Neuro-Fuzzy Load Flow Estimation - Project Summary

B.Tech Final Year Project

Title: A Neuro-Fuzzy System for Real-Time Load Flow Estimation

Team: Abhinav Jha, Akshin Saxena, Akshat Garg

Date: November 2025

Project Overview

This project implements a **hybrid neuro-fuzzy system** for real-time load flow estimation in smart grids using sparse mobile sensor data. The system combines fuzzy logic preprocessing with deep neural networks to predict complete grid states (voltage magnitudes and angles) from limited, noisy sensor measurements.

Key Objectives

- Develop a robust load flow estimation system for disaster-resilient smart grids
- Handle 30-50% missing sensor data with 5-10% noise
- Achieve real-time inference (<100ms per prediction)
- Outperform traditional ANN approaches through fuzzy logic integration

System Architecture

1. Fuzzy Logic Preprocessor

File: `fuzzy_preprocessor.py`

Features:

- **Membership Functions:** Voltage (Low/Normal/High), Current (Low/Medium/High), Power (Low/Medium/High), Data Availability (Sparse/Medium/Dense)

- **Fuzzy Rule Base:** 13 inference rules for confidence and quality scoring
- **Output:** 12 fuzzy features per sample (availability, confidence, quality, membership degrees)

Key Statistics:

- Processes 5,000 samples in 0.5 seconds
- Zero NaN values in output features
- Adaptive normalization based on training data

2. Neural Network Architecture

File: `neurofuzzy_model.py`

Model Specifications:

- **Input:** 52 features (20 sensor + 20 binary masks + 12 fuzzy)
- **Hidden Layers:** 128 → 256 → 128 neurons (ReLU + Dropout 0.2)
- **Output:** 66 grid states (33 voltages + 33 angles)
- **Parameters:** 81,218 trainable parameters
- **Loss Function:** Weighted MSE (2x voltage, 1x angle)

Baseline Comparison:

- Baseline ANN: 79,682 parameters (no fuzzy features)
- Neuro-fuzzy adds only 1,536 parameters (+2%)

3. Training Pipeline

File: `train.py`

Configuration:

- Train/Validation Split: 80/20 (4,000 / 1,000 samples)
- Batch Size: 64
- Optimizer: Adam (lr=0.001, weight_decay=1e-5)
- Scheduler: ReduceLROnPlateau (factor=0.5, patience=5)
- Early Stopping: Patience=15 epochs

Training Results:

Model	Best Val Loss	Best Epoch	Training Time
Neuro-Fuzzy	**1.548**	33	3.23s
Baseline ANN	1.896	21	2.25s

Improvement: 18.38% reduction in validation loss

Performance Metrics

Validation Set Results (1,000 samples)

Overall Performance

Metric	Neuro-Fuzzy	Baseline	Improvement
Overall MAE	0.001309	0.001388	**5.69%**
Overall RMSE	0.018132	0.021226	**14.58%**
Overall R^2	0.228628	0.153159	**49.28%**

Voltage Predictions

Metric	Neuro-Fuzzy	Baseline	Improvement
Voltage MAE (pu)	**0.000337**	0.000373	**9.88%**
Voltage RMSE (pu)	0.003092	0.003662	**15.58%**
Voltage MAPE (%)	0.0348%	0.0385%	**9.77%**
Voltage Max Error (pu)	0.082	0.130	**36.67%**
Voltage R^2	**0.475**	0.263	**80.52%**

Angle Predictions

Metric	Neuro-Fuzzy	Baseline	Improvement
Angle MAE (degrees)	**0.002281**	0.002402	**5.03%**

Metric	Neuro-Fuzzy	Baseline	Improvement
Angle RMSE (degrees)	0.025456	0.029795	**14.56%**
Angle Max Error (deg)	0.995	1.426	**30.20%**
Angle R^2	**0.277**	0.010	**2646.70%**

Inference Time

Model	Mean Time	Std	Meets <100ms Target
Neuro-Fuzzy	**0.089 ms**	0.004 ms	✅ Yes (1120x faster)
Baseline	0.087 ms	0.004 ms	✅ Yes (1150x faster)

File Structure

Core Implementation

```
├── fuzzy_preprocessor.py          # Fuzzy logic system (membership functions, rules)
├── neurofuzzy_model.py            # Neural network architecture (hybrid + baseline)
├── train.py                       # Training pipeline with early stopping
├── evaluate.py                    # Comprehensive evaluation metrics
├── inference.py                   # Production-ready inference script
```

Test Scripts

```
├── test_phase1_fuzzy.py           # Fuzzy preprocessor validation
├── test_phase2_neural_network.py  # Neural network architecture tests
```

Data Generation

```
├── data_generation/
│   ├── main.py                   # Pandapower data generation script
│   └── ieee_33_bus_system.py      # IEEE 33-bus system definition
├── output_generation/
│   ├── sensor_inputs_ieee_33-bus.csv   # 5,000 sparse sensor samples
│   └── grid_states_ieee_33-bus.csv     # 5,000 full grid state labels
```

Trained Models & Results

```
├── checkpoints/
│   ├── neurofuzzy_best.pth        # Best neuro-fuzzy model (966 KB)
│   ├── neurofuzzy_final.pth       # Final epoch model
│   ├── baseline_best.pth          # Best baseline model (947 KB)
│   └── baseline_final.pth         # Final baseline model
├── fuzzy_preprocessor.pkl         # Fitted fuzzy preprocessor
├── neurofuzzy_model.onnx          # Exported ONNX model (for deployment)
├── neurofuzzy_model_stats.pkl     # Normalization statistics
```

Results & Visualizations

```
├── training_histories.json        # Training curves data
├── evaluation_results.json        # Complete evaluation metrics
├── training_history.png           # Training/validation loss plots
├── prediction_comparison.png      # Scatter plots (predicted vs actual)
├── error_analysis.png             # Per-bus error analysis (33 buses)
├── sparsity_impact.png            # Accuracy vs data sparsity
├── fuzzy_membership_functions.png # Fuzzy logic visualization
├── fuzzy_feature_distributions.png# Fuzzy feature histograms
```

Usage Guide

1. Testing the Fuzzy Preprocessor

```
python test_phase1_fuzzy.py
```

Output:

- Validates membership functions
- Tests on 5,000 samples
- Generates fuzzy feature visualizations

2. Testing the Neural Network

```
python test_phase2_neural_network.py
```

Output:

- Validates architecture (52 inputs → 66 outputs)
- Tests forward pass and prediction functions
- Benchmarks inference time

3. Training Models

```
python train.py
```

Output:

- Trains both neuro-fuzzy and baseline models
- Saves best checkpoints to `checkpoints/`
- Generates training history plots

4. Evaluation

```
python evaluate.py
```

Output:

- Computes detailed metrics (MAE, RMSE, R^2, MAPE)
- Per-bus error analysis
- Sparsity impact analysis
- Inference time benchmarking
- Generates 3 visualization plots

5. Real-Time Inference

Demo Mode (Test with validation data)

```
python inference.py --demo
```

Predict from CSV file

```
python inference.py --input sensor_data.csv --output predictions.csv
```

Export to ONNX

```
python inference.py --export-onnx --onnx-path model.onnx
```

Python API Example

```python
from inference import LoadFlowPredictor

# Initialize predictor
predictor = LoadFlowPredictor(
    model_path='checkpoints/neurofuzzy_best.pth',
    fuzzy_processor_path='fuzzy_preprocessor.pkl'
)

# Single prediction
sensor_dict = {
    'meas_0': 0.97,   # Voltage measurement
    'meas_1': 0.95,
    'meas_5': 15.3,   # Current measurement
    # ... (use np.nan for missing sensors)
}

result = predictor.predict_single(sensor_dict, verbose=True)
print(result['voltages'])  # Predicted voltages for all 33 buses
print(result['angles'])     # Predicted angles for all 33 buses
print(result['metadata'])   # Confidence scores
```

Key Achievements

✅ Technical Success

1. **Real-time Performance:** 0.089 ms inference time (1120x faster than 100ms target)
2. **High Accuracy:** Voltage MAE of 0.000337 pu (0.03% error)
3. **Robustness:** Handles 75% data sparsity with minimal accuracy degradation
4. **Scalability:** Processes 1,049,521 samples/second

✅ Fuzzy Logic Benefits

1. **9.88% improvement** in voltage prediction accuracy
2. **36.67% reduction** in maximum voltage error
3. **80.52% improvement** in voltage R^2 score
4. Confidence scoring enables uncertainty quantification

✅ Production Ready

1. Complete inference API with CLI
2. ONNX export for deployment
3. Comprehensive error handling
4. Detailed logging and metrics

Research Contributions

Novel Aspects

1. **Hybrid Architecture:** First application of fuzzy-enhanced features for sparse load flow estimation
2. **Disaster Resilience:** Optimized for 50%+ sensor loss scenarios
3. **Real-Time Capability:** Sub-millisecond inference on CPU
4. **IEEE 33-Bus Validation:** Complete benchmark on standard distribution system

Comparison with Traditional Methods

Method	Voltage MAE	Angle MAE	Inference Time	Handles Sparsity
Newton-Raphson	N/A*	N/A*	~10-50ms	❌ No
Gauss-Seidel	N/A*	N/A*	~20-100ms	❌ No
Simple ANN	0.000373 pu	0.002402°	0.087 ms	⚠️ Partial
Neuro-Fuzzy	**0.000337 pu**	**0.002281°**	**0.089 ms**	✅ Yes

*Traditional methods require full observability (cannot handle missing data)

Dataset Characteristics

IEEE 33-Bus System

- **Voltage Level:** 12.66 kV distribution
- **Total Buses:** 33
- **Total Lines:** 32
- **Total Load:** Variable (50-150% scaling)

Training Dataset

- **Total Scenarios:** 5,000
- **Training Samples:** 4,000 (80%)
- **Validation Samples:** 1,000 (20%)
- **Sparsity:** 53.67% average missing data
- **Noise Level:** 5-10% Gaussian noise
- **Sensors per Sample:** 5-10 mobile sensors (voltages, currents, power flows)

Feature Engineering

- **Input Features:** 20 sparse sensor measurements
- **Binary Masks:** 20 missing data indicators
- **Fuzzy Features:** 12 derived confidence/quality scores
- **Output Targets:** 66 grid states (33 V + 33 θ)

Future Work

Potential Enhancements

1. **Dynamic Sensor Placement:** Optimize mobile sensor locations based on grid topology
2. **Multi-Grid Support:** Extend to IEEE 69-bus, 118-bus systems
3. **Temporal Modeling:** Add LSTM/Transformer layers for time-series prediction
4. **Adversarial Robustness:** Test against sensor spoofing attacks
5. **Hardware Deployment:** Deploy on edge devices (Raspberry Pi, NVIDIA Jetson)

Research Extensions

1. **Adaptive Fuzzy Rules:** Learn membership functions from data
2. **Explainable AI:** Visualize which fuzzy rules contribute most to predictions
3. **Multi-Objective Optimization:** Balance accuracy, speed, and interpretability
4. **Transfer Learning:** Pre-train on multiple grids, fine-tune on specific systems

Dependencies

Core Libraries

```
torch==2.9.1
numpy==2.3.1
pandas==2.3.1
scikit-fuzzy==0.5.0
scikit-learn==1.7.2
matplotlib==3.10.3
```

Data Generation

```
pandapower==3.1.2
scipy==1.16.0
networkx==3.5
```

Model Export

```
onnx==1.19.1
onnxscript==0.5.6
```

Installation

```
pip install -r requirements.txt
```

Citation

If you use this work in your research, please cite:

```
@project{jha2025neurofuzzy,
  title={A Neuro-Fuzzy System for Real-Time Load Flow Estimation},
  author={Jha, Abhinav and Saxena, Akshin and Garg, Akshat},
  year={2025},
  institution={Delhi Technological University},
  type={B.Tech Final Year Project},
  note={Disaster-Resilient Smart Grid Load Flow Estimation using Sparse Mobile Sensors}
}
```

Contact

Project Team:

- Abhinav Jha (2K22/EE/10) - Lead Developer
- Akshin Saxena (2K22/EE/36) - Data Preprocessing
- Akshat Garg (2K22/EE/35) - Signal Processing

Institution: Delhi Technological University
Department: Electrical Engineering
Project Duration: July 2025 - November 2025

License

Acknowledgments

- IEEE for providing standard test systems (IEEE 33-bus)
- Pandapower development team for power system simulation tools
- PyTorch and scikit-fuzzy communities for deep learning and fuzzy logic frameworks

Project Status: ✅ Complete
Last Updated: November 14, 2025
Version: 1.0.0