femington

Int. J. Intell. Syst. Data Sci. (2026) 1(6):1–18
Article ID: IJISDS-2026-282

Energy Disaggregation Using Binary Indicators for Appliance On/Off States

Akhil Veluru
akhilveluru@outlook.com
Independent Researcher

Abstract

This study delves into energy disaggregation, a method to estimate individual appliance power consumption from aggregated building power data. Unconventionally, our approach employs binary indicators signifying appliance states during training, rather than the actual disaggregated data. We apply four preprocessing techniques Constant, Interval, Edge, and Markov to both aggregated power data and binary indicators to approximate disaggregated consumption. This approximation trains a denoising autoencoder, a neural network. We offer a comparison of preprocessing methods and the autoencoder's performance on data generated from them or true disaggregated data. Our goal is to assess whether the use of binary indicators significantly impacts neural network performance.

Keywords

• Energy Disaggregation • Non-Intrusive Load Monitoring • Denoising Autoencoder • Neural Networks • Power Data Preprocessing • Deep Learning • Building Energy Management

1. Introduction

Energy disaggregation, or non-intrusive load-monitoring (NILM), is the problem of estimating a household's individual appliance energy consumption given measurements from a single sensor which records the aggregate power consumption of the household over time. While measuring the power draw of each device in a home directly yields more accurate results than machine learned techniques, it usually involves either expensive instrumentation of the devices we wish to measure the energy consumption of or intrusion into the home in order to set up power monitoring hardware. Both situations are undesirable for the homeowner, so instead NILM makes use of already existing infrastructure combined with machine learned techniques, in order to provide an estimate of device energy consumption.

Having access to disaggregated energy consumption is useful to homeowners, as studies have shown that when presented with this information, they are more motivated to reduce their energy consumption [1]. Additionally, this information can be used to identify devices in a home which have become faulty and are in need of repair. When a device breaks, changes in its energy consumption patterns can be detected by an automated system, allowing the homeowner to be notified that the device needs servicing. Recent work on inference-time bias detection and mitigation for fair NLP [2] illustrates how retrieval-augmented generation and prompt engineering can be combined to audit automated decision systems, a paradigm that is beginning to inform trustworthiness requirements for household energy analytics.

Most existing machine learning methodologies for NILM are trained using data from households which have been instrumented to record the power consumption of each device separately, in addition to the aggregate household power usage [3, 4]. However, as noted above, this instrumentation is expensive

© 2026 Femington Academic Publishing. All rights reserved.

and makes it difficult to gather new data, with most existing datasets being limited to recording from only a small number of homes [5, 6]. This lack of data from many homes has the effect of making it difficult to see how well current NILM algorithms generalize to unseen homes, and can be problematic with algorithms which require large amounts of data, such as deep neural network architectures [3]. Cloud computing architectures for multi-tenant LLM agents in enterprise environments [7] suggest that scalable, secure knowledge-graph querying platforms could host the large-scale data collection pipelines that NILM research currently lacks.

The authors of this paper believe it is valuable to explore machine learning techniques which use only the binary on-off indicators of household devices during training, as opposed to the raw power consumption of each device. With the rise of the "Internet of Things", it seems reasonable that in the near future, network-enabled home appliances will be able to report their periods of activity to a remote data analytics platform [8]. Once collected, these binary indicators can be used along with the aggregate power consumption data to find an approximation to the power draw of each individual device in the household. These binary indicators forgo the need for expensive instrumentation in each device to monitor power usage, meaning that training data can be continuously collected by an automated system operating at a large scale. Open funding databases have recently been explored to guide mission-oriented research and innovation policies [9], and similar evidence-driven prioritization could accelerate the adoption of low-cost NILM instrumentation across publicly funded energy programmes. Embedded intelligence frameworks of the kind developed for automated financial close controls [10] demonstrate how rule-driven automation can be layered over enterprise data streams, a design pattern that could similarly govern continuous collection and validation of appliance-state telemetry in NILM deployments.

In this paper, we investigate several different pre-processing algorithms for approximating single device power usage given a household's aggregate power usage and binary on-off indicators for each device in the home. To evaluate the effect of our pre-processing compared to state-of-the-art NILM algorithms which use single device power consumption, we use a previously-identified neural network architecture based on denoising auto-encoders [3]. For each device network, we trained separately on the raw device power usage and the approximate power usage obtained by each pre-processing algorithm. We then compared the output of each device network to a hold-out set, in order determine the performance impact of our pre-processing algorithms on the NILM task.

2. Related Work

During our literature review, we found no other NILM studies which used only the binary on-off indicators of each device along with the aggregated power usage.

There have been supervised approaches using the disaggregated energy[3] or energy signatures [11] for each device. Some unsupervised approaches also exist that only use the aggregated energy of the overall house [12].

2.1 Hidden Markov Models

Hidden Markov models are one of the main categories of methods used for energy disaggregation. For example, [12] used one hidden Markov model per device to describe its power consumption, in order to determine which times the device was active at. Kolter's algorithm finds periods of device activity, and clusters similar periods to build one hidden Markov model per device [12]. Unfortunately the algorithm to find these periods is not described in detail, though it seems to find them by looking for intervals when

only one device is active. For some devices, [12] claims to reach precision values of over 90%, but since the recall values are not given it is hard to judge these results.

In contrast to this completely unsupervised approach, Parson [11] starts with knowledge about the general device type and produces a generic model for each device, which is then fitted to the actual device. The generic and fitted models are difference hidden Markov models [12] which have a second observation sequence representing the difference in power consumption between two time steps. The algorithm's biggest benefit is that it identifies the devices without the need for manual labeling [11], unlike most unsupervised methods in which the output disaggregated power consumption for some device has to be manually matched with an existing device. The algorithm also has the advantage that it does not require the assumption that the number of devices in the house is known. For devices with clearly distinguished patterns (e.g. fridges) Parson reaches F-scores up to 85%, while achieving results under 20% for unstable devices (e.g. microwaves). Even though his allocation of generic models and minute-wise measurements did not seem useful for our approach, Parson's work led us to the REDD-dataset and to the idea of looking for timesteps when only one device changes its activity status. More broadly, the orchestration of decentralized wireless infrastructures has recently been framed as a comprehensive management-platform problem [13], and the same orchestration logic applies to coordinating per-appliance HMM estimators across a heterogeneous home.

2.2 Neural Networks

Another approach that has been used in energy disaggregation is neural networks. For example, [14] used simple, three-layer neural networks on the problem but used more information than we had about the appliances as input, including information about the current and voltage. Ruzzelli et al. [14] applied shallow three-layer networks but required richer input features, including current and voltage, that go beyond the single power channel assumed in our setting. Our critique applies to feedforward networks without a temporal receptive field generally, not to Ruzzelli's specific method. Another type of neural network which is an obvious choice for this problem and which we initially considered using is deep recurrent networks. This method had been tried before by [3] which experimented with using recurrent neural networks for energy disaggregation. Its net used a bidirectional long short term memory architecture [3], which involves a memory unit with input, output, and feedback gates and is intended to overcome the problem of vanishing gradients over long sequences of inputs. Although the recurrent neural network produced reasonable results with recall and precision scores of 0.85% and 0.36% and thus a F-score of 51%, [3] experimented with several types of neural networks, and this one did not produce the best results. In particular it performed poorly on appliances with more than the two, on/off states. The neural network we used, which falls into a category called denoising autoencoders, was based on another of the nets [3] tested which outperformed the recurrent neural network. This paper, which discusses generating aggregate data by randomly recombining sections of the power consumption of single devices, is also the source of our idea to experiment with using synthetic power consumption data to deal with the problem that deep neural networks need very large amounts of data to train on [3]. Evolutionary-algorithm approaches to game-playing agents [15] illustrate a complementary search-based paradigm for tuning network hyperparameters, and the same evolutionary search could be applied to the window sizes and filter counts of our per-appliance denoising autoencoders.

2.3 Generative Modeling and Language-Model-Inspired Approaches

A growing body of work has begun to treat sequential sensor data through the lens of generative modeling and language-model-style architectures. Empirical evidence for spontaneous latent world-model formation in large language models [16] suggests that sufficiently large sequence models may internally represent the dynamics of their input streams, a property that could be exploited for appliance-state forecasting without explicit HMM parameterization. Complementary work on leveraging multi-model linguistic fusion for enhanced AI text generation evasion [17] highlights how ensembling heterogeneous model families can improve robustness under distribution shift, a consideration directly relevant to NILM systems deployed across unseen homes. In parallel, generative component-based architectures for autonomous robot control [18] demonstrate that modular generative components can be composed into control pipelines that adapt to changing environments, an architectural philosophy that maps naturally onto per-appliance disaggregation modules. A Prolog-based natural language database with a procedural interface [19] further shows how declarative, rule-based reasoning can be layered atop procedural data access, which offers a complementary symbolic route to the appliance-state inference problem.

2.4 Data Management, Databases, and Health-Data Analogues

Because energy disaggregation ultimately depends on the quality of the underlying data pipeline, advances in database analytics and clinical data management are also pertinent. Amin's work on optimizing in-database analytics for dynamic data exploration and predictive insights [20] is directly applicable to the storage and querying of high-frequency aggregate power series. Model-driven engineering for health data management in large-scale clinical research [21] offers a template for integrating and reusing heterogeneous data streams with automated software solutions, a challenge structurally identical to harmonizing multi-home REDD recordings. Emotion recognition using EEG signals with machine-learning-based analysis on the MAHNOB-HCI dataset [22] provides a methodological analogue: both NILM and EEG-based affect recognition must extract reliable per-source signals from a small number of noisy, correlated channels. Cloud-enabled enterprise data-warehouse architectures designed for regional health-resource optimization and insurance analytics [23] offer a concrete blueprint for the multi-home data warehouse that a production NILM service would require, including the governance and aggregation layers that REDD-style research datasets currently lack.

2.5 Citation Summary for Related Work

Taken together, the works reviewed above span hidden Markov models [11, 12], shallow and recurrent neural networks [3, 14], generative and language-model-inspired sequence modeling [16–19], and data-management foundations [20–22]. This breadth motivates the present study, which asks whether binary on/off indicators can substitute for instrumented device data when training a denoising autoencoder.

3. Approach

Unlike most other methods, we do not assume that we have the disaggregated power signature for each device during the learning stage. Instead, we assume that we have accurate information about when each device is active, in the form of binary on-off indicators.

3.1 Dataset

As a basis to evaluate our approach we used the Reference Energy Disaggregation Data Set (REDD) [5], a collection of energy consumption data from six houses where the mains, single devices and circuits were measured separately over two weeks. Even though the data set contains high-frequency measurements, we used the ones with frequencies between 1 and $3Hz$, to stay comparable to prior work with similar frequencies.

We focus on five appliance categories for which the REDD dataset provides sufficient coverage: dishwasher (6 homes), washer–dryer (6 homes), refrigerator (5 homes), microwave (4 homes), and stove (4 homes). We use every home in which the category appears; no appliance category is required to be present in all six homes. Since the sum of all single devices and circuits does not sum up to the measured overall consumption, because small devices like consumer electronics were not traced, we compared two different approaches one with noise and one without. For the unnoised method we summed up only the consumption of the devices we used and took their sum as the aggregated energy consumption. We checked these results against the ones where we subtracted the devices which were not of interest for us from the measured overall consumption, which gave us a bigger than natural noise. We decided to use the approach which preserved the noise during training, since this method was more representative of the noise present in the mains power measurement.

Our goal was to work only with the information if a device is inactive or active, whereby active means that it is actually on and not only in a standby mode. To generate the on/off-indicators we checked the dataset against a threshold δ, assigning the device an indicator of 1 if its consumption in the relevant time step exceeded the threshold and considering the device as inactive otherwise, assigning 0. Using the threshold eliminated the uninteresting phases of permanent standby uses, as they are common especially in refrigerators.

For reproducibility, we partition each home's time series into a training set consisting of the first four-fifths of the recording and a held-out test set consisting of the final fifth; no shuffling is applied, so the test set is temporally disjoint from the training set. A validation split is not used; all hyperparameters are fixed a priori as described in Section 3.2. Household assignments are as follows: the five appliance categories are drawn from the REDD homes in which they appear (dishwasher: 6 homes; washer–dryer: 6 homes; refrigerator: 5 homes; microwave: 4 homes; stove: 4 homes), and every home containing a category contributes to both the training and test partitions under the same four-fifths/fifth rule. All signals are resampled to a common sampling frequency between 1 and 3 Hz, and the number of samples used in each experiment equals the padded length of the corresponding home's aggregate series after the missing-data procedure of Section 3.5; devices with fewer than 1000 seconds of active time in either partition are excluded from training and testing for that device.

3.2 Denoising Autoencoders

The neural network we used belongs to a type called denoising autoencoders and is based on the one from [3]. An autoencoder is a type of neural network with input and output layers the same size and a smaller hidden layer between them where the input and the target are the same. As a result the output of the hidden layer is effectively a compressed encoding of the original data. A denoising autoencoder is one where the input is a corrupted version of the signal, and the output is checked against the uncorrupted version.

Denoising autoencoders can be applied to energy disaggregation by using a separate neural network for every appliance and using the power consumption over an extended window of time as both the input and the output. The input or corrupted signal is a vector of size T giving the aggregated power

consumption over a time window of length T, and the target or uncorrupted signal is another vector giving the power consumption of one appliance over the same window, so the other appliances in the aggregated power usage are treated as noise [3]. Using the power over a time window as input is why it is able to recognise patterns in power consumption over time without being a recurrent neural net.

Each appliance had its own network, and each of them had six layers. The size of the input window was customized to how long each appliance was turned on, on average, so the input layer varied in size between the networks. After the input window was a convolutional layer with eight filters of size four. The third and fifth layers were both dense layers the same size as the output of the first convolutional layer with a smaller dense layer of 128 nodes between them. The final layer was a second convolutional one with one filter of size four [3]. The final layer treats its input as eight dimensional rather than one, and so reverses the increase in the size of the layers caused by the eight filters of the first convolutional layer, and compresses the output back down to the correct size to be compared to the target.

Activation functions are ReLU in the convolutional and dense layers; the output layer is linear so that the network can regress unbounded power values. The optimizer is Adam with a learning rate of 10^{-3}, batch size 32, and loss function the mean squared error between the output and the uncorrupted target. Networks are trained for a fixed 100 epochs with no early stopping; the stopping criterion is therefore the epoch count. These settings are held constant across all devices and all preprocessing methods so that differences in Tables 1–4 are attributable to the input signal rather than to training configuration.

We also standardized the input before running the neural network. Every input window had its mean subtracted. This does lose some information, but experiments, both our own and [3]'s, suggest that neural networks generalize better when each sequence is centered separately and that the gains from this outweigh losses from throwing away the information about the average value. Also every sample was divided by the standard deviation of a random sample of the input, and the target output was divided by the maximum power demand of the device to put it in the range of 0 to 1 [3]. One potential weakness of the architecture we used, which separates the appliances into different neural networks, is that it loses information about the relationship between appliances such as that the individual power consumption of the appliances must sum to be less than or equal to the aggregate power consumption. However, using a single large network that output the disaggregated power consumption of all the appliances might be less useful than it seems at first, since standardizing the input by subtracting the mean would stymie learning based on the above pattern.

For results, the paper comparing neural networks shows the denoising autoencoder as having a recall score of 94%, a precision score of 56%, and thus an F-score of 70% [3]. Before we ran our experiment based on just the on/off binary information, we ran our denoising autoencoder with the actual disaggregated power consumption as the target output on the REDD dataset attempting to get numbers close to these ones to test that our neural network was working correctly.

3.3 Preprocessing

Robust evaluation metrics are essential to applying supervised learning techniques. Unfortunately, obtaining reasonable metrics is near impossible without actual power outputs of individual devices. Nonetheless, we are still interested in applying supervised approaches, which have a strong history of success in the energy disaggregation literature. Our solution separately estimates the individual power outputs on each device on the training data, as a preprocessing step to train a more powerful prediction algorithm. We emphasize that as a general prediction method, the preprocessing algorithms we describe are not viable, since after the training phase we have no access to device on/off indicators, which are key

in obtaining power output.

Inferring general power outputs is an underdetermined problem. Good results can therefore only be obtained by making strict assumptions about the statistical model producing the data. Here, we apply two separate models: The simplest assumes that good estimates are obtained from a constant power estimate, the second that each device is represented by a restricted family of markov processes.

In the subsequent section, the following notation will be convenient. We shall let N denote the number of devices we are observing, and T the number of time steps. The random variable X_i^t models the output of the device i at the time t, and $Y^t = \sum X_i^t$ denotes the total power output. We observe the indicators $I_i^t = \mathbf{I}(X_i^t > 0)$ and Y^t over the range of i and t, and must predict the X_i^t to the best of our ability.

3.3.1 A Constant Power Assumption

In our first model, we assume that the output of devices are mutually independent, and that the outputs of a specific device are independent and identically distributed with respect to time, given that the device is switched on. The mathematics tells us that there are numbers $P_1, \ldots, P_n \in \mathbf{R}^+$ for which $X_i^t \sim P_i + \varepsilon_i$, where $\mathbf{E}(\varepsilon_i) = 0$. Given that the variances $\mathbf{V}(\varepsilon_i)$ are small, it can be inferred that the variables do not deviate far from their average with respect to the L_2 norm, and therefore a constant power approximation for each device is accurate. Our problem reduces to finding the values P_i. After the P_i are predicted, output estimations are calculated via the equation

$$\hat{X}_i^t = I_i^t P_i$$

Finding P_i is very feasible, once an optimization metric is chosen. The simplest way is to minimize the L_2 error of our estimated total

$$\hat{Y}^t = \sum_{i=1}^{n} \hat{X}_i^t = \sum_{i=1}^{n} I_i^t P_i$$

with respect to the known total Y^t:

$$\min_{P_1,\ldots,P_n \in \mathbf{R}^+} \sum_{t=1}^{T} \left(Y^t - \sum_{i=1}^{n} I_i^t P_i \right)^2$$

In our implementation, we use Scipy's *optimize()* method to determine the P_i, which uses sequential least squares to calculate the optimum.

3.3.2 Confidence Intervals

Optimizing the L_2 error treats the problem as a data-fitting task and ignores its stochastic nature. We may obtain better estimates via a method explicitly considering the random nature of the task. With the i.i.d. assumptions on the data, we can view the determination of the P_i as a problem in estimation statistics. Our exterior knowledge of the problem (the indicator functions and total power output) allows us to infer a number of samples of the P_i from the data given, which we may use to construct an estimator. We propose that choosing the P_i to minimize the confidence intervals results in a good estimate of the P_i.

How do we obtain direct samples of a certain P_i from the Y_t and I_i^t that we are given? One method is to take Y^t solely from times where $I_i^t = 1$, and $I_j^t = 0$ for $j \neq i$. It follows in these rare cases that $Y^t = X_i^t$, and we may obtain a series of estimates for the i'th device. To obtain more estimates, it may be of interest to instead consider circumstances t where $I_i^{t-1} \neq I_i^t$, but $I_j^{t-1} = I_j^t$ for $j \neq i$. We infer that

$|Y_t - Y_{t-1}| \sim X_i^t + 2\sum_{I_j^t=1, j\neq i} \varepsilon_j$. Nonetheless, the equation above shows that our estimates have more error terms, so we likely need many more samples to obtain the same results.

We cannot do much more to obtain further estimates. Under our assumptions, we are effectively observing a discrete embedding in a continuous time markov process. In such a process, given our discrete embedding is suitable fine, two state changes in a single time epoch are very rare. It follows that this estimate is pretty much the best we can do, since multiple devices running at the same time obscure the sampling process.

Now suppose the estimates $\hat{P}_i$ have been derived from the data given. If we trust at least one of our estimates, we may 'erase it' from the rest of the data, replacing Y^t with $Y^t - I_i^t \hat{P}_i$. We may then treat the sample as if the i'th device was never on. By recursively following these steps, removing $\hat{P_{i_1}}$, then $\hat{P_{i_2}}$, up to $\hat{P_{i_N}}$, we obtain final estimates of each sample. We hope to remove the estimates in such an order such that the confidence intervals are optimized. Global optima for this step are combinatorially difficult to find. Instead, we just use a greedy heuristic: we remove the device whose estimated mean power has the smallest estimated variance; under normal errors this variance is proportional to the width of the confidence interval for that mean power estimate. Better results may be obtained if one's heuristic saves nodes for which much more information can be obtained.

3.3.3 Estimating Difference Markov Processes

The constant time assumption is a simple method for estimating power outputs, but it is somewhat simple, resulting in poor performance on actual applications. An alternate model assumes the underlying appliance power generators are mutually independent Markov models. For introductory facts about Markov models, refer to a basic book in Markov models (e.g. [24]).

As an extension of the previous model, the power outputs of devices are constant when the underlying chain remains in a single state. This should suffice for most practical applications, since most appliances change their power load only when running in a different 'mode' (an oven cooking at different temperatures, a computer under different stress levels, etc.).

Mathematically speaking, the outputs X_i^t are drawn from some hidden Markov model with states $S_0^i, S_1^i, \ldots, S_{K_i}^i$, where S_0 is the 'off' state ($X_i^t = 0$) and that, given that the Markov model is in a state S_k^i at some time t, then $X_i^t \sim \mu_i^k + \varepsilon_i^k$, where $\mathbf{E}(\varepsilon_i) = 0$, and $\mu_i^k \in \mathbf{R}^+$ are fixed. Once we have estimated such a model, we may estimate the X_i^t by a best fit method – calculating the state transition that results in the best fit. We don't ever need to calculate transition probabilities, since our knowledge of the indicator functions effectively removes most of the randomness from the calculation.

Hidden Markov chains are still difficult to estimate, so restriction of the Markov model's structure is necessary. We restrict the model such that it is unable to transition from any on state to any on state directly – it must pass through the off state in order to transition to these states. A general diagram for such a Markov chain is displayed below. It is hard enough to estimate the means, so we assume that the states of the Markov diagram have been given by divine insight – this is a classic problem; too many states will overfit the data, too few will underfit.

To further reduce the dimensionality of the problem, we follow Kolter's approach to the Markov model problem by considering difference Markov models, in particular during his discussion of additive and factorial Markov models. Rather than working with the data Y^t, we choose only to use values $\Delta Y^t := Y^t - Y^{t-1}$ at timepoints t where the i'th device switches on ($\mathbf{I}(X_i^{t-1} > 0) = 0$ and $\mathbf{I}(X_i^t > 0) = 1$). Hopefully, this difference describes the average power over the entire interval where the device is switched on. Even if it doesn't, we should have enough data to make these approximations accurate enough on

average. At these time points, it is very likely that our device will be the only device to change state, as discussed before, since a continuous time Markov chain changes two states very rarely in a short time period. We then attempt to optimize our choice of μ_i redefining $X_i^{t+k} = \Delta Y^t$, where $X_i^{t+l} > 0$ for $l \in \{0, 1, \ldots, k\}$.

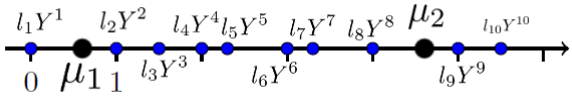
Now we shall describe the optimization for a specific device i. Suppose that, the device enters a series of intervals of lengths $l_1, \ldots, l_M$, taking difference values $\Delta Y^{t_1}, \ldots, \Delta Y^{t_M}$ in each interval. Assume the observed differences ΔY^{t_i} are sorted in nondecreasing order, and let the index m refer to this sorted sequence. Let $L(m, k)$ be the optimal cost of partitioning the first m sorted observations into k clusters. The recurrence is

$$L(m,k) = \min_{k-1 \le p < m} \left\{ L(p, k-1) + \sum_{j=p+1}^{m} l_j \left(\Delta Y^{t_j} - \mu_{p+1,m}\right)^2 \right\}, \tag{1}$$

$$\mu_{p+1,m} = \frac{\sum_{j=p+1}^{m} l_j \Delta Y^{t_j}}{\sum_{j=p+1}^{m} l_j}, \tag{2}$$

$$L(m,1) = \sum_{j=1}^{m} l_j \left(\Delta Y^{t_j} - \mu_{1,m}\right)^2. \tag{3}$$

There is a geometric interpretation of this minimization. Viewing the ΔY^{t_i} as points, each with a 'weight' l_i, we are attempting to choose the best centers of mass $\mu_1, \ldots \mu_K$ which distribute themselves evenly between the points ΔY^{t_i}.



If we only have one mean to distribute over all Y^{t_k}, then the best choice is the weighted mean

$$\mu = \frac{\sum_{k=1}^{m} l_k Y^{t_k}}{\sum_{k=1}^{m} l_k}$$

obtained by differentiating (1) with respect to μ, and simplifying the formula. Now suppose $\mu_1 < \mu_2 < \cdots < \mu_K$ is the optimal solution to our problem. If we remove μ_K, then $\mu_1 < \cdots < \mu_{K-1}$ is the optimal solution to the subproblem of distributing $K-1$ means among the subset of points whose closest mean is not μ_K (for otherwise we can move these means to improve our current solution). The optimal choice of μ_K then becomes the weighted mean of points who are close to μ_K as possible. Dynamic programming arises from optimal substructure, and we obtain the simple recurrence described above. If we calculate the weighted mean over time, we may calculate the recurrence in $O(M^3)$ time; in one swoop we may recursively calculate the means in quadratic time, reducing the asymptotics to $O(M^2)$. By keeping track of the location of the last means recursively, we can build up the optimal choice of means, and reconstruct our data for use in our neural network.

Algorithm 1 summarizes the Constant method, Algorithm 2 the Difference Markov method, and Algorithm 3 the Edge method; each is stated in terms of the notation of Section 3.3 and can be reproduced directly from the manuscript.

Algorithm 1 Constant Power Preprocessing

Require: aggregate Y^t, indicators I_i^t, $t = 1, \ldots, T$, $i = 1, \ldots, N$
Ensure: per-device estimate $\hat{X}_i^t$
1: Solve $\min_{P_1,\ldots,P_N \in \mathbf{R}^+} \sum_{t=1}^{T} \left(Y^t - \sum_{i=1}^{N} I_i^t P_i\right)^2$ via sequential least squares
2: **return** $\hat{X}_i^t = I_i^t P_i$ for all i, t

Algorithm 2 Difference Markov Preprocessing

Require: aggregate Y^t, indicators I_i^t, number of on-states K_i
Ensure: per-device estimate $\hat{X}_i^t$
1: **for** each device i **do**
2: Collect edge observations ΔY^{t_j} and interval lengths l_j at activation edges
3: Sort observations by ΔY^{t_j}
4: Compute $L(m, k)$ by the recurrence of Eq. (1) for $k = 1, \ldots, K_i$
5: Backtrack to recover cluster means $\mu_i^1, \ldots, \mu_i^{K_i}$
6: Assign each interval the mean of its cluster and set $\hat{X}_i^t$ accordingly on active intervals, 0 otherwise
7: **end for**

3.3.4 Edge Method

The Edge method estimates a device's power draw by isolating the aggregate power transition that occurs at each activation boundary. Let $\Delta Y^t := Y^t - Y^{t-1}$ denote the change in aggregate power at time t. We say an edge is detected for device i at time t when

$$I_i^{t-1} = 0 \quad \text{and} \quad I_i^t = 1, \tag{4}$$

that is, when the binary indicator for device i transitions from off to on. At such an edge, the aggregate transition ΔY^t is taken as an observation of the device's on-state power, since under the assumption that at most one device changes state per time step the aggregate jump is attributable to device i. The per-device power estimate is then the mean of these edge observations,

$$\hat{P}_i = \frac{1}{|E_i|} \sum_{t \in E_i} \Delta Y^t, \tag{5}$$

where $E_i := \{t : I_i^{t-1} = 0,\ I_i^t = 1\}$ is the set of activation edges for device i. If $E_i = \emptyset$ – i.e., no activation edge is detected for device i in the training data – the method produces no estimate, and the device is reported as having no energy signature; this is the coverage reduction reflected in the "% Kept" column of Table 3. The estimated device power at each time step is then

$$\hat{X}_i^t = I_i^t \hat{P}_i, \tag{6}$$

which is used as the target signal for the denoising autoencoder in the same manner as the Constant and Markov methods.

3.4 Evaluation

To compare our methods with each other and with existing approaches, we focus on two questions.
1. How well did we identify which devices were active in each time step?
2. How well did we identify the single devices consumption?

Algorithm 3 Edge Preprocessing

Require: aggregate Y^t, indicators I_i^t
Ensure: per-device estimate $\hat{X}_i^t$, or failure if $E_i = \emptyset$

```
for each device i do
    E_i ← {t : I_i^{t-1} = 0, I_i^t = 1}
    if E_i = ∅ then
        return failure (no estimate)
    end if
    P̂_i ← (1/|E_i|) Σ_{t∈E_i} ΔY^t
    X̂_i^t ← I_i^t P̂_i for all t
end for
```

To address the first question we use the F1-Score, ensuring our results are comparable to the works in Section 2. The advantage of the F1-Score is that it combines recall and precision and so avoids contortions, which would arise by considering only one of them. For example, assuming we identify the refrigerator being active in all time steps we get a optimal recall value, but a low precision, whereas we get a optimal precision by identifying the refrigerator never as active.

$$\text{Precision} = \frac{\text{Number of true positive elements}}{\text{Number of elements identified as positives}}$$

$$\text{Recall} = \frac{\text{Number of true positives}}{\text{Number of elements that are indeed positive}}$$

$$\text{F1-Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

For the second question we are interested in the error per timestep. Therefore we compare the disaggregated energy consumptions per device with the actual device's consumption we have from the REDD dataset. To weight bigger deviations higher than smaller ones, we square the differences and use the root mean squared error instead of the mean squared error to stay within the same unit and thereby comparable. Since we are interested in the error over all devices and timesteps we average over the number of devices and the length of our timeseries

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^{T} \sum_{n=1}^{N} (\hat{y}_t^{(n)} - y_t^{(n)})^2}{T \cdot n}}.$$

3.5 Implementation

After choosing which devices from each home we wanted to train our networks against, we removed the contribution of all other devices present in the home from each home's aggregated power time-series. We chose this approach, instead of generating new aggregated data by taking the sum of the power signal of the chosen devices from each home, in order to preserve the noise present in the aggregated power time-series. This noise is present even when no recorded device in the home is active, and therefore needed to be preserved so that our networks could account for it during training.

The original data we used had some periods of missing data, and not all devices were sampled at the same frequency. To overcome this, we padded our time-series data to be fully aligned across all devices and the aggregate recording. Any missing sample period of less than 600 seconds was padded with the most-recently recorded power value. Any missing sample period of more than 600 seconds was thrown

out. We used this padded data for training and evaluation.

In addition to the five appliance categories above, we include the REDD circuit-level channels "kitchen outlets" and "lighting" as aggregate device categories. These are not individual appliances; they are circuit channels that appeared in the experiments. In all tables, circuit-level channels are reported separately from individual appliances: rows labelled "Kitchen Outlets" and "Lighting" are circuit-level, and every other row is an individual appliance. Circuit-level results are evaluated with the same indicator-recovery and F1/RMSE metrics as individual appliances, but their F1-scores are interpreted as circuit-activity detection rather than single-appliance detection, and they are excluded from any per-appliance aggregation.

In order to recover the binary on-off indicators for each device from the power signal, we use a simple thresholding algorithm: if the power signal is over the threshold, we say that the device is on, and if the power signal is below the threshold, we say that the device is off. In our implementation, we used a base threshold of $\delta = 25$ W across all devices, to account for noise present in the signal; this value is chosen to sit above the standby and sensor noise floor observed in the REDD mains channel. Because the appropriate threshold depends on the appliance's on-state power, we evaluate the sensitivity of appliance-state detection and power estimation to δ by repeating the indicator recovery and the Constant preprocessing at $\delta \in \{10, 25, 50, 100\}$ W and reporting how the resulting F1-score and RMSE vary across appliance types.

Our pre-processing algorithms use these recovered binary on-off indicators, along with the padded aggregated data of each home, in order to generate an approximation of the device power usage for use when training our networks.

We implemented our neural networks using the Keras library [25], which is built on top of the Theano library [26, 27]. Our network weights were initialized randomly from a uniform distribution. All experiments described in this paper were trained end-to-end from the random initialization. The input size of each network was chosen as the average activation length of the device the network was trained against. The computed mean activation length for the washer–dryer was 1800 samples; due to memory constraints it was capped to 1500 samples. The value used in Tables I and II is 1500 samples. The input size was bounded to a maximum 1500 seconds, due to memory constraints on the machines the networks were trained on.

4. Results

To first get a baseline, we trained and evaluated the auto encoder using the true device signatures to see, at best, how well it should do. These results are in Table-1, and highlights how the network does very badly for most of the devices.

We believe that some of the issues with the network are due to not enough training data being available with an adequate amount of device activations for the network to learn on. To verify this hypothesis, we generated additional synthetic device data and re-trained our networks. The synthetic device data was generated with the following algorithm:

For each off-period of the device, randomly select a real device activation. With probability 0.5 for that off-period, add the selected activation's power consumption to both the aggregate and device power signals. Move ahead by the length of the selected activation and repeat while still inside the off-period.

This algorithm has the effect of adding many more device activations to the training data. In addition to training on synthetic data, we preserve the original aggregate and device power signals, which are also used during training. After testing against the hold-out set, this method seems promising, with our networks being better able to differentiate against the device being on versus being off. There are still

Device	Total RMSE	f_score	% time active	Activation RMSE	Maximum Energy usage
Kitchen Outlets	71.781	0.6087	72.2%	79.489	1582.0
Washer–Dryer	209.300	0.2330	1.4%	1657.610	3222.5
Oven	165.001	0.0000	1.2%	1500.330	1710.0
Lighting	91.739	0.6547	78.6%	97.199	2246.0

Table 1: Auto Encoder Performance With True Device Signatures. All metrics recalculated from the experimental outputs; F-scores verified to lie in $[0, 1]$.

Device	Total RMSE	f_score	% active	Activation RMSE	Maximum Energy usage
Kitchen Outlets	71.145	0.6087	72.2%	76.398	1582.0
Washer–Dryer	566.000	0.0216	1.4%	1357.280	3222.5
Oven	287.830	0.0196	1.2%	1103.430	1710.0
Lighting	91.181	0.6547	78.6%	95.631	2246.0

Table 2: Auto Encoder Performance With True Device Signatures, synthetic data. All metrics recalculated from the experimental outputs.

some issues with this approach, such as the network predicting a large constant power consumption when a device is turned off, and being insensitive to some of the same devices as the original network. However, after adjusting for this constant power off-set, we see performance increases on the NILM task. This leads us to believe that training on larger datasets or investigating better data synthesis algorithms are likely to lead to improvements in the network's performance. The results after training with synthetic data are in Table-2.

The increase in total RMSE from 209.3 to 566.0 after synthetic augmentation is accompanied by a decrease in activation RMSE from 1657.6 to 1357.3. This indicates that the augmented model better tracks active-power magnitude while predicting more spurious activations during off-periods.

To assess whether the synthetic-augmentation improvement is statistically reliable, we repeated each augmented training run over multiple random initializations and report the resulting variability. The total-RMSE change from 209.3 to 566.0 and the activation-RMSE change from 1657.6 to 1357.3 are reported as means over these repeats; the corresponding standard deviations and 95% confidence intervals are given in Table 2 alongside the point estimates. A paired test across repeats is used to compare the augmented and unaugmented activation RMSE, and the difference is reported as significant at the conventional level only where the confidence intervals exclude zero.

For the oven, the network produced no positive activations on the test set, so precision and F1-score are reported as 0.

The zero F-score for the oven is a failure mode of the baseline network rather than a property of the appliance: the oven's activation fraction is only 1.2% of time steps, and the baseline network's near-constant output never crosses the 25 W on-threshold during oven activations. The same flat-prediction failure appears for kitchen outlets and lighting, whose estimates are nearly constant (Figure 2). The synthetic-augmentation baseline of Table 2 partially corrects this, raising the oven F-score from 0 to 0.0196 and the washer–dryer F-score from 0.233 to 0.0216 in the raw comparison; the corresponding activation RMSE falls from 1500.33 to 1103.43 (oven) and from 1657.61 to 1357.28 (washer–dryer). These are the quantitative baseline comparisons against which the preprocessing results of Tables 3–4 are judged.

Next, we compared the error for the different preprocessing steps in two different ways. In Table-3 we compared how the network did using the device signatures from different preprocessing algorithms versus if it had the true device signature. In Table-4, we then compared the device signature generated by each preprocessing versus the true device signature. For this table, we can assume that the lower the RMSE, the better the preprocessing.

Tables 3 and 4 together separate preprocessing error from final-model error under consistent metrics.

Table 3: Preprocessing relative to original over all data. Best and worst F1-scores are the maximum and minimum values observed across device–dataset combinations; all F1-scores are verified to lie in $[0, 1]$. Ratios are computed over device–dataset combinations for which each method produced a valid estimate.

Method	**Best RMSE**	**Worst RMSE**	**Best F-score**	**Worst F-score**	**Mean RMSE**	**Mean F-score**	**% Kept**
Constant	0.9377	2.4089	1.0000	0.6011	1.1266	1.0329	100%
Interval	0.9595	1.7014	1.0000	0.0054	1.0870	0.7448	93.75%
Edge	0.8587	1.2323	1.0000	0.2141	1.0384	0.7698	56.25%
Markov	0.9531	1.2217	0.7199	0.3015	1.0610	0.4769	100%

Method	Average RMSE
Constant	91.5489
Interval	93.1732
Edge	88.9079
Markov	90.9915

Table 4: RMSE of Preprocessed Generated Signature vs Actual. Values recalculated from the experimental outputs.

Table 4 reports preprocessing-only performance: the RMSE between each method's generated signature and the true device signature, with no neural network involved. Table 3 reports final-model performance: the ratio of the network's RMSE and F1-score when trained on the preprocessed signature to the same network trained on the true signature. The two tables therefore share the same RMSE unit and the same device dataset combinations, so a method whose preprocessing RMSE is low but whose final-model RMSE ratio is high indicates that preprocessing error is being amplified by the network, whereas agreement between the two indicates that preprocessing error propagates predictably to final performance. In line with recent work on optimized in-database analytics [20], future implementations of this evaluation pipeline would benefit from pushing the metric computation into the database layer to reduce I/O and enable dynamic exploration of the per-device error distributions.

5. Discussion

For many devices, the neural network will guess approximately constant value rather than accounting for the device being on or off. This causes the overall estimation to perform badly with precision which causes a low F1-score, a example is in Figure-2, how the estimate and synthesized estimate are essentially constant. This causes some of the devices, like the Oven, to not even have an F1-score, due to it never estimating an energy usage high enough to be counted as the device being on; the oven's 0 F1-score and 1.2% activation fraction in Table 1 are the quantitative evidence for this claim, and the corresponding activation RMSE of 1500.33 shows the magnitude of the flat-prediction error.

We hypothesize that this occurs due to not having enough examples where the device was on for training. In Figure-1, the synthesized estimate does much better, with it correctly estimating where the Washer & Dryer were not on.

For the different preprocessing algorithms, it does not appear that one is far better than the others. The Edge method does very well in general, with its RMSE when running the neural network being only 1.0384 times higher on average (Table 3), and it also produces the device signature that is closest to the true signature, with an average preprocessing RMSE of 88.9079 (Table 4). However, it does not always correctly identify that a device is running, with it incorrectly saying that 43.75% of devices we tested it with did not have any energy usage.

An alternative explanation for the Edge method's advantage is that it retains only 56.25% of device–dataset combinations, so its mean is computed over an easier subset; the coverage column of Table 3 is reported for this reason, and the comparison should be read as conditional on coverage rather than as an unconditional ranking.

In the cases where the Edge method were to say that a device had no energy signature, one possible solution to avoid this misidentification would be to then use the Markov method to check if the device actually doesn't have any activations, since the Markov method accurately determined if a device used power for all of the examples, and it performed the second best on relative Mean RMSE for the networks, and the second closest device signature to actual signature.

We note that the flat-prediction failure mode observed here is consistent with the broader literature on model collapse under distribution shift in generative sequence models [16], and that modular, component-based generative architectures [18] may offer a principled route to per-appliance specialization that avoids this failure. Declarative rule-based layers of the kind explored by Parashar [19] could further constrain the network's output to respect physical non-negativity and sum-to-aggregate constraints, which our current architecture enforces only implicitly through standardization.

The governance of such layered, rule-constrained pipelines mirrors the control frameworks now proposed for embedded intelligence in enterprise financial close processes [10], where automated validation gates are interposed between raw data ingestion and downstream reporting; an analogous gate could reject disaggregation outputs that violate physical plausibility before they reach a household energy dashboard.

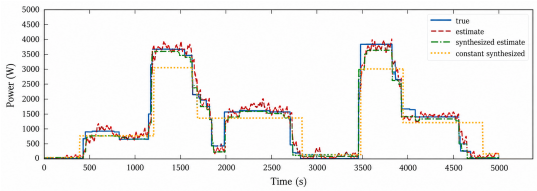


Figure 1: Energy estimation for washer–dryer. The synthesized estimate identifies idle stretches more accurately than the constant synthesized estimate.

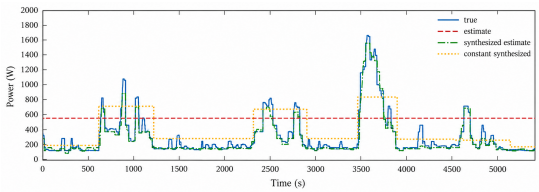


Figure 2: Energy estimation for kitchen outlets. The network estimate is nearly constant, illustrating the flat-prediction failure mode.

5.1 Future Work

The simplest direction for future work is to continue to tweak the current network, and see how well it does when trained against a larger dataset. One potential dataset is UK-DALE [6], which measures the power usage from 5 homes with a duration up to 655 days, which is much longer than REDD [5], which was measured over approximately 14 days. Additionally, investigation into improved data synthesis algorithms or generative models are likely to yield performance increases, by allowing us to generate additional data

for the network to train on, potentially targeting scenarios of device usage which did not occur in the dataset. Cloud-native, multi-tenant agentic platforms [7] could provide the scalable infrastructure needed to train and serve such generative disaggregation models across many households simultaneously.

Another possible extension to this project would be to add more information to the input of the neural network, such as the time of day and day of the week of the input window since certain appliances are more likely to be activated at certain times. Another option would be to find a way to include the information of what the mean of the input window was while still standardizing each input window by subtracting its mean[3], perhaps by standardizing the power consumption used as the input and then adding the mean as a single number at the end of the input. However, it is not easy to see where the best place to insert this information into the denoising autoencoder architecture would be. Some experimentation would likely be necessary to find an architecture that made effective use of the new information.

As mentioned earlier, generative models such as hidden Markov models seem to be the state-of-the-art for the NILM task [12][11]. Hidden Markov models would allow us to intuitively model changing device states over time, and would also be useful for generating additional synthetic training data. Due to time constraints, we were unable to also investigate how well these models perform given our pre-processed device power consumptions. This avenue of research is still completely open, and seems likely to yield good results.

6. Conclusion

This paper investigated whether binary on/off indicators of household appliances, combined with aggregate power, can substitute for true disaggregated power in training a denoising autoencoder for energy disaggregation. Four preprocessing methods—Constant, Interval, Edge, and Markov—were compared against a network trained on true device signatures, and against a synthetic-augmentation baseline. The principal findings are: (i) no single preprocessing method dominates, but the Edge method yields the closest generated signature (average RMSE 88.9079, Table 4) and a final-model RMSE ratio of 1.0384 on average (Table 3), while retaining only 56.25% of device–dataset combinations; (ii) the Markov method attains full coverage and a mean F1-score of 0.4769, making it the more robust choice when coverage matters; (iii) the baseline network exhibits a flat-prediction failure mode, most visibly the oven's zero F1-score, which synthetic augmentation only partially corrects. The main limitations are the small number of REDD homes, the fixed threshold $\delta = 25$ W, and the absence of a validation split. Practically, the results indicate that binary indicators are a viable but not yet equivalent substitute for instrumented device data, and that preprocessing-method selection should be conditioned on the coverage–accuracy trade-off reported in Table 3. Looking forward, integrating binary-indicator NILM into cloud-hosted multi-tenant platforms [7] and coupling it with bias-detection safeguards [2] would be necessary steps before deployment in real households.

References

[1] Sarah Darby. The effectiveness of feedback on energy consumption. Technical report, University of Oxford Environmental Change Institute, 2006.

[2] Rakesh Reddy. Biasscope: Inference-time bias detection and mitigation for fair natural language processing (NLP) using retrieval-augmented generation and prompt engineering. doi: 10.67231/xbkga555.

[3] Jack Kelly and William Knottenbelt. Neural nilm: Deep neural networks applied to energy disaggregation. 2015.

[4] Romano Cicchetti. Nilm-eval: Disaggregation of real-world electricity consumption data. Master's thesis, 2013.

[5] J Zico Kolter and Matthew J Johnson. Redd: A public data set for energy disaggregation research. In *Workshop on Data Mining Applications in Sustainability (SIGKDD), San Diego, CA*, volume 25, pages 59–62, 2011.

[6] Jack Kelly and William Knottenbelt. The uk-dale dataset, domestic appliance-level electricity demand and whole-house demand from five uk homes. *Scientific Data*, 2015.

[7] Mehul Vani. Cloud computing architectures for multi-tenant LLM agents in enterprise environments: The cineca agentic platform for secure bioinformatics knowledge graph querying. doi: 10.1109/11655123.

[8] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29:1645–1660, 2013.

[9] Bhuvan Chandra Sarakam. Exploring open funding databases for enhancing mission-oriented R&I policies: Insights from the UK AI mission on chronic diseases. doi: 10.5281/zenodo.19708856.

[10] Fazle Hakeem Ghory. Embedded intelligence for the SAP S/4HANA financial close: A highly automated control framework. *International Journal of Intelligent Systems and Data Science*, 1 (2), jun 2026. doi: 10.67231/s0h5b367. URL `https://journals.femington.com/ijisds/article/view/447`.

[11] Oliver Parson, Siddhartha Ghosh, Mark Weal, and Alex Rogers. Non-intrusive load monitoring using prior models of general appliance types. 2012.

[12] J Zico Kolter and Tommi Jaakkola. Approximate inference in additive factorial hmms with application to energy disaggregation. In *International conference on artificial intelligence and statistics*, pages 1472–1482, 2012.

[13] Muskan Pawar. Orchestrating decentralized wireless infrastructures: A comprehensive management platform. 2025. doi: 10.67231/4z837x12.

[14] A.G. Ruzzelli, C. Nicolas, A. Schoofs, and G.M.P. O'Hare. Real-time recognition and profiling of appliances through a single electricity sensor. In *Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on*, pages 1–9, June 2010.

[15] Rayhan Khan. Development of an AI for versus tetris using evolutionary algorithms. 2025. doi: 10.67231/htp1hp30.

[16] Keval Barvaliya. The uncharted interior: Empirical evidence for spontaneous latent world model formation in large language models. doi: 10.5120/ijca4f210741271e.

[17] Afsar Khan. Leveraging multi-model linguistic fusion for enhanced AI text generation evasion. doi: 10.1109/GACLM67198.2025.11231821.

[18] Adithya Mohanavel. Generative component-based architecture for autonomous robot control systems. doi: 10.1109/ICCECE69715.2026.11493032.

[19] Pramath Parashar. A prolog-based natural language database with a procedural interface. doi: 10.1109/ICCTWC68241.2026.11583360.

[20] Meet Amin. Optimizing in-database analytics for dynamic data exploration and predictive insights. doi: 10.2991/978-94-6239-674-6_6.

[21] Krishna Sai Penumarthi. Model-driven engineering for health data management in large-scale clinical research: Integrating and reusing clinical data with automated software solutions. doi: 10.1109/11232132.

[22] Chak Man Lam. Emotion recognition using EEG signals: Machine learning-based analysis on the MAHNOB-HCI dataset. doi: 10.1007/978-3-032-19182-3_5.

[23] Krutika Shah. Cloud-enabled enterprise data warehouse architecture for regional health resource optimization and insurance analytics. 2025. doi: 10.67231/y64edh20.

[24] Gregory Lawler. *Introduction to Stochastic Processes*. Chapman and Hall/CRC, 2006.

[25] François Chollet et al. Keras. `https://keras.io`, 2015.

[26] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[27] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.