Appliance-Level Load Disaggregation using Machine Learning Models SustLabs

Project Report

Group Project 4

Guide: Prof. Anupama Kowli

Full Name	Roll No.
Abhishek Savaliya	21d070065
Simran Singh	21D070073
Chinta Siva Madhav	21D070020

Acknowledgements

We would like to express our sincere gratitude to **Prof. Anupama Kowli** for providing us with the opportunity to work on this project *"Appliance-Level Load Disaggregation using Machine Learning Models"*, in collaboration with **Sustlabs**. Her guidance and the knowledge gained through this course have been invaluable and greatly contributed to the successful completion of this project. We would also like to thank SustLabs team for providing us the valuable datasets and the opportunity to work on the real-world domain application. We would also like to thank TAs Gopal, Ojo and Deepti for their help in coordinating with the SustLab team.

Contents

1 | Introduction and Problem Statement

This project revolves around solving the Load desegregation problem. The project is part of EE467 course in collaboration with SustLabs. We aim to build a machine learning model that can solve the load-desegregation problem with best possible performance.

SustLab team has provided us the dataset. It contains appliance level data which means we have data of active and reactive power usage of different household electronics appliances that they have gathers through rigorous expeimentations. Now it is our task to aggregate these data to create real time household simulations and train Machine Learning models over them for classification and regression tasks.

2 | Data Aggregation

2.1 | Creating Daily scenarios based on Real-world situations

First we aim to create a daily profile of equipment usage based on hour of the day. We acheive this by creating hourly probabilities data as follows:

```
def get_realistic_usage_probability(self, hour, appliance_type):
    """Define realistic usage probabilities based on time of day
       """
    base_probabilities = {
        'AC': 0.3, 'Fridge': 0.8, 'Kettle': 0.2, 'Iron': 0.1,
        'Washing_machine': 0.15, 'Oven': 0.1, 'Geyser': 0.2, '
           Induction': 0.15
    }

    time_adjustments = {
        'AC': {**{h: 0.8 for h in [12, 13, 14, 15, 16]},
               **{h: 0.1 for h in [0, 1, 2, 3, 4, 5]},
               **{h: 0.5 for h in [20, 21, 22]}},

        'Fridge': {h: 0.9 for h in range(24)},

        'Kettle': {**{h: 0.4 for h in [7, 8, 18, 19]},
                   **{h: 0.05 for h in [0, 1, 2, 3, 4, 5]}},

        'Iron': {**{h: 0.3 for h in [8, 9, 18, 19]},
                 **{h: 0.0 for h in [0, 1, 2, 3, 4, 5]}},

        'Washing_machine': {**{h: 0.3 for h in [9, 10, 15, 16,
           19, 20]},
                            **{h: 0.0 for h in [0, 1, 2, 3, 4,
                               5]}},

        'Oven': {**{h: 0.25 for h in [12, 13, 19, 20]},
                 **{h: 0.0 for h in [0, 1, 2, 3, 4, 5]}},

        'Geyser': {**{h: 0.4 for h in [6, 7, 20, 21]},
                   **{h: 0.1 for h in range(24)}},
```

```
        'Induction': {**{h: 0.3 for h in [12, 13, 19, 20]},
                      **{h: 0.0 for h in [0, 1, 2, 3, 4, 5]}}
    }

    base_prob = base_probabilities.get(appliance_type, 0.1)
    adjustment = time_adjustments.get(appliance_type, {}).get(
       hour, 1.0)

    return base_prob * adjustment
```

- **Base Usage Probability:**
  - Every appliance has a default probability that reflects how often people generally use it. For example, fridge is 0.8 since there is high probability that it is on most times. On contrary, iron has 0.1 base probability since it is run at very small amount of time during the day.
  - Examples:
    - Washing Machine: 0.15
    - AC: 0.3
    - Kettle: 0.2
    - Iron: 0.1
    - Geyser: 0.2
  - This base value acts as the general likelihood of usage throughout the day.
- **Time-of-Day Adjustment:**
  - Human usage patterns are not uniform throughout the day. So the function modifies the base probability according to the hour of the day (0–23). For example, Geyser:
    - High usage in early morning 6–7 and evening 20–21
    - Low baseline usage (0.1) at all hours
    - Adjustment multiplies the base accordingly

    .
  - Other Examples:
    - **AC:** Higher probability in the afternoon (12–16), lower during night hours (0–5).
    - **Kettle:** Higher usage during morning (7–8) and evening (18–19); very low at night.
    - **Iron:** Peaks during morning and evening routines; near zero at night.
    - **Washing Machine:** Moderate use in late morning and evening.
    - **Fridge:** Constant adjustment (0.9) as it remains active at all hours.
- **Final Probability Calculation:**

- The final usage probability is computed as:

$$P_{\text{usage}} = P_{\text{base}} \times A_{\text{time}}$$

- This ensures that both general appliance behaviour and hour-specific patterns are captured realistically.

- **Importance for Simulation:**
  - Prevents unrealistic activation patterns (e.g., kettle at midnight or AC at 3 AM).
  - Produces household energy profiles that match real-world human routines.
  - Enhances the realism and diversity of aggregate training data.

2.2 | Simulation of Multi-State Appliance Behaviour

We created a function that models appliance behavior as a small state machine with probabilistic transitions and realistic power sampling: simple appliances flip rarely, multi-state appliances can change levels after being on for a while, and off→on decisions depend on time-of-day usage probabilities and real measured samples. We first calculated the states for each appliance, the procedure for which is shown in section 2.2.2.
The function `simulate_multi_state_appliance` models the realistic power consumption of an appliance over time. It uses a probabilistic state-transition mechanism, combined with real appliance measurements, to generate active and reactive power values at each timestep. The behaviour can be divided into two major categories: simple ON/OFF appliances and multi-state appliances. The overall logic is described below.

2.2.1 | Mathematical model of state transitions

Define the discrete appliance state set $S = \{0, 1, \ldots, N\}$ where 0 means *OFF* and $1, \ldots, N$ are ON states. Let $s_t \in S$ be the state at time index t, and let d_t denote the duration (in samples) that the appliance has continuously remained in the current state.

Simple (ON/OFF) appliances: For appliances with at most two states ($N \leq 1$), transitions follow fixed probabilities:

$$\Pr(s_{t+1} = 0 \mid s_t = 1) = p_{\text{off}}^{\text{simple}} = 0.02,$$
$$\Pr(s_{t+1} = 1 \mid s_t = 0) = p_{\text{on}}^{\text{simple}} = 0.10.$$

When the appliance is ON and remains ON, the simulator returns a measured sample (P_t, Q_t) from the real trace at the index $\min(t, \ |D| - 1)$, where P_t is active power and Q_t is reactive power.

Multi-state appliances: For $N \geq 2$, let $T_{\text{thresh}} = 5 \cdot (N + 1)$ (the code uses $5 \cdot |states|$ as the threshold). If $s_t > 0$ and $d_t > T_{\text{thresh}}$ then the appliance attempts a state change with probability

$$\Pr(\text{attempt change} \mid s_t) = p_{\text{change}} = 0.10.$$

Given an attempt to change:

$$\Pr(s_{t+1} = 0 \mid \text{change}) = p_{\text{off|change}} = 0.30,$$

and otherwise the appliance transitions to one of the other ON states uniformly at random:

$$\Pr(s_{t+1} = j \mid \text{change},\ j \in \{1, \ldots, N\} \setminus \{s_t\}) = \frac{0.70}{N-1}.$$

If the appliance is OFF ($s_t = 0$), it may start operating based on a time-dependent usage probability:

$$\Pr(s_{t+1} = j \mid s_t = 0) = u(h) \cdot \frac{1}{N} \quad \text{for } j \in \{1, \ldots, N\},$$

where h is the hour-of-day derived from the sample index and $u(h) \in [0, 1]$ is provided by `get_realistic_usage_probability`(h).

Power selection: When a transition to a target ON state j is chosen, the simulator searches the appliance's measurement dataset D for a sample $s \in D$ with active power $P(s)$ satisfying

$$|P(s) - P_{\text{target},j}| < 50 \text{ W}.$$

If such samples exist the simulator uses one of them (the code picks the first match); otherwise it uses the nominal target power $P_{\text{target},j}$ with reactive power set to zero.

Combined rule: The function returns at each time t a triple

$$\big(P_t,\ Q_t,\ s_t\big),$$

where s_t is sampled according to the probabilistic rules above and (P_t, Q_t) are selected from the real traces when available, ensuring realistic active/reactive power behaviour.

2.2.2 | State Selection Method

As seen in the function `simulate_multi_state_appliance`, we are using the states of each appliance for simulation. In this section, we are showing logic of how states are generated for each appliance. It is done using K-means clustering on the NILM data.

- **Preprocessing:**
  - Active power samples below 5 W are removed to eliminate noise and the OFF state.
  - Remaining values represent meaningful consumption levels.
- **Cluster-Based State Detection (Preferred Method):**
  - K-means clustering is applied for $k = 2$ to 5 clusters.
  - For each k, the silhouette score is computed to measure how well-separated the clusters are.
  - The value of k with the highest silhouette score is chosen as the optimal number of ON states.
  - Cluster centers represent the power levels of the operational states.
  - Thresholds between states are computed as midpoints between consecutive cluster centers.

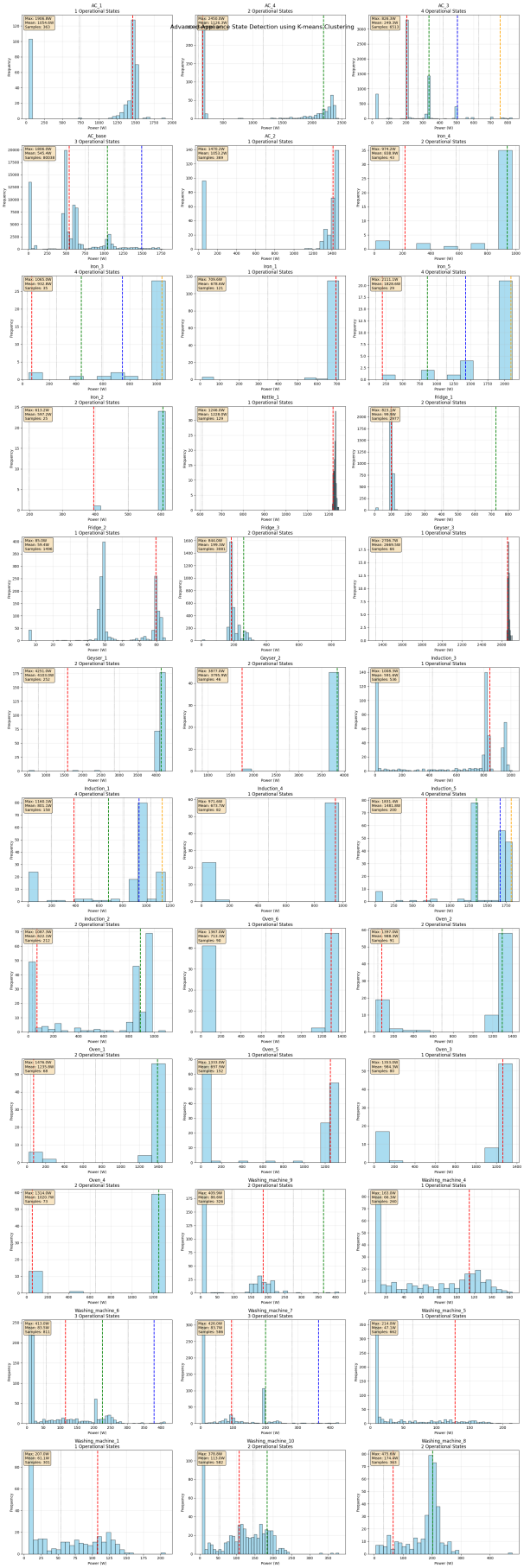

Figure 2.1: State Detection for each Appliance

- **Fallback Method (Used when clustering is unreliable):**
  - If clustering is unstable (silhouette score < 0.4) or only one cluster is found: a simpler statistical approach is used.
  - If the power variance is low, a single ON state is selected using the mean power.
  - If variance is high, two ON states are estimated using the 25th and 75th percentiles.
- **State Filtering:**
  - States that are too close to each other (difference $< 50\,\text{W}$) are removed to ensure meaningful separation between power levels.
  - Final states always include the OFF state ($0\,\text{W}$) followed by sorted ON power levels.
- **Final Threshold Computation:**
  - For each pair of consecutive states, a threshold is computed as the midpoint between their power values.
  - During simulation, these thresholds determine which state corresponds to a given power sample.

2.3 | Simulating for multiple households

The function `create_realistic_households` synthesizes realistic aggregate power consumption data for multiple household types. It mimics how real homes differ in size, appliance ownership, and daily usage patterns. The overall pipeline is summarised below:

2.3.1 | Household Type Definitions

Three representative household categories are defined:

- **Small Apartment** — few appliances with probabilistic presence.
- **Medium Family Home** — stable set of essential appliances.
- **Large Household** — multiple appliances of each type.

Each category specifies how many units of each appliance type may appear, using lists such as $[0, 1]$ or fixed values. These act as sampling distributions for appliance ownership.

```
household_configs = [
        {
            'type': 'Small_Apartment',
            'appliance_options': {
                'Fridge': [1],  # Always have fridge
                'Kettle': [0, 1],  # 50% chance
                'Iron': [0, 1],   # 50% chance
                'AC': [0, 1],     # 50% chance in small
                   apartment
                'Washing_machine': [0, 1]  # 50% chance
```

```
            },
            'description': 'Small apartments with varying
               appliance combinations'
        },
        {
            'type': 'Medium_Family',
            'appliance_options': {
                'Fridge': [1],       # Always have fridge
                'AC': [1, 2],        # 1 or 2 ACs
                'Geyser': [1],       # Always have geyser
                'Iron': [1],         # Always have iron
                'Kettle': [1],       # Always have kettle
                'Washing_machine': [1],  # Always have washing
                   machine
                'Induction': [0, 1] # 50% chance
            },
            'description': 'Medium family homes with consistent
               core appliances'
        },
        {
            'type': 'Large_Household',
            'appliance_options': {
                'AC': [2, 3],            # 2 or 3 ACs
                'Fridge': [1, 2],        # 1 or 2 fridges
                'Geyser': [1, 2],        # 1 or 2 geysers
                'Induction': [1],        # Always have induction
                'Iron': [1],             # Always have iron
                'Kettle': [1],           # Always have kettle
                'Oven': [1],             # Always have oven
                'Washing_machine': [1, 2]  # 1 or 2 washing
                   machines
            },
            'description': 'Large households with multiple
               appliances'
        }
    ]
```

2.3.2 | Random Appliance Assignment

For every household instance, a random number of appliances is sampled from the options provided. Specific appliance *models* (e.g., AC_3, Fridge_2) are chosen at random from the dataset.

2.3.3 | Per-Household Simulation Setup

For each selected appliance:

- its state is initialized to OFF,
- its last power value is stored (initially 0),
- a "duration counter" tracks how long the appliance has been active.

These variables evolve through the day according to the state-transition model described in section 2.2.1

2.3.4 | Time-Series Generation

The simulation runs at 5-second resolution. Thus, for each household, one day contains:

$$\texttt{samples\_per\_day} = 17280 \text{ samples.}$$

At each timestep, the following occur:

1. Compute hour-of-day and contextual labels (timestamp, day-of-week, household ID, etc.).
2. For every appliance:
   - retrieve its previous state,
   - apply the `simulate_multi_state_appliance` logic,
   - update power, reactive power, and categorical state,
   - accumulate power into the household aggregate.
3. Store a complete row containing:
   - all per-appliance power values,
   - all per-appliance states,
   - aggregate active and reactive power.

This produces a fine-grained synthetic NILM dataset with realistic variability.

3 | Machine Learning on the Data

The second phase in our work is building machine learning model for classification and regression on the aggregate data. First, we will do feature engineering

3.1 | Feature Engineering

The function `create_features_for_ml` constructs a rich set of descriptive features from the aggregated household load data. These features encode temporal patterns, statistical behaviour, transient events, and electrical characteristics that are essential for NILM classification and regression tasks. The major feature categories are summarized below.

3.1.1 | Aggregate Power Features

- **Active Power:** Total active power at each timestep.
- **Reactive Power:** Total reactive power at each timestep.

These serve as the fundamental measurements from which all higher-level features are derived.

3.1.2 | Time-Based Features

- **Minute of Day** ($0 \leq m < 1440$) captures daily cycles.
- **Hour Embeddings:** Sine and cosine transforms:

$$\sin\left(\frac{2\pi h}{24}\right), \qquad \cos\left(\frac{2\pi h}{24}\right),$$

  producing a smooth 24-hour representation.
- **Weekend Indicator** distinguishes weekday vs weekend behaviour.

These features help models learn time-of-use patterns typical of each appliance.

3.1.3 | Statistical Features (Rolling Window)

Using a centered window of size w, the following properties of active power are computed:

- Rolling mean and standard deviation,
- First differences for active and reactive power,
- Higher-order moments: skewness and kurtosis,
- Rolling variance of active power.

These capture local fluctuations and steady-state vs. dynamic load behaviour.

3.1.4 | Power Signature Features

To characterize the electrical nature of the load:

- **Active-to-Reactive Ratio** indicates inductive/capacitive trends.
- **Power Magnitude:**

$$\sqrt{P^2 + Q^2},$$

  capturing the complex power vector length.
- **Power Factor Estimate:**

$$\frac{P}{\sqrt{P^2 + Q^2}}.$$

Such features help distinguish appliances with different electrical signatures.

3.1.5 | State-Aware Features

- **State Change Indicator** flags abrupt changes ($|P_t - P_{t-1}| > 50$ W).
- **Power Level Category** assigns power ranges (e.g., 0–10 W, 10–100 W, 100–500 W, etc.).

These encode implicit state transitions associated with individual appliances.

3.1.6 | Event-Based Features

To capture appliance on/off events and transient behaviour, the following are computed:

- **Event Onset:** Detected when the power jump exceeds a threshold.
- **Event Magnitude:** Difference between instantaneous and local rolling mean.
- **Rise Time and Fall Time:** Short-term accumulation of rapid increases or decreases in power.

These features emulate the signatures used in classical NILM event-based methods.

3.1.7 | Additional Load Dynamics

Several helper features quantify long-term or structural behaviour:

- **Spike Duration** and **Event Duration** measure how long power remains above a given threshold.
- **Periodicity Estimate** via simple autocorrelation to detect repeated daily cycles.
- **State Complexity** quantifies density of local state changes over a sliding window.
- **Load Type Classification** (resistive / inductive / capacitive) based on power ratio.

These enhance the model's ability to distinguish appliances with repetitive or complex operating behaviour.

Label Construction

For each appliance type (e.g., AC, Iron, Kettle, etc.):

- **Classification labels:** a binary indicator (ON/OFF) based on whether any appliance of that type exceeds 10 W.
- **Regression labels:** total active power of all appliances of that type.

Final Output

The resulting feature matrix includes all engineered features and spans tens of thousands of samples. This forms a high-quality training dataset enabling both classification (appliance ON/OFF detection) and regression (power estimation) models to learn detailed appliance signatures.

3.2 | Classification task

The classification stage aims to identify which appliance types are ON or OFF at each timestep based on the engineered feature set. The model is trained in a multi-label setting since multiple appliances may be active simultaneously.

3.2.1 | Model Design

- **Input:** The feature matrix produced in section 3.1, containing aggregate power statistics, temporal encodings, transient signatures, and event-related features.
- **Labels:** Binary ON/OFF indicators for each appliance type (AC, Iron, Kettle, Fridge, Geyser, Induction, Oven)
- **Data Split:** Instead of random row-wise splitting, the dataset is divided *by household*:

  $$\text{Train households} \cup \text{Test households} = \text{All households}, \qquad \text{Train} \cap \text{Test} = \varnothing.$$

  This ensures the model generalizes to unseen households.
- **Model:** A `RandomForestClassifier` wrapped in a `MultiOutputClassifier`, allowing simultaneous prediction of multiple appliance types:
  - 100 decision trees
  - Maximum depth = 15
  - Minimum samples to split = 10
  - Parallel execution
- **Evaluation Metrics:**
  - **Accuracy** per appliance type
  - **F1-score** per appliance
  - **Macro F1-score** for overall performance
  - **Confusion matrix** per appliance type

3.2.2 | Training Summary

- Total households: 90
- Training households: 63
- Testing households: 27
- Training samples: 1,088,640
- Testing samples: 466,560

3.2.3 | Overall Performance

Accuracy = 0.7939, **Macro F1-score** = 0.7113

Appliance Type	Accuracy	F1-score
AC	0.9901	0.9934
Iron	0.9635	0.4426
Kettle	0.9991	0.8856
Fridge	0.9211	0.9366
Geyser	0.9878	0.9608
Induction	0.9492	0.6942
Oven	0.9912	0.0796
Washing_machine	0.9118	0.6980
Overall	**0.7939**	**0.7113**

Table 3.1: Classification performance per appliance type.

3.2.4 | Brief interpretation

- High accuracy for many appliances (AC, Kettle, Geyser, Fridge, Induction) indicates the classifier predicts the dominant class well, but accuracy alone can be misleading for imbalanced labels. F1 score serves as a better metric here.
- Very low F1 for Oven (0.0796) suggests slight class imbalance (very few ON examples) or ambiguous event signatures.
- This shows that high power appliances like AC, Fridge or washing machine are much easier to segregate as compared to low-powered and irregular-usage-pattern appliances whose patterns can easily hide deep within the noise of other appliances.

3.3 | Regression Model

The regression stage estimates the power contribution (in watts) of each appliance type from the aggregate features. A multi-output random forest regressor is trained to predict the continuous power for each appliance type.

3.3.1 | Model and training setup

- **Input:** Feature matrix produced in Step 4 (temporal, statistical, event-based and electrical-signature features).
- **Targets:** Per-appliance-type total active power (continuous).
- **Split strategy:** Household-wise train/test split (no household appears in both sets) to avoid data leakage and evaluate generalization to unseen homes.
- **Model:** `MultiOutputRegressor(RandomForestRegressor)` with:
  - 100 trees,
  - maximum depth = 15,
  - minimum samples split = 10,
  - minimum samples leaf = 5,

- parallel training.

- **Evaluation metrics:**
  - Root Mean Squared Error (RMSE) — in watts,
  - Mean Absolute Error (MAE) — in watts,
  - Coefficient of determination R^2.

3.3.2 | Training summary

- Training households: 63
- Testing households: 27
- Training samples: 1,088,640
- Testing samples: 466,560

3.3.3 | Overall performance

Overall RMSE = 160.56 W, **Overall MAE** = 42.22 W, $\overline{R^2} = 0.6624$.

Appliance Type	RMSE (W)	MAE (W)	R^2
AC	374.29	130.30	0.8547
Iron	116.37	26.80	0.5795
Kettle	32.93	1.95	0.8175
Fridge	72.37	23.19	0.9052
Geyser	374.09	66.04	0.9232
Induction	181.38	55.85	0.6103
Oven	84.77	12.72	0.0911
Washing_machine	48.30	20.89	0.5172
Overall	**160.56**	**42.22**	**0.6624**

Table 3.2: Regression performance per appliance type.

3.3.4 | Interpretation of results

- **High R^2 (Geyser, Kettle, AC, Fridge):** These appliances have strong, distinguishable signatures and the model explains much of their variance. Except oven, all other appliances have very good R^2, indicating that the regression model is successful for these appliances.
- **Low R^2 (Oven):** Low R^2 values indicate the model is not performing well over these appliances. Causes may include:
  - Highly irregular or rare operating events (class imbalance in ON durations),
  - Strong overlap of signature with other appliances in the aggregate signal.
- **RMSE vs MAE:** Appliances with large maximum power (AC, Geyser) naturally exhibit larger RMSE despite acceptable relative performance (high R^2).

4 | Conclusion

The overall project gave us a rich experience of working with real-world data. The data generation process created a synthetic dataset that feels close to real life. Each household type was given its own set of appliances and daily routines. This made the final dataset rich, varied, and natural. Appliances were turned on and off in realistic ways, using time-of-day patterns and state transitions. Because of this, the aggregate load traces captured both regular habits and sudden events. Overall, the system produced realistic training data without needing real-world measurements.

The models performed well for appliances that have strong and clear power signatures. ACs, kettles, geysers, and fridges were easy for the system to identify. Their high accuracy and F1-scores show that the features captured their behaviour well. Most appliances were captured easily by the models, especially the larger appliances like AC and fridge who also exhibit strong seasonal patterns. However, some appliances were harder to detect. Ovens had fewer events and weaker patterns, so the model struggled. Since ovens performed bad for both classification and regression task, we may need to update original NILM dataset to better capture oven data. Even with these difficulties, the overall performance was promising. The results show that the approach works and can be implemented on meter data for load desegregation in real time.