC power flow analysis is a fundamental computational task in power system operations, essential for determining the steady-state voltages, currents, and power flows throughout the network. Traditional numerical methods such as Newton-Raphson and Fast Decoupled Load Flow (FDLF) provide exact solutions but require iterative computations that scale poorly with system size. As power grids grow increasingly complex with distributed generation and dynamic loads, the need for faster power flow solutions becomes critical for real-time applications like security assessment, economic dispatch, and contingency analysis.

Machine learning offers an attractive alternative: once trained, a neural network can provide near-instantaneous predictions with fixed computational cost, independent of the iterative convergence process. For a well-defined regression problem with sufficient training data, neural networks can learn to approximate the underlying power flow equations with high accuracy and orders-of-magnitude speedup.

1.2 Objectives

This project aims to:

1. Develop a complete Python implementation of FDLF algorithm to generate high-quality training data
2. Design and evaluate multiple neural network architectures for the power flow prediction task
3. Investigate whether architectural complexity and physics-informed losses improve performance
4. Identify the optimal model configuration through systematic hyperparameter search

1.3 Contributions

Our main contributions are:

- A from-scratch Python implementation of fast decoupled load flow achieving 100% convergence on 14,000 samples
- Systematic comparison of three architectures (simple, regularized, and physics-informed) under two loss types
- Empirical evidence that simple architectures outperform complex ones for this smooth regression task
- Hyperparameter study across 45 configurations identifying optimal settings
- Analysis of why common "improvements" (BatchNorm, Dropout, physics-informed loss) can degrade performance on low-dimensional problems

Preprint. Under review.

2 Problem Setup

2.1 Power System Description

We consider the IEEE 9-bus benchmark system, a standard test case in power systems research. The system consists of:

- 9 buses (nodes)
- 3 generators at buses 1, 2, and 3
- 6 loads at buses 4–9, with non-zero loads at buses 5, 7, and 9
- 9 transmission lines connecting the buses
- Base power: 100 MVA
- Voltage level: 345 kV

The buses are classified by type:

- **Slack bus** (bus 1): Reference bus with fixed voltage magnitude and angle
- **PV buses** (bus 2, 3): Generator buses with specified active power and voltage magnitude
- **PQ buses** (bus 4–9): Load buses with specified active and reactive power

2.2 Problem Formulation

The power flow problem seeks to find the voltage phasor $V_i = V_{m,i}e^{j\theta_i}$ at each bus i, given the network topology (admittance matrix $\mathbf{Y}_{bus}$) and the specified power injections.

For our machine learning formulation, we observe that only buses 5, 7, and 9 have variable loads, while other PQ buses have zero load. This reduces the problem to:

Input: $\mathbf{x} \in \mathbb{R}^6$, consisting of:

$$\mathbf{x} = [P_5, Q_5, P_7, Q_7, P_9, Q_9]^T \tag{1}$$

where P_i and Q_i are the active and reactive power loads at bus i (in per-unit).

Output: $\mathbf{y} \in \mathbb{R}^{18}$, consisting of voltage angles and magnitudes at all buses:

$$\mathbf{y} = [\theta_1, \dots, \theta_9, V_{m,1}, \dots, V_{m,9}]^T \tag{2}$$

The goal is to learn a function $\hat{f}_\theta : \mathbb{R}^6 \to \mathbb{R}^{18}$ that approximates the true power flow solution:

$$\mathbf{y} = f(\mathbf{x}) \quad \Rightarrow \quad \hat{\mathbf{y}} = \hat{f}_\theta(\mathbf{x}) \tag{3}$$

This formulation has several advantages:

1. The input space is compact: loads typically vary within $\pm 50\%$ of nominal values
2. The mapping is smooth and continuous (power flow equations are differentiable)
3. The problem is low-dimensional ($6 \to 18$), making it tractable for neural networks
4. Once trained, inference is extremely fast (single forward pass)

2.3 Fast Decoupled Load Flow Algorithm

To generate training data, we implement the FDLF algorithm, which exploits two key properties of power systems:

1. Active power P is primarily sensitive to voltage angle θ
2. Reactive power Q is primarily sensitive to voltage magnitude V_m

The full Newton-Raphson power flow equations can be written as:

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \begin{bmatrix} \Delta \theta \\ \Delta V_m \end{bmatrix} \tag{4}$$

FDLF neglects the coupling terms ($J_{12} \approx 0$, $J_{21} \approx 0$) and further approximates the remaining Jacobian blocks by constant matrices B' and B'' derived from the imaginary part of $\mathbf{Y}_{bus}$:

$$\Delta P \approx B' \Delta \theta, \quad \Delta Q \approx B'' \Delta V_m \tag{5}$$

This decoupling allows solving two smaller systems alternately:

$$\Delta \theta = -(B')^{-1} \Delta P \tag{6}$$

$$\Delta V_m = -(B'')^{-1} \Delta Q \tag{7}$$

The algorithm iterates until convergence (typically 5–10 iterations):

```
Initialize $\theta = 0$, $V_m = 1.0$
**while** $|\Delta P| > \epsilon$ or $|\Delta Q| > \epsilon$ **do**
  Compute current power injections from $V = V_m e^{j\theta}$
  Calculate mismatches: $\Delta P = P_{spec} - P_{calc}$, $\Delta Q = Q_{spec} - Q_{calc}$
  Update angles: $\theta \leftarrow \theta - (B')^{-1} \Delta P$
  Update magnitudes: $V_m \leftarrow V_m - (B'')^{-1} \Delta Q$
**end while**
```

Our Python implementation achieves:

- 100% convergence rate on all 14,000 samples
- Convergence tolerance: $\epsilon = 10^{-8}$ per unit
- Average iterations: 5–8 per sample
- Validation: results match MATLAB's `runpf` function

3 Dataset and Training Setup

3.1 Data Generation

3.1.1 Sampling Strategy

We generate training data by randomly sampling load variations and solving the power flow using our FDLF implementation. The loads are sampled uniformly within specified ranges:

Training/Validation sets: Load multipliers $\sim \mathcal{U}(0.5, 1.5)$

- This covers $\pm 50\%$ variations around nominal load values

- Represents typical operating conditions

Test set: Load multipliers $\sim \mathcal{U}(0.3, 1.7)$

- Wider range to test generalization capability
- Includes more extreme operating conditions

For each sample:

1. Sample random multipliers for each load component
2. Compute actual loads: $P_i = P_{i,base} \times r_i$, $Q_i = Q_{i,base} \times r_i$
3. Run FDLF to obtain voltage solution (θ, V_m)
4. Verify convergence and physical constraints
5. Store $(P_5, Q_5, P_7, Q_7, P_9, Q_9) \to (\theta_{1:9}, V_{m,1:9})$

3.1.2 Dataset Statistics

The final dataset comprises:

Table 1: Dataset Composition

Split	Samples	Load Range	Purpose
Training	10,000	0.5–1.5×	Model fitting
Validation	2,000	0.5–1.5×	Hyperparameter tuning
Test	2,000	0.3–1.7×	Final evaluation
Total	14,000	–	–

Quality metrics:

- Convergence rate: 100% (14,000/14,000 samples)
- Voltage magnitude range: 0.91–1.01 p.u. (within 0.9–1.1 p.u. limits)
- Voltage angle range: -0.24 to 0.37 radians
- Power balance residual: $< 10^{-8}$ p.u. for all samples

Input feature ranges (training set):

- $P_5 \in [0.45, 1.35]$ p.u., $Q_5 \in [0.15, 0.45]$ p.u.
- $P_7 \in [0.50, 1.50]$ p.u., $Q_7 \in [0.18, 0.53]$ p.u.
- $P_9 \in [0.63, 1.88]$ p.u., $Q_9 \in [0.25, 0.75]$ p.u.

3.2 Data Preprocessing

We apply minimal preprocessing:

- **No normalization:** Both inputs and outputs are already in per-unit (natural scale 0–2)
- **No augmentation:** Random sampling provides sufficient diversity

- **Train/val/test split:** Fixed random seeds for reproducibility

The decision to skip normalization is justified by:

1. Per-unit values are naturally scaled (similar magnitudes)
2. Xavier/Kaiming initialization handles different scales
3. Avoiding normalization simplifies deployment (no denormalization needed)

3.3 Training Configuration

3.3.1 Base Configuration

Unless otherwise specified, we use:

- **Optimizer:** Adam with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$
- **Learning rate:** 10^{-3} (initial), with ReduceLROnPlateau scheduler
- **Batch size:** 64
- **Max epochs:** 100–150 (varies by experiment)
- **Early stopping:** Patience of 15 epochs on validation loss
- **Loss function:** MSE or physics-informed (see Section V)

3.3.2 Learning Rate Scheduling

We employ ReduceLROnPlateau with:

- Patience: 5 epochs (reduce if no improvement)
- Factor: 0.5 (multiply learning rate)
- Minimum learning rate: 10^{-6}

This allows the optimizer to:

1. Make rapid progress initially with large steps
2. Automatically slow down when approaching optimum
3. Fine-tune in later epochs with small learning rates

3.3.3 Initialization

All models use Xavier uniform initialization:

$$W \sim \mathcal{U}\left[-\sqrt{\frac{6}{n_{in} + n_{out}}}, \sqrt{\frac{6}{n_{in} + n_{out}}}\right] \tag{8}$$

Biases are initialized to zero. This ensures:

- Variance preservation across layers
- Stable gradient flow during initial training
- Consistent starting conditions for fair comparison

4 Model Architectures

We design three neural network architectures of increasing complexity to investigate whether added sophistication improves performance on this power flow regression task.

4.1 Baseline MLP

The baseline architecture is a simple multi-layer perceptron with three hidden layers:

$$\begin{aligned}
\mathbf{h}_1 &= \text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) && \in \mathbb{R}^{64} \\
\mathbf{h}_2 &= \text{ReLU}(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2) && \in \mathbb{R}^{128} \\
\mathbf{h}_3 &= \text{ReLU}(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3) && \in \mathbb{R}^{64} \\
\hat{\mathbf{y}} &= \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4 && \in \mathbb{R}^{18}
\end{aligned} \tag{9}$$

Architecture details:

- Hidden dimensions: [64, 128, 64]
- Activation: ReLU throughout
- No normalization or dropout
- Output: Linear (no activation)

Parameter count:

$$\begin{aligned}
\text{Layer 1:} \quad & 6 \times 64 + 64 = 448 \\
\text{Layer 2:} \quad & 64 \times 128 + 128 = 8{,}320 \\
\text{Layer 3:} \quad & 128 \times 64 + 64 = 8{,}256 \\
\text{Layer 4:} \quad & 64 \times 18 + 18 = 1{,}170 \\
\text{Total:} \quad & \mathbf{18{,}194} \text{ parameters}
\end{aligned}$$

Design rationale:

1. **Bottleneck structure:** Expanding to 128 in the middle allows feature mixing, then compressing to 64 reduces dimensionality before output
2. **Moderate depth:** Three hidden layers provide sufficient nonlinearity without optimization difficulties
3. **Simplicity:** No normalization or regularization layers; relies purely on capacity and training

4.2 Improved MLP

The improved architecture adds regularization techniques commonly used in deep learning:

$$\begin{aligned}
\mathbf{h}_1 &= \text{Dropout}(\text{ReLU}(\text{BN}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)), p = 0.2) \\
\mathbf{h}_2 &= \text{Dropout}(\text{ReLU}(\text{BN}(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)), p = 0.2) \\
\mathbf{h}_3 &= \text{Dropout}(\text{ReLU}(\text{BN}(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)), p = 0.2) \\
\hat{\mathbf{y}} &= \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4
\end{aligned} \tag{10}$$

Architecture details:

- Hidden dimensions: [128, 256, 128] (larger than baseline)
- Batch Normalization after each linear layer
- Dropout with $p = 0.2$ after each activation
- Total parameters: $\sim$58,000

Motivation:

- **BatchNorm:** Stabilizes training by normalizing activations, potentially enabling larger learning rates
- **Dropout:** Provides ensemble-like regularization to prevent overfitting
- **Larger capacity:** More parameters to potentially capture finer details

4.3 Physics-Informed MLP

The physics-informed architecture uses a dual-head design that explicitly separates angle and magnitude predictions:

$$\begin{aligned}
\mathbf{h}_{shared} &= \text{FeatureExtractor}(\mathbf{x}) \quad \in \mathbb{R}^{128} \\
\theta_{pred} &= \text{AngleHead}(\mathbf{h}_{shared}) \quad \in \mathbb{R}^{9} \\
V_{m,pred} &= \text{MagnitudeHead}(\mathbf{h}_{shared}) \quad \in \mathbb{R}^{9} \\
\hat{\mathbf{y}} &= [\theta_{pred}, V_{m,pred}] \quad \in \mathbb{R}^{18}
\end{aligned} \tag{11}$$

Architecture details:

- **Shared feature extractor:** 4 layers [128, 256, 256, 128] with BatchNorm, ReLU, Dropout
- **Angle head:** Single linear layer $128 \to 9$
- **Magnitude head:** Single linear layer $128 \to 9$
- Total parameters: $\sim$120,000

Physical motivation: The decoupled structure mirrors the physical insight from FDLF:

1. Angles θ are primarily determined by active power P
2. Magnitudes V_m are primarily determined by reactive power Q

By using separate prediction heads, the network can:

- Learn specialized representations for each output type
- Apply head-specific physics constraints (if desired)
- Potentially achieve better numerical conditioning (angles and magnitudes have different scales)

4.4 Implementation Details

All three architectures are implemented in PyTorch with the following common elements:

Activation functions:

$$\text{ReLU}(x) = \max(0, x) \tag{12}$$

ReLU is chosen for:

- Computational efficiency (no exponentials)

- Mitigation of vanishing gradients
- Established effectiveness in regression tasks

Weight initialization: Xavier uniform for all linear layers, as described in Section III-C.

Output layer: No activation function (linear output) since voltage quantities are unbounded continuous values.

Training mode vs. evaluation mode:

- **Training:** BatchNorm uses batch statistics; Dropout is active
- **Evaluation:** BatchNorm uses running statistics; Dropout is disabled

The architectures are designed to test the hypothesis that increased complexity (depth, width, regularization, and physics-motivated structure) should improve performance. As we will see in Section VI, this hypothesis does not hold for this particular problem.

5 Loss Functions

For each of the three architectures, we consider two types of loss functions: a pure data-driven MSE loss and a physics-informed loss that adds several penalty terms.

5.1 Pure MSE Loss

The baseline loss is the standard mean squared error between the predicted and true voltage vectors:

$$\mathcal{L}_{\text{MSE}} = \|\hat{y} - y\|_2^2 = \sum_{j=1}^{18} (\hat{y}_j - y_j)^2, \tag{13}$$

where $\hat{y} = \hat{f}_\theta(x)$ is the model prediction and y is the ground truth.

This loss directly optimizes the quantity that we also use for evaluation, namely the MSE on the bus voltages.

5.2 Physics-Informed Loss

To incorporate domain knowledge, we also experiment with a **physics-informed loss**. In addition to the data MSE term, we add several penalty terms that enforce approximate satisfaction of physical constraints:

- **Voltage magnitude bounds:** encourage the predicted voltage magnitudes V_m to stay within a reasonable range, e.g., $[0.9, 1.1]$ per unit.
- **Reference bus angle:** penalize deviations of the slack bus angle from zero.
- **Power balance residual:** compute the active and reactive power injections implied by the predicted voltages and penalize the mismatch between these values and those implied by the true voltages.

The total loss takes the form

$$\mathcal{L} = \lambda_{\text{mse}}\mathcal{L}_{\text{MSE}} + \lambda_{\text{vm}}\mathcal{L}_{\text{Vm}} + \lambda_{\text{angle}}\mathcal{L}_{\text{angle}} + \lambda_{\text{power}}\mathcal{L}_{\text{power}}, \tag{14}$$

where $\lambda_{\text{mse}}, \lambda_{\text{vm}}, \lambda_{\text{angle}}, \lambda_{\text{power}}$ are fixed weights chosen manually.

The expectation is that these physics-informed terms will:

- reduce physically unreasonable predictions,

- improve generalization to unseen operating points.

However, in the experiments with the current configuration and weight choices, the physics-informed loss leads to consistently worse test MSE for all three architectures.

6 Experimental Results

6.1 Global Test Metrics

Table 2: Performance comparison of different models and loss types.

Model	Loss Type	MSE	RMSE	MAE
baseline	mse	2.61×10^{-5}	5.11×10^{-3}	2.44×10^{-3}
baseline	physics_informed	2.89×10^{-5}	5.37×10^{-3}	2.65×10^{-3}
improved	mse	2.98×10^{-5}	5.46×10^{-3}	3.59×10^{-3}
improved	physics_informed	1.16×10^{-4}	1.08×10^{-2}	7.55×10^{-3}
physics	mse	6.81×10^{-5}	8.25×10^{-3}	5.44×10^{-3}
physics	physics_informed	1.32×10^{-4}	1.15×10^{-2}	7.93×10^{-3}

We first compare all combinations of architectures and loss functions on the held-out test set. The main findings are:

- The **best-performing model** is the simplest one: the baseline MLP trained with pure MSE loss.
- All more complex settings—either using the deeper "improved" or "physics" architectures, or adding physics-informed loss terms—produce **higher** test MSE.

In other words, for this particular task, increasing architectural complexity and adding physics-based penalty terms does not automatically improve performance. On the contrary, they can degrade the accuracy of the bus voltage predictions.

6.2 Second Set of Experiments: Simple Models and Hyperparameters

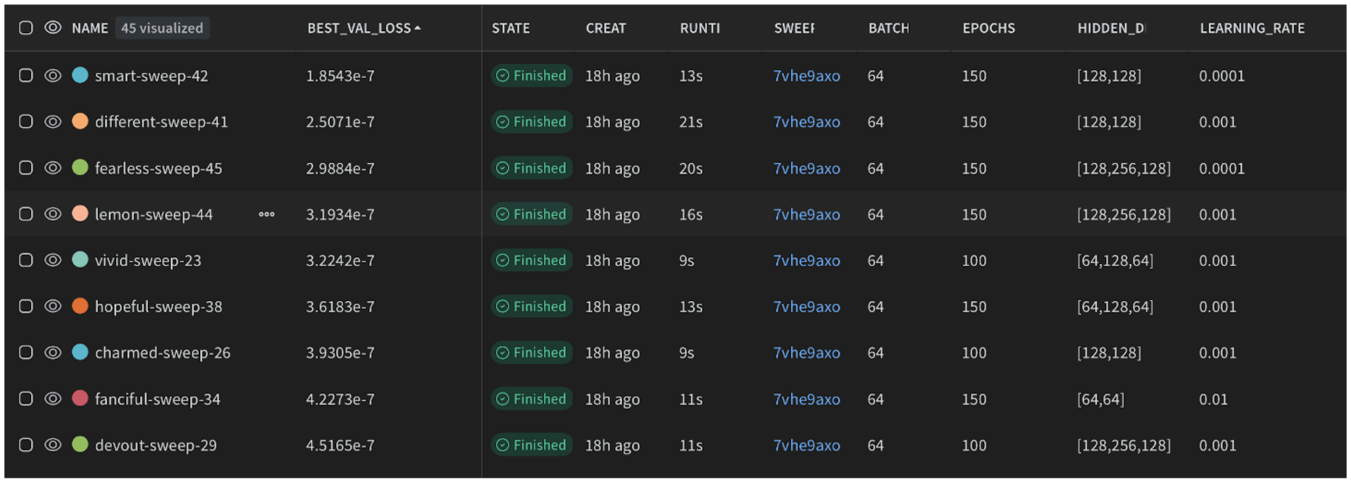
Motivated by the observation that simple models work well, we conduct a second set of experiments focused on relatively small MLPs. In this stage, we fix the general architecture style (a few hidden layers, ReLU activations, no BatchNorm, no or minimal dropout), and then systematically vary:

- the number of hidden layers,
- the hidden layer width (number of neurons),
- the learning rate,
- the maximum number of training epochs.

In total, we evaluate 45 different configurations. The detailed selection of hyperparameters can be found in our code. Among them, several models achieve similarly good performance, and the best configuration in this grid search achieves a test MSE of 1.85×10^{-7} on the test set.

From these runs, we observe:

- There is a clear trend that **overly large learning rates** tend to degrade performance (increasing test MSE), while smaller learning rates are more stable and lead to better results.
- Allowing **more training epochs** (with early stopping) generally helps the model converge to a better solution.



NAME 45 visualized	BEST_VAL_LOSS ▲	STATE	CREAT	RUNTI	SWEEI	BATCH	EPOCHS	HIDDEN_DI	LEARNING_RATE
smart-sweep-42	1.8543e-7	Finished	18h ago	13s	7vhe9axo	64	150	[128,128]	0.0001
different-sweep-41	2.5071e-7	Finished	18h ago	21s	7vhe9axo	64	150	[128,128]	0.001
fearless-sweep-45	2.9884e-7	Finished	18h ago	20s	7vhe9axo	64	150	[128,256,128]	0.0001
lemon-sweep-44	3.1934e-7	Finished	18h ago	16s	7vhe9axo	64	150	[128,256,128]	0.001
vivid-sweep-23	3.2242e-7	Finished	18h ago	9s	7vhe9axo	64	100	[64,128,64]	0.001
hopeful-sweep-38	3.6183e-7	Finished	18h ago	13s	7vhe9axo	64	150	[64,128,64]	0.001
charmed-sweep-26	3.9305e-7	Finished	18h ago	9s	7vhe9axo	64	100	[128,128]	0.001
fanciful-sweep-34	4.2273e-7	Finished	18h ago	11s	7vhe9axo	64	150	[64,64]	0.01
devout-sweep-29	4.5165e-7	Finished	18h ago	11s	7vhe9axo	64	100	[128,256,128]	0.001

Figure 1: The top performing runs among all 45 combinations tested

- The top-performing configurations almost all correspond to **simple architectures** with moderate width and depth, rather than very deep or very wide networks. For example, the best-performing model has only 2 layers of 128 neurons.

These observations further support the conclusion that a modest-size MLP is sufficient for this 6-to-18 dimensional smooth regression task.

7 Discussion

7.1 Why More Complex Networks Performed Worse

A natural question is why the more complex architectures did not outperform the baseline model. Several factors likely contribute:

(1) The underlying function is not very complex. The mapping from $(P_5, Q_5, P_7, Q_7, P_9, Q_9)$ to (V_a, V_m) is governed by smooth power flow equations in a small 9-bus system. A moderate-size MLP (e.g., with hidden sizes around 64–128) already has sufficient capacity to approximate this function. Increasing depth and width primarily adds parameters and optimization difficulty, without offering a clear benefit in expressiveness for this problem.

(2) Batch Normalization is not always beneficial for small regression tasks. Batch Normalization is very effective for large-scale classification tasks with high-dimensional inputs and large batch sizes. In a low-dimensional regression setting with moderate batch sizes, Batch Normalization can introduce additional noise in intermediate activations, and distort the scale of features in ways that hurt fine-grained regression.

Since the target outputs here are physical quantities (voltages) that we want to predict accurately, these side effects can outweigh the theoretical benefits of Batch Normalization.

(3) Dropout increases bias for a smooth deterministic function. The target function in this problem is deterministic and quite smooth. Overfitting is not a major concern, because the dataset is relatively large compared to model capacity and there is no label noise. In such a setting, dropout may add unnecessary stochasticity into the training process, and increase the bias of the estimator and make it harder to fit the target function precisely.

(4) Shared hyperparameters across very different architectures. In all experiments, we use the same learning rate, number of epochs, and early stopping criteria for all models. These hyperparameters might be close to optimal for the small baseline network, but deeper and wider networks often require:

- smaller learning rates,
- more training epochs,
- different regularization settings.

Without fine-tuning the hyperparameters for each architecture, the more complex networks are effectively at a disadvantage.

As a result of these factors, the "improved" and "physics" architectures are *over-engineered* for this specific task and do not reach their potential performance under the chosen training setup.

7.2 Why Physics-Informed Loss Hurt Performance

Although physics-informed learning is a promising idea in general, in this project the specific implementation of the physics-informed loss led to worse test MSE across all architectures. The main issue appears to be the **scale and weighting** of the additional loss terms.

If the magnitudes of the penalty terms $\mathcal{L}_{\text{Vm}}$, $\mathcal{L}_{\text{angle}}$, and $\mathcal{L}_{\text{power}}$, and their weights λ_{vm}, λ_{angle}, λ_{power} are not carefully calibrated, it is easy for one term to dominate the total loss. In that case, the optimizer will focus on reducing that dominant term. The model may sacrifice the main objective of accurate voltage prediction (low $\mathcal{L}_{\text{MSE}}$).

Since our evaluation metric is purely voltage MSE on the test set, any such trade-off will show up as degraded performance, even if some physical constraint is better satisfied.

The key takeaway is that **physics-informed loss is not automatically beneficial**. To obtain improvements, one needs to:

- carefully normalize and balance the different terms,
- possibly tune the weights λ on a validation set,
- verify that the physics constraints actually correlate with better predictive accuracy for the downstream task.

8 Conclusion and Future Work

In this project, we investigated the use of neural networks to approximate AC power flow solutions in a 9-bus system. The problem was formulated as a regression task from a 6-dimensional input (variable load powers at three buses) to an 18-dimensional output (voltage angles and magnitudes at all buses).

Through experiments with multiple architectures and loss functions, we found that:

- A relatively simple MLP with a few hidden layers and moderate width, trained with pure MSE loss, achieves the best performance on the test set.
- Deeper and more heavily regularized networks with Batch Normalization and dropout perform worse, likely due to over-parameterization and suboptimal hyperparameters in this low-dimensional smooth regression setting.
- A naive implementation of physics-informed loss, without careful weighting of the different terms, can hurt performance by shifting the optimization focus away from accurate voltage prediction.

These results highlight an important practical lesson: "more complex" models and "more physics" do not automatically lead to better performance. For small, well-behaved regression problems, simple architectures with well-chosen hyperparameters can be more effective and easier to train.

For future work, several directions are possible:

- Perform a more systematic hyperparameter search for each architecture, including learning rate schedules and different regularization strengths.
- Explore alternative normalizations (e.g., layer normalization) that may be better suited for small regression tasks.
- Redesign the physics-informed loss with normalized terms and trainable or adaptive weights, and evaluate whether a carefully tuned physics-guided model can outperform the pure MSE baseline.
- Extend the approach to larger and more realistic power system test cases, where the benefits of faster approximate power flow solutions may be more significant.

Overall, this project provides a useful case study in applying neural networks to power systems problems and illustrates some of the challenges in turning intuitive "improvements" into actual performance gains in practice.