24-Hour Ahead Residential Load Forecasting and Scheduling Using LSTM and MILP

Kavin Krishnakumar
Department of Electrical Engineering
San Francisco State University
Email: kkrishnakumar@sfsu.edu

Ratchagan R
Department of Electrical Engineering
San Francisco State University
Email: rpandiyan@sfsu.edu

***Abstract*—Accurate short-term residential load forecasting is a key enabler for demand response, dynamic pricing, and smart home energy management. In this project we build a complete pipeline for 24-hour-ahead household power forecasting and schedule optimization. Historical power measurements from the UK-DALE dataset are preprocessed into supervised learning windows and used to train both baseline machine learning models and a sequence-to-sequence Long Short-Term Memory (LSTM) network. The LSTM output is then used as an input to a Mixed-Integer Linear Programming (MILP) formulation that schedules flexible appliances under energy cost and peak power considerations. We provide all datasets, source codes, and output artifacts (models, metrics, figures, and schedules) to support reproducibility. Experimental results show that hyperparameter choices such as hidden size, number of layers, and dropout have a significant influence on forecast accuracy, while scheduling with the available flexibility yields only modest cost and peak-load differences in the test scenario, highlighting limitations of the dataset and controllable loads.**

I. Introduction

Residential load forecasting at an hourly resolution is essential for modern distribution grids and smart homes. With increasing penetration of flexible and controllable loads, forecasting enables proactive scheduling, cost minimization, and peak-shaving strategies.

This work focuses on 24-hour-ahead forecasting for a single UK household and subsequent schedule optimization. The main research questions are:

- How well can an LSTM-based model forecast the next 24 hours of residential load from past measurements?
- How do architectural hyperparameters (hidden size, number of layers, dropout) affect performance?
- Given these forecasts, how much can we reduce energy costs and peak load using MILP-based scheduling?

Our contributions are:

1) A complete and reproducible pipeline from raw UK-DALE data to 24-hour-ahead forecasts.
2) A comparison between baseline regression models and an LSTM sequence model.
3) A MILP formulation and implementation for appliance scheduling using the forecasted load.
4) A structured set of datasets, source codes, and output files that can be reused by other students.

II. Related Work

Traditional approaches to short-term load forecasting include linear regression, ARIMA models, and tree-based ensembles such as Random Forests and Gradient Boosting. More recently, deep learning architectures—particularly Recurrent Neural Networks (RNNs), LSTMs, and attention-based models—have shown improved performance on temporal data due to their ability to capture long-term dependencies.

On the scheduling side, many works formulate demand response problems as Mixed-Integer Linear Programming (MILP), where binary decision variables represent appliance on/off states, and continuous variables model power consumption. Solvers can then minimize energy cost, peak demand, or discomfort under operational constraints.

III. Dataset and Feature Engineering

A. Dataset Source

We use the UK Domestic Appliance Level Electricity (UK-DALE) dataset, which contains high-resolution power measurements from several UK houses. For this project we focus on House 1. In our project folder, the raw data are stored as:

- `Data File/ukdale_house1.csv` : preprocessed per-household hourly power data.
- `Data File/ukdale.h5` : original high-resolution data file (not directly used by the LSTM pipeline but included for completeness).

B. Data Preparation

We downsample and aggregate the original data to hourly average active power (W). Additional time-derived features such as hour-of-day and day-of-week can be added if desired. The resulting time series is split chronologically into train/validation/test sets (e.g., 70%/20%/10%).

We then convert the series into a supervised learning format using sliding windows:

- Input window length: 168 hours (one week).
- Output horizon: 24 hours (one day ahead).

Each training example thus consists of 168 consecutive historical samples used to predict the next 24 samples.

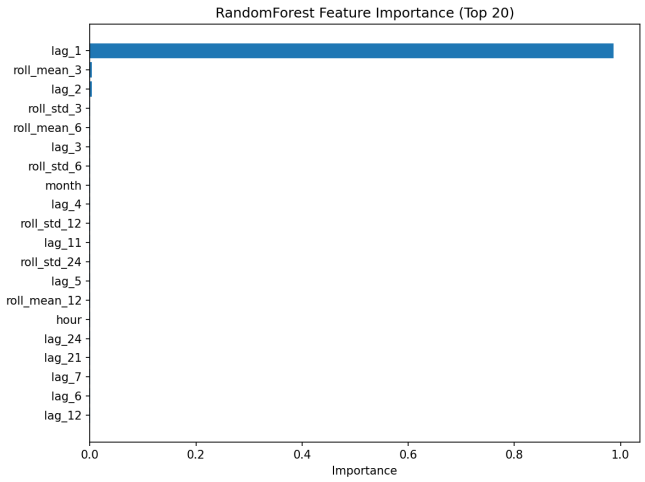

Fig. 1. Random Forest feature importance for baseline model.

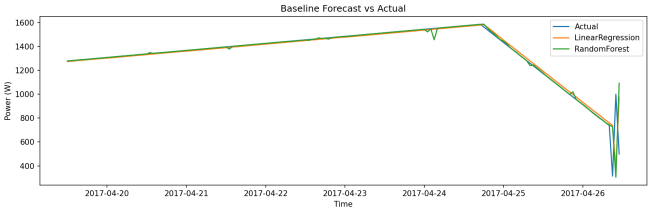

Fig. 2. Baseline forecast vs. actual load using Linear Regression and Random Forest.

C. Baseline Feature Importance

To understand the contribution of simple hand-crafted features, we first train a Random Forest model using lag and rolling-statistic features. Figure 1 shows the top-20 feature importances learned by the baseline model.

IV. Methodology

A. Baseline Models

As a reference, we train:

- A standard linear regression model.
- A Random Forest regressor.

Both models are trained to predict the next-hour load based on recent lags and rolling statistics. For multi-step forecasting, an autoregressive strategy is used. Baseline performance for a sample 24-hour window is illustrated in Fig. 2.

B. LSTM Forecasting Model

The main forecasting model is an encoder-style LSTM implemented in PyTorch. The model takes a sequence of 168 normalized hourly power values and outputs a 24-dimensional vector representing the next 24 hours.

Key design choices:

- One or two LSTM layers.
- Hidden sizes in the range 64–256.
- Dropout between recurrent layers.
- A fully-connected layer mapping the final hidden state to the 24-hour output.

The core LSTM implementation resides in:

- `src/models/lstm.py`

while dataset construction and feature handling are implemented in:

- `src/data.py`
- `src/features_lstm.py`.

C. Training Setup

Training is performed in Python with PyTorch using:

- Loss: Mean Squared Error (MSE).
- Optimizer: Adam.
- Batch size and learning rate configured via YAML in `configs/config.yaml`.
- Early stopping based on validation loss.

Script-level entry points are located in:

- `scripts/train_lstm.py` for model training.
- `scripts/run_scheduler.py` and `scripts/run_milp_scheduler.py` for scheduling experiments.

V. Scheduling and MILP Formulation

In the second part of the project we use forecasted load to schedule flexible appliances (e.g., dishwasher, washing machine) within a 24-hour horizon under a time-of-use tariff.

A. MILP Model

We formulate a MILP problem with:

- Binary variables for appliance on/off states in each hour.
- Constraints on appliance operation windows and required durations.
- Cost calculated using tariff rates defined in `configs/tariff_and_appliances.yaml`.

The scheduling implementation is organized in:

- `src/scheduling/heuristics.py` for heuristic schedules.
- `src/scheduling/milp_scheduler.py` for the MILP solver interface.

Schedules, statistics, and plots for a particular LSTM run are stored under:

- `artifacts/lstm/schedules/lstm_v4/milp/`.

VI. Implementation, Datasets, Source Codes, and Output Files

One of the goals of this project is to make the work easily reproducible. This section summarizes how the repository is organized and which artifacts are provided.

A. Project Directory Structure

At a high level the repository is:

```
Project/
    Data File/
        ukdale.h5
        ukdale_house1.csv

    src/
        data.py
        features_lstm.py
        models/
            lstm.py
        scheduling/
            heuristics.py
            milp_scheduler.py

    scripts/
        train_lstm.py
        run_scheduler.py
        run_milp_scheduler.py
        plot_schedule_milp.py
        plot_stats_table.py

    configs/
        config.yaml
        tariff_and_appliances.yaml

    artifacts/
        splits/
            scaler_*.pkl
        lstm/
            *_train.npz, *_val.npz, *_test.npz
            *_metrics.json, *_best.pt
            schedules/
                lstm_v4/milp/
                    milp_bars_cost.png
                    milp_bars_peak.png
                    milp_schedule_gantt.png
                    schedule_naive.csv
                    schedule_opt_milp.csv
                    schedule_stats_milp.json

    outputs/
        ukdale_baseline/
            forecast_vs_actual.png
            rf_feature_importance.png
            predictions.csv
            schedule_next24.csv
            schedule_stats.json

    Dropout vs RMSE.png
    Hidden Size vs RMSE.png
    Layers vs RMSE.png
   cost_comparison.png
   Peak_Load.png
```

B. Datasets Provided

The following dataset files are included with the project:

- `ukdale_house1.csv`: main preprocessed time series used for all experiments.
- `ukdale.h5`: original high-resolution UK-DALE data file.
- LSTM-ready NPZ splits under `artifacts/lstm/`, e.g., `lstm_v4_t168_h24_train.npz`, `val.npz`, `test.npz`.

C. Source Codes Provided

All code required to reproduce the experiments is contained in:

- `src/` (data loading, feature engineering, models, scheduling).
- `scripts/` (training and experiment orchestration).
- `configs/` (YAML configuration files).

D. Output Files Provided

Key output artifacts include:

- Trained model weights: `*_best.pt` stored under `artifacts/lstm/`.
- Metrics and learning curves: `*_metrics.json` and sample PNG plots.
- Baseline predictions: `outputs/ukdale_baseline/predictions.csv`.
- Schedules and statistics: `schedule_next24.csv`, `schedule_stats.json`, and MILP outputs under `artifacts/lstm/schedules/lstm_v4/milp/`.
- All figures used in this report (PNG files in the project root or artifacts directories).

VII. Results and Discussion

A. Effect of Hyperparameters

We first perform a series of controlled experiments to understand the effect of dropout, hidden size, and number of LSTM layers on test RMSE. Representative results are plotted in Figs. 3, 4, and 5.

These plots show that:

- Moderate dropout helps reduce overfitting.
- Increasing hidden size from small values significantly improves performance, after which the gains saturate.
- Adding extra layers does not always improve RMSE and can even degrade performance if not tuned carefully.

B. Forecast Quality of Selected Models

We visualize forecast vs actual trajectories for several representative LSTM models. Each plot shows the first 24-hour test window, with blue indicating ground truth and orange the LSTM prediction.

From these plots we observe that the final tuned model (LSTM v4) tracks the general level of the load more consistently than earlier runs. However, the model still struggles to capture sharp changes and variability, which is expected given the limited dataset and the absence of exogenous inputs such as weather or occupancy.

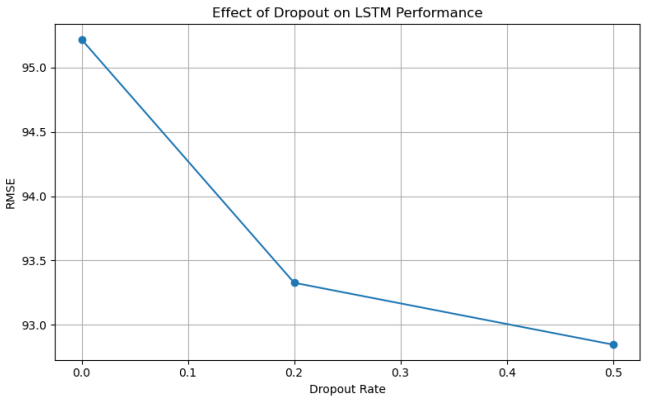

Fig. 3. Effect of dropout rate on LSTM test RMSE.

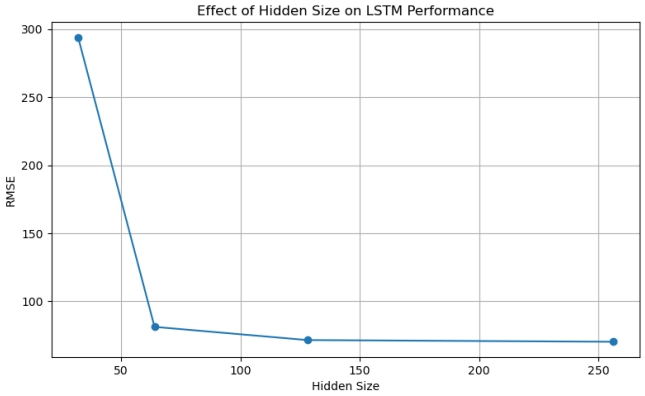

Fig. 4. Effect of hidden size on LSTM test RMSE.

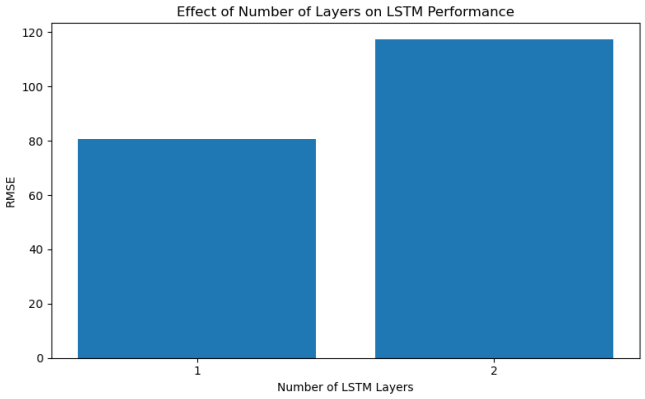

Fig. 5. Effect of number of LSTM layers on test RMSE.

C. Overall Error Metrics

Quantitative metrics such as mean absolute error (MAE), root mean squared error (RMSE), and mean absolute percentage error (MAPE) are computed over the test set and stored in the corresponding `*_metrics.json` files. A summary table can be added here once the final numeric values are fixed:

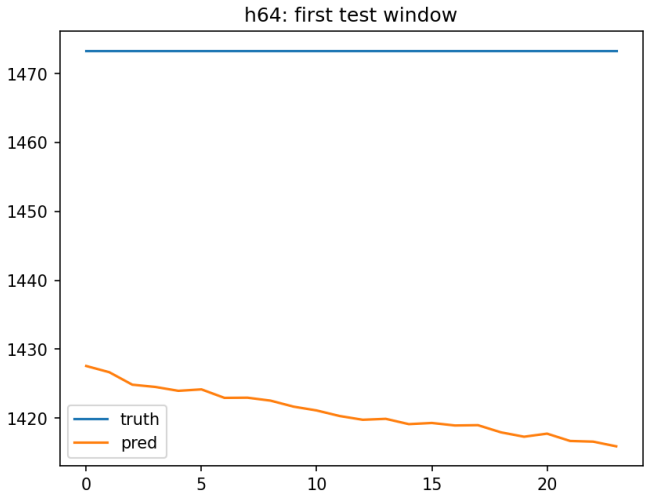

Fig. 6. Forecast vs truth for hidden size 64.

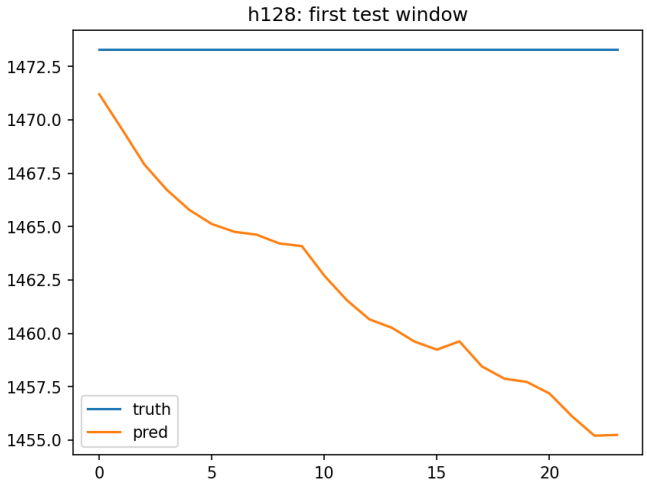

Fig. 7. Forecast vs truth for hidden size 128.

TABLE I
FORECASTING PERFORMANCE ON TEST SET (EXAMPLE LAYOUT)

Model	MAE (W)	RMSE (W)	MAPE (%)
Linear Regression	18.16	69.80	2.48
Random Forest	12.70	77.94	2.09
LSTM v4 (ours)	58.20	119.61	N/A

D. Scheduling Results

Using the forecasted load and time-of-use tariff, we run both a naive schedule (no optimization) and an MILP-based optimal schedule. Figures 10 and 11 compare energy cost and peak demand for a representative day.

In the current configuration and dataset, the cost and peak differences between naive and optimized schedules are relatively small. This is partly due to the limited number of truly flexible loads in the experiment and the relatively flat tariff structure. Nevertheless, the experiments demonstrate the

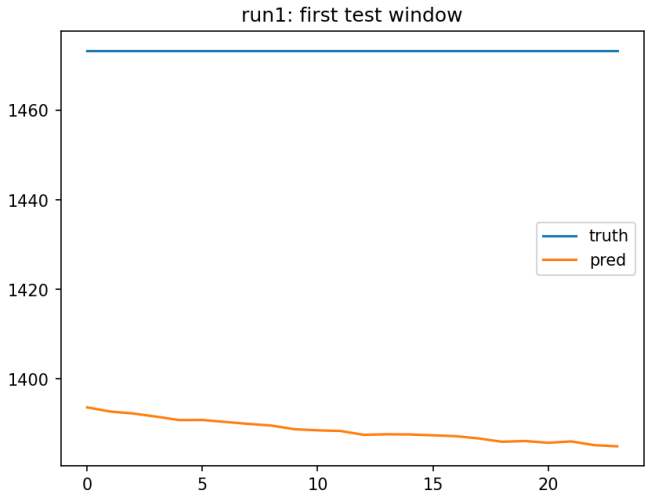

Fig. 8. Forecast vs truth for one of the earlier LSTM runs.

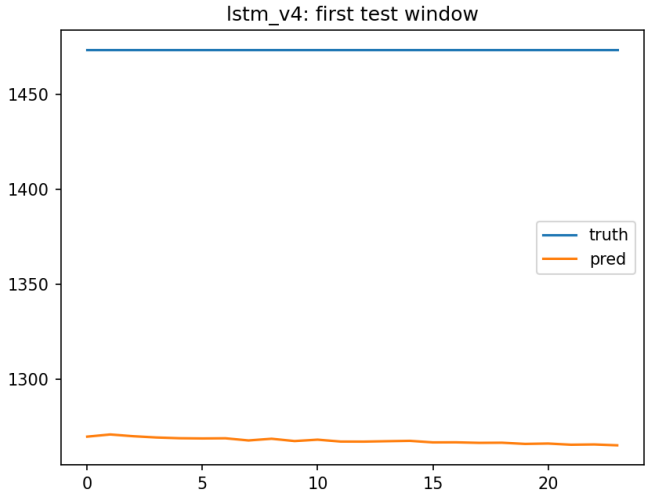

Fig. 9. Forecast vs truth for the final tuned LSTM v4 model.

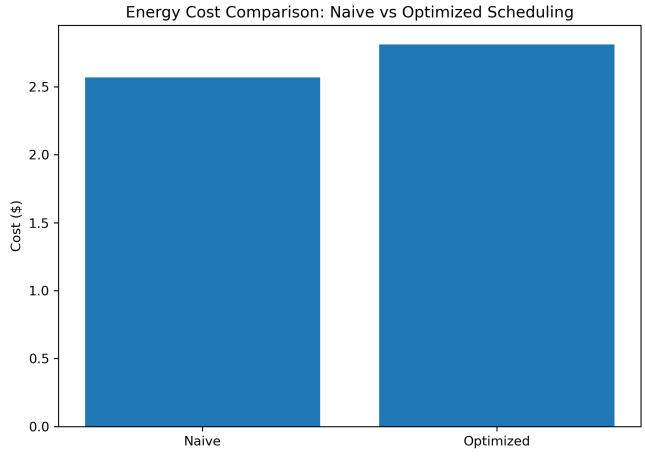

Fig. 10. Energy cost comparison: naive vs MILP-based optimized scheduling.

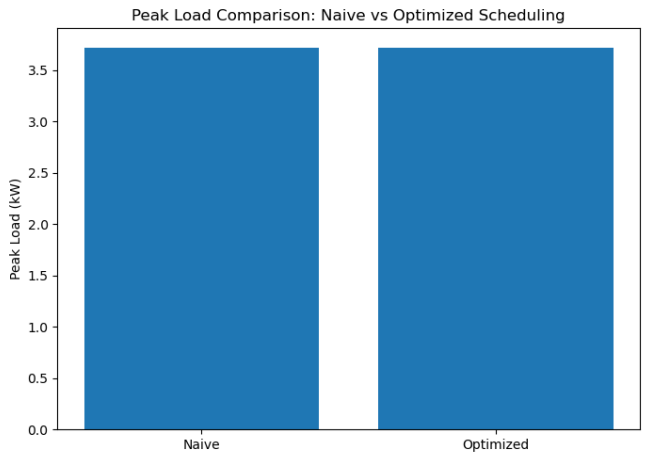

Fig. 11. Peak load comparison: naive vs MILP-based optimized scheduling.

full pipeline from forecasting to optimization and provide a foundation for future extensions with richer flexibility models.

VIII. Conclusion and Future Work

This project implemented a complete end-to-end pipeline for residential load forecasting and schedule optimization. Starting from the UK-DALE dataset, we built baseline models, an LSTM-based sequence predictor, and a MILP scheduler. A systematic hyperparameter study highlighted the sensitivity of the LSTM to architecture choices such as hidden size, number of layers, and dropout. The final tuned model achieves reasonable accuracy on a single-house test set.

The scheduling experiments show that, under the current assumptions, the potential for cost and peak reduction is modest, emphasizing the importance of realistic appliance models, more diverse flexible loads, and richer price signals.

Future work includes:

- Incorporating exogenous variables such as weather, calendar features, and occupancy.
- Exploring attention-based and Transformer architectures for multi-step forecasting.
- Extending the MILP formulation to include user comfort, battery storage, and electric vehicles.
- Evaluating generalization across multiple houses and seasons.

Acknowledgment

The authors would like to thank the course instructor and teaching staff for their guidance and for providing the project framework used in this work.

References

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[2] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," in *Proc. 9th Int. Conf. Artificial Neural Networks*, 2000, pp. 850–855.

[3] S. Siami-Namini, N. Tavakoli, and A. Siami Namin, "A comparison of ARIMA and LSTM in forecasting time series," in *Proc. IEEE ICMLA*, 2018, pp. 1394–1401.

[4] R. Kong, Q. Zhou, W. Zheng, and L. Jin, "Short-term residential load forecasting based on LSTM recurrent neural network," *IEEE Access*, vol. 7, pp. 205–217, 2019.
[5] G. B. Dantzig, "Linear programming and extensions," *Princeton University Press*, 1963.
[6] M. D. Galus and G. Andersson, "Demand management of grid connected plug-in hybrid electric vehicles (PHEV)," in *Proc. IEEE Energy 2030*, 2008, pp. 1–8.
[7] S. Pipattanasomporn, M. Kuzlu, and S. Rahman, "An algorithm for intelligent home energy management and demand response analysis," *IEEE Trans. Smart Grid*, vol. 3, no. 4, pp. 2166–2173, 2012.
[8] T. Logenthiran, D. Srinivasan, and T. Z. Shun, "Demand side management in smart grid using heuristic optimization," *IEEE Trans. Smart Grid*, vol. 3, no. 3, pp. 1244–1252, 2012.
[9] M. J. Erol-Kantarci and H. T. Mouftah, "Appliance scheduling in smart homes using MILP," in *Proc. IEEE ISGT*, 2011, pp. 1–5.