Deep Learning for Electricity Load Forecasting: An Early-Warning System

Deep Learning Project Report

Karthik Kunnamkumarath Aswin Anil Bindu Sreelakshmi Nair
Hari Saravanan Kopinath Sithamparanathan
Nithin Reddy Muskala Yuvraj Handa

February 2026

Contents

1 Abstract

This project presents the design, development, and evaluation of a deep learning-based early-warning system for detecting high electricity load events using multivariate time-series data. Leveraging the UCI LD2011_2014 dataset, which contains 15-minute interval electricity consumption readings from 370 clients over four years (2011–2014), we developed and compared eight distinct models spanning classical machine learning, recurrent neural networks, convolutional networks, and attention-based architectures. Our pipeline includes data preprocessing, exploratory analysis, event definition using statistical thresholds, sliding window construction, class imbalance handling, and systematic threshold optimization. The Transformer model achieved the best balanced performance (F1 = 0.817), while the Temporal Convolutional Network (TCN) achieved the highest recall (0.928), making it ideal for safety-critical deployments. All experiments are fully reproducible with fixed random seeds.

2 Introduction

2.1 Background

Electricity grid operators face the constant challenge of anticipating sudden spikes in electricity demand. Failure to predict high-load events can lead to grid instability, rolling blackouts, equipment damage, and financial losses. Traditional statistical methods often fall short in capturing the complex temporal dependencies present in electricity consumption data.

Deep learning offers a promising alternative by learning hierarchical temporal representations directly from raw time-series data. Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, have been shown to be effective at modeling sequential dependencies. More recently, Temporal Convolutional Networks (TCNs) and Transformer architectures have emerged as competitive or superior alternatives.

2.2 Problem Statement

Given multivariate electricity consumption time-series data from multiple clients, the objective is to predict whether a high-load event will occur within the next hour (4 timesteps at 15-minute intervals). This is framed as a binary classification problem:

- **Class 0 (Normal):** No high-load event expected within the prediction horizon.
- **Class 1 (Event):** A high-load event is predicted to occur within the next hour.

2.3 Objectives

1. Clean and preprocess the electricity consumption dataset.
2. Define high-load events using statistical thresholds and persistence criteria.
3. Construct early-warning labels with a look-ahead prediction horizon.
4. Build and compare multiple deep learning architectures for early-warning classification.
5. Address class imbalance using weighted training and threshold optimization.
6. Evaluate models using precision, recall, F1-score, and accuracy.

3 Dataset

3.1 Description

The dataset used is the **UCI Electricity Load Diagrams 2011–2014 (LD2011_2014)**, which contains electricity consumption data from 370 clients recorded at 15-minute intervals from 2011 to 2014. Each row represents a timestamp, and each column represents a client's electricity consumption in kilowatts (kW).

Property	Value
Source	UCI Machine Learning Repository
Time Range	2011 – 2014
Sampling Interval	15 minutes
Number of Clients	370
Total Records	~140,256 per client
File Format	Semicolon-separated, comma-decimal

3.2 Data Loading and Cleaning

The raw data required several preprocessing steps:

1. **Column Renaming:** The unnamed first column was identified as the datetime index.
2. **Datetime Parsing:** Timestamps were converted to pandas datetime format.
3. **Decimal Conversion:** European comma-decimal format was converted to standard float representation.
4. **Missing Value Check:** Missing values were identified and handled appropriately.

3.3 Data Subsetting

For computational efficiency during model development, a subset of the data was selected:

- **Clients:** MT_001 through MT_005 (5 clients)
- **Time Period:** June 1 – August 31, 2013 (summer months)
- **Rationale:** Summer months exhibit a higher density of high-load events due to air conditioning demand, making them suitable for supervised learning while preserving realistic class imbalance.

4 Methodology

4.1 Exploratory Data Analysis

Comprehensive exploratory analysis was conducted to understand consumption patterns:

1. **System-Level Total Load:** Aggregate consumption across all clients over time, revealing seasonal trends and an overall upward trajectory.
2. **Short-Window Analysis:** A one-week view (January 2013) showing daily cyclical patterns and intra-day variability.
3. **Individual Consumer Profiles:** Comparison of consumption patterns among sample clients (MT_001–MT_005), revealing heterogeneous usage behaviors.
4. **Hourly Patterns:** Average consumption by hour of day, showing peak demand during business hours (9 AM – 6 PM) and minimum consumption during early morning hours.
5. **Weekly Patterns:** Average consumption by day of week, showing reduced consumption on weekends compared to weekdays.

4.2 Event Definition

High-load events were defined using the following criteria:

1. **Threshold:** The 95th percentile of total system-level consumption was used as the high-load threshold.
2. **Persistence Requirement:** A high-load event is confirmed only when total consumption exceeds the threshold for at least 4 consecutive timesteps (1 hour), filtering out momentary spikes.
3. **Binary Labeling:** Each timestep is labeled as 1 (event) or 0 (normal).

4.3 Early-Warning Label Construction

To transform this into a predictive task, early-warning labels were generated:

- **Prediction Horizon:** 4 timesteps (1 hour ahead)
- **Label Logic:** A timestep receives label 1 if a high-load event occurs within the next 4 timesteps.
- **Implementation:** A forward-looking rolling maximum was applied to the event labels and shifted backward by the prediction horizon.

4.4 Data Preprocessing

4.4.1 Normalization

MinMaxScaler was applied to scale all features to the [0, 1] range. The scaler was fit on the training data only to prevent data leakage.

4.4.2 Sliding Window Construction

Input sequences were created using a sliding window approach:

- **Window Size:** 24 timesteps (6 hours of historical data)
- **Input Shape:** Each sample is a 3D tensor of shape (24 timesteps $\times$ 5 features)
- **Target:** Binary label (0 or 1) at the end of each window

4.4.3 Train–Validation Split

Data was split chronologically (not randomly) to prevent information leakage:

- **Training Set:** First 70% of samples
- **Validation Set:** Last 30% of samples

This time-aware splitting ensures the model is evaluated on future data, reflecting real-world deployment conditions.

4.5 Class Imbalance Handling

Early-warning tasks are inherently imbalanced — high-load events are rare compared to normal operation. Two strategies were employed:

1. **Class-Weighted Training:** Scikit-learn's `compute_class_weight('balanced')` was used to assign higher loss weight to the minority (event) class during training.
2. **Decision Threshold Tuning:** Instead of using the default 0.5 threshold, thresholds were systematically swept from 0.10 to 0.55 to find the optimal precision-recall tradeoff for each model.

5 Model Architectures

5.1 Model 1: Logistic Regression (Baseline)

A logistic regression classifier was trained on flattened sliding windows ($24 \times 5 = 120$ features) to establish a non-temporal benchmark. This model cannot capture sequential dependencies but provides a reference for evaluating the added value of deep learning.

Parameter	Value
Input	Flattened window (120 features)
Max Iterations	1,000
Class Weighting	Balanced

5.2 Model 2: LSTM (Base)

The Long Short-Term Memory network processes the input sequence step-by-step, maintaining a cell state that selectively retains or forgets information through input, forget, and output gates.

Layer	Configuration
LSTM	64 units, return_sequences=False
Dropout	0.3
Dense	32 units, ReLU activation
Output Dense	1 unit, Sigmoid activation
Optimizer	Adam (default lr)
Decision Threshold	0.35

5.3 Model 3: GRU (Base)

The Gated Recurrent Unit is a lighter alternative to LSTM that combines the forget and input gates into a single update gate, reducing the number of parameters while maintaining comparable performance.

Layer	Configuration
GRU	64 units, return_sequences=False
Dropout	0.3

Layer	Configuration
Dense	32 units, ReLU activation
Output Dense	1 unit, Sigmoid activation
Optimizer	Adam (default lr)
Decision Threshold	0.35

5.4 Model 4: LSTM (Tuned)

An enhanced LSTM with increased capacity and tuned hyperparameters:

Layer	Configuration
LSTM	128 units
Dropout	0.3
Dense	32 units, ReLU activation
Output Dense	1 unit, Sigmoid activation
Optimizer	Adam (lr = 5e-4)
Decision Threshold	0.35

5.5 Model 5: GRU (Tuned)

An enhanced GRU with optimized threshold:

Layer	Configuration
GRU	64 units
Dropout	0.3
Dense	32 units, ReLU activation
Output Dense	1 unit, Sigmoid activation
Optimizer	Adam (lr = 0.001)
Decision Threshold	0.20

5.6 Model 6: Temporal Convolutional Network (TCN)

TCNs use causal and dilated convolutions to model long-term dependencies without recurrence, enabling parallel computation.

Parameter	Configuration
Filters	64
Kernel Size	3
Dilations	[1, 2, 4, 8, 16]
Activation	ReLU
Dropout (built-in)	0.2
Skip Connections	Yes
Additional Dropout	0.3
Dense	32 units, ReLU
Output Dense	1 unit, Sigmoid
Optimizer	Adam (lr = 5e-4)
Decision Threshold	0.15 (high-recall mode)

The dilated convolutions with factors [1, 2, 4, 8, 16] create an exponentially growing receptive field, allowing the network to capture dependencies across the full 6-hour input window.

5.7 Model 7: Bidirectional LSTM (BiLSTM)

The BiLSTM processes the input sequence in both forward and backward directions, capturing both past and future context within the input window.

Layer	Configuration
Bidirectional LSTM	64 units per direction (128 total)
Dropout	0.3
Dense	32 units, ReLU activation
Output Dense	1 unit, Sigmoid activation
Optimizer	Adam (lr = 0.001)
Early Stopping Patience	15 epochs

5.8 Model 8: Transformer

The Transformer architecture uses multi-head self-attention to capture both short- and long-range dependencies simultaneously, without the sequential processing constraint of RNNs.

Architecture Components:

1. **Input Projection:** Linear layer projecting 5 input features to 64-dimensional space.
2. **Positional Encoding:** Sinusoidal positional encodings added to provide sequence order information.
3. **Transformer Encoder Blocks (×2):** Each block contains:
   - Multi-Head Self-Attention (4 heads)
   - Layer Normalization with residual connections
   - Feed-Forward Network (128 hidden units)
4. **Global Average Pooling:** Aggregates the sequence into a fixed-size vector.
5. **Classification Head:** Two dense layers (64 $\rightarrow$ 32 units) with dropout, followed by sigmoid output.

Parameter	Configuration
Model Dimension	64
Attention Heads	4
Feed-Forward Dimension	128
Encoder Blocks	2
Dropout	0.2
Optimizer	Adam (lr = 5e-4)

6 Unsupervised Anomaly Detection

In addition to supervised classification, an LSTM Autoencoder was implemented for unsupervised anomaly detection.

6.1 Architecture

The autoencoder learns to reconstruct normal electricity load patterns:

- **Encoder:** LSTM (64 units) compresses the input sequence to a fixed-length vector.
- **Bottleneck:** RepeatVector layer replicates the encoding across all timesteps.
- **Decoder:** LSTM (64 units) reconstructs the original input sequence.
- **Output:** TimeDistributed Dense layer maps back to the original feature space.

6.2 Training

The autoencoder was trained exclusively on normal samples (class 0) from the training set, learning the patterns of typical electricity consumption.

6.3 Anomaly Detection

- Reconstruction error (MSE) was computed for each validation sample.
- The 95th percentile of the training reconstruction error was used as the anomaly threshold.
- Samples with reconstruction error above the threshold were flagged as anomalies.

6.4 Correlation with Events

Cross-tabulation between autoencoder-detected anomalies and actual high-load events was used to evaluate whether the unsupervised approach captures meaningful patterns that correlate with the supervised labels.

7 Training Configuration

All deep learning models shared the following training configuration:

Parameter	Value
Maximum Epochs	30–200 (model-dependent)
Batch Size	64
Loss Function	Binary Cross-Entropy
Early Stopping	Patience 5–15 epochs, restore best weights
Class Weighting	Balanced (sklearn)
Random Seed	42 (NumPy, TensorFlow, Python)
Hardware	CPU (GPU disabled for reproducibility)
TensorFlow Determinism	Enabled

8 Results

8.1 Model Comparison

The table below summarizes the performance of all models on the validation set:

Model	Accuracy	Precision (Event)	Recall (Event)	F1 (Event)
Logistic Regression	0.710	0.791	0.318	0.453
LSTM (Base)	0.751	0.686	0.629	0.657
GRU (Base)	0.796	0.679	0.872	0.764
LSTM (Tuned)	0.746	0.642	0.741	0.688
GRU (Tuned)	0.811	0.796	0.672	0.729
TCN	0.709	0.572	0.928	0.708
BiLSTM	0.842	0.746	0.883	0.809
Transformer	0.852	0.771	0.869	0.817

8.2 Key Observations

1. **All deep learning models outperformed the baseline** logistic regression, confirming the importance of temporal pattern modeling for early-warning prediction.

2. **The Transformer achieved the best F1 score (0.817)** and highest accuracy (0.852), demonstrating that the self-attention mechanism effectively captures both short- and long-range temporal dependencies in electricity load patterns.

3. **The TCN achieved the highest recall (0.928)** when configured with a low decision threshold (0.15), catching the most events. This makes it ideal for safety-critical applications where missing an event is more costly than a false alarm.

4. **LSTM and GRU models performed comparably.** The base GRU (F1 = 0.764) outperformed the base LSTM (F1 = 0.657), while the tuned GRU (F1 = 0.729) achieved the highest precision (0.796) among all models. The LSTM Tuned (F1 = 0.688) showed modest improvement over the base LSTM.

5. **The BiLSTM model (F1 = 0.809)** leveraged bidirectional processing to capture context from both directions within the input window, achieving the second-best F1 score and a recall of 0.883.

6. **Threshold tuning significantly impacts the precision-recall tradeoff.** The optimal threshold varies across models (0.15 for TCN, 0.20 for GRU Tuned, 0.25

for BiLSTM, 0.35 for LSTM, 0.50 for Transformer), highlighting the importance of application-specific calibration.

8.3 Threshold Analysis

Systematic threshold sweeps were conducted for each model, revealing:

- **Low thresholds (0.15–0.20):** Maximize recall at the cost of precision — suitable for high-safety applications.
- **Medium thresholds (0.30–0.35):** Balance precision and recall — suitable for general monitoring.
- **High thresholds (0.45–0.50):** Maximize precision — suitable when false alarms are costly.

9 Discussion

9.1 Strengths of the Approach

1. **End-to-end pipeline:** From raw data to trained models with complete reproducibility.
2. **Multi-architecture comparison:** Fair evaluation of seven architectures under identical conditions.
3. **Practical event definition:** Statistical threshold with persistence requirement reduces noise.
4. **Flexible deployment options:** Different models and thresholds for different use cases.
5. **Unsupervised complement:** The LSTM autoencoder provides anomaly detection capability for novel events not seen during training.

9.2 Limitations

1. **Subset evaluation:** Models were trained on 5 out of 370 clients during a 3-month summer period. Performance on the full dataset and across seasons may differ.
2. **Single prediction horizon:** Only a 1-hour look-ahead was evaluated. Longer horizons may require different architectures.
3. **Fixed event definition:** The 95th percentile threshold is domain-independent. Real-world thresholds would be set by grid operators based on infrastructure capacity.
4. **No external features:** Weather data, calendar events, and economic indicators were not incorporated but could improve prediction accuracy.

9.3 Practical Recommendations

Use Case	Recommended Model	Threshold	Rationale
Balanced early warning	Transformer	~0.50	Best F1 (0.817) — strong precision (0.771) AND recall (0.869)
Maximum event detection	TCN	~0.15	Highest recall (0.928) — misses fewest events

Use Case	Recommended Model	Threshold	Rationale
Minimize false alarms	GRU (Tuned)	~0.30–0.50	Highest precision (0.796) — fewer alerts
Lightweight deployment	GRU (Base)	~0.35	Good F1 (0.764) with only 15,745 parameters

10 Conclusion

This project successfully developed and evaluated an end-to-end deep learning pipeline for early-warning detection of high electricity load events. Seven models were compared under identical experimental conditions, revealing that:

- **Deep learning models capture temporal dependencies** that classical models miss, leading to substantially better early-warning performance. The best deep learning model (Transformer, F1 = 0.817) nearly doubled the baseline logistic regression F1 score (0.453).
- **The Transformer architecture provides the best balanced performance** (F1 = 0.817, Accuracy = 0.852), leveraging self-attention to model complex temporal relationships.
- **The TCN is optimal for high-recall deployments**, catching 92.8% of critical events.
- **Class imbalance handling and threshold tuning are essential** for practical early-warning systems. Class weights of 0.622 (normal) and 2.543 (event) were used to address the inherent imbalance.
- **The LSTM autoencoder offers a complementary unsupervised approach** for detecting novel anomalies, identifying 120 anomalies in the validation set with 69.2% of detected anomalies corresponding to actual high-load events.

The pipeline is fully reproducible, with fixed random seeds ensuring consistent results across runs. The modular design allows easy extension to additional architectures, longer prediction horizons, and real-time deployment.

10.1 Future Work

- Expand to the full dataset (all 370 clients, 2011–2014).
- Integrate external features (weather, calendar, economic data).
- Explore advanced Transformer variants designed for time series (Informer, Autoformer, PatchTST).
- Implement ensemble methods combining Transformer (best F1) and TCN (best recall).
- Deploy as a real-time streaming monitoring dashboard for electricity grid operators.
- Investigate multi-horizon prediction (15 min, 1 hr, 4 hr ahead).

11 References

1. UCI Machine Learning Repository. (2015). *Electricity Load Diagrams 2011–2014*. https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014

2. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780.

3. Cho, K., et al. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. *EMNLP*.

4. Bai, S., Kolter, J. Z., & Koltun, V. (2018). An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *arXiv:1803.01271*.

5. Vaswani, A., et al. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems (NeurIPS)*.

6. Malhotra, P., et al. (2016). LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection. *ICML Workshop on Anomaly Detection*.

7. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

12 Appendix

12.1 A. Environment and Dependencies

Package	Purpose
Python 3.13	Programming language
TensorFlow / Keras	Deep learning framework
NumPy	Numerical computing
Pandas	Data manipulation
Matplotlib	Visualization
Scikit-learn	Preprocessing, baseline model, metrics
keras-tcn	Temporal Convolutional Network implementation

12.2 B. Reproducibility

All random seeds were fixed for full reproducibility:

```
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)
os.environ['PYTHONHASHSEED'] = '42'
tf.config.experimental.enable_op_determinism()
```

12.3 C. Group Member Contributions

Member	Contribution
Karthik Kunnamkumarath	Project management, model implementation & documentation
Aswin Anil Bindu	Data preprocessing, EDA
Sreelakshmi Nair	LSTM and GRU implementation
Hari Saravanan	Dataset preparation, BiLSTM implementation
Kopinath Sithamparanathan	TCN model, threshold tuning
Nithin Reddy Muskala	Transformer model, evaluation
Yuvraj Handa	Autoencoder, visualization