

École polytechnique de Louvain

Towards Energy Disaggregation at Home: A Self-Hosted NILM System Integrated with Home Assistant

Author: **Gombeer CHARLES**
Supervisors: **Tom BARBETTE, Ramin SADRE**
Readers: **Rémi VAN BOXEM, Clément DELZOTTI**
Academic year 2025–2026
Master [120] in Computer Science

Contents

List of Acronyms	IV
Acknowledgments	V
1 Introduction	1
1.1 Background and Motivation	1
1.2 Problem Statement	2
1.3 Research Questions and Contributions	3
2 Related Work	6
2.1 NILM Problem Definition	6
2.1.1 Mathematical Formulation	6
2.2 Appliance Signatures and Taxonomy	8
2.2.1 Implications for this work.	8
2.3 Datasets	9
2.3.1 REDD	9
2.3.2 UK-DALE	9
2.3.3 REFIT	9
2.3.4 ECO	9
2.3.5 GREEND	10
2.3.6 Implications for This Work	10
2.4 Deep Learning Approaches for NILM	10
2.4.1 Convolutional Architectures	11
2.4.2 Recurrent Architectures	11
2.4.3 Dilated and Causal Convolutional Architectures	12
2.4.4 Transformer-Based Architectures	12
2.4.5 Summary	13
2.5 Temporal Convolutional Networks	13
2.5.1 General Architecture and Properties	14
2.5.2 Application to NILM	17
2.5.3 Implications for this Work	18
2.6 Domain Shift Problem in NILM	19
2.6.1 Sources of Domain Shift in NILM	20
2.6.2 Mitigation Strategies	20
2.6.3 Implications for This Work	21

2.7	Class Imbalance Problem in NILM	21
2.7.1	Description	21
2.7.2	Mitigation Strategies	22
2.7.3	Implications for This Work	22
2.8	From Research to Practical Deployment	23
2.8.1	Reproducibility and Open Toolkits	23
2.8.2	Embedded and Edge-Deployed NILM	23
2.8.3	Commercial Deployments	24
2.8.4	Home Assistant	24
2.8.5	Implications for this Work	25
2.9	Summary : Research Gap Addressed by This Work	25
3	Backend Architecture and Implementation	27
3.1	System Design Goals and Constraints	27
3.2	Architecture Overview	28
3.2.1	Design Decisions Summary	29
3.3	API Layer	30
3.4	Orchestrator	31
3.5	Data Layer	32
3.6	Model Architecture	33
3.7	Training	35
3.7.1	Pipeline Overview	35
3.7.2	Loss Function	38
3.7.3	Hyperparameter Summary	39
3.8	Cold-Start from Pre-Trained Reference Models	41
3.9	Incremental Fine-Tuning	41
3.9.1	Pipeline Overview	41
3.9.2	Cursor Mechanism	42
3.9.3	Reversibility and Backup Strategy	42
3.10	Inference	43
3.11	Deployment	44
4	Home Assistant Integration	45
4.1	Architecture	45
4.1.1	Configuration Flow	46
4.1.2	Entity Model	49
4.1.3	Data Pipeline	50
4.1.4	Backend Communication and API Layer	51
4.1.5	Persistence and User Labels	52
4.2	End-to-End Validation	52
5	Experimental Setup	55
5.1	Dataset Selection	55
5.2	Data Preprocessing	56
5.2.1	Load and Sampling	56

5.2.2	Window selection	57
5.3	Appliance Selection	57
5.4	Evaluation Metrics	58
5.4.1	Evaluation Procedure	58
5.4.2	Regression Metrics	59
5.4.3	Signal Aggregate Error (SAE)	60
5.4.4	State-Based Classification Metrics	60
5.5	Reference Results from Prior Work	62
5.5.1	Caveats on Direct Comparison	62
5.6	Cross-House Evaluation Protocol	64
5.7	In-House Training Protocol	65
5.7.1	Effect of Appliance-meter Delay	65
5.7.2	Effect of Training Duration	65
5.8	Simulated Finetuning Protocol	66
5.9	Computational Cost Protocol	67
5.9.1	Hardware Configurations	68
5.9.2	Measured Dimensions	68
6	Results and Discussion	70
6.1	Cross-House Evaluation	70
6.2	In-House Training Evaluation	76
6.2.1	Effect of Training Duration	76
6.2.2	Effect of Appliance Meter Delay	80
6.3	Simulated Finetuning	85
6.4	Computational Cost	93
6.4.1	Deployment Topologies	94
6.4.2	Training Cost	94
6.4.3	Inference Latency	95
6.4.4	CPU and RAM Utilisation	96
6.5	General Discussion	97
6.6	Answers to Research Questions	98
6.7	Limitations	99
6.8	Threats to Validity	100
7	Conclusion	102
7.1	Perspectives	103
	Use of Generative AI Tools	105
A	Loss comparaisn	113
B	Home Assistant Integration: User Workflow	114
B.1	First Configuration	114
B.2	How Appliances Are Represented	117
B.3	Training	119
B.4	Appliance Parameters	120

List of Acronyms

BCE	Binary Cross-Entropy.	18
CNN	Convolutional Neural Network.	11
GELU	Gaussian Error Linear Unit.	17
GRU	Gated recurrent unit.	11
HA	Home Assistant.	24
LSTM	Long Short-Term Memory.	11
MAE	Mean Absolute Error.	22
MSE	Mean Squared Error.	18
NAS	Network-Attached Storage.	44
NILM	Non-Intrusive Load Monitoring.	2
RELU	Rectified Linear Unit.	16
RNN	Recurrent Neural Network.	11
RSS	Resident Set Size.	68
SAE	Signal Aggregate Error.	37
SGN	Subtask-Gated Network.	22
TCN	Temporal Convolutional Network.	3

Acknowledgments

I would like to thank my supervisors, Prof. Tom Barbette and Prof. Ramin Sadre, for their guidance throughout this thesis. Their feedback during our meetings and their willingness to engage with the practical deployment dimension of this work shaped the direction of this project at several key moments

I am also grateful to my readers, Rémi Van Boxem and Clément Delzotti, for accepting to review this work and for the time they devoted to evaluating it.

I am thankful to my family and to my partner Erine for their support during the longer stretches of this work, and for the patience that the final months of writing demanded.

Finally, I acknowledge the open-source communities behind NILMTK, PyTorch, and Home Assistant (among others), whose tools made the experimental work of this thesis possible.

Chapter 1

Introduction

1.1 Background and Motivation

European households accounted for 26.2% of final energy consumption in the European Union in 2023 [1], making the residential sector a critical lever for energy reduction. Within this share, electricity accounts for 25.9% of household final energy consumption [1]. Beyond the environmental dimension, energy bills weigh directly on household purchasing power, and the price volatility observed in recent years has made this issue increasingly sensitive from a social standpoint.

Yet for most households, consumption remains opaque: a single monthly figure on an electricity bill, with no indication of which appliances are responsible. Reducing this opacity is not merely a matter of convenience. Behavioural research has shown that detailed, appliance-level feedback leads to measurable reductions in consumption [2]. However, the magnitude of this effect is closely tied to the granularity of the information provided. Knowing that a household consumed 400 kWh over the past month conveys little that is directly actionable: it is at the level of individual appliances that concrete decisions arise, like turning off a device left on standby, replacing an ageing energy-intensive refrigerator, or shifting the use of a washing machine to off-peak hours. Empirical evidence confirms that appliance-level disaggregated feedback achieves substantially greater consumption reductions than aggregate metering data alone [3].

Obtaining this level of granularity through conventional means is, however, costly and complex. Deploying a dedicated smart meter per circuit or per outlet amounts to installing a full home automation infrastructure, with the associated hardware and installation costs. Such a solution remains out of reach for the vast majority of households. There is therefore a genuine need for a solution capable of

providing detailed, appliance-level consumption data from a single measurement point, without any additional hardware infrastructure. This is precisely the problem addressed by *Non-Intrusive Load Monitoring (NILM)*, illustrated in Fig. 1.1.

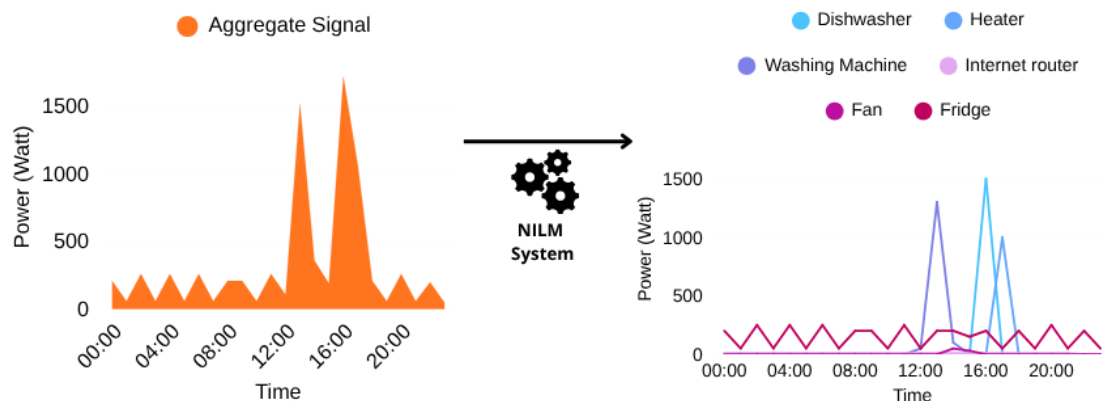


Figure 1.1: Concept of NILM: From Aggregate Signal to Appliance-Level Disaggregation

The founding principle of NILM traces back to the work of George Hart in the early 1990s [4]. The underlying idea is both simple and powerful: the aggregate power signal measured at a household’s main meter is the sum of the individual consumption of each active appliance. By analysing this single signal, it is theoretically possible to recover the contribution of each appliance. This decomposition of the aggregate signal into per-appliance contributions is referred to as energy *disaggregation*. In practice, NILM remains an intrinsically challenging problem. Overlapping appliance activations, similar load signatures across devices, and the limited resolution of low-frequency aggregate measurements make reliable disaggregation difficult in real-world settings. Recent advances in NILM have therefore largely focused on deep learning approaches, which have consistently demonstrated stronger performance and greater robustness than traditional model-based methods as discussed in Section 2.4.

1.2 Problem Statement

Despite the maturity reached by NILM research, a significant gap remains between academic performance and real-world residential adoption. State-of-the-art models achieve strong results on reference benchmarks, yet they are virtually absent from ordinary homes. This thesis does *not* attempt to push the performance frontier further. Instead, it addresses the question of *why* these systems have not translated

into practical deployment, and proposes a concrete answer.

Three obstacles, considered jointly, explain this gap.

The first is the **domain shift problem**. A model trained on a large public dataset may exhibit significant performance degradation when deployed in a different home, because both aggregate-load patterns and appliance-level signatures can differ across households, datasets, user habits, and appliance models [5, 6].

The second is **data scarcity**. Household-specific calibration requires simultaneous recordings of the aggregate signal and the individual consumption of each target appliance, a form of ground truth that can only be collected with dedicated sub-metering hardware. This infrastructure is costly and intrusive by nature, which directly contradicts the non-intrusive promise of NILM [7, 8]. In practice, this barrier makes it unrealistic for most ordinary users to collect the labelled data required to train or adapt a model in their own home without additional hardware or technical effort.

The third is **deployment complexity**. Many NILM works remain primarily evaluated as research systems, not for production use by non-expert users. Running such systems requires substantial expertise in machine learning and systems administration, placing them out of reach for the vast majority of households. Furthermore, deployment-oriented NILM solutions often rely at least partly on cloud infrastructure, requiring households to transmit fine-grained energy data to third-party servers. Such data is now recognised as privacy-sensitive, since it can reveal appliance usage patterns, occupancy patterns, and daily routines [9, 10].

Together, these three obstacles explain why NILM has remained largely confined to academic research. This thesis argues that closing the gap between NILM research and residential adoption requires explicit attention to deployment architecture, in addition to algorithmic accuracy.

1.3 Research Questions and Contributions

This thesis addresses the question of why state-of-the-art NILM systems have not yet found their way into ordinary households, and proposes a system architecture that, through targeted design choices, prioritizes ease of deployment over generality. The investigation is organised around three research questions, and the corresponding contributions are split into research and engineering contributions to make their respective scopes explicit.

Three questions guide the experimental work of this thesis :

- **Research Question 1.** Can a *Temporal Convolutional Network (TCN)* of

compact parameter count, trained on the data of a single household, reach a level of disaggregation performance compatible with residential deployment, while remaining executable on consumer-grade hardware?

- **Research Question 2.** How does the disaggregation accuracy of an in-house-trained model vary with the duration of the recording period and with the synchronisation delay between smart plugs and the aggregate meter, and what data-collection budget can be recommended for residential deployment?
- **Research Question 3.** When a pre-trained appliance model is deployed in a target environment whose appliance signature differs from the source, whether due to a different household or a replaced appliance instance, can fine-tuning on a small sample of target-environment data recover the disaggregation performance of a model trained from scratch on that target?¹

These questions are answered in Chapter 6. Research Question 1 is addressed along its two complementary dimensions: its disaggregation-accuracy dimension is evaluated through the cross-house and in-house experimental protocols (Sections 6.1 and 6.2), and its hardware-feasibility dimension is evaluated through the computational cost protocol (Section 6.4). Research Question 2 is addressed in Section 6.2 through the in-house training protocol. Research Question 3 is addressed in Section 6.3 through the simulated fine-tuning protocol.

Research contributions. The following contributions are evaluated experimentally and constitute the research output of this work :

1. **A causal TCN architecture sized for single-household training on consumer hardware.** A dual-head causal TCN is adapted to operate on active power signals sampled at 0.2 Hz (a rate within reach of mainstream residential smart-meter integrations) with a parameter count dimensioned to permit both training on the data of a single household and inference on consumer-grade hardware. The architecture is evaluated under a cross-house transfer protocol on datasets UK-DALE and ECO, characterising the regime in which a single-household-trained model retains operational disaggregation accuracy on appliances of Type I, Type II and Type III (Research Question 1).

¹No public dataset captures the same household before and after appliance replacement; the cross-house transfer scenario is therefore used as a strict upper bound on the difficulty of intra-house brand change.

2. **A smart-plug-supervised training pipeline for residential deployment.** An in-house training pipeline is proposed that uses the aggregate household power signal together with ground-truth measurements from one or more smart plugs attached to the target appliances. This provides the supervisory signal needed to train or adapt the appliance models without manual annotation or a fully sub-metered installation. The pipeline is evaluated with respect to recording duration and smart-plug/aggregate-meter synchronisation delay, providing a practical data-collection budget for residential deployment (Research Question 2).
3. **A fine-tuning pipeline for adaptation to target signature shift.** A fine-tuning protocol is proposed to adapt a pre-trained appliance model to the signatures encountered in a target environment, whether the shift originates from a different household or from a replaced appliance instance within the same household. The protocol is evaluated through a cross-house transfer experiment, used as a strict upper bound on the difficulty of the intra-household brand-change scenario (Research Question 3).

Engineering contributions. The following contributions support the research contributions but are not themselves the object of experimental evaluation. They are nevertheless required for the system to be deployable in a residential setting :

1. **A Home Assistant custom component.** A custom component is implemented to expose the NILM pipeline within the Home Assistant ecosystem, comprising a configuration flow, an entity model for per-appliance power and state sensors, a data pipeline based on a single shared coordinator, and a management panel for training and labelling operations (Chapter 4).
2. **A Dockerised local inference, training and fine-tuning backend.** The complete model lifecycle (training from scratch, inference, and incremental fine-tuning with reversible checkpoints) is packaged as a Docker container designed to run on consumer-grade hardware typical of residential deployments, from Raspberry Pi-class edge devices to home servers. All processing remains local to the household, with no dependency on external cloud services (Chapter 3).

Chapter 2

Related Work

2.1 NILM Problem Definition

Non intrusive load monitoring refers to the set of techniques used to infer the individual electricity consumption of each appliance from a single measurement point on a building's electrical circuit [4]. It is therefore a signal disaggregation problem: decomposing a global signal into its elementary contributions.

2.1.1 Mathematical Formulation

The aggregated signal measured at the meter, denoted $P(t)$, represents the sum of instantaneous power consumed by all N active appliances at time t :

$$P(t) = \sum_{i=1}^N P_i(t) + \varepsilon(t) \quad (2.1)$$

where :

- $P(t)$ is the aggregated power measured
- $P_i(t)$ is the power consumed by appliance i
- $\varepsilon(t)$ is a residual noise term encompassing unmodelled appliances and measurement disturbances

The NILM task consists of estimating each $P_i(t)$ from the sole observation of $P(t)$; the estimate produced for appliance i is denoted $\hat{P}_i(t)$.

Nature of the Problem

In its general formulation, this problem is fundamentally ill-posed: an infinite number of combinations $\{P_i(t)\}$ can theoretically produce the same aggregated signal $P(t)$. Classical approaches have long tried to resolve this indeterminacy by introducing additional constraints, namely:

- **electrical signatures** specific to each appliance type (start-up transients, harmonics, power factor);
- **behavioural models** based on discrete states (ON/OFF cycles, typical usage patterns);
- **prior knowledge** about the appliances present in the household.

Deep Learning Approach

In the context of this thesis, the problem is not approached from the angle of analytical signal inversion, but rather through *supervised learning*. The model learns directly from labelled examples a projection function:

$$f_{\theta_i} : P_{t-W+1:t} \mapsto \hat{P}_i(t), \quad (2.2)$$

where θ_i denotes the neural network parameters and W , the size of the input windows. This function is optimised to reproduce the actual consumption of the target appliance, without needing to explicitly model other appliances or solve the underlying ill-posed system. Deep learning thereby allows the model to implicitly capture temporal signatures and long-range dependencies in the signal, without manual feature engineering.

This approach reframes the NILM problem as a task of *regression*, where the input is a window of the aggregated signal and the output is the estimated power of the target appliance on a single point of that same window, a paradigm known as *seq2point* (sequence-to-point) [11]. In practice, this seq2point formulation is instantiated as one independent model f_{θ_i} per target appliance. A direct consequence is that the sum constraint of equation 2.1 is no longer enforced at the architectural level : in general, $\sum_i \hat{P}_i(t) \neq P(t)$, since each model is trained against its own appliance signal without any coupling to the others. This relaxation is a deliberate trade-off in favour of the modular per-appliance training, fine-tuning, and cold-start workflows exploited in Chapter 3.

2.2 Appliance Signatures and Taxonomy

The literature classifies domestic appliances according to their *electrical signature* : the characteristic evolution of their power consumption over time. In his seminal work, Hart [4] introduced a foundational taxonomy organised initially around three categories, later extended to a fourth to account for permanently energised loads [12]. Table 2.1 summarises the four types, their characteristic behaviour, representative examples, and their implications for NILM.

Table 2.1: Hart’s four-type appliance taxonomy and its implications for NILM.

Type	Behaviour	Examples	NILM challenge
I (ON/OFF)	Binary states, abrupt transitions	Kettle, toaster, incandescent lighting	Easy to detect ; amplitude ambiguity (e.g. kettle vs. toaster)
II (Finite-state)	Sequence of discrete operating phases	Washing machine, dishwasher, fridge	Multi-phase signatures with variable-duration transitions
III (Continuous)	Smooth continuous variations	Dimmable lights, variable-speed motors, EV chargers	No discrete events to detect in the aggregate signal
IV (Always-on)	Permanently energised, steady baseline	Routers, smoke detectors, alarm systems	Better handled by static baseline subtraction

Fridges occupy a borderline position between Type-II and Type-IV : they exhibit two well-defined thermostat-controlled states (active compressor vs. idle) that fit the finite-state model, but their permanently energised nature also aligns with Type-IV. In this thesis, fridges are treated as Type-II on the basis of their discrete states and the direct relevance of state-transition modelling to disaggregation.

2.2.1 Implications for this work.

Appliance signatures are highly heterogeneous in both temporal structure and amplitude. No single modelling assumption captures this diversity, motivating the use of flexible sequence modelling approaches such as deep neural networks (Section 2.4).

2.3 Datasets

Several public datasets are available for NILM research, including *REDD*, *UK-DALE*, *REFIT*, *ECO*, and *GREENEND*. These datasets are widely used to train and evaluate energy disaggregation models and to study cross-building generalisation [13]. Table 2.2 reports datasets and their details.

2.3.1 REDD

The *Reference Energy Disaggregation Dataset* (REDD) [14], introduced by Kolter and Johnson in 2011, is one of the earliest and most cited public NILM benchmarks. It covers six US households at 1 Hz / 15 kHz aggregate and 1/2 Hz to 1 Hz appliance-level sampling, over a few months per household.

2.3.2 UK-DALE

The *UK Domestic Appliance-Level Electricity* (UK-DALE) [15] dataset, introduced in 2015, covers five homes in the United Kingdom and is noted for its long monitoring duration. It provides whole-home aggregate signals at both 16 kHz and 1/6 Hz, alongside appliance-level sub-metered measurements at 1/6 Hz.

2.3.3 REFIT

REFIT [16] is a large-scale UK residential dataset covering 20 homes with aggregate and appliance-level active power measurements sampled every 8 s over approximately 2 years.

2.3.4 ECO

The *Electricity Consumption and Occupancy* (ECO) [17] dataset was collected from six Swiss households over approximately 8 months. It records both aggregate and appliance-level measurements at 1 Hz. Its key distinguishing feature is the simultaneous recording of occupancy information, enabling research that combines NILM with presence or activity detection.

2.3.5 GREEND

GREEND [18] is a cross-country European dataset collected from 9 residential households in Italy and Austria over approximately 1 year, at 1 Hz for both aggregate and appliance-level active power.

Table 2.2: Overview of publicly available NILM datasets.

Dataset	Year	Country	Houses	Appl. / house	Duration	Variables	Aggregate rate	Appliance rate
REDD [14]	2011	US	5	10–24	Few months	Active power, voltage	1 Hz / 15 kHz	1/2 Hz / 1 kHz
UK-DALE [15]	2015	UK	5	5–54	1–2 years	Active power, voltage	16 kHz / 1/6 Hz	1/6 Hz
REFIT [16]	2015	UK	20	9	2 years	Active power	1/8 Hz	1/8 Hz
ECO [17]	2014	CH	6	7–10	8 months	Active power	1 Hz	1 Hz
GREEND [18]	2014	IT/AT	9	6–9	1 year	Active power	1 Hz	1 Hz

2.3.6 Implications for This Work

Two datasets are retained for the experiments conducted in this thesis: **UK-DALE** and **ECO**. UK-DALE is selected on the basis of its long monitoring duration, which supports data efficiency studies across appliance activation counts, and its sustained adoption in the deep learning NILM literature, ensuring direct comparability with prior work [11, 19, 20]. ECO complements UK-DALE with a geographically and electrically distinct source (six Swiss households on a three-phase grid), making it a meaningful target for cross-dataset transfer experiments. European datasets are preferred throughout, as they reflect the electrical infrastructure most representative of the deployment context.

The remaining datasets surveyed in this section are not retained. REDD is excluded as both non-European and limited in monitoring duration. REFIT and GREEND were considered but excluded on technical grounds related to sampling period and recording duration, detailed in Section 5.1.

2.4 Deep Learning Approaches for NILM

Since 2015, the application of deep learning techniques to energy disaggregation has been concentrated on low-frequency NILM methods [21]. The literature can be

organised along two complementary axes : the output paradigm specifying how a window of aggregate signal is mapped to an appliance signal, and the architecture family specifying the inductive bias of the learning model.

Two output paradigms were introduced respectively by Kelly and Knottenbelt [22] and Zhang et al. [11]. In the *seq2seq* (sequence-to-sequence) setting, the model maps a fixed-length window of the aggregate signal to a window of the target appliance’s power, producing one value per input timestep. In the *seq2point* setting, the output reduces to a single scalar either at the midpoint of the input window in the original formulation or at its last point in a causal variant. Concentrating the model’s capacity on a single prediction target yields substantial reductions in MAE compared to the seq2seq baseline [11].

The sections that follow review the main architecture families in turn : convolutional, recurrent, dilated causal, and attention-based. Each subsection reviews one family along with its inductive bias on appliance signatures, before a comparative synthesis motivates the architectural choice of this work.

2.4.1 Convolutional Architectures

Convolutional Neural Network (CNN) treats NILM as a temporal pattern-matching problem in which the network learns appliance-specific activation signatures regardless of their position in the input window. The reference seq2point implementation of Zhang et al. [11] consists of five one-dimensional convolutional layers followed by two fully-connected layers, totalling approximately 29 M parameters. Subsequent variants (pruned, residual, and multi-scale architectures) have been proposed to improve accuracy or reduce model size [21]. CNNs benefit from parallel processing of the input window, which makes them considerably faster to train than recurrent alternatives. However, the parameter count of the canonical seq2point CNN is essentially driven by the final dense layer and remains a constraint for memory-bounded deployment

2.4.2 Recurrent Architectures

Recurrent Neural Network (RNN) capture sequential dependencies through an explicit hidden state that is propagated across timesteps. In NILM, Kelly and Knottenbelt [22] introduced an *Long Short-Term Memory (LSTM)* based recurrent architecture as part of their Neural NILM framework, alongside denoising autoencoders and activation-regression networks. Later work, including Krystalakos et al. [23], extended recurrent modelling to online-oriented sliding-window inference and considered both LSTM and *Gated recurrent unit (GRU)* variants.

Bidirectional and context-aware recurrent models were also later proposed. However, because recurrent computation is sequential along time, these models are less parallelisable than convolutional alternatives, which can be a practical drawback in computationally constrained or edge-oriented settings.

2.4.3 Dilated and Causal Convolutional Architectures

A distinct line of work has applied *dilated and causal convolutions* to NILM, motivated by the need for a wide receptive field at lower parameter count than convolutional architectures. *WaveNILM* [24] adapted the dilated causal convolutions of *WaveNet* to multi-channel power disaggregation, establishing the viability of this paradigm for energy disaggregation. Yang et al. [20] and Qian et al. [25] subsequently combined dilated convolutions with the seq2point and seq2seq output formulations respectively, while *Conv-NILM-Net* [26] extended the family to multi-appliance prediction. More broadly, the generic TCN framework formalised by Bai et al. [27], based on causal dilated convolutions and residual blocks, provides the architectural backbone adopted in this thesis, as described in Section 2.5.

2.4.4 Transformer-Based Architectures

The introduction of the *self-attention* mechanism by Vaswani et al. [28] opened a new direction for sequence modelling that has been actively explored in NILM. *BERT4NILM* [29] adapted the bidirectional transformer architecture to energy disaggregation through a masked training procedure and a composite regression–classification loss. *ELECTRICity* [30] proposed a computationally lighter transformer variant with a two-phase training routine (unsupervised pre-training followed by supervised fine-tuning) and reported strong performance without requiring explicit data balancing. *NILMFormer* [31] mitigates the non-stationarity of smart meter data by applying per-window normalization and learning to reintroduce the corresponding statistics during reconstruction. It also incorporates timestamp-based positional encodings to embed temporal context.

Transformer-based architectures achieve good accuracy on standard NILM benchmarks. They share, however, two characteristics that make them poorly suited to the deployment scenario considered in this work : their parameter count and computational cost are substantial, and the strongest results are often obtained with multi-household pre-training, an assumption incompatible with the single-household scenario targeted here. These limitations motivate the synthesis presented in the following subsection.

2.4.5 Summary

Table 2.3 summarises the architectures surveyed above along the dimensions that are decisive for the deployment scenario considered in this work : the output paradigm, the causality of inference, the parameter count, and the training regime under which the architecture is typically evaluated. The table also previews the dilated causal convolutional family (WaveNILM, Yang et al., Qian et al., Conv-NILM-Net), which is the subject of Section 2.5.

Table 2.3: Representative deep learning architectures for low-frequency NILM, ordered chronologically.

Architecture	Ref.	Family	Output	Causal	Params	Training regime	Training burden
Neural NILM	[22]	DAE, RNN, CNN	seq2seq	no	<1 M	single-house	moderate
Seq2Point CNN	[11]	CNN	seq2point	no	≈ 30 M	single-house	high
GRU sliding window	[23]	RNN	seq2point	yes	≈ 0.4 M	single-house	moderate
WaveNILM	[24]	dilated CNN	seq2seq	yes	<1 M	single-house	moderate
BERT4NILM	[29]	Transformer	seq2seq	no	≈ 1.94 M	single-house	high
Improved TCN	[25]	residual TCN	seq2seq	yes	≈ 0.132 M	single-house	moderate
ELECTRICity	[30]	Transformer	seq2seq	no	≈ 1.94 M	multi-house	high
Conv-NILM-Net	[26]	TCN-based	multi-output	yes	≈ 0.041 M	single-house	moderate
NILMFormer	[31]	Transformer	seq2seq	no	≈ 0.383 M	multi-house	moderate-high

2.5 Temporal Convolutional Networks

Section 2.4.3 identified the dilated causal convolutional family as the architectural family best matching the constraints of this work, with several independent NILM contributions converging on it as the underlying processing block despite differing output paradigms. This section formalises the generic framework on which the architecture developed in Chapter 3 is built : the Temporal Convolutional Network,

introduced by *Bai et al.* [27].

Three properties of this framework are decisive for the deployment scenario considered here. First, the exponentially increasing dilation factors yield a receptive field that grows exponentially with the number of stacked layers, sufficient to cover the activation duration of slow appliances such as washing machine and dishwasher cycles, at a depth substantially lower than what non-dilated or recurrent architectures would require for an equivalent effective memory. Second, the causal padding scheme constrains the prediction at time t to depend only on past samples, which is a prerequisite for online inference on streaming aggregate data. Third, the absence of recurrence preserves the parallelism of convolutional training and avoids the gradient-propagation pathologies associated with long sequences in recurrent architectures.

The remainder of this section describes the TCN architecture in detail, before the NILM-specific adaptations developed in Chapter 3 are presented.

2.5.1 General Architecture and Properties

Temporal Convolutional Networks were formally introduced as a general-purpose sequence modelling architecture by Bai, Kolter, and Koltun [27], building on earlier work in dilated causal convolutions for audio generation [32]. Their central finding was that a simple convolutional architecture can outperform canonical recurrent networks such as LSTMs across a diverse range of sequence modelling tasks, while exhibiting a longer effective memory.

A TCN is a hierarchical architecture in which the temporal resolution of the input is preserved at every layer. The full input sequence is processed in parallel, and three architectural mechanisms jointly enable long-range modelling on consumer hardware: causal convolutions, dilated convolutions, and residual connections.

Causal convolutions

In a causal convolution, the output at time t depends only on elements at time t and earlier in the input sequence, ensuring that no future information leaks into the prediction. This property is essential for online inference and was popularised by the WaveNet architecture [32].

To preserve the input length across layers, the input is padded with $(k - 1) \cdot d$ zeros on the left, where k is the kernel size and d the dilation factor. This left-padding scheme maintains causality while keeping the output sequence aligned with the input.

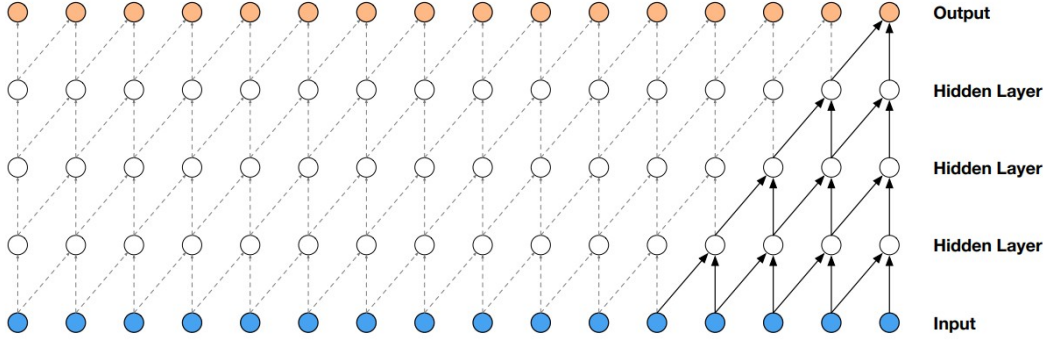


Figure 2.1: Stack of causal convolutional layers.

Dilated convolutions

Causal convolutions alone require very deep networks to capture long-range dependencies, as the receptive field of a stack of standard convolutions grows only linearly with depth. Dilated convolutions, introduced by Yu and Koltun [33] for semantic segmentation and adapted to causal sequence modelling in WaveNet [32], address this limitation by introducing a dilation factor d that controls the spacing between kernel elements.

Formally, the dilated convolution operator F at sequence position s is defined as:

$$F(s) = \sum_{i=0}^{k-1} f(i) x_{s-di}, \quad (2.3)$$

where $f : \{0, \dots, k-1\} \rightarrow R$ is the convolution filter, x is the input sequence, k the kernel size, and d the dilation factor [27]. With $d = 1$, the operator reduces to a standard convolution.

In TCN, the dilation factor follows an exponential schedule across layers, typically $d_l = 2^l$ where l denotes the layer index [27]. Combined with a fixed kernel size k , this yields a receptive field that grows exponentially with depth:

$$\text{RF} = 1 + (k-1) \sum_{l=0}^{L-1} d_l, \quad (2.4)$$

for a stack of L dilated causal layers. This property allows TCN to cover long input sequences with a parameter count that scales only linearly with depth, in

contrast to recurrent architectures whose effective memory is bounded by gradient propagation through time.

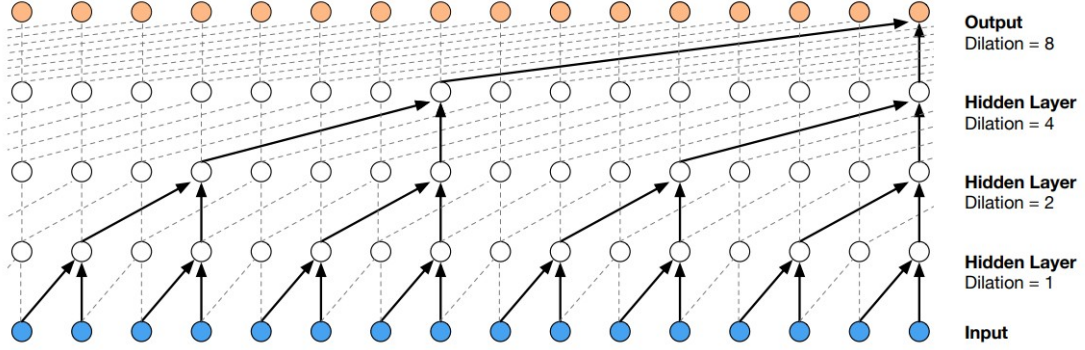


Figure 2.2: Stack of dilated causal convolutional layers with exponentially increasing dilation factors.

Residual connections

As depth increases, training deep convolutional stacks becomes challenging due to gradient degradation. Residual connections, introduced by He et al. [34], address this by reformulating each block as a residual mapping: given an input x and a learned transformation $F(x)$, the block output is

$$o = (x + F(x)). \quad (2.5)$$

In the canonical TCN formulation [27], F is implemented as two dilated causal convolutions, each followed by weight normalisation, a *Rectified Linear Unit (RELU)* activation, and spatial dropout (see Figure 2.3). Spatial dropout zeroes entire channels rather than individual activations, which has been shown to improve regularisation in convolutional sequence models [27].

When the number of input and output channels of a block differs, the identity shortcut cannot be applied directly. A 1×1 convolution $W_{1 \times 1}$ is then introduced to project x onto the output channel dimension prior to addition:

$$o = (W_{1 \times 1} x + F(x)). \quad (2.6)$$

When channel dimensions match, the shortcut reduces to the identity mapping.

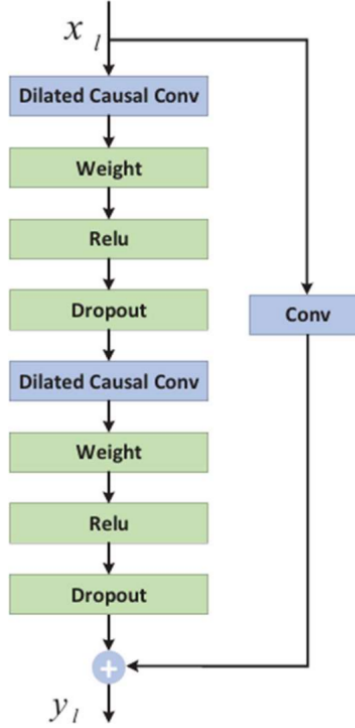


Figure 2.3: Residual block in a TCN, with two dilated causal convolutions, weight normalisation, ReLU activations, and spatial dropout.

2.5.2 Application to NILM

Among the early NILM works to adopt dilated causal convolutions, WaveNILM [24] adapted the WaveNet paradigm to multi-channel power disaggregation. It established that causal convolutions, combined with an exponentially growing receptive field through dilation, could capture the long-range temporal dependencies characteristic of appliance activation cycles without recurrence.

Yang et al. [20] subsequently combined dilated causal convolutions with the seq2point output paradigm of Zhang et al. [11]. Their architecture is a five-layer stack of dilated causal convolutions with dilations $\{1, 2, 4, 8, 16\}$, followed by a fully connected head producing a single midpoint prediction. Two design choices distinguish their work from earlier seq2point models: the use of dilated convolutions enlarges the receptive field while keeping the model parameter count below that of the standard seq2point CNN, and the *Gaussian Error Linear Unit (GELU)* activation function [35] replaces ReLU to retain a portion of the negative-input gradient signal. The model is evaluated on UK-DALE (House 1 for training, House 2

for testing) on five appliances.

One limitations of this architecture are relevant to the present work : although the dilated convolutions are causal at the layer level, the midpoint output combined with a fully-connected head that aggregates the entire convolved sequence breaks operational causality, preventing real-time deployment on streaming aggregate signals.

Beyond seq2point, dilated causal architectures have also been investigated for seq2seq disaggregation. Qian et al. [25] propose a three-block residual TCN with dilations $\{1, 2, 4\}$ and channel widths $\{32, 32, 16\}$ trained on REDD House 1, and report training-time reductions of more than four times relative to GRU and LSTM baselines, with comparable or improved MAE on the four target appliances.

More recent work has extended the TCN backbone with multi-task structures borrowed from the Subtask-Gated Network of Shin et al. [19]. The dual-head TCN of Gireesha et al. [36] trains a regression subnetwork (six dilated TCN layers with dilations $\{1, 2, 4, 8, 16, 32\}$) and a classification subnetwork (six Conv1D layers) jointly under a combined *Mean Squared Error (MSE) + Binary Cross-Entropy (BCE)* loss, reporting performance gains on REDD and UK-DALE. Although their experimental protocol does not fully isolate the contribution of the dual-head structure from the deeper backbone, the work illustrates the convergence of TCN architectures with the gated-subtask paradigm.

2.5.3 Implications for this Work

The TCN architecture was selected as the backbone of this system over alternative deep learning approaches on the basis of four criteria jointly required by the per-household deployment scenario targeted in this work: causal inference, receptive field coverage, data efficiency, and hardware compatibility.

Causal inference. The target deployment scenario requires online inference on a continuously streamed aggregate signal, which forbids access to future samples at prediction time. Many seq2point architectures in the NILM literature use centered windows that exploit both past and future context [11], an assumption incompatible with real-time disaggregation. Dilated causal convolutions enforce this constraint by construction [32], while preserving full parallelism during training, unlike recurrent alternatives, which are causal by nature but sequential, and unlike causally-masked Transformers, which retain the data and compute costs discussed above.

Receptive field coverage. Dilated causal convolutions provide a wide receptive field without a proportional increase in parameter count [27, 32]. This allows the model to capture long-range temporal dependencies in the aggregate signal, a property necessary to disambiguate appliances with overlapping or slowly evolving consumption patterns.

Data efficiency. The limited parameter count improves data efficiency. Transformer-based models such as NILMFormer [31] or ELECTRICity [30] are typically trained on data pooled from multiple households simultaneously, an assumption incompatible with the per-household scenario targeted here, where only data from the target home is available. The reduced capacity of the TCN architecture used in this work makes single-household training viable, as demonstrated experimentally in Chapter 6.

Hardware compatibility. The compact architecture is compatible with the deployment on consumer-grade hardware. Transformer-based models require substantial computational resources for both training and finetuning [31], whereas the TCN architecture used here is designed to run on the deployment device itself, as demonstrated in Section 6.4.

The proposed architecture follows the canonical TCN formulation of Bai et al. [27]: a stack of residual blocks combining dilated causal convolutions, weight normalization, and dropout. Its application to NILM extends the line of work initiated by WaveNILM [24], refined by Yang et al. [20], and extended to multi-appliance prediction by Conv-NILM-Net [26], which established the viability of causal convolutional architectures for energy disaggregation. The inference paradigm, predicting the appliance power at the last timestep of a past-only input window, follows the online disaggregation approach of Krystalakos et al. [23], with their GRU backbone replaced by a TCN to benefit from its parallelizable training, explicit receptive-field control, and improved gradient stability.

The resulting architecture is detailed in Chapter 3.

2.6 Domain Shift Problem in NILM

Traditional machine learning assumes that training and target data share the same statistical distribution. This assumption rarely holds in NILM applications. In practice, the statistical properties of household energy data vary substantially across deployments due to differences in user behavior, appliance signatures, and

environmental noise. This mismatch between the source domain (training data) and the target domain (deployment data) is known as *domain shift* [6]. Because a model implicitly encodes distributional assumptions during training, any departure from those assumptions at inference time can lead to significant performance degradation, a particularly acute problem in NILM, where a model trained on one household rarely transfers reliably to another.

2.6.1 Sources of Domain Shift in NILM

The NILM literature identifies several distinct sources of domain shift, which can be grouped into two categories :

Appliance signature variability: The same category of appliance (e.g. a washing machine or a kettle) can exhibit substantially different power profiles depending on the manufacturer, model, or age of the device. For example, an older refrigerator with a reciprocating compressor produces sharp on/off cycles with a pronounced startup spike, whereas a modern inverter model exhibits a smoothly modulated power draw. A model trained on one particular device signature may therefore fail to recognise the same appliance type in a different household [7, 6].

Household behavioural variability: Consumption patterns depend heavily on the number of occupants, their daily routines, and concurrent appliance usage. Two households owning identical appliances may produce very different aggregate signals due to differences in usage frequency, time of use, and simultaneous operation of multiple devices [6].

2.6.2 Mitigation Strategies

The literature proposes approaches at three levels to address domain shift in NILM, which differ in the availability of target data and the cost of adaptation.

Domain generalisation methods aim to train a model that performs well on unseen domains without any access to target data. This is typically achieved through multi-source training across diverse datasets [6] or data augmentation to artificially broaden the training distribution. While theoretically appealing, these approaches offer no guarantee of performance on a specific unseen household.

Domain adaptation methods assume access to unlabelled target data and attempt to align source and target distributions. In the NILM context, techniques such as adversarial approaches have been proposed [6]. These methods reduce the need for target labels but introduce architectural complexity.

Fine-tuning and retraining constitute the most straightforward strategy when labelled target data is available. Two variants are considered : fine-tuning, where a pre-trained model is updated on a small sample of target-household data ; and full retraining, where a new model is trained from scratch on the target household [5, 37]. Both approaches are simple, effective, and particularly well-suited to practical deployment scenarios where a limited amount of ground-truth data can be collected.

2.6.3 Implications for This Work

In this work, we adopt the fine-tuning and retraining strategy.

For **cross-building transfer**, two scenarios are considered. When no target data is available, the source model is applied directly, optionally with a calibration step that corrects for signal-level offsets in the input distribution. When labelled target data is available, a model is either retrained from scratch or fine-tuned on the target household. The cross-building scenario without target data is studied experimentally as a negative baseline, illustrating the performance gap that motivates per-household training rather than serving as the primary deployment regime.

For **intra-building appliance shift**, a model trained on a known appliance instance is either retrained or fine-tuned when a new instance of the same appliance category is introduced, producing a distinct power signature.

2.7 Class Imbalance Problem in NILM

2.7.1 Description

In the NILM context, *class imbalance* arises from the intrinsic usage patterns of domestic appliances: most devices are inactive the vast majority of the time. Short-duration appliances such as kettles or microwaves may be active for only a few minutes per day, producing ON-state samples that represent less than 2–5% of the total recording duration.

While class imbalance is most commonly discussed in the context of classification,

it manifests equally in regression settings, where the target variable spends the majority of its time in a dominant value range [38, 39]. In the NILM context, this corresponds to the near-zero power region : the model is over-exposed to OFF-state samples during training, which biases gradient updates towards systematically under-predicting active power levels [40]. This produces models that achieve acceptable global *Mean Absolute Error (MAE)* while failing precisely on the ON-state windows that are most informative for disaggregation [41].

Shang et al. [40] demonstrate empirically that the degree of class imbalance in NILM training data varies significantly depending on the recording period selected. For a washing machine on UK-DALE House 1, a seq2point model trained on a period with lower ON-state density yields a MAE of 14.05 W, compared to 6.89 W for a period with higher ON-state density, a twofold difference attributable solely to the imbalance ratio of the training window. This finding underscores the importance of a rigorous and fully documented data split protocol, as results reported in the literature may not be directly comparable if the training period is not explicitly specified.

2.7.2 Mitigation Strategies

The literature proposes countermeasures at two levels.

At the **data level**, oversampling active windows or undersampling inactive ones ensures that ON-state examples are sufficiently represented at each gradient step [42, 13].

At the **algorithm level**, a recurrent strategy is to complement the regression objective with an auxiliary classification objective. The *Subtask-Gated Network (SGN)* of Shin et al. [19] formalises this idea by training a regression subnetwork and a classification subnetwork jointly, with a binary cross-entropy term explicitly supervising the ON/OFF decision. This dual-head paradigm has since been extended in subsequent work [41, 36].

2.7.3 Implications for This Work

Class imbalance is addressed at both levels.

At the **data level**, ON-state oversampling during batch construction ensures that active windows are sufficiently represented at each gradient step, as described in Section 3.7. Oversampling is preferred over undersampling to preserve the OFF-state context required to learn the appliance-absent baseline.

At the **algorithm level**, the dual-head paradigm of Shin et al. [19] is adopted:

a dedicated classification head, supervised by a binary cross-entropy term, is trained jointly with the regression head. The BCE term provides explicit supervision on the ON/OFF state classification, a task that purely regressive objectives cannot solve directly. Beyond improving state estimation, this auxiliary classification signal regularises the shared TCN backbone, encouraging representations that discriminate ON from OFF windows and indirectly benefiting the regression head through multi-task learning. The complete training objective and its additional terms are detailed in Section 3.7.2.

2.8 From Research to Practical Deployment

While the bulk of NILM research focuses on improving disaggregation accuracy on benchmark datasets, a complementary literature has emerged around the practical question of deploying NILM systems in real households. This section reviews three relevant strands: open research toolkits supporting reproducibility, academic studies on embedded and edge deployment, and the commercial systems that brought NILM to end users.

2.8.1 Reproducibility and Open Toolkits

Empirically comparing disaggregation algorithms across publications has historically been impeded by inconsistent data splits, incompatible dataset formats, and the absence of reference implementations. The *NILMTK* toolkit, introduced by Batra et al. [43] and substantially extended in *NILMTK-Contrib* [44], addresses these issues by providing a unified Python API for parsing public datasets (REDD, UK-DALE, ECO, REFIT, and others). This thesis uses NILMTK as the data loading and preprocessing infrastructure for the experimental evaluation in Chapter 5, ensuring direct reproducibility by third parties. The deployed system itself does not depend on NILMTK at inference time.

2.8.2 Embedded and Edge-Deployed NILM

A distinct line of research investigates NILM deployment on resource-constrained hardware, motivated by the privacy exposure and latency of cloud-based inference, the recurring service cost at scale, and integration into embedded smart-metering products. *Winkler et al.* [45] provide the most directly comparable reference for the present work : end-to-end classification on a Raspberry Pi, building on the earlier YoMoPie smart-metering platform [46]. The same viability has since been

demonstrated on Jetson-class hardware [47] and on microcontroller targets through aggressive feature trimming [48], framing the envelope within which the present system operates.

2.8.3 Commercial Deployments

Sense was a home energy monitor developed by an American startup of the same name. The system relied on the installation of current clamps on the home’s electrical panel, enabling high-frequency sampling of the aggregate power signal [49]. However, the hardware cost and the requirement for professional installation by a licensed electrician represented significant barriers to widespread adoption [50]. Furthermore, all processing relied on a proprietary cloud infrastructure, implying dependency on a third-party service and raising data privacy concerns [51]. Ultimately, these obstacles were confirmed when Sense stopped marketing its consumer product in December 2025 [52].

Smappee is a Belgian company that also offers an energy monitor based on NILM technology [53, 54]. The system uses a patented technology to identify up to five appliances from the aggregate signal, with a reported average accuracy of 70% under their internal metric, which is not publicly specified [53]. Like Sense, Smappee relies on a cloud architecture for data storage and analysis, and its disaggregation algorithm is not publicly accessible [54, 53]. Smappee offers a partial local integration with Home Assistant and a limited household-specific adaptation mechanism [53, 55], yet the underlying NILM model remains proprietary and cannot be retrained or customised by the end user, and real-time data access depends in most configurations on the cloud API [53, 55].

2.8.4 Home Assistant

Home Assistant (HA) is an open-source home automation platform designed to run locally on a home server, typically on hardware such as a Raspberry Pi [56]. Its primary goal is to provide a unified interface for monitoring and controlling all connected devices in a household, without relying on external cloud infrastructure.

The platform is based on a component-oriented architecture. Each integration is responsible for connecting a specific device or service to the Home Assistant ecosystem. Home Assistant also supports custom integrations, i.e., user-defined components that extend the platform’s capabilities [57].

Home Assistant is selected as the deployment target on the basis of three properties relevant to the constraints identified in Section 2.8: its local-first architecture eliminates the cloud dependency that characterises commercial NILM products; its

plugin-based component model supports the addition of custom integrations without modifying the core platform; and its existing user base of energy-conscious households provides a natural deployment context for an appliance-level disaggregation system.

2.8.5 Implications for this Work

The reviewed literature exposes a consistent gap that the system proposed in this thesis specifically targets.

Commercial systems demonstrate that NILM is technically viable in consumer-facing products but rely on cloud infrastructure for inference and storage, introducing privacy exposure, service and internet dependency and latency incompatible with local automation, and expose proprietary, non-adaptable models that cannot accommodate household-specific signature shifts when appliances are replaced.

Academic edge-NILM research establishes the feasibility of running deep models on consumer hardware and has produced open hardware platforms, but few studies combine deep learning accuracy, fully local inference on consumer hardware, and Home Assistant integration into a mature ecosystem.

2.9 Summary : Research Gap Addressed by This Work

The literature reviewed in this chapter establishes three observations:

1. Deep learning architectures span a clear accuracy-vs-deployability spectrum, with Transformers at one end and dilated causal convolutional networks at the other ; the latter family is more compatible with the single-household, edge-deployed scenario targeted in this work (Section 2.5).
2. Domain shift is a documented obstacle; the mitigation strategies that fully sidestep it are fine-tuning and retraining on target-household data (Section 2.6).
3. Practical deployment has produced academic prototypes demonstrating edge feasibility and commercial systems offering end-user products, but neither addresses the full triple of edge inference, household-specific retraining, and open home automation integration (Section 2.8).

The system proposed in this thesis directly addresses these three observations: it adopts a compact dilated causal TCN backbone (Chapter 3), trains on target-household data collected through smart plugs to bypass domain shift, and exposes the full pipeline through a Home Assistant custom component running on consumer hardware (Chapter 4).

Chapter 3

Backend Architecture and Implementation

3.1 System Design Goals and Constraints

The architecture of the proposed NILM system was shaped by a set of practical constraints that differ significantly from those encountered in classical research settings. Unlike academic implementations that run on dedicated GPU servers with pre-labelled benchmark datasets, this system is designed to *operate in a residential environment on commodity hardware*, typically deployed on a separate always-on machine on the local network (see Section 6.4 for deployment topologies). This imposes strict limits on memory footprint, computational budget, and storage capacity, and rules out any design that relies on always-on GPU acceleration or large in-memory data structures.

A second constraint is *the partial or total absence of labelled ground truth at deployment time*. In a residential setting, appliance-level labels are collected progressively through smart plugs and may never be available for appliances whose outlet is physically inaccessible (e.g., a built-in oven). The architecture must therefore support two complementary workflows: a training pipeline, in which data is collected continuously and used to train models as sufficient observations accumulate; and a cold-start mechanism, which allows appliances without any ground-truth label to be initialised from a pre-trained reference model and used for limited disaggregation until household-specific data becomes available.

Finally, the system must remain *adaptable over time*. Appliances are added, replaced, or modified; usage patterns shift across seasons and occupancy changes. A static, one-shot trained model would inevitably degrade in accuracy. The

architecture therefore exposes explicit mechanisms for registering new appliances, triggering retraining, and performing lightweight fine-tuning, all without requiring a full system reset or model retrain. Critically, each appliance is trained and fine-tuned independently: updating one model does not affect the others.

Together, these three constraints (limited hardware, absence of *a priori* labels, and long-term adaptability) motivate every major design decision described in the sections that follow.

3.2 Architecture Overview

The system is organised into four loosely coupled layers, each with a clearly delimited responsibility, as illustrated in Figure 3.1. The full implementation is available at <https://github.com/klaertus/nilm-docker>.

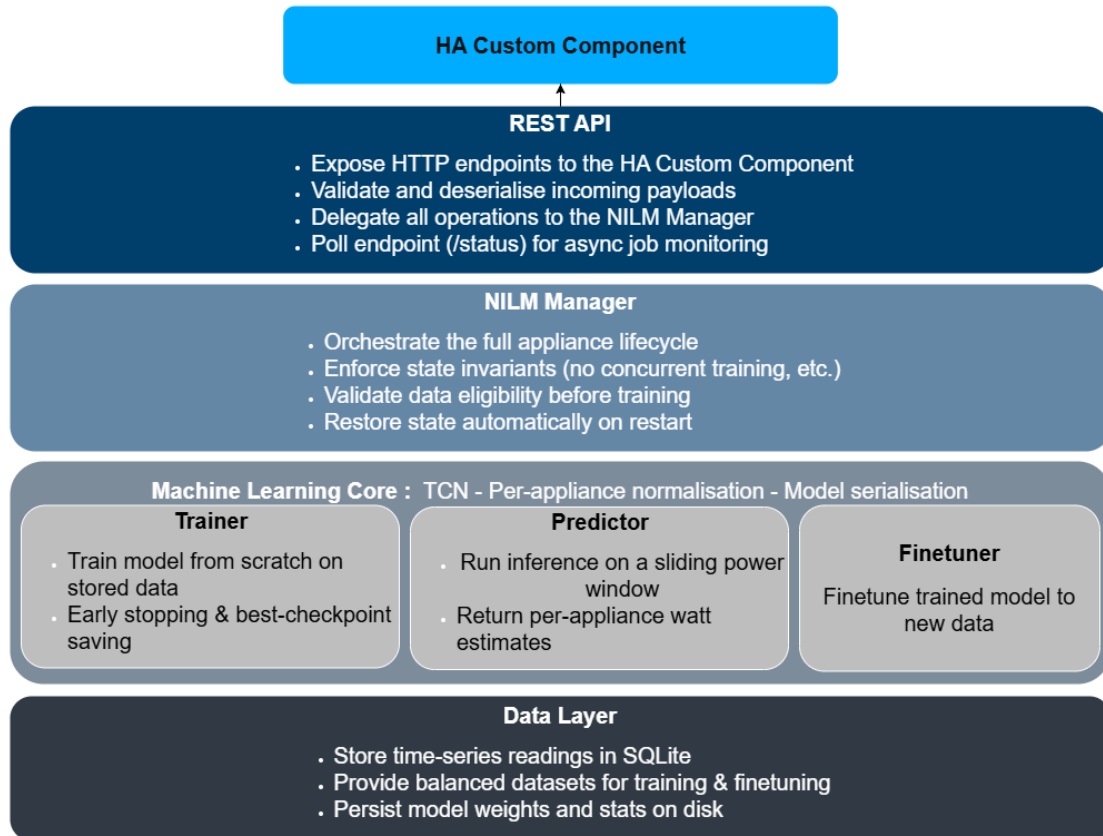


Figure 3.1: Conceptual architecture of the backend system.

The outermost layer is the *API*, which acts as the sole public interface of the

system. Behind it sits the *NILMManager*, a thread-safe orchestrator that owns all runtime state and coordinates the sub-modules without itself implementing any neural network logic. The ML core consists of three single-responsibility components: *NILMTrainer*, *NILMFineTuner*, and *NILMPredictor*, each operating on plain Python arrays and returning plain Python dictionaries, with no knowledge of HTTP or database internals. The innermost layer is a *SQLite database* that persists the raw time series, while model weights and normalisation statistics are stored as flat files on disk.

An important distinction underlies this architecture: the system operates around the concept of a *general model*, which acts as a named container grouping one or more appliances. Each appliance encapsulates its own independently trained or fine-tuned model weights alongside its own normalisation statistics. Training, fine-tuning, and prediction therefore operate independently across the individual appliance models that compose the currently loaded general model. This design makes it straightforward to maintain several general models simultaneously, for instance a *my_house1* model containing a kettle, an oven, and a washing machine, alongside a *my_house2* model with an entirely different set of appliances and corresponding models.

3.2.1 Design Decisions Summary

Decisions internal to the model itself, namely the output paradigm, training loss, and inference gating, are not summarised here but justified in the sections where they are introduced (Section 3.6, Section 3.7.2, and Section 3.10 respectively). The present section focuses on the architectural and systems-level decisions affecting the backend as a whole.

Model granularity. One model per appliance is trained and served independently, rather than a single multi-output model jointly predicting all targets. This preserves the modular training, fine-tuning, and cold-start workflows exploited in Sections 3.8 and 3.9 : a new appliance can be added, retrained, or replaced without touching the others. The trade-off is that the architectural sum constraint $\sum_i \hat{P}_i(t) = P(t)$ is no longer enforced, as already discussed in Section 2.1.1.

Normalisation scope. Each model carries its own normalisation statistics (μ_a, σ_a) , rather than a single set of global statistics shared across all appliances. This is required for trained, fine-tuned, and cold-started models to coexist in the same deployment : each appliance was fitted under its own data distribution, and forcing a unified preprocessing would either invalidate the existing checkpoints or

constrain all future training to a fixed scaling. The cost is that statistics must be persisted and reloaded per appliance, which is handled transparently by the data layer (Section 3.5).

Data ingestion. Aggregate readings and per-plug ground-truth values are pushed from Home Assistant to the backend through `POST /data` (Section 3.3), rather than pulled by the backend from the Home Assistant recorder. The push model makes the backend independent of the recorder’s retention policy and configuration, at the cost of a more constrained one-way data flow.

Storage backend. A dedicated SQLite database is used (Section 3.5) rather than reusing the Home Assistant recorder. Multi-week training windows must remain available under the backend’s own control, independent of any pruning rule set on the Home Assistant side (the default detailed retention on Home Assistant is 10 days). The trade-off is one additional component to maintain, which is acceptable given that the alternative would couple training-data retention to a system the backend does not control.

Training execution. Training jobs are dispatched to a background thread and their progress is polled by the client through `GET /status` (Section 3.3), rather than blocking the HTTP request until completion. A single training epoch may take half an hour on a Raspberry Pi 4, which makes a blocking endpoint incompatible with the responsiveness expected from a residential automation system. The trade-off is a more complex request/response protocol on the Home Assistant-side component.

3.3 API Layer

The API layer is intentionally kept thin. Its responsibility is limited to exposing the backend’s operations over HTTP : every route validates its input through a Pydantic model and delegates immediately to the NILMManager, so that no training, inference, or database logic resides in the routing layer itself. This keeps the HTTP contract simple and stable, which is the property the Home Assistant component relies on : its implementation is coupled to the API surface, not to the ML internals behind it.

The same API boundary is also used for data ingestion. The backend does not actively poll Home Assistant; it acts as a passive receiver. Power readings are pushed to the backend through `POST /data` by the Home Assistant integration (see Chapter 4). The request body contains two collections of timestamped readings:

`total`, a list of (`timestamp`, `watts`) pairs for the aggregate household signal, and `appliances`, a dictionary mapping appliance names to their own (`timestamp`, `watts`) series. The backend does not impose a fixed ingestion frequency at the HTTP level; in the deployed configuration the same sampling period used for inference is expected, i.e. approximately one sample every 5 seconds, corresponding to roughly 17,280 samples per day. Batches may therefore be sent periodically rather than sample by sample.

Training and fine-tuning are exposed as asynchronous endpoints. Rather than blocking the client for the duration of a training job, which may last several hours on a CPU-only device, the API returns immediately and dispatches work to a background thread. Progress is polled through `GET /status`. This design keeps the system responsive during long-running operations, which is a practical requirement when the backend shares hardware with Home Assistant.

Error handling is centralised at the API boundary. Domain exceptions are mapped to HTTP status codes: 409 for lifecycle conflicts such as a concurrent training job, 404 for unknown appliances, 400 for malformed requests, so that client-side error handling remains predictable regardless of which internal component raised the exception.

3.4 Orchestrator

The `NILMManager` is the central orchestrator of the backend, instantiated once at startup and injected into the API routes. It owns all runtime state : the active model name, the loaded appliance models and their statistics, the database wrapper, and the live predictor instance. It also enforces lifecycle invariants such as rejecting inference calls when no model is loaded or refusing a new training job while one is already running.

The manager is accessed concurrently from several threads. The API layer dispatches long-running training and fine-tuning jobs to a daemon background thread (Section 3.3), while inference, status polling, and management requests continue to be served by the server. For example, the Home Assistant component polls `/status` throughout a training run while still requesting `/prediction` on its regular sensor update cycle. Without explicit protection, these accesses could read a partially-installed predictor reference or allow two conflicting lifecycle operations to interleave.

Thread safety is therefore handled through a single `threading.RLock` that protects all mutations to the in-memory model state. A reentrant lock is required because some internal helpers re-acquire it while the caller already holds it. Long-

running operations are designed to run outside the critical section : training and fine-tuning acquire the lock only to swap in the newly fitted model once it is ready, leaving the previous predictor fully operational in the meantime. Inference similarly snapshots the predictor reference under the lock and then runs the forward pass outside it, so prediction latency is never blocked by a background training job.

Per-appliance independence is preserved throughout. During fine-tuning, each appliance model and its statistics are deep-copied before any update, fine-tuned in isolation, and written back into the live predictor atomically under the lock. Updating one appliance therefore never affects the serving state of the others, which satisfies the adaptability constraint identified in Section 3.1.

State survives container restarts through a `state.json` file that records the active model name. At startup, the system reads this file and, if the referenced model directory exists, reconstructs the full manager state from disk without requiring any retraining or manual API call.

3.5 Data Layer

Persistence is implemented on SQLite rather than delegating storage to Home Assistant’s built-in recorder. By default, Home Assistant retains only a limited amount of high-resolution power readings, which is insufficient to accumulate the weeks of on/off cycles required for supervised training. A dedicated SQLite database decouples NILM data retention from the home automation platform and keeps the training window under the backend’s control. SQLite was chosen over a server-based alternative for its single-file deployment model, absence of a daemon process, and native Python support, all well-suited to a single-container self hosted installation. In the current implementation, retention is explicit: rows remain in the mounted SQLite volume until the user clears them through `DELETE /data`; there is no automatic time-based purge inside the backend.

Data enters this database through the endpoint `POST /data`. This endpoint is called by the Home Assistant side of the integration, which pushes batches of power readings to the backend. The backend therefore follows a push model rather than a pull model: it never queries Home Assistant’s recorder directly. Each request contains a list of aggregate readings and, optionally, one or more appliance-level series collected from smart plugs.

The schema consists of two tables. The `aggregate` table stores one household-level power value per Unix timestamp:

```
aggregate(
```

```

    timestamp INTEGER PRIMARY KEY,
    total REAL NOT NULL
)

```

The `appliances` table stores per-appliance smart-plug readings when available:

```

appliances(
    timestamp INTEGER NOT NULL,
    appliance TEXT NOT NULL,
    value REAL NOT NULL,
    PRIMARY KEY (timestamp, appliance)
)

```

with `(timestamp, appliance)` as primary key and an additional index on `(appliance, timestamp)` to accelerate the time-range scans performed during training and fine-tuning. Writes use `INSERT OR REPLACE`, which makes ingestion idempotent: if Home Assistant resends the same batch, the existing readings are overwritten rather than duplicated. Connections use WAL journal mode so that ingestion, training queries, and inference-related reads can overlap without serialising the whole database.

When a training dataframe is built, aggregate readings are joined with the selected appliance series on timestamp. Missing appliance values are filled with 0 W, which corresponds to the absence of a labelled smart-plug reading for that timestamp. This keeps the aggregate timeline dense while allowing appliance labels to be collected progressively.

3.6 Model Architecture

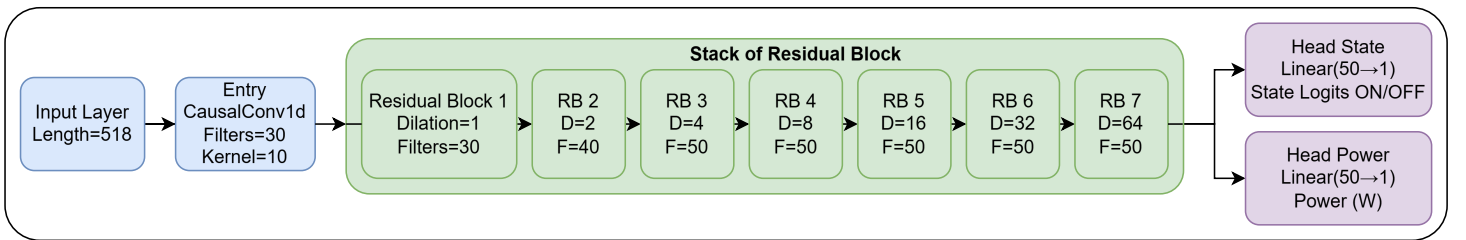


Figure 3.2: High-level model architecture.

The model is a dilated causal TCN backbone [27] adapted to the seq2point output paradigm of Zhang et al. [11] and dimensioned for the deployment constraints

described in Section 3.1 (Figure 3.2). A fixed-length window of aggregate power readings is mapped to a single appliance power estimate at the last time step. Causality is enforced by left-only padding in every convolution, ensuring that the output at time t depends only on samples at times $\leq t$. This makes the model suitable for streaming inference without requiring future samples.

The design choices are summarised below; entries marked as *inherited* follow established practice in the TCN-NILM literature, while *adapted* reflects modifications introduced in this work.

Window size $T = 518$. (*adapted*) The window length is set equal to the network’s receptive field so that every input sample influences the prediction at the last timestep. The receptive field is itself determined by the backbone depth and dilation schedule (derivation below); the resulting 518 samples at 5s correspond to approximately 43 minutes of context, comparable to the 60 minutes used by Yang et al. [20] at 6s resolution.

Entry convolution. (*inherited / empirical*) Kernel size 10, 30 output channels. The wide entry kernel follows the entry-layer pattern of Zhang et al.’s [11] seq2point CNN (kernel of size 10 in their first layer), and captures short local activation patterns (rising edges, bursts, transients) before the dilated stack. The channel count of 30 is set empirically to keep the input-side parameter count low while providing sufficient feature diversity for the deeper blocks.

Backbone. (*inherited / empirical*) Seven residual blocks with dilations $d \in \{1, 2, 4, 8, 16, 32, 64\}$ and channel widths $\{30, 40, 50, 50, 50, 50, 50\}$. The block structure follows the canonical TCN formulation of Bai et al. [27]: two causal convolutions of kernel size 3 with weight normalisation, ReLU activation, and dropout, followed by a residual skip connection (a 1×1 projection when the channel count changes, identity otherwise). The depth of seven blocks is the minimum yielding a receptive field that covers the full input window under the exponential dilation schedule [27]. Channel widths progress from 30 to a plateau of 50 in deep blocks, chosen empirically to balance expressive capacity against parameter count on the deployment target ; the plateau reflects that exponential dilation already provides wide receptive coverage in deep layers, so further channel growth adds parameters without proportional capacity gain.

Dual linear projection heads. (*adapted*) Following the dual-output paradigm of the Subtask Gated Network of Shin et al. [19], the feature vector at the last timestep is passed to two parallel projections producing respectively the regression

output (estimated power in normalised space) and a logit for the on/off classification subtask. Unlike prior seq2point architectures [11, 20] which apply a multi-layer dense block before the regression output, each head is implemented as a single linear projection from the 50-channel feature vector. This reduces the head parameter count by approximately two orders of magnitude with no observed accuracy loss on the appliances evaluated in Chapter 6. The two heads are jointly optimised through the combined loss described in Section 3.7.2, which couples them via a gated regression term and supervises each head directly against its target.

Weight normalisation. Preferred over batch normalisation because it is independent of batch statistics, ensuring identical behaviour at training and inference time [27].

The receptive field is derived as follows. The entry convolution (kernel 10, dilation 1) contributes 9 samples beyond the prediction point. Each residual block contributes $4d$ samples (two causal convolutions of kernel size 3, each adding $2(k - 1)d = 4d$ samples of left context). With seven blocks at dilations 1, 2, 4, 8, 16, 32, 64 :

$$\text{RF} = 1 + 9 + 4 \cdot (1 + 2 + 4 + 8 + 16 + 32 + 64) = 1 + 9 + 4 \cdot 127 = 518.$$

Since the input window contains 518 samples, the receptive field spans the full input sequence : every sample influences the prediction at the last timestep.

3.7 Training

3.7.1 Pipeline Overview

The training pipeline fits one independent model per appliance. Each appliance owns its dataset, normalisation statistics, and persisted checkpoint, so adding or retraining one device leaves all others unchanged. The pipeline proceeds through the following sequential stages.

Data retrieval. The *NILMManager* queries the data layer to fetch the aggregate and appliance time series for the requested training window.

Feasibility check. Before any computation, the manager runs three sequential gates : the database must hold at least one full input window for the appliance

(`window_size` = 518; the OFF→ON transitions count in the requested time range must exceed `min_on_events` (default 5 for training, 3 for fine-tuning); and once the balanced sampler is built, both the training and validation loaders must contain at least `min_on_windows` ON-labelled windows (default 100). This guard is essential in practice : for infrequently used appliances such as a kettle, the collected series may consist (almost) entirely of zero-watt readings, which would otherwise let a trivial always-OFF predictor win the early-stopping selection. If any condition is not met, the pipeline aborts with an explicit error rather than producing a silently degenerate model.

Statistics computation. The aggregate mean and standard deviation, and the appliance mean and standard deviation, are computed from the selected time range and persisted to `stats.json`. The ON/OFF threshold is not estimated from the data by a method such as a median split or Variance-Sensitive thresholding [41]. It is a user-defined appliance parameter: it is provided when the appliance is registered through `POST /appliances`, can later be updated with `PATCH /appliances/{name}`, and falls back to the configured default threshold of 10 W when omitted. This threshold is used exclusively at training time : it defines the ON/OFF labels for the balanced sampler and for the BCE classification target. At inference time, the ON/OFF decision is taken by the trained classifier head and not by any threshold on the aggregate signal (Section 3.10). This low default favours noise rejection across heterogeneous appliances; users are expected to override it based on the nominal power of their specific appliance. During training, this watt-level threshold θ is transformed into the normalised appliance space as

$$\theta_{\text{norm}} = \frac{\theta - \mu_{\text{app}}}{\sigma_{\text{app}}},$$

and labels with power above this value are treated as ON. In practice, the threshold is the minimum power level below which the appliance is considered to be off.

Dataset construction. A *NILMDataset* is built by sliding a window of size $T = 518$ over the normalised aggregate signal with stride 1. Each sample pairs one aggregate window with the appliance power at its last time step, following the seq2point formulation [11].

Temporal train/validation split. The dataset is split chronologically into 80 % training and 20 % validation windows. A gap of one full window is inserted at the boundary to ensure that no raw sample appears in both subsets, making the validation error a realistic estimate of out-of-sample performance.

Balanced sampling. Class imbalance is the central challenge in NILM training: for appliances such as kettles or ovens, the ON state may represent less than 5 % of all windows. A *WeightedRandomSampler* up-weights ON-state windows so that the sampled ON fraction reaches a minimum of 1 %. For naturally imbalanced appliances (kettle, dishwasher, washing machine) this is an upsampling operation that prevents the always-OFF predictor from dominating the loss; for naturally balanced appliances (fridge), where the ON fraction already exceeds 1 %, the sampler leaves the distribution unchanged. The total number of samples drawn per epoch is capped at 50,000 windows to bound training time on the deployment device. Validation is evaluated without resampling. An additional check verifies that both the training and validation subsets contain a minimum of ON-state windows; if either condition fails, training is aborted rather than producing a trivially degenerate model.

Training. The model is optimised with Adam ($\alpha = 10^{-3}$, batch size 1,024) using the combined loss described in Section 3.7.2. A *ReduceLROnPlateau* scheduler halves the learning rate after five epochs without improvement on the validation loss, down to a minimum of 10^{-6} . and `torch.compile` are enabled when a CUDA device is available; on CPU (e.g. Raspberry Pi), compilation is skipped as the Inductor CPU backend incurs a >30 s first-call overhead with no throughput benefit on this model size.

Evaluation during training. The validation split is used for model selection and early stopping. At each epoch, the trainer reports the combined validation loss and its three components (gated MSE, power MSE, BCE), together with the mean absolute error rescaled back to watts. The checkpoint with the lowest combined validation loss is retained. These validation metrics are internal training diagnostics; broader NILM evaluation metrics such as ON/OFF F1 score or *Signal Aggregate Error (SAE)* are not used to optimise the model and are reported separately in Chapter 6.

Early stopping and checkpointing. The checkpoint achieving the lowest validation loss is saved to disk at each improvement. Training halts if no improvement is observed for 15 consecutive epochs, after which the best checkpoint is reloaded as the final model.

Persistence. Model weights (`model.pt`) and normalisation statistics (`stats.json`) are written to the appliance-specific directory, making the trained state immediately available for inference and fine-tuning.

3.7.2 Loss Function

Training combines three terms, building on the SGN loss of Shin et al. [19]:

$$\mathcal{L} = \lambda_{\text{gated}} \mathcal{L}_{\text{gated}} + \lambda_{\text{power}} \mathcal{L}_{\text{power}} + \lambda_{\text{state}} \mathcal{L}_{\text{state}}$$

The first term is the *gated regression loss* of Shin et al. (their eq. 7a). The raw power prediction $\hat{y}$ is multiplied element-wise by the predicted ON probability $\hat{p} = \sigma(\text{logits})$, and the MSE is computed between this gated output and the normalised target:

$$\mathcal{L}_{\text{gated}} = \text{MSE}(\hat{p} \cdot \hat{y}, y).$$

This term penalises any deviation of the gated product $\hat{p}\hat{y}$ from y , and thereby couples the two heads. From its gradients, $\partial \mathcal{L}_{\text{gated}} / \partial \hat{y} = 2(\hat{p}\hat{y} - y) \hat{p}$ and $\partial \mathcal{L}_{\text{gated}} / \partial \hat{p} = 2(\hat{p}\hat{y} - y) \hat{y}$, the regression head is corrected in proportion to the predicted ON probability, and the classifier in proportion to the predicted power: each head is held accountable for the product error to the degree that the other commits to a non-trivial output.

The second term is the *regression loss* on the raw, ungated power head:

$$\mathcal{L}_{\text{power}} = \text{MSE}(\hat{y}, y).$$

$\mathcal{L}_{\text{power}}$ is defined in Shin et al. (eq. 7b) but is not part of their training objective. We retain it as a regularizer motivated by an identifiability issue that our deployment configuration exposes. As shown above, $\partial \mathcal{L}_{\text{gated}} / \partial \hat{y}$ vanishes on OFF samples once the classifier converges and $\hat{p} \rightarrow 0$, leaving the regression head unconstrained on the majority of training samples. Under the soft gating used by Shin et al. at inference this is inconsequential, since the arbitrary OFF-state value of $\hat{y}$ is masked by the near-closed gate. The backend instead applies a hard gate $g(\hat{p}) = \mathbf{1}[\hat{p} \geq 0.5]$ at inference. Because this threshold is permissive, a single classifier false positive opens the gate fully and propagates the unconstrained $\hat{y}$ as a spurious high-magnitude prediction, observed as ~ 3000 W kettle spikes in preliminary runs. $\mathcal{L}_{\text{power}}$ anchors $\hat{y}$ to the ground truth on all samples regardless of the gate, bounding the magnitude of these false-positive predictions at no additional parameter cost.

The third term is the *state classification loss*, a binary cross-entropy on logits whose target is obtained on-the-fly by thresholding the normalised power label at the per-appliance ON/OFF threshold θ_{norm}

$$\mathcal{L}_{\text{state}} = \text{BCE}(\text{logits}, \mathbf{1}[y > \theta_{\text{norm}}]).$$

All three weights are fixed to 1.0 ($\lambda_{\text{gated}} = \lambda_{\text{power}} = \lambda_{\text{state}} = 1.0$). The three losses operate in the normalised appliance space; during validation the absolute error

is rescaled to watts via the appliance standard deviation, giving an interpretable metric across appliances of differing power ranges.

We validate $\mathcal{L}_{\text{power}}$ against the Shin eq. 8 baseline ($\lambda_{\text{power}} = 0$, i.e. $\mathcal{L}_{\text{gated}} + \mathcal{L}_{\text{state}}$) on four UK-DALE House 1 appliances (Appendix A, Table A.1). The three-term formulation is the stronger objective: it improves *glsRMSE* and recall on all four appliances and MAE, F1 and precision on three of the four. On the kettle, the MAE and F1 differences (+0.3 W, -0.002) are within single-seed noise, and precision drops in exchange for higher recall. On average, MAE falls 17% and RMSE 19%. $\mathcal{L}_{\text{power}}$ also raises both precision and recall despite acting only on the regression head (e.g. dishwasher recall $0.271 \rightarrow 0.475$), a cross-head effect mediated by the shared backbone.

The kettle illustrates the intended mechanism. Its RMSE stays an order of magnitude above its MAE in both configurations (136.5 vs 11.1; 121.9 vs 11.4), a heavy-tailed error distribution consistent with the false-positive spikes described above. $\mathcal{L}_{\text{power}}$ shortens this tail (RMSE -11% , SAE -45%) and raises recall at a small precision cost: it admits more false positives but bounds their magnitude.

Only SAE is mixed : improved on kettle and washing machine, worse on fridge and dishwasher and the two regressions have distinct causes. On the fridge, every per-sample metric improves while SAE rises ($0.037 \rightarrow 0.059$); with recall near 0.9 this points to a small systematic sign bias in the residuals that no longer cancels over the aggregation window, not to under-detection. On the dishwasher, the baseline reaches a lower SAE despite a recall of 0.271; with only 87 events the metric is dominated by a handful of aggregation windows, and we treat it as unreliable rather than as evidence of better energy estimation.

A residual limitation is the soft-train / hard-infer gating mismatch. We compute $\mathcal{L}_{\text{gated}}$ on the soft product $\hat{p}\hat{y}$ and apply the hard gate $g(\hat{p}) = \mathbf{1}[\hat{p} \geq 0.5]$ only at inference, relying on $\mathcal{L}_{\text{power}}$ to keep $\hat{y}$ well-behaved when the gate opens. The *Hard SGN* variant of Shin et al. removes this mismatch by applying $g(\hat{p})$ already in the regression loss : training and inference both operate on the hard-gated product $\hat{y} \cdot g(\hat{p})$ at the cost of a non-differentiable gate in the training objective. A direct comparison is left for future work.

3.7.3 Hyperparameter Summary

Table 3.1 consolidates all training hyperparameters. Each entry indicates whether the value was inherited from prior literature, adapted from a related work, or set empirically during preliminary experiments on the datasets used in this thesis.

Table 3.1: Training hyperparameters. Status: I = inherited, A = adapted, E = empirical.

Hyperparameter	Value	Status	Details
Window size T	518	A	Set equal to the 7-block receptive field
Batch size	1024, 512	E	Largest power-of-two fitting Pi 4 RAM (512) and GPU standard (1024)
Learning rate α	10^{-3}	I	Standard Adam
Max epochs	100	E	Bounded by early stopping in practice
Early stopping patience	15	E	Empirically robust across appliance categories
LR scheduler patience	5	E	Halves LR once or twice before early stopping
LR scheduler factor	0.5	I	Standard ReduceLROnPlateau configuration
Min learning rate	10^{-6}	I	Prevents oscillation at end of training
Val split	20 %	I	Standard chronological split
Leakage gap	1 window	E	Prevents temporal leakage between subsets
ON ratio target	1 %	E	Minimum to activate sampler; prevents trivial-zero predictor
Max samples per epoch	50,000	E	Bounds epoch duration on Pi 4; no significant accuracy loss
Stride	1	E	Maximum temporal coverage; storage cost is acceptable
Dropout p	0.2	I	Standard configuration
Loss weights $\lambda_{\text{gated}}, \lambda_{\text{power}}, \lambda_{\text{state}}$	1.0	E	Equal weighting; validated empirically in Appendix A
Default ON/OFF threshold	10 W	E	Training-time default used for ON/OFF labels (sampler + BCE); user overrides per-appliance

No systematic ablation study was conducted on these hyperparameters except for the loss weights (Appendix A); their values were fixed after preliminary experiments and held constant across all appliances and datasets reported in Chapter 6. A systematic sensitivity analysis would be valuable future work.

3.8 Cold-Start from Pre-Trained Reference Models

For appliances without smart-plug coverage, ground-truth labels are never available locally and incremental training is impossible. To support this case, the system accepts user-supplied `model.pt` checkpoints placed in the appliance directory. Upon model loading, the system reads the associated `stats.json` containing the normalisation parameters (μ_{agg} , σ_{agg} , μ_{app} , σ_{app}) and ON/OFF threshold of the source training environment, and uses them as-is for inference.

Two recalibration mechanisms are exposed to mitigate domain shift between the source and target households:

- **Aggregate normalisation recalibration.** The user can recompute μ_{agg} and σ_{agg} from a window of locally collected aggregate readings, replacing the source statistics. This addresses the input-scale component of domain shift without requiring any appliance-level label. As shown in Section 6.1, this recalibration improves energy estimation (SAE) but has limited and uneven effects on per-sample regression accuracy.
- **Threshold recalibration.** The ON/OFF threshold can be adjusted manually by the user based on knowledge of the target appliance’s nominal power range.

If labelled data subsequently becomes available (e.g. via smart plug measurement), the standard fine-tuning or training pipeline applies. Otherwise, the reference model remains in use indefinitely, with the accuracy limitations inherent to cross-household model transfer.

Model selection is the user’s responsibility: no automated verification ensures that the loaded checkpoint corresponds to the appliance type or measurement conditions of the target installation. This is an explicit design choice that prioritises flexibility over guardrails, on the assumption that users deploying a self-hosted NILM system have sufficient technical context to make this judgement.

3.9 Incremental Fine-Tuning

3.9.1 Pipeline Overview

Fine-tuning adapts a pre-trained appliance model to newly collected data without retraining from scratch. The pipeline is identical to the training procedure described

in Section 3.7, with two differences: the model received is the one already loaded in memory rather than a freshly initialised network, and the optimal checkpoint is kept in a memory buffer (a `BytesIO` object) throughout training and reloaded at the end, avoiding unnecessary disk writes during the optimisation loop. Fine-tuning-specific hyperparameters are listed in Table 3.2.

Table 3.2: Fine-tuning hyperparameters that differ from training defaults.

Hyperparameter	Value	Details
Learning rate	2×10^{-5}	Low rate to avoid overwriting trained features
Max epochs	30	Fewer epochs; new data batches are small
Early stopping patience	10	Tighter budget on small incremental batches

3.9.2 Cursor Mechanism

To ensure that each fine-tuning run processes only data collected since the previous one, the system maintains a timestamp cursor per appliance. The cursor resolves to the timestamp of the last fine-tuning run if one has occurred, to the timestamp of the original training run otherwise, and to zero if neither is available, in which case all stored data is used. At the end of each run, the cursor is updated to the maximum timestamp of the processed data, so successive fine-tuning calls consume strictly new observations.

3.9.3 Reversibility and Backup Strategy

Before overwriting the model checkpoint, the current `model.pt` is copied to `model_backup.pt`. This backup is written only on the first fine-tuning run and never overwritten thereafter, preserving the original trained state regardless of how many fine-tuning cycles follow. Reverting to the original model restores the backup checkpoint, clears the fine-tuning metadata from `stats.json`, and immediately reloads the restored weights into the active predictor. If no backup exists, the revert operation is rejected.

Before fine-tuning starts, the currently served appliance model and its statistics are deep-copied under the manager lock, and all optimisation is then performed on this private copy. This prevents a partially updated model from being visible to inference requests. The copy is created per appliance and fine-tuning proceeds sequentially, so the temporary memory overhead is bounded to one additional appliance model at a time.

3.10 Inference

At inference time, the *NILMManager* receives a window of $T = 518$ aggregate power samples spaced 5 seconds apart from the API layer and delegates to the *NILMPredictor*. All appliances are processed sequentially within the same call, with gradient tracking disabled. For each appliance, the pipeline proceeds as follows.

The `/prediction` endpoint exposes a high-level interface : the client submits a window of aggregate samples and receives one power estimate per appliance. Internal classification, hard gating, and denormalisation are encapsulated server-side, so the integration client treats the backend as a pure regression service.

```
POST /prediction
{
  "power_window": [184.2, 187.5, 191.0, ..., 312.7]  // 518 samples
}

Response:
{ "appliances":
  "kettle":      0.0,
  "fridge":      112.3,
  "washing_machine": 0.0
}
```

Normalisation. The raw aggregate window is standardised using the appliance-specific statistics ($\mu_{\text{agg}}, \sigma_{\text{agg}}$) stored in `stats.json`. Using per-appliance normalisation rather than a global statistic allows each model to remain compatible with its own training distribution even when multiple models with different provenance (trained, cold-started) coexist in the same general model.

Forward pass. The normalised window is passed through the *TCN*, which returns a power estimate $\hat{y}$ and a classification logit at the last causal timestep.

Gated output. The ON/OFF state is determined by thresholding $\sigma(\text{logit})$ at $\tau = 0.5$ (configurable). If the appliance is classified as OFF, the reported power is forced to 0 W regardless of the regression output. Otherwise, the denormalised power estimate $\hat{y} \cdot \sigma_{\text{app}} + \mu_{\text{app}}$ is returned, clamped to $[0, +\infty)$.

The result is a dictionary mapping each appliance name to its estimated instantaneous consumption in watts, which the API layer returns to the Home Assistant integration for entity updates.

The compact architecture (approximately 92 000 parameters per appliance model) keeps per appliance-model relatively light.

3.11 Deployment

The backend is packaged as a single *Docker* container whose target platform and PyTorch variant (CPU-only or CUDA) are selected at build time via configuration variables. This makes the same image definition deployable across a range of hardware, from a resource-constrained Raspberry Pi to a Network-Attached Storage (NAS) or desktop machine with GPU acceleration. Model checkpoints, normalisation statistics, and the SQLite database are stored in host-mounted volumes, so all trained state survives container restarts without requiring any retraining or manual intervention.

Integration with Home Assistant is achieved through the backend’s API, consumed locally over the home network without any cloud dependency. The architecture and configuration of this integration are described in detail in Chapter 4.

Chapter 4

Home Assistant Integration

The NILM integration is implemented as a Home Assistant custom component, designed to bridge the core Home Assistant ecosystem with the external NILM backend service described in Chapter 3. Its architecture follows Home Assistant conventions while adding a complete pipeline for data acquisition, model interaction, inference, and user-facing management tools. The present chapter focuses on this architecture ; the end-user workflow, including the configuration steps and screenshots of the management panel, is documented in Appendix B. The full implementation is available at <https://github.com/klaertus/nilm-ha>.

4.1 Architecture

The architecture is organised into clear functional layers.

- **Configuration layer** : guides first-time setup, validates backend connectivity, handles model selection/creation, and stores persistent configuration.
- **Backend communication layer**: provides an asynchronous HTTP client abstraction for all model-server endpoints (status, models, prediction, training, data upload, appliance management).
- **Data and orchestration layer**: maintains the rolling power window, resamples recorder history, runs periodic inference through a coordinator, normalises predictions, and schedules historical data pushes.
- **Entity layer**: exposes NILM outputs as native Home Assistant entities (appliance power sensors, appliance state binary sensors, aggregate and predicted hub sensors), including dynamic entity discovery.

- **API/View layer:** exposes Home Assistant-side HTTP endpoints (`/api/nilm/...`) used by the frontend and service workflows.
- **Frontend layer:** provides a custom management panel for model operations, appliance settings, labeling workflows, and manual data actions.
- **Persistence layer:** stores user-defined labels and entity links, and reuses this metadata to assemble supervised datasets sent to the backend.

A key architectural choice is the central use of shared domain state plus a single *DataUpdateCoordinator*, which ensures consistent data propagation across entities and avoids redundant polling. This provides a scalable integration model: backend calls are centralised, entity updates remain coherent, and UI/API features can reuse the same internal state and services.

Overall, the component architecture combines Home Assistant-native patterns (config entries, entities, coordinator, services) with a modular backend-facing design, enabling both real-time NILM inference and interactive model lifecycle management from within the Home Assistant environment.

4.1.1 Configuration Flow

During the first installation of the component, the user must configure the integration through what Home Assistant calls a *config flow*. This standardised mechanism guides the user through a series of configuration steps, validates inputs at each step, and ultimately creates a persistent *config entry* that survives Home Assistant restarts. The flow consists of three main steps. The full procedure is summarised in Algorithm 1.

The first step collects the backend connection parameters : the host address, the port, the aggregate power entity to monitor, the prediction interval, and the data push interval. Upon submission, the flow immediately attempts a connection to the backend to verify reachability. If the connection fails, the user is redirected to the same form with an error message. Only after a successful connection does the flow advance.

The second step concerns model selection. The user can either select an existing model from those already present on the backend, or create a new one by providing a name. If no model exists yet, the selection step is skipped and the user is taken directly to the name input form.

The third step allows the user to choose which appliances to activate among those already trained in the selected model. If the model contains no trained

appliances yet, this step is skipped and the config entry is created immediately with an empty appliance list.

Algorithm 1 Home Assistant configuration flow

Require: User initiates the integration setup

Ensure: A persistent config entry is created

Step 1 — Backend connection

```
1: repeat  
2:   User submits (host, port, aggregate_entity, prediction_interval,  
   push_interval)  
3:    $ok \leftarrow \text{GETSTATUS}(host, port)$   
4: until  $ok$ 
```

Step 2 — Model selection

```
5:  $models \leftarrow \text{LISTMODELS}()$   
6: if  $models = \emptyset$  then  
7:   User provides a new model name  
8: else  
9:   User picks an existing model or a new name  
10: end if  
11: if new name then  
12:   Validate  $\text{ISSAFE\_NAME}(model\_name)$ ; re-prompt on failure  
13: end if  
14: repeat  
15:    $ok \leftarrow \text{LOADMODEL}(model\_name)$   
16:   if  $\neg ok$  then  
17:     Re-display Step 2 with error  
18:   end if  
19: until  $ok$   
20:  $(sampling\_rate, window\_size) \leftarrow \text{FETCHBACKENDMETADATA}()$ 
```

Step 3 — Appliance activation

```
21:  $appliances \leftarrow \text{LISTREGISTEREDAPPLIANCES}(model\_name)$   
22: if  $appliances = \emptyset$  then  
23:    $active \leftarrow \emptyset$   
24: else  
25:   User deselects unwanted appliances (default: all enabled)  
26:    $active \leftarrow \text{user selection}$   
27: end if
```

```
28:  $\text{CREATECONFIGENTRY}(host, port, \dots, model\_name, active)$ 
```

4.1.2 Entity Model

In Home Assistant, an *entity* is a software abstraction representing an object, a service, or a piece of information within the home automation system. It constitutes the fundamental unit of modelling and interaction. Each entity is uniquely identified, holds a state at any given moment, and may carry a set of attributes describing additional properties. Entities thus provide a standardised way to represent and interact with the various components integrated into the platform [58].

A *sensor* is a particular type of entity whose sole role is to expose a measurement or a piece of information (temperature, humidity, battery level, power consumption, etc.). Unlike other entity types, it does not allow direct actuation; it only surfaces data that can be consumed by dashboards or automations [59].

Each appliance detected by the component is represented in Home Assistant by two entities:

- A **power sensor**, which exposes the estimated power consumption of the appliance at time t , expressed in watts.
- A **binary sensor**, which exposes the estimated on/off state of the appliance. This state is derived locally from the power sensor as $\text{state} = 1$ if $p > 0$, 0 otherwise. The backend already applies the classification head of the model to gate the power output (Section 3.10), so a non-zero predicted power directly reflects an ON decision and no additional thresholding is required at the entity level.

Both entity types subscribe to the coordinator and are updated dynamically as new appliances are added or removed, without requiring a component restart.

In addition to the per-appliance entities, a central device named *NILM Hub* is created during initialisation, grouping all appliance entities visually in the Home Assistant interface. The Hub exposes two additional sensors. The first is a real-time mirror of the configured *aggregate power entity*; unlike the appliance sensors, it subscribes directly to state change events rather than the prediction polling cycle, giving it immediate reactivity. The second reports the *sum of all estimated appliance powers*, allowing the measured aggregate consumption to be compared against the total reconstructed by the model (if all appliances are available).

4.1.3 Data Pipeline

The Coordinator

In Home Assistant, the `DataUpdateCoordinator` is a technical component used by integrations to centralize and optimize data retrieval from an API, a device, or an external service.

Its primary role is to prevent each entity from querying the data source independently. Instead, the coordinator performs a single periodic update (for example every 30 seconds or every 5 minutes) and then distributes the retrieved data to all related entities. This reduces network load, improves performance, and ensures data consistency across entities [60].

The coordinator's role is to maintain a fresh power window and trigger predictions at the configured scan interval. A persistent state-change listener subscribes to the configured aggregate power entity and appends every reported value to an in-memory raw-event buffer trimmed to a `window_size` $\times$ `freq_sample` horizon. At each scan interval, the coordinator resamples this buffer onto a uniform grid via zero-order-hold (forward-fill), consistent with the recorder's change-only logging semantics and submits the resulting `window_size` samples to the backend's `POST /prediction` endpoint. The response is post-processed: the on/off state of each appliance is derived locally as `state = (power > 0)`. The backend already zeroes the power of OFF appliances through its own classifier head (sigmoid output thresholded against the global `on_probability_threshold`), so a non-zero predicted power unambiguously means the appliance is ON. No per-appliance threshold cache is maintained on the HA side, keeping the classification logic in a single place. The resulting `{name  $\rightarrow$  {power, state}}` dictionary, optionally filtered to the appliances the user enabled at setup, is the data distributed to all entities subscribed to the coordinator.

History Pre-fill on Startup

A particular case arises when the component starts up : the buffer is initially empty, which prevents any prediction from being issued until it is progressively filled by successive coordinator cycles. Depending on the configuration, this delay can be significant; in this context, with a scan interval of 5 seconds and a `window_size` of 518, the buffer would take approximately 43 minutes to fill naturally.

To avoid this delay, the `prefill_buffer_from_history` function is called once during initialization, before the first coordinator cycle. Its goal is to query the recorder over the preceding `window_size` $\times$ `freq_sample` seconds. Resampling

is performed via forward-fill, consistent with the recorder’s change-only logging semantics. The buffer is then handed to the state-change listener for further updates. As a result, the component is fully operational and able to emit predictions from the very first coordinator cycle (if there is any data from before).

Historical Data Export to the Backend

Besides real-time inference, the integration periodically exports historical measurements from Home Assistant to the NILM backend. This is necessary because Home Assistant’s recorder retains detailed sensor history for a limited period only (10 days by default), which is insufficient for training or fine-tuning a model [61]. By regularly pushing this data to the backend, the system progressively builds a long-term dataset on which training and fine-tuning operations can be performed.

The export pipeline first loads the user label mapping and extracts the appliance-to-entity links. Only appliances explicitly linked to a Home Assistant entity are included, which avoids sending ambiguous or unlabeled channels. Given a `[start, end]` interval, the component queries the Home Assistant recorder for all relevant entities and converts each state history into a numeric time series, discarding invalid values in the process.

Once assembled, the dataset is transmitted to the backend. The same mechanism is exposed through three entry points : periodic automatic push, manual service call, and frontend-triggered upload. This design unifies the data export logic while supporting both unattended background collection and user-driven uploads.

4.1.4 Backend Communication and API Layer

The communication between the Home Assistant component and the NILM backend is handled by two distinct layers : an internal HTTP client and a set of HA-side API views.

The internal client exposes one method per backend endpoint (model management, prediction, training, fine-tuning, data upload, appliance CRUD). Each method defines its own timeout appropriate to the expected response time of the operation.

The HA-side views serve the frontend panel and act as proxy endpoints toward the backend. Beyond simple proxying, some views enrich the response with local HA state. For instance, the devices view aggregates the coordinator’s latest predictions, the user-defined appliance labels, and the current aggregate power reading into a unified response. Others trigger side effects on success, such as dynamically registering or removing HA entities when an appliance is added or deleted.

The current prototype does not implement authentication between the Home Assistant integration and the local backend. While this is consistent with the trust assumption commonly made in self-hosted home automation deployments, it is a known weakness given the privacy-sensitive nature of disaggregated consumption data. Adding token-based authentication to the backend API is a straightforward engineering task that should be addressed before any production deployment.

4.1.5 Persistence and User Labels

The NILM component maintains a lightweight persistence layer to store user-defined metadata for each appliance. This data is kept in a JSON file located in Home Assistant’s `.storage` directory, mapping each appliance identifier to an optional display name and an optional linked HA entity identifier.

Read and write access is provided by asynchronous wrappers around synchronous I/O functions to avoid blocking the event loop. Labels are read on every prediction cycle to build the full appliance list, and written whenever the user links, unlinks, or renames an appliance. When the config entry is removed, the labels file is deleted to ensure no stale data is left behind.

4.2 End-to-End Validation

The integration was validated through manual end-to-end testing on a personal Home Assistant deployment running on a Raspberry Pi 4. The aggregate household power was measured by a Shelly Pro EM energy meter installed at the main electrical panel, while two individual appliances (an LED lamp and a desk fan) were instrumented with Antela smart plugs providing per-appliance ground truth. Both the aggregate meter and the smart plugs were exposed to Home Assistant as power sensors, and the smart plugs were declared as labelled appliance sources through the integration’s management panel. This deployment exercised the full operational chain documented in Appendix B (configuration flow, data ingestion, dashboard display, threshold-based state derivation, and management panel workflows).

Custom appliance training. To validate the model training pipeline beyond the standardised appliance set studied in Chapter 6, two custom appliance models were trained on smart plug recordings collected over four days in the same household. The state-derivation thresholds were configured at 15 W for the fan and 4 W for the lamp, reflecting their respective baseline-versus-active power gaps. The resulting training datasets contained 17 distinct activation events for the fan ($n_{\text{on}} = 20\,563$

samples, $n_{\text{off}} = 38\,909$ samples) and 12 events for the lamp ($n_{\text{on}} = 19\,203$ samples, $n_{\text{off}} = 40\,269$ samples).

Live disaggregation. Figure 4.1 compares the ground-truth power recordings of the two instrumented appliances, displayed alongside the household aggregate, with the per-appliance estimates produced by the deployed NILM service. The upper panel shows the smart-plug measurements of the fan (yellow) and the lamp (blue), together with the aggregate signal from the Shelly Pro EM (cyan). The lower panel shows the disaggregated power estimates emitted by the NILM service and ingested by Home Assistant as virtual sensors.



Figure 4.1: End-to-end qualitative validation in Home Assistant. **Top :** ground-truth power recordings from the smart plugs on the fan (yellow) and lamp (blue), and the household aggregate signal from the Shelly Pro EM (cyan). **Bottom :** per-appliance disaggregation estimates produced by the deployed NILM service and ingested as Home Assistant sensors.

The temporal structure of the activations is correctly recovered for both appliances : the fan is detected on its three ON periods, and the lamp follows the ground-truth pattern over the entire interval. Two characteristic features of the causal dual-head architecture are visible. First, a reaction lag of approximately two minutes at activation events, corresponding to the time required for the activation to populate a sufficient fraction of the receptive field for the state classifier to consistently exceed its threshold. Second, a transient unstable phase immediately after activation, during which the gate output flickers around the threshold and

the regression head produces noisy power estimates due to the still sparse on-state evidence in the window. Both effects subside, after which the prediction stabilises.

Scope of this validation. This evaluation is qualitative and intended to confirm that the full deployment chain operates correctly in a live environment, including with custom-trained appliance models outside the standardised evaluation set of Chapter 6. The quantitative performance of the underlying model is established on the UK-DALE and ECO benchmarks rather than on this personal deployment, since the latter does not provide a comparable reference for the literature. Automated unit and integration testing of the codebase is acknowledged as a gap and left to future work.

Chapter 5

Experimental Setup

This chapter defines the experimental framework used to evaluate the proposed system. It first specifies the datasets and preprocessing pipeline, the appliances retained for evaluation, and the metrics against which performance is measured, including representative reference figures from the prior NILM literature (Section 5.5). It then defines four experimental protocols, each targeting a distinct question raised by residential deployment.

The protocols are ordered to follow the decision a household actually faces. The cross-house protocol (Section 5.6) measures how far a model trained on another home can be reused with no local supervision, isolating domain shift as the sole source of degradation. The in-house protocol (Section 5.7) instead trains directly on the target household and quantifies the sensitivity of accuracy to the two parameters a deployment can control : the amount of data collected and the smart-plug to mains synchronisation delay. The simulated fine-tuning protocol (Section 5.8) addresses the intermediate case, in which a pre-trained model is adapted to a new appliance signature from a short target-house sample. Finally, the computational cost protocol (Section 5.9) verifies that the resulting models remain trainable and executable on consumer hardware.

5.1 Dataset Selection

The experimental evaluation uses the two datasets identified in Section 2.3: UK-DALE and ECO. Both are supported by NILMTK, ensuring a reproducible and unified data loading pipeline across all experiments.

Two additional candidates were considered but excluded on protocol-level grounds. GREEND, despite its European origin, provides only approximately 30

days of usable recordings per household in the publicly available version, insufficient for the combined 28-day training plus 7-day evaluation protocol adopted in this work. REFIT was excluded because its 8 s sampling period requires more aggressive imputation when resampled to the 5 s target grid: approximately 37% of the resampled bins would be filled by forward-fill, against only 17% for UK-DALE’s 6 s period. This higher proportion of synthetic samples introduces a step-function artefact frequency comparable to the duration of brief transients, degrading the model’s ability to learn fast appliance dynamics.

Table 5.1: Households selected for experimental evaluation. ECO House 1 provides sub-metered usable data only for the kettle.

Dataset	House	Appliances	Role
UK-DALE	House 1	fridge, kettle, washing machine, dishwasher, laptop	in-house, fine-tuning, cross-house
UK-DALE	House 2	fridge, kettle, washing machine, dishwasher, laptop	in-house, fine-tuning, cross-house
ECO	House 1	kettle	cross-house

Households were selected based on the availability of the target appliance categories defined in Section 5.3, the presence of overlapping households within the same dataset to enable cross-house evaluation, and sufficient recording duration to support the training windows and the validation windows used in the experiments.

5.2 Data Preprocessing

5.2.1 Load and Sampling

All datasets are loaded via NILMTK (see Section 2.8.1). Aggregate mains and per-appliance channels are extracted and resampled to a common 5 s grid (0.2 Hz) using NILMTK’s standard `Resample` preprocessor. The resampling combines 5 s bin-mean aggregation (`pandas.resample('5S').mean()`) with forward-fill imputation of empty bins, which arise when the source sampling rate is lower than the target rate. This is equivalent to a step-function reconstruction of the signal between consecutive readings, consistent with the assumption that appliance power consumption is approximately constant on sub-sampling-period timescales. For ECO households, the three-phase mains sub-meters are summed prior to resampling to obtain total household active power. All experiments use the real mains aggregate rather than a synthetic appliance sum.

5.2.2 Window selection

The temporal segments listed in Table 5.2 were selected to ensure a minimum number of activations per appliance in each of the train, finetune and test windows. This selection ensures that no experimental condition degenerates to a trivial always-OFF prediction, but it also means that the reported metrics correspond to households evaluated on activation-rich periods rather than on arbitrary calendar windows. Shang et al. [40] have shown that NILM disaggregation accuracy can vary by a factor of two depending on the chosen training window for a given household; the figures reported here should therefore be interpreted as representative of the system’s performance under favourable activation density, not as upper bounds achievable on any 28-day window of the same household.

Table 5.2: Temporal segments used per household. All date ranges are inclusive on the start and exclusive on the end (e.g., "11 Sep–09 Oct" denotes the half-open interval [11 Sep 00:00, 09 Oct 00:00), strictly non-overlapping with the test window beginning at 09 Oct 00:00). The first column gives the training or calibration window and the held-out test window used for the source/in-house and cross-house protocols. The second column gives the fine-tuning pool and the held-out test segment used for the fine-tuning protocol.

House	Train/Calibration and Test (28 d + 7 d)	Fine-tuning and Test (14 d + 7 d)
UK-DALE H1	Train/Calibration: 11 Sep–09 Oct 2014 Test: 09–16 Oct 2014	Fine-tune: 13–27 Nov 2014 Test: 27 Nov–04 Dec 2014
UK-DALE H2	Train/Calibration: 21 May–18 Jun 2013 Test: 18–25 Jun 2013	Fine-tune: 11–25 Jul 2013 Test: 25 Jul–01 Aug 2013
ECO H1	Train/Calibration: 01–29 Nov 2012 Test: 29 Nov–06 Dec 2012	–

5.3 Appliance Selection

Five appliances are retained for evaluation: kettle, washing machine, dishwasher, fridge, and laptop. The selection covers three of the four signature types defined in Section 2.2 and reflects both the diversity of disaggregation challenges and the availability of sub-metered ground-truth in UK-DALE and ECO.

The **kettle** is the canonical Type-I appliance, with sharp ON/OFF transitions and short activation duration. It serves as a tractable baseline against which more complex signatures can be contrasted.

The **washing machine** and **dishwasher** are Type-II appliances cycling through programmed phases of variable duration. Their multi-state signatures and infrequent activations make them representative of the central disaggregation challenge in residential NILM.

The **fridge** is also classified as Type-II in this work (see Section 2.2), with two thermostat-controlled states. Unlike washing machines and dishwashers, its activations are continuous throughout the recording period and not user-driven, providing a complementary evaluation regime: the abundance of state transitions favours classifier training, while the low compressor power amplitude tests the regression head’s calibration on small signals.

The **laptop** is the only Type-III appliance retained. Its consumption varies continuously with workload, with no well-defined state structure. It is included specifically to characterise the behaviour of the architecture on a signature type that violates the discrete-state assumption underlying most NILM benchmarks.

Type-IV permanently active appliances (routers, smoke detectors, landline equipment) are not retained. Their near-constant baseline behaviour offers no informative temporal structure for a sequence modelling architecture and is more appropriately handled by static baseline subtraction at the aggregate level.

Low-power devices (lamps, smartphone chargers, small networking equipment) are excluded from this evaluation, as their consumption typically falls below the noise floor of aggregate measurements, making robust disaggregation infeasible without specialised techniques [5]. Furthermore, such devices contribute marginally to total household energy consumption. A possible extension, not explored here, would be to aggregate them under a single virtual meter, for instance by grouping all lighting circuits into a single lamps meter.

5.4 Evaluation Metrics

Disaggregation performance is inherently multi-dimensional: a model may reproduce instantaneous power levels accurately yet systematically misestimate total energy consumption, or it may track energy budgets well while failing to detect appliance state transitions. The evaluation protocol therefore combines regression, energy, and classification metrics, each capturing a distinct failure mode.

5.4.1 Evaluation Procedure

All experiments reported in Chapter 6 are executed against the deployed backend through its public HTTP API. Training data is pushed through `POST /data`,

training is triggered through `POST /train`, inference through `POST /prediction`, and fine-tuning through `POST /finetune`. The experimental protocol therefore exercises the same code paths invoked by the Home Assistant integration (Chapter 4) in a residential deployment, with the public datasets standing in for the live aggregate and smart-plug streams. The reported accuracy and latency figures correspond to the deployed system’s actual behaviour rather than to a separate research-only implementation.

Three aspects of a live residential deployment are not exercised by this protocol: data acquisition variability introduced by the Home Assistant recorder (event-based logging gaps, intermittent sensor unavailability), long-term operation across appliance brand changes and seasonal drift, and sustained multi-appliance concurrent load on the inference path. These are addressed only by the qualitative end-to-end validation of Section 4.2.

The model is queried through a sliding window over the held-out evaluation period. At each step, a window of $W = 518$ samples (approximately 43 minutes at 5 s resolution) is extracted from the aggregate mains signal and submitted to the `/prediction` endpoint. The predicted active power for each appliance corresponds to the *last* sample of that window, i.e. the model produces a point estimate $\hat{p}_t$ for time step $t = \text{start} + W - 1$. The window is then advanced by a stride of $s = 6$ samples (30 seconds), yielding a dense prediction sequence that covers the entire evaluation window. A stride of six rather than one reduces evaluation cost by a factor of six without materially affecting the reported metrics: the appliance dynamics of interest in this work evolve on timescales of tens of seconds or more (compressor cycles, washing-machine phases, kettle activations of one to three minutes), well above the resulting 30 seconds prediction interval. Ground-truth values g_t are taken from the corresponding smart-plug recordings, aligned to the same timestamps.

Let $\hat{\mathbf{p}} = (\hat{p}_1, \dots, \hat{p}_N)$ and $\mathbf{g} = (g_1, \dots, g_N)$ denote the predicted and ground-truth power sequences for a given appliance over N evaluation steps.

5.4.2 Regression Metrics

Mean Absolute Error (MAE) MAE measures the average magnitude of the instantaneous prediction error in watts:

$$\text{MAE} = \frac{1}{N} \sum_{t=1}^N |\hat{p}_t - g_t|. \quad (5.1)$$

MAE is reported as the primary regression metric throughout this work. It is interpretable directly in watts and is robust to outliers relative to squared-error

alternatives.

Root Mean Square Error (RMSE) RMSE penalises large deviations more heavily than MAE:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{t=1}^N (\hat{p}_t - g_t)^2}. \quad (5.2)$$

RMSE is computed for completeness and is reported in the detailed per-appliance tables; however, MAE is preferred for cross-condition comparisons as it is less sensitive to sporadic large errors caused by brief power spikes that fall outside the model’s receptive field.

5.4.3 Signal Aggregate Error (SAE)

SAE measures the relative error in total energy consumption estimated over the evaluation period [11]:

$$\text{SAE} = \frac{\left| \sum_{t=1}^N \hat{p}_t - \sum_{t=1}^N g_t \right|}{\sum_{t=1}^N g_t + \varepsilon}, \quad (5.3)$$

where $\varepsilon = 10^{-8}$ guards against division by zero for appliances that are never active during the evaluation window. SAE equals zero when total predicted energy matches total ground-truth energy exactly, and grows without bound as the discrepancy increases. A value of 0.1, for instance, indicates that the model over- or underestimates the appliance’s total energy consumption by 10 % over the evaluation period.

SAE is particularly informative in the context of smart-home energy monitoring, where users and utility providers care more about cumulative consumption figures than about instantaneous accuracy at every sample.

5.4.4 State-Based Classification Metrics

For appliances with low duty cycles, MAE can be misleading: a model that systematically predicts 0 W achieves an artificially low error while completely missing the appliance, a qualitatively different failure mode that regression metrics fail to expose. Classification metrics address this by binarising both the predicted and ground-truth sequences using a per-appliance power threshold τ_a :

$$\hat{y}_t = \mathbf{1}[\hat{p}_t \geq \tau_a], \quad y_t = \mathbf{1}[g_t \geq \tau_a]. \quad (5.4)$$

The threshold values τ_a (Table 5.3) follow the standard NILM convention [22, 11] and are used only for the evaluation reported in this chapter; they are not exposed in the Home Assistant integration (Chapter 4).

Table 5.3: Per-appliance on-power thresholds τ_a , set conservatively above the standby/baseline level so as to reliably detect any genuine ON state. For Type-I appliances (kettle), the threshold is set sufficiently below the steady-state ON power to capture full activations. For Type-II appliances with multiple internal states (washing machine, dishwasher), the threshold is set above the lowest active phase to include all on-states without mistaking standby noise for activity. Thresholds are fixed prior to evaluation by visual inspection of the training-house signals and never tuned post-hoc.

Appliance	τ_a (W)
Fridge	20
Kettle	500
Washing machine	10
Dishwasher	10
Laptop	10

From the binary sequences, the standard confusion matrix quantities are derived: true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN).

Precision and Recall.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \quad (5.5)$$

Precision quantifies the fraction of predicted on-states that correspond to a genuine activation; recall quantifies the fraction of true activations that are detected. Both are set to zero when their denominators vanish (i.e. the appliance is never predicted to be on, or never truly on, respectively).

F1 Score The F1 score is the harmonic mean of precision and recall:

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (5.6)$$

$F_1 = 0$ when either precision or recall is zero; $F_1 = 1$ indicates perfect detection. F1 is the primary classification metric reported in this work. Beyond evaluating

activation detection quality, it additionally serves as a diagnostic tool for validating the per-appliance threshold τ_a , which is used internally by the training pipeline to separate ON and OFF samples during data resampling. To prevent post-hoc threshold optimisation from artificially inflating results, all thresholds are fixed prior to evaluation and never adjusted to maximise classification performance [41].

5.5 Reference Results from Prior Work

For the reader to interpret the absolute error figures reported in the remainder of this work, Table 5.4 summarises representative results from the deep-learning NILM literature on UK-DALE, ordered from the first architecture applied to NILM (Kelly and Knottenbelt 2015) to the more recent classification-aware variants. These figures are intended as a reading aid rather than a direct ranking : the goal of the present work is the practical deployment of a NILM system on edge hardware (Section 5.9), not the optimisation of any single regression metric.

5.5.1 Caveats on Direct Comparison

Several protocol differences preclude a strict ranking against the proposed system :

- **Train/test split.** The cited baselines train on UK-DALE houses 1, 3, 4 and 5 and test on house 2 (cross-house, multi-source) ; the in-house results of this work train and test on disjoint windows of the same household, an inherently easier task.
- **Window length and causality.** The proposed model uses a $W = 518$ input window at 5 s resolution and predicts the last sample (causal seq2point). The references use longer windows (599 samples at 6 s) and often predict the midpoint (non-causal). Causal prediction is strictly harder at equal capacity.
- **Aggregate construction.** Conv-NILM-Net [26] uses a synthetic aggregate constructed from selected appliance channels, which removes most of the unknown-load contamination present in real mains and yields systematically lower MAE figures. The other papers do not explain how they calculate the aggregates.
- **State threshold.** F1 depends on the per-appliance on-power threshold τ_a , which varies between papers and is sometimes optimised post-hoc [41]. Cross-paper F1 comparison is therefore unreliable, and only MAE and SAE are reproduced in Table 5.4.

Table 5.4: Mean Absolute Error (MAE, in watts) and Signal Aggregate Error (SAE) reported in the literature for the NILM disaggregation task on UK-DALE house 2, under a train-on-houses-1-3-4-5 / test-on-house-2 cross-house protocol. Seq2seq(Kelly) figures are as re-evaluated by Zhang et al. [11]; Seq2point(Zhang) is from the same paper. SGN-sp values are from Shin et al. [19]. Improved TCN values are from Yang et al. [20], under their causal seq2point configuration on UK-DALE house 2. SAE values are only reported for Zhang et al., as Shin et al. define a time-normalised variant (SAE_{δ}) that is not directly comparable to the energy-normalised SAE used in this work (equation 5.3). The laptop is not included in this table: to the best of our knowledge, no published deep-learning NILM study reports per-appliance results for a laptop on UK-DALE, as the appliance is rarely retained in the standard evaluation subset [11, 19, 22].

Method	Causal	Kettle	Fridge	Dish w.	Wash. m.
<i>MAE (W)</i>					
Seq2seq (Kelly 2015)	no	13.00	38.45	237.96	163.47
Seq2point (Zhang 2018)	no	7.44	20.89	27.70	12.66
SGN-sp (Shin 2019)	no	5.96	15.32	13.49	11.04
Improved TCN (Yang 2021)	yes	4.11	16.38	23.26	9.50
<i>SAE (energy-normalised)</i>					
Seq2seq (Kelly)	no	0.085	0.502	4.237	13.831
Seq2point (Zhang)	no	0.069	0.121	0.645	0.284

5.6 Cross-House Evaluation Protocol

The goal of this experiment is to quantify the performance degradation induced by domain shift when a model trained on one household is deployed in a different home without any target-specific retraining. Two transfer scenarios are evaluated: one within-dataset transfer (UK-DALE House 1 $\rightarrow$ House 2) and one cross-dataset transfer (UK-DALE House 2 $\rightarrow$ ECO House 1), so as to characterise the compounded effect of household and dataset shift.

In all conditions, models are trained on 28 days of source-household data and evaluated on the fixed 7 days eval window from the target household. The smart-plug to mains synchronisation delay is set to 0 seconds throughout, so as to isolate domain shift as the sole source of performance degradation.

Two transfer conditions are compared.

Uncalibrated. The model is applied directly to the target household without modification. Both the model weights and the normalisation statistics (μ_{agg} , σ_{agg} , and per-appliance statistics) are those learned on the source household.

Calibrated. The aggregate normalisation statistics μ_{agg} and σ_{agg} are recomputed from a 28 days of train windows target-household aggregate data, correcting for amplitude differences in the input distribution without modifying the model weights. The calibration window precedes the evaluation window in time and the two are strictly non-overlapping. Per-appliance statistics (μ_a , σ_a) and the threshold τ_a are kept unchanged, as no target-household ground truth is assumed available at deployment time.

Performance is reported using MAE, RMSE, SAE, F1, precision and recall. Per-appliance thresholds τ_a are those determined on the source household and listed in Table 5.3; classification metrics should therefore be interpreted with caution in conditions where target-household power levels differ significantly from the source.

Table 5.5: Cross-house and cross-dataset evaluation scenarios. The cross-dataset transfer (UK-DALE H2 $\rightarrow$ ECO H1) is evaluated on the appliance available in ECO House 1 (kettle); the dishwasher and laptop are excluded from this condition.

Source	Target	Transfer type
UK-DALE H1	UK-DALE H2	Within-dataset
UK-DALE H2	ECO H1	Cross-dataset

5.7 In-House Training Protocol

The goal of this experiment is to characterise the performance of the proposed system under realistic in-house deployment conditions, and to quantify the sensitivity of disaggregation accuracy to two practical parameters: the synchronisation delay between the smart plug and the aggregate meter, and the duration of the training data collection period. All experiments are conducted independently on two households, UK-DALE Houses 1 and 2. In all conditions, the model is evaluated on the fixed 7 days described in Table 5.2.

5.7.1 Effect of Appliance-meter Delay

Smart-plug measurements are subject to a variable transmission delay relative to the aggregate mains signal, induced by wireless protocol overhead, device-side processing, and integration broker buffering. To quantify the impact of this misalignment on disaggregation accuracy, the appliance smart-plug signal is artificially shifted by 0, 5, and 10 seconds relative to the aggregate during training. The first value (0 seconds) serves as a perfect-synchronisation upper-bound reference (assuming that the dataset has a 0-second delay). The second (5 s) reflects the steady-state delay observed on a typical Wi-Fi consumer plug (TP-Link Tapo P110) under normal network conditions. The third (10 seconds) is included as a stress-test condition representative of degraded scenarios such as low end smart meter, congested 2.4 GHz networks, multi-hop Zigbee deployments, or transient broker backpressure. Training duration is fixed at 28 days across all delay conditions.

Table 5.6: Experimental conditions for the appliance-meter delay. Each (delay, dataset, house) configuration is trained on 28 days of smart-plug data and evaluated independently.

Delay (s)	Dataset	House
0, 5, 10	UK-DALE	H1
0, 5, 10	UK-DALE	H2

5.7.2 Effect of Training Duration

To characterise the relationship between the volume of in-home training data and disaggregation accuracy, models are trained on 14 and 28 days of smart-plug

data. These values correspond respectively to an optimistic early deployment scenario, a standard collection period beyond which diminishing returns are expected. Synchronisation delay is fixed at 5 s.

Table 5.7: Experimental conditions for the training duration ablation. Each configuration is evaluated at a fixed synchronisation delay of 5 s, across two households and two training window sizes.

Train days	Dataset	House
14, 28	UK-DALE	H1
14, 28	UK-DALE	H2

5.8 Simulated Finetuning Protocol

The goal of this experiment is to evaluate the system’s ability to adapt to a new appliance instance of the same category but with a different power signature, a scenario referred to as a *brand change*. Two variants are relevant in practice. The first, *intra-house* brand change, occurs when a household replaces an appliance with a functionally equivalent device from a different manufacturer: the home environment is unchanged, but the device under observation has a new signature. The second, *cross-house transfer*, occurs when a pre-trained model is deployed in a new household that contains the same appliance category but has never been seen during training. In this case, both the home environment and the specific device instance differ from the source.

Of these two variants, only the cross-house transfer can be simulated using public NILM datasets, as no publicly available dataset captures the same household before and after appliance replacement. The cross-house transfer is therefore used as a proxy for both scenarios: it constitutes a strict upper bound on the difficulty of intra-house brand change, since it introduces both a device shift and a household shift simultaneously.

Each transfer is evaluated in three stages. First, a model is trained from scratch on the source house, using the full 28 days training split (see Table 5.2). Then, the trained model is fine-tuned on a short window drawn from the target house, which contains the same appliance category but with a distinct power signature. Finally, both the source-trained and the fine-tuned models are evaluated on a fixed 7-day held-out test split of the target house, strictly non-overlapping with the fine-tuning window (see Table 5.2). The synchronisation delay is fixed at 5 s during fine-tuning.

Three fine-tuning durations are evaluated (5, 10 and 14 days) to characterise the trade-off between adaptation data volume and target-house performance.

For each (appliance, source-target) combination, results are reported under two conditions:

1. **Baseline:** the source-trained model is evaluated directly on the target house without any adaptation.
2. **Fine-tuned:** the model undergoes the fine-tuning procedure described in Section 3.9.

Two source-target pairs are evaluated, listed in Table 5.8. These are within-dataset transfer (UK-DALE H1 $\rightarrow$ H2), (UK-DALE H2 $\rightarrow$ H1), isolating the effect of household and device shift under matched recording conditions.

Table 5.8: Source-target pairs for the simulated brand-change scenario. Each pair is evaluated under the two conditions described above (baseline, fine-tuned), and at fine-tuning durations of 5, 10 and 14 days. The smart meter delay is fixed to 5 s during finetuning.

Source	Target	Transfer type
UK-DALE H2	UK-DALE H1	Within-dataset
UK-DALE H1	UK-DALE H2	Within-dataset

5.9 Computational Cost Protocol

The proposed system targets deployment on consumer-grade edge hardware within a Home Assistant environment. This experiment therefore assesses the practical feasibility of the system by measuring training time, inference latency, and memory footprint across three hardware configurations representative of realistic user profiles.

The computational cost evaluation focuses on the deployment constraints of the proposed system. Training is benchmarked across two desktop configurations representing realistic user profiles (with and without a discrete GPU), since the training cost varies with the user’s available hardware and is borne once per appliance during onboarding. Inference, in contrast, is benchmarked on the Raspberry Pi 4 alone : as this platform is the slowest of the hardware ensemble considered, validating real-time inference here is sufficient to establish feasibility on any more

capable deployment target. Measurements on the desktop platforms would not add information to this lower-bound argument and are therefore omitted to keep the analysis focused on the binding deployment constraint.

5.9.1 Hardware Configurations

Desktop GPU. An AMD Ryzen 7 7700X processor paired with an NVIDIA RTX 3060 (12 GB VRAM), representing a user who trains locally on a gaming or workstation PC.

Desktop CPU-only. The same AMD Ryzen 7 7700X without GPU acceleration, representing a user without a discrete GPU.

Edge (Raspberry Pi). A Raspberry Pi 4 (8 GB RAM), which represents the primary deployment target for edge-based Home Assistant installations.

All configurations run Python 3.13.5 with PyTorch 2.9.1; the GPU configuration uses CUDA 12.8, while the CPU and Raspberry Pi configurations use the default PyTorch CPU backend. No hardware-specific quantisation or pruning is applied, so as to reflect out-of-the-box deployment conditions.

5.9.2 Measured Dimensions

Training cost. Wall-clock duration of one training epoch on the 28-day training set, averaged over five consecutive epochs (batch size $B = 1024$ on Desktop PC and batch size $B = 512$ on Raspberry), together with the peak Resident Set Size (RSS) of the training process during those epochs. Reporting per-epoch time rather than to-convergence isolates the compute cost of the architecture from stochastic effects such as early stopping and weight initialisation. Peak training RSS validates that the deployment platform has sufficient memory to perform the training step under the chosen batch size, which is the relevant constraint on the Raspberry Pi 4 with its 8 GB of system memory budget. The reduced batch size doubles the number of optimisation steps per epoch (98 against 49) but preserves the total number of samples processed per epoch (50,000 in both configurations, set by the balanced sampler cap).

Inference latency. Time required to produce a complete disaggregation across all five appliance models for one input window of size $W = 518$ samples. The five models are loaded simultaneously into memory and executed sequentially within

a single inference call, so the per-window latency reported here is the sum of the five individual forward-pass latencies (no batching or parameter sharing across appliances). Each measurement consists of 1000 consecutive inference calls preceded by 50 warmup calls discarded from the statistics. We report the mean, median, p95, p99, maximum, standard deviation, and throughput (windows per second). A new aggregate sample arrives at a 5 s interval and must be disaggregated before the next sample. Four sub-configurations of the Raspberry Pi 4 deployment are evaluated to characterise the impact of network transport and concurrent Home Assistant workload.

Memory and CPU footprint. Peak resident set size (RSS) of the inference process during execution, expressed in megabytes, reported for the multi-appliance deployment configuration in which the five appliance models are loaded simultaneously into memory and executed via five sequential forward passes per call. Both the framework baseline (Python interpreter and PyTorch runtime, no model loaded) and the absolute peak RSS during inference are reported, with the difference giving the application-level contribution to the memory footprint. Model size on disk is reported per appliance (in kilobytes) and as a combined total across the five appliances. On the Raspberry Pi 4 configuration, mean and peak CPU utilisation during inference are additionally reported, since this is the only platform where CPU saturation is a meaningful deployment constraint.

Table 5.9: Hardware configurations evaluated for the computational cost benchmarks, and the measurement type performed on each.

Configuration	Hardware	Measured
Desktop GPU	Ryzen 7700X + RTX 3060 + 32 GB DDR5	Training
Desktop CPU	Ryzen 7700X + 32 GB DDR5	Training
Edge	Raspberry Pi 4 (Cortex-A72) + 8 GB LPDDR4	Training and Inference

Chapter 6

Results and Discussion

6.1 Cross-House Evaluation

This section evaluates the disaggregation performance of models trained on one household and deployed on another without target-specific retraining. Table 6.1 and Table 6.2 report the results, alongside Figures 6.1, 6.2 and 6.3.

Input recalibration improves energy estimation. Recalibrating the input statistics on the target house consistently reduces SAE across the appliance set (Figure 6.2: dishwasher $0.870 \rightarrow 0.752$, kettle $0.287 \rightarrow 0.263$, fridge $0.176 \rightarrow 0.165$, washing machine $0.413 \rightarrow 0.372$), consistent with the role of input normalisation in reducing regression bias on total energy.

Recalibration has limited and uneven effect on regression accuracy. The effect on MAE (Figure 6.1) is smaller than on SAE and goes in either direction. Only the dishwasher improves substantially ($32.2 \rightarrow 28.4$ W, -12%), consistent with its mid-power multi-phase signature being the most sensitive to aggregate distributional shift. The washing machine and laptop are essentially unchanged ($5.4 \rightarrow 5.3$ and $9.1 \rightarrow 9.1$ W), the kettle marginally degrades ($6.1 \rightarrow 6.2$ W), and the fridge degrades by 1.7 W ($17.1 \rightarrow 18.8$ W) : these signatures (the kettle’s ~ 2 kW spike, the fridge’s tight operating range) are robust enough to scale shift that recomputing input statistics adds variance without resolving any exploitable mismatch. F1 follows the same logic, with kettle, fridge, dishwasher and laptop showing degradation or marginal improvement under calibration; only the washing machine F1 stays essentially constant ($0.818 \rightarrow 0.816$).

Cross-house transfer degrades MAE relative to the in-house baseline on every

Table 6.1: Cross-house disaggregation performance when models trained on UK-DALE House 1 are evaluated on UK-DALE House 2. The in-house baseline corresponds to models trained and evaluated on the target house. The uncalibrated setting uses the source normalisation statistics, whereas the calibrated setting recomputes the aggregate input statistics on the target house. MAE and RMSE are expressed in watts. Appliance-meter delay is fixed at 0 s.

Appliance	Setting	MAE	RMSE	SAE	F1	P	R
kettle	In-house baseline	2.0	38.0	0.086	0.974	0.989	0.959
	Uncalibrated	6.1	78.8	0.287	0.882	0.841	0.928
	Calibrated	6.2	79.8	0.263	0.870	0.818	0.928
fridge	In-house baseline	11.0	25.2	0.224	0.965	0.981	0.950
	Uncalibrated	17.1	32.3	0.176	0.889	0.846	0.936
	Calibrated	18.8	33.3	0.165	0.846	0.772	0.936
washing machine	In-house baseline	7.9	74.6	0.136	0.476	0.530	0.432
	Uncalibrated	5.4	54.2	0.413	0.818	0.813	0.824
	Calibrated	5.3	53.3	0.372	0.816	0.801	0.833
dishwasher	In-house baseline	5.5	64.9	0.113	0.739	0.954	0.602
	Uncalibrated	32.2	235.5	0.870	0.303	0.992	0.179
	Calibrated	28.4	214.0	0.752	0.412	0.972	0.261
laptop	In-house baseline	4.9	9.0	0.207	0.747	0.800	0.699
	Uncalibrated	9.1	16.3	0.992	0.016	0.970	0.008
	Calibrated	9.1	16.3	0.974	0.044	0.831	0.023

appliance except the washing machine: kettle $2.0 \rightarrow 6.2$ W, fridge $11.0 \rightarrow 18.8$ W, dishwasher $5.5 \rightarrow 28.4$ W, laptop $4.9 \rightarrow 9.1$ W. Input recalibration alone is therefore insufficient to close the cross-house gap on regression quality. Event detection (F1) follows an amplitude-dependent pattern visible in Table 6.1: high-power signatures (kettle) detect robustly under shift and low-power signatures (laptop) collapse.

The washing machine inversion is the most pronounced of the cross-house results. The H1-source model outperforms the H2 in-house baseline on the target house in regression accuracy (MAE 5.4 W vs. 7.9 W, Figure 6.1) and in event detection by a wide margin (F1 0.818 vs. 0.476, Figure 6.3). The inversion does *not* hold for cumulative energy estimation: SAE is markedly worse under cross-house transfer (0.413–0.372) than for the in-house baseline (0.136, Figure 6.2).

Two complementary mechanisms explain this pattern. First, the washing machine cycle is dominated by physically constrained phases (heating-element

pulses, motor starts, drum rotation, spin) whose temporal structure is largely conserved across brands and houses; the signature transfers well under domain shift, which explains the per-sample regression and event-detection gains. Second, the H2 in-house model is itself weak: its precision of 0.530 and recall of 0.432 both signal that the model misses most activations and fires unreliably on those it catches. The H2 training window contains only 419 activations versus 2,510 in House 1 (Table 6.3), so the in-house detector is undertrained. The H1-source model, exposed to a richer event repertoire, becomes substantially more sensitive (recall 0.824 vs. 0.432, precision 0.813 vs. 0.530). The cost of this higher sensitivity is energy over-estimation: more detected events combined with the systematic overestimation of power magnitudes propagated by the source model produces a cumulative energy error roughly three times larger than the in-house baseline.

The apparent SAE advantage of the H2 baseline should therefore be read with care: it stems in part from error compensation between under-detection and over-estimation on the events it does catch, rather than from genuinely better energy estimation. The overall finding is unchanged: event count during training is a stronger predictor of detection and per-sample regression quality than calendar duration or geographic proximity of the training house (Section 6.2.1), at the cost of an energy calibration trade-off that practical deployments must manage.

The laptop failure on House 1 is rooted in a within-window distribution shift, not in cross-house transfer. The cross-house laptop model is essentially unusable (MAE 9.1 W vs. in-house baseline 4.9 W, SAE 0.974, F1 0.044, Figure 6.1). The H1 in-house baseline at 28 days (Table 6.3) performs no better, ruling out cross-house shift as the cause. The bottleneck is a mismatch between the H1 training and evaluation windows: training contains 57 long bouts (mean 105 min, 39 W mean power), but the test window contains 278 short bouts (mean 10.5 min, 29 W). The source model learns a stable, infrequent, high-power signature and is then evaluated on rapid low-power pulses outside its training distribution; the high SAE confirms that predictions collapse to zero across the test period. This within-H1 distribution shift is structural and is not resolved by any protocol evaluated in this work: fine-tuning on H1 data (Section 6.3) draws from the same training window and therefore inherits the same mismatch with the test period. The H1 28-day in-house baseline (Table 6.3) confirms this: doubling the training window from 14 to 28 days adds activations of the same long-bout regime and metrics improves negligibly or not at all. Recovery would require an H1 training window whose bout-length distribution matches the test period, or a sampling strategy that explicitly balances bout-length regimes.

The cross-dataset transfer from UK-DALE House 2 to ECO House 1 is reported

separately in Table 6.2, restricted to the kettle: it is the only appliance with sufficient activation density and submeter consistency in ECO House 1.

Table 6.2: Cross-house disaggregation performance when the UK-DALE House 2 kettle model is evaluated on ECO House 1. Appliance-meter delay is fixed at 0 s.

Setting	MAE	RMSE	SAE	F1	P	R
In-house baseline	15.7	133.5	0.368	0.309	0.319	0.301
Uncalibrated	18.2	185.4	0.457	0.039	0.062	0.028
Calibrated	15.3	168.5	0.698	0.045	0.121	0.028

The UK-DALE H2 to ECO H1 kettle transfer fails at the appliance level.

The calibrated MAE of 15.3 W is marginally below the in-house baseline (15.7 W), but this reflects the kettle’s above 95% OFF duty cycle rather than disaggregation quality. SAE confirms the failure (0.368 baseline $\rightarrow$ 0.457 uncalibrated $\rightarrow$ 0.698 calibrated): calibration actively worsens energy estimation by shrinking the magnitude of mostly false-positive detections. Event detection collapses (F1 0.309 $\rightarrow$ 0.039/0.045, R = 0.028), missing roughly 97% of the 19 kettle activations in the 7-day test window. The failure is overdetermined: the ECO H1 kettle operates at 1 730 W (vs. 2 947 W in source UK-DALE H2), a 1.2 kW gap that calibration cannot bridge; the three-phase ECO aggregate peaks at 8 825 W, diluting the signal further; and the in-house baseline (F1 = 0.309) confirms the kettle is hard to detect in this dataset even without transfer.

Practical implication for Home Assistant deployment. Cross-house transfer degrades performance on every in-house appliance evaluated except the washing machine. The washing machine inversion is not driven by inherent signature transferability but by the much larger event count, as discussed above. Even high-amplitude signatures sometimes assumed robust to domain shift, such as the kettle, see their MAE multiplied by approximately three under transfer (2.0 $\rightarrow$ 6.2 W). Input recalibration alone does not close this gap on any appliance.

The recommended deployment strategy depends on the availability of labeled data for the target home. When reference data from smart plugs can be collected on the target device, training on the target home or fine-tuning a source model is the preferred strategy (see section 6.3 and section 6.4.2). When smart plug coverage is not possible, typically for built-in appliances such as ovens, wired heating, transfer can serve as a fallback solution for a cold start, with the accuracy trade-offs mentioned above. Note that in such cases, pre-training on a dataset resembling the target home is necessary to reduce domain shift. Transfer between

datasets (different countries, measurement architecture, or sampling protocols) is unreliable in all cases and should not be deployed without adaptation to the target.

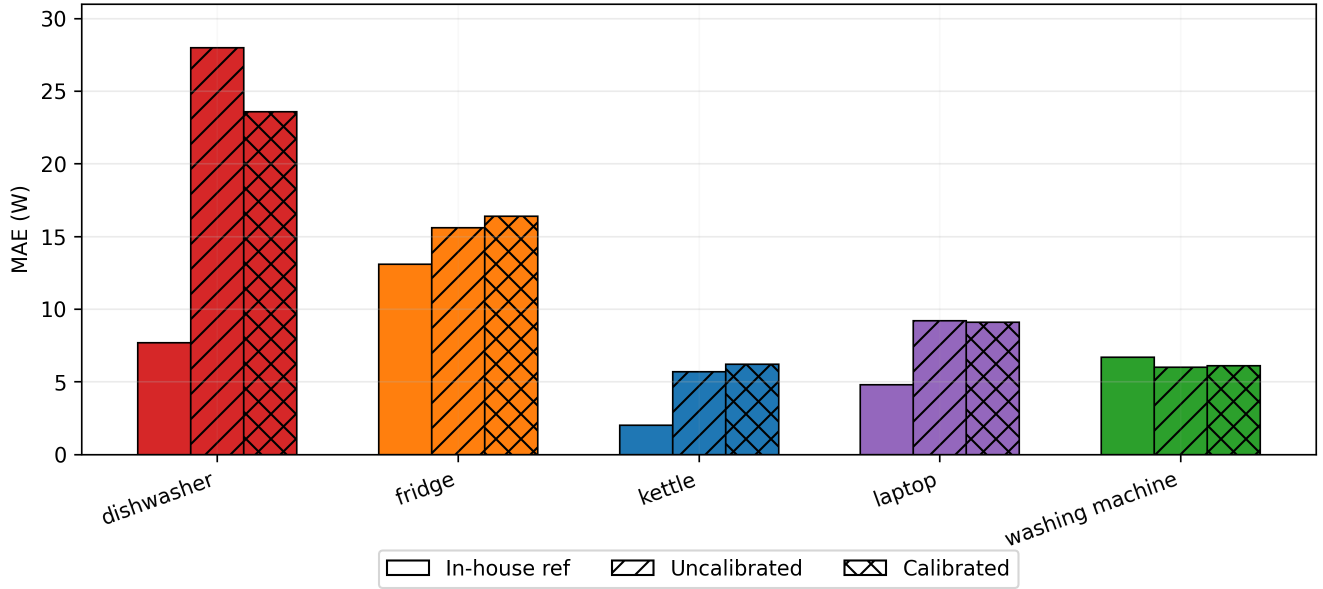


Figure 6.1: MAE comparison for cross-house transfer from UK-DALE House 1 to House 2.

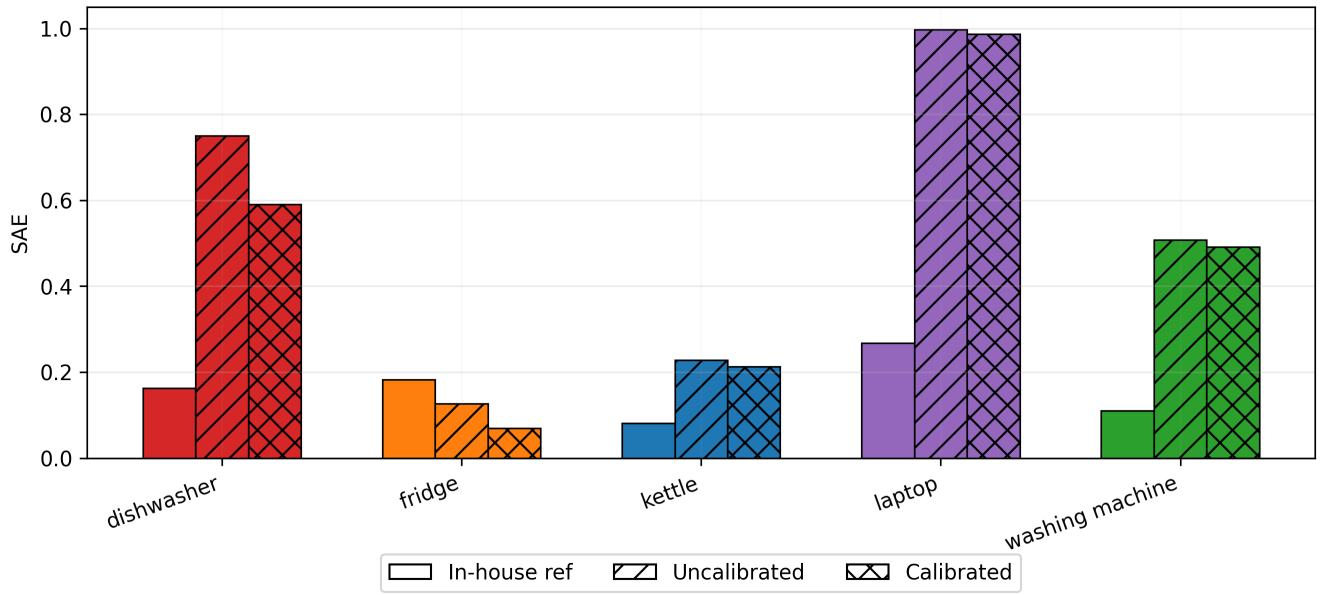


Figure 6.2: SAE comparison for cross-house transfer from UK-DALE House 1 to House 2.

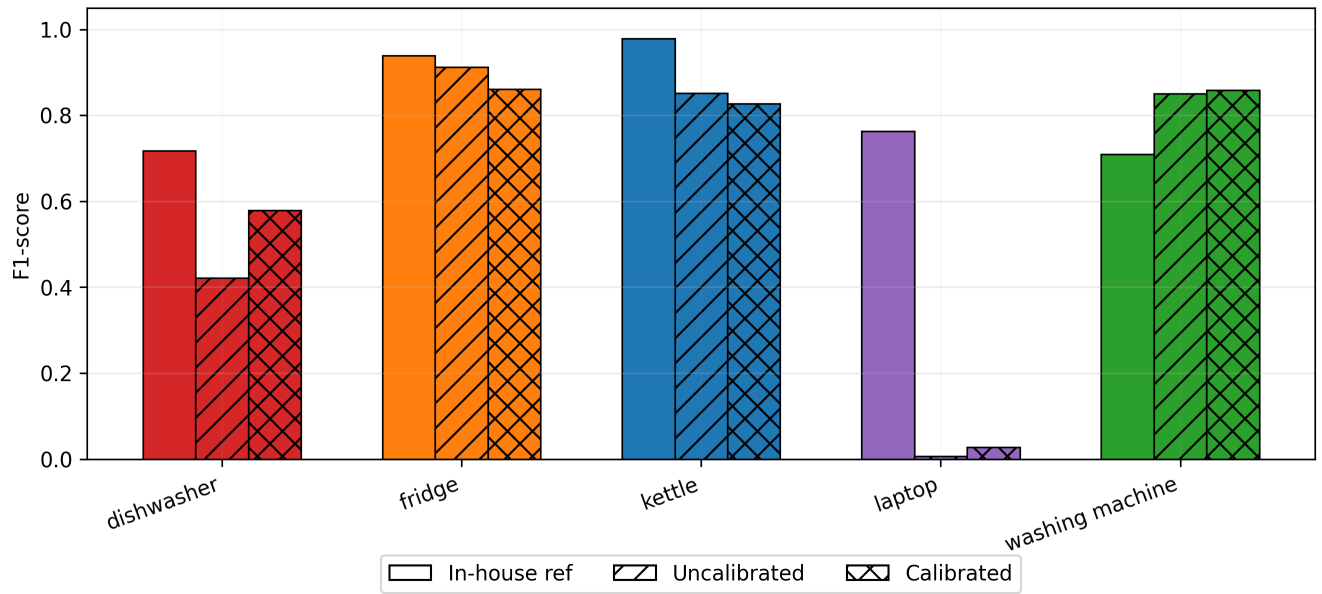


Figure 6.3: F1-score comparison for cross-house transfer from UK-DALE House 1 to House 2.

6.2 In-House Training Evaluation

6.2.1 Effect of Training Duration

This experiment quantifies how the volume of locally recorded training data affects in-house disaggregation accuracy. Tables 6.3 and 6.4 report per-appliance metrics for House 1 and House 2 respectively, alongside the number of activation events present in each training window and with Figures 6.4, 6.5 and 6.6.

Table 6.3: In-house disaggregation performance on UK-DALE House 1 for two training durations. MAE and RMSE are expressed in watts. *Events* denotes the number of activation events present in the training window.

Appliance	Duration	Events	MAE	RMSE	SAE	F1	P	R
kettle	14 d	79	11.6	118.2	0.125	0.846	0.785	0.918
	28 d	177	13.1	123.7	0.135	0.823	0.735	0.936
fridge	14 d	361	23.5	46.2	0.063	0.794	0.747	0.847
	28 d	716	19.5	41.2	0.015	0.835	0.772	0.908
washing machine	14 d	1189	39.8	173.8	0.155	0.403	0.429	0.381
	28 d	2510	13.8	100.2	0.058	0.826	0.844	0.808
dishwasher	14 d	31	11.8	103.3	0.164	0.365	0.682	0.250
	28 d	87	6.8	63.1	0.118	0.452	0.803	0.315
laptop	14 d	32	12.3	18.2	0.787	0.164	0.399	0.104
	28 d	57	11.9	18.0	0.880	0.118	0.451	0.068

Table 6.4: In-house disaggregation performance on UK-DALE House 2 for two training durations. MAE and RMSE are expressed in watts. *Events* denotes the number of activation events present in the training window.

Appliance	Duration	Events	MAE	RMSE	SAE	F1	P	R
kettle	14 d	70	2.8	58.2	0.036	0.939	0.929	0.949
	28 d	145	2.6	55.5	0.045	0.949	0.949	0.949
fridge	14 d	698	13.4	30.8	0.213	0.932	0.954	0.911
	28 d	1403	11.4	28.5	0.208	0.961	0.972	0.950
washing machine	14 d	225	7.9	72.9	0.349	0.391	0.478	0.330
	28 d	419	8.8	86.8	0.228	0.360	0.485	0.286
dishwasher	14 d	76	6.2	80.5	0.133	0.725	0.955	0.584
	28 d	152	6.4	76.3	0.132	0.729	0.965	0.586
laptop	14 d	66	5.0	9.0	0.123	0.744	0.791	0.703
	28 d	91	4.8	8.7	0.229	0.759	0.829	0.701

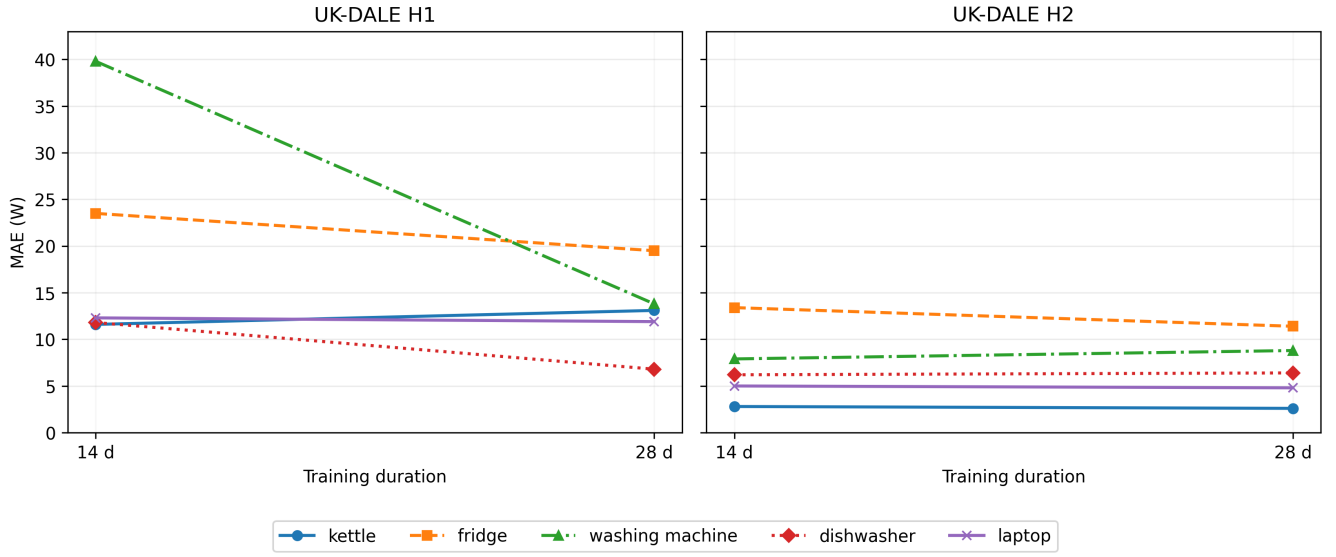


Figure 6.4: MAE comparison across training durations on UK-DALE House 1 and House 2

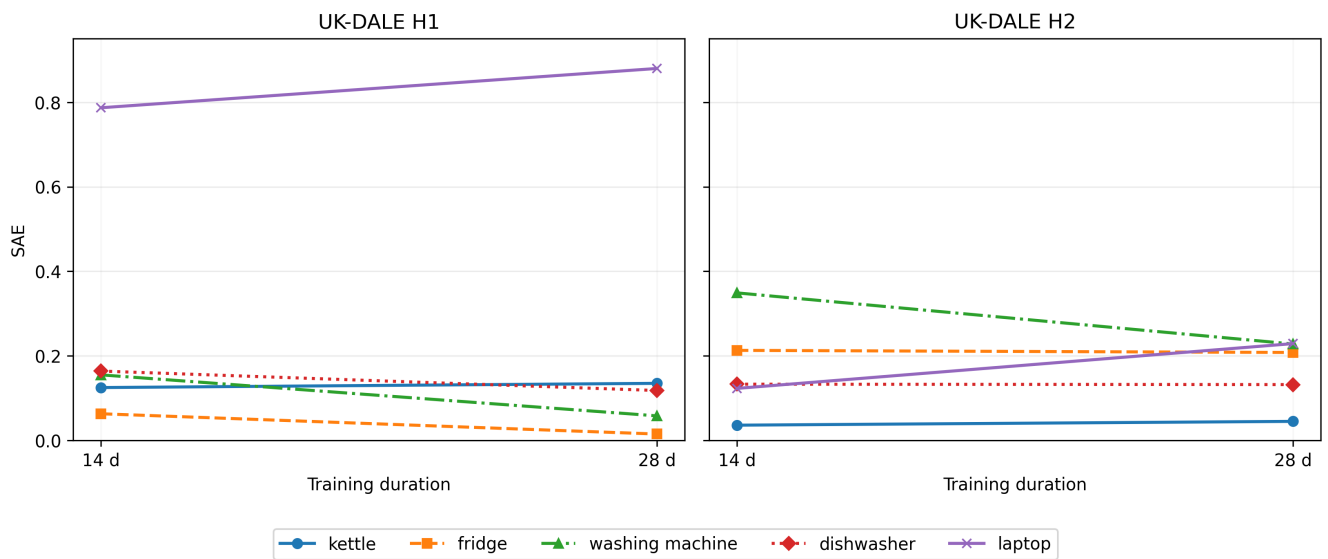


Figure 6.5: SAE comparison across training durations UK-DALE House 1 and House 2

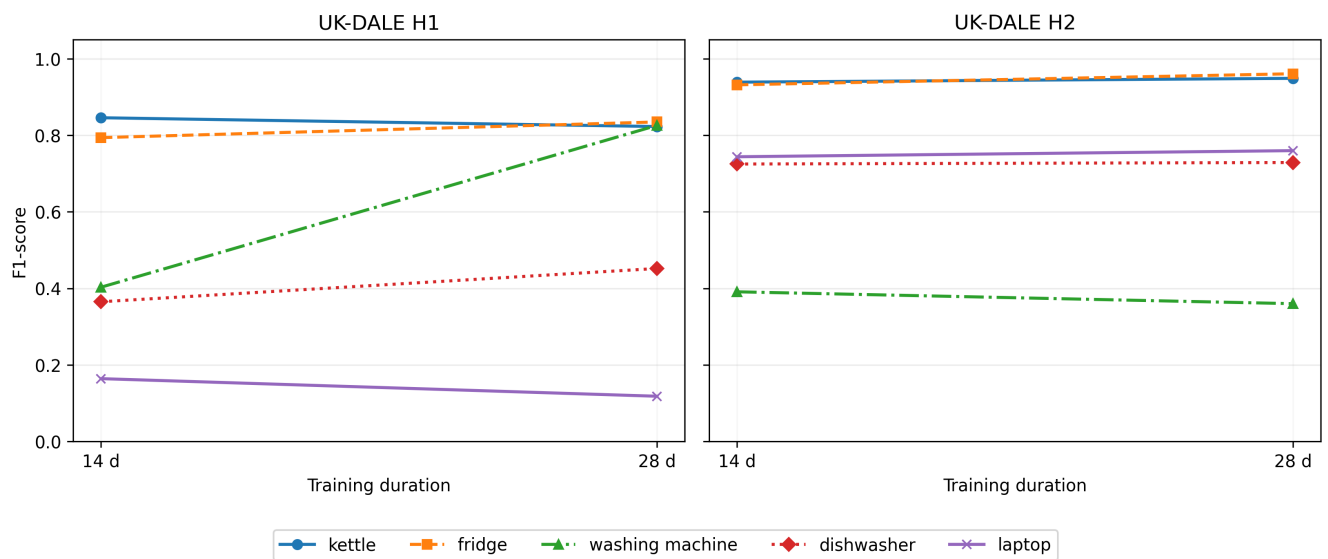


Figure 6.6: F1-score comparison across training durations on UK-DALE House 1 and House 2

Training-duration gains depend on both event count and aggregate noise level. Doubling the training window from 14 to 28 days produces uneven gains

(Figure 6.4). On House 1, the largest MAE reductions are on the washing machine ($39.8 \rightarrow 13.8$ W, $\times 2.1$ events), the dishwasher ($11.8 \rightarrow 6.8$ W, $\times 2.8$) and the fridge ($23.5 \rightarrow 19.5$ W). The kettle is an exception: MAE *increases* ($11.6 \rightarrow 13.1$ W) and F1 drops modestly ($0.846 \rightarrow 0.823$), with recall up ($0.918 \rightarrow 0.936$) but precision down ($0.785 \rightarrow 0.735$): the longer window shifts the decision boundary toward higher sensitivity, capturing more activations at the cost of more false positives. On House 2, MAE changes stay within 2 W in either direction across all appliances, because the cleaner aggregate already exposes the signatures within 14 d. The cross-house contrast is informative: the H2 washing machine reaches MAE 7.9 W at 14 d with only 225 events, whereas the H1 washing machine still records 39.8 W at 14 d despite five times as many activations (1,189). The event-count threshold for adequate performance is therefore higher on H1 than H2: the noisier aggregate (more sub-metered and unmetered loads, see Section 5.1) dilutes per-event signal and makes disaggregation intrinsically harder regardless of training volume. F1 follows the same pattern (Figure 6.6).

SAE moves in inconsistent directions across appliances (Figure 6.5). Several see clear improvements (H1 washing machine $0.155 \rightarrow 0.058$, H1 fridge $0.063 \rightarrow 0.015$, H1 dishwasher $0.164 \rightarrow 0.118$, H2 washing machine $0.349 \rightarrow 0.228$), while others increase despite gains in MAE and F1: H1 kettle $0.125 \rightarrow 0.135$, H1 laptop $0.787 \rightarrow 0.880$, H2 laptop $0.123 \rightarrow 0.229$. This reflects the same trade-off observed in the cross-house analysis: as more training data makes the detector more sensitive, additional detected activations introduce cumulative energy bias before the regression head fully calibrates the predicted magnitudes. Low SAE values for the 14-day models should be interpreted with caution: they often reflect near-zero predictions, or an almost coincidental match in total energy, rather than genuinely accurate energy estimation.

The H2 washing machine sits below the event-count threshold for stable training. The H2 washing machine is a second case where additional duration degrades performance: MAE rises from 7.9 to 8.8 W and F1 from 0.391 to 0.360 between 14 and 28 d. A per-fortnight inspection of the training window shows that the first and second fortnights are statistically indistinguishable (225 vs. 194 events, 688 vs. 693 W mean active power, identical bout-duration distribution): the added data are not qualitatively different. The instability instead reflects the appliance’s $\sim 1\%$ duty cycle, which leaves the chronological 20% validation tail with only a handful of activations and amplifies stochastic differences in early-stopping checkpoints, the same mechanism behind the V-shape laptop result in Section 6.3. The 0.9 W and 0.03 F1 gap should therefore be read as variance rather than a deterministic ordering; per-seed averaging would be needed to confirm.

The laptop on House 1 reaches a ceiling that duration cannot lift.

Despite doubling activations from 32 to 57, the laptop on House 1 shows no useful improvement: MAE marginally drops from 12.3 to 11.9 W, F1 *decreases* from 0.164 to 0.118, and SAE rises from 0.787 to 0.880. The cause is the within-window distribution mismatch analysed in paragraph 6.1. Additional calendar duration cannot recover the regime the model has never been trained on.

Practical implication for Home Assistant deployment. The minimum useful recording duration depends on the noise level of the household aggregate. On House 2, 14 d of local data already produces models within 2 W of the 28 d MAE on every appliance, in either direction; two weeks is therefore a reasonable default. On House 1, the second fortnight delivers much larger gains (-26.0 W on the washing machine, -5.0 W on the dishwasher, -4.0 W on the fridge), so noisier aggregates require more data to reach comparable accuracy. Since users cannot determine their regime in advance, 28 d is the safer default and 14 d a lower bound for early experimentation. Two caveats: infrequent appliances should be sized by event count rather than calendar duration (the H1 dishwasher accumulates only 31 activations in 14 d, insufficient regardless of noise level); and the H1 laptop shows that additional data cannot compensate for a training window structurally unrepresentative of the evaluation period.

6.2.2 Effect of Appliance Meter Delay

This experiment quantifies the system’s tolerance to misalignment between the aggregate signal and the per-appliance submeter. Training duration is fixed at 28 days. Tables 6.5 and 6.6 report results on House 1 and House 2 respectively, alongside Figures 6.7, 6.8, 6.9.

Table 6.5: In-house disaggregation performance on UK-DALE House 1 for three appliance meter delays. Training duration is fixed at 28 days. MAE and RMSE are expressed in watts.

Appliance	Delay	MAE	RMSE	SAE	F1	P	R
kettle	0 s	12.4	122.7	0.151	0.834	0.764	0.918
	5 s	13.1	123.7	0.135	0.823	0.735	0.936
	10 s	13.7	133.9	0.121	0.795	0.715	0.895
fridge	0 s	20.5	41.8	0.045	0.831	0.766	0.908
	5 s	19.5	41.2	0.015	0.835	0.772	0.908
	10 s	22.0	44.0	0.115	0.818	0.771	0.870
washing machine	0 s	10.7	81.7	0.117	0.912	0.883	0.942
	5 s	13.8	100.2	0.058	0.826	0.844	0.808
	10 s	15.9	108.9	0.026	0.721	0.794	0.660
dishwasher	0 s	6.8	59.0	0.147	0.426	0.828	0.287
	5 s	6.8	63.1	0.118	0.452	0.803	0.315
	10 s	7.3	66.6	0.115	0.397	0.776	0.267
laptop	0 s	11.9	18.0	0.914	0.090	0.440	0.050
	5 s	11.9	18.0	0.880	0.118	0.451	0.068
	10 s	12.0	18.1	0.913	0.073	0.377	0.041

Table 6.6: In-house disaggregation performance on UK-DALE House 2 for three appliance meter delay. Training duration is fixed at 28 days. MAE and RMSE are expressed in watts.

Appliance	Delay	MAE	RMSE	SAE	F1	P	R
kettle	0 s	2.0	38.0	0.086	0.974	0.989	0.959
	5 s	2.6	55.5	0.045	0.949	0.949	0.949
	10 s	3.8	82.4	0.050	0.907	0.907	0.907
fridge	0 s	11.0	25.2	0.224	0.965	0.981	0.950
	5 s	11.4	28.5	0.208	0.961	0.972	0.950
	10 s	10.9	28.1	0.188	0.958	0.965	0.951
washing machine	0 s	7.9	74.6	0.136	0.476	0.530	0.432
	5 s	8.8	86.8	0.228	0.360	0.485	0.286
	10 s	8.2	80.4	0.126	0.586	0.595	0.577
dishwasher	0 s	5.5	64.9	0.113	0.739	0.954	0.602
	5 s	6.4	76.3	0.132	0.729	0.965	0.586
	10 s	6.3	77.9	0.126	0.726	0.960	0.584
laptop	0 s	4.9	9.0	0.207	0.747	0.800	0.699
	5 s	4.8	8.7	0.229	0.759	0.829	0.701
	10 s	4.8	8.7	0.212	0.758	0.819	0.704

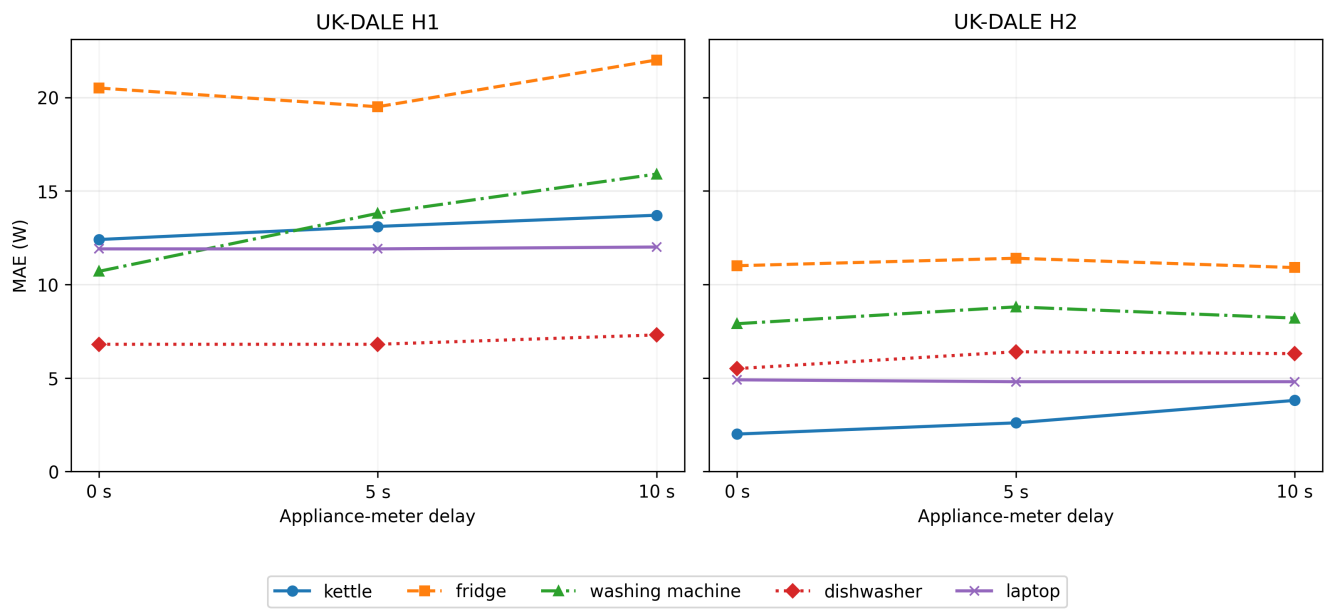


Figure 6.7: MAE comparison across appliance-meter delays on UK-DALE House 1 and House 2

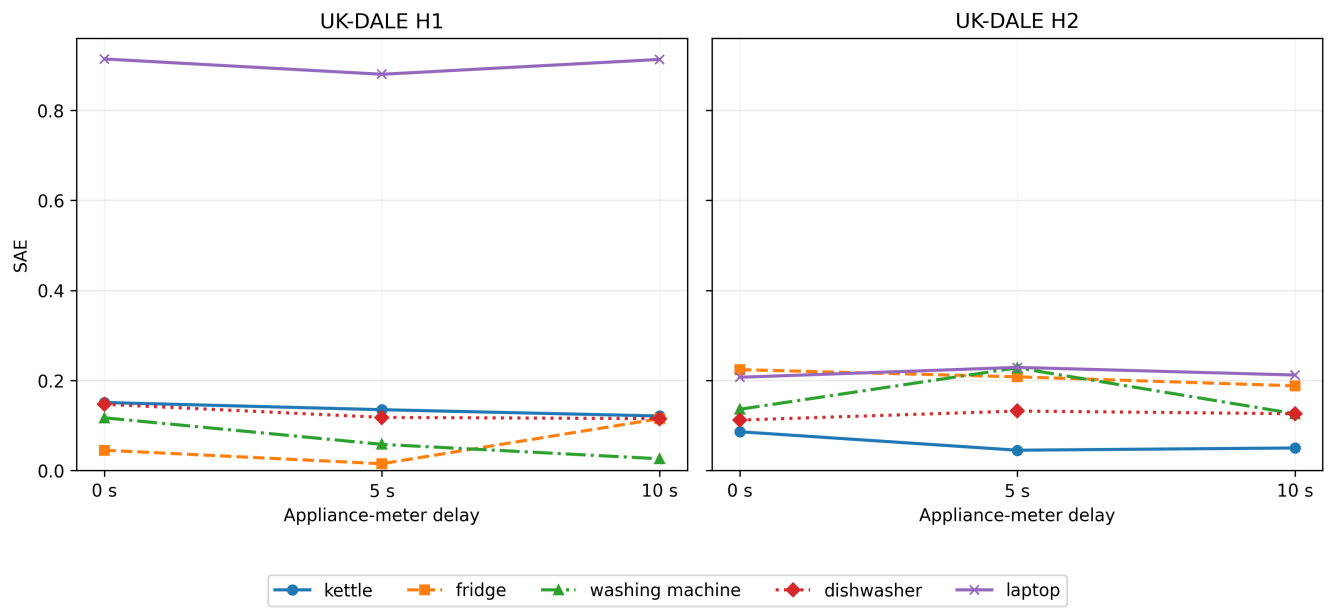


Figure 6.8: SAE comparison across appliance meter delays on UK-DALE House 1 and House 2

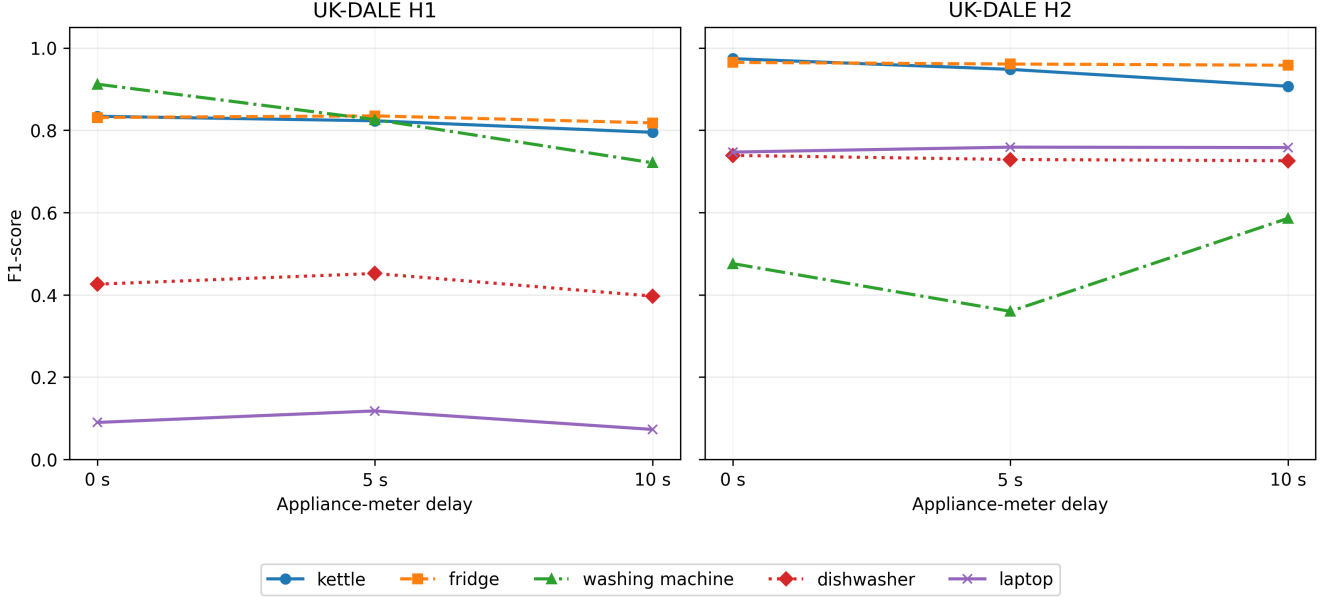


Figure 6.9: F1-score comparison across appliance meter delays on UK-DALE House 1 and House 2

Delay sensitivity is governed by event sharpness Sustained, slowly evolving signatures (fridge, dishwasher, laptop) are essentially insensitive to delay: MAE varies by less than 2.6 W across 0–10 s on both houses (Figure 6.7), because a 10 s misalignment is negligible when an activation lasts tens of minutes. The kettle degrades gracefully on both houses (H1: 12.4 $\rightarrow$ 13.7 W; H2: 2.0 $\rightarrow$ 3.8 W) since its high-amplitude spike remains detectable even with a label shifted by one or two samples. The washing machine is the most delay-sensitive appliance: on House 1, MAE rises monotonically from 10.7 W at 0 s to 15.9 W at 10 s (+49%), because its multi-phase cycle contains sharp transition boundaries whose alignment with the label is critical to learn correct onset and offset timing. F1 follows the same pattern (Figure 6.9): flat for sustained loads, mild monotone drop for the kettle, and sharp degradation on the House 1 washing machine (0.912 $\rightarrow$ 0.721).

Improvements in SAE with increasing latency are misleading. A counterintuitive phenomenon is evident in Figure 6.8: SAE often improves as latency increases, even when MAE and F1 deteriorate. On House 1, the SAE of the kettle goes from 0.151 to 0.121, that of the washing machine from 0.117 to 0.026 (−78%), and that of the dishwasher from 0.147 to 0.115, while the corresponding MAE and F1 scores deteriorate. The mechanism is the same as that already observed in sections 6.1 and 6.2.1: since misalignment makes the detector less sensitive, it is

possible that the resulting underprediction compensates for the slight overestimation of the model at zero delay, producing a lower cumulative energy error that does not reflect better disaggregation. The SAE is therefore an unreliable single indicator of delay sensitivity; the mean absolute error (MAE) and F1 scores should be considered the authoritative measures.

The House 2 washing machine exhibits a non-monotonic response. On House 2, the washing machine F1 at 5 s (0.360) falls below both the 0 s (0.476) and 10 s (0.586) values, reproducing the V-shape pattern. The model is already weak at every delay configuration (in-house F1 of 0.360 at 28 d/5 s, Table 6.4), and the small number of training activations (419 events) makes the result sensitive to the exact validation tail drawn by each delay configuration. The non-monotonicity should not be interpreted as evidence that a 5 s delay is structurally harmful (see 6.2.1).

Practical implication for Home Assistant deployment. The delay sweep models the timestamp misalignment between the aggregate meter and a per-appliance smart plug, not a change in sampling rate: the resolution is held at 5 s throughout and only the label stream is offset. In a Home Assistant deployment the dominant source of such misalignment is the plug’s reporting cadence: a plug reporting every 5–10 s can be offset from the aggregate by up to one reporting period. So the 0–10 s sweep serves as a proxy for plugs of increasing reporting coarseness. The proxy is conservative: a genuinely coarse plug also loses temporal resolution through downsampling, an effect not captured here (except in end-to-end validation detailed in 4.2 where Antella Smart plugs have a resolution of $\tilde{0.1}$ Hz), so the degradations reported above are a lower bound on what such a plug would incur. Under this reading, a 5 s offset is tolerable for every appliance except the washing machine, where it already begins to erode recall (House 1 R: $0.942 \rightarrow 0.808$). Transition-rich appliances should therefore be paired with a plug that reports as frequently as possible to keep the offset small; for sustained loads (fridge, dishwasher, laptop) the offset is immaterial and meter choice is not a priority.

6.3 Simulated Finetuning

This section evaluates the fine-tuning mechanism described in Section 3.9 under cross-house transfer conditions. Table 6.7 and Table 6.8 reports the finetuning results, alongside Figures 6.10, 6.11, 6.12, 6.13, 6.14, 6.15.

Table 6.7: Cross-house fine-tuning performance when a model trained on UK-DALE House 1 is adapted to UK-DALE House 2. The baseline corresponds to direct transfer without target-house fine-tuning. Appliance-meter delays are fixed to 5 s. *Events* denotes the number of activation events available in the fine-tuning window. MAE and RMSE are expressed in watts.

Appliance	Setting	Events	MAE	RMSE	SAE	F1	P	R
kettle	Baseline	–	9.9	104.2	0.288	0.927	0.893	0.964
	Fine-tuned 5 d (skipped)	–	–	–	–	–	–	–
	Fine-tuned 10 d	34	4.5	81.7	0.037	0.952	0.946	0.958
	Fine-tuned 14 d	47	4.3	78.9	0.029	0.955	0.946	0.964
fridge	Baseline	–	23.5	38.0	0.199	0.853	0.909	0.804
	Fine-tuned 5 d	209	23.0	37.6	0.201	0.840	0.896	0.791
	Fine-tuned 10 d	401	21.4	36.5	0.177	0.848	0.897	0.804
	Fine-tuned 14 d	552	21.0	35.7	0.166	0.856	0.896	0.819
washing machine	Baseline	–	24.9	151.8	0.697	0.303	0.234	0.431
	Fine-tuned 5 d (skipped)	–	–	–	–	–	–	–
	Fine-tuned 10 d	160	9.4	86.1	0.672	0.322	0.895	0.197
	Fine-tuned 14 d	221	9.4	87.4	0.676	0.319	0.905	0.194
dishwasher	Baseline	–	37.4	254.5	0.852	0.303	0.853	0.184
	Fine-tuned 5 d	22	16.7	125.4	0.008	0.770	0.746	0.795
	Fine-tuned 10 d	40	17.3	125.1	0.039	0.741	0.717	0.767
	Fine-tuned 14 d	63	17.5	125.3	0.047	0.761	0.723	0.803
laptop	Baseline	–	11.4	17.5	0.940	0.118	0.983	0.063
	Fine-tuned 5 d	10	5.0	9.8	0.311	0.876	0.939	0.821
	Fine-tuned 10 d	18	10.8	16.8	0.862	0.242	0.981	0.138
	Fine-tuned 14 d	24	5.9	11.0	0.427	0.850	0.950	0.769

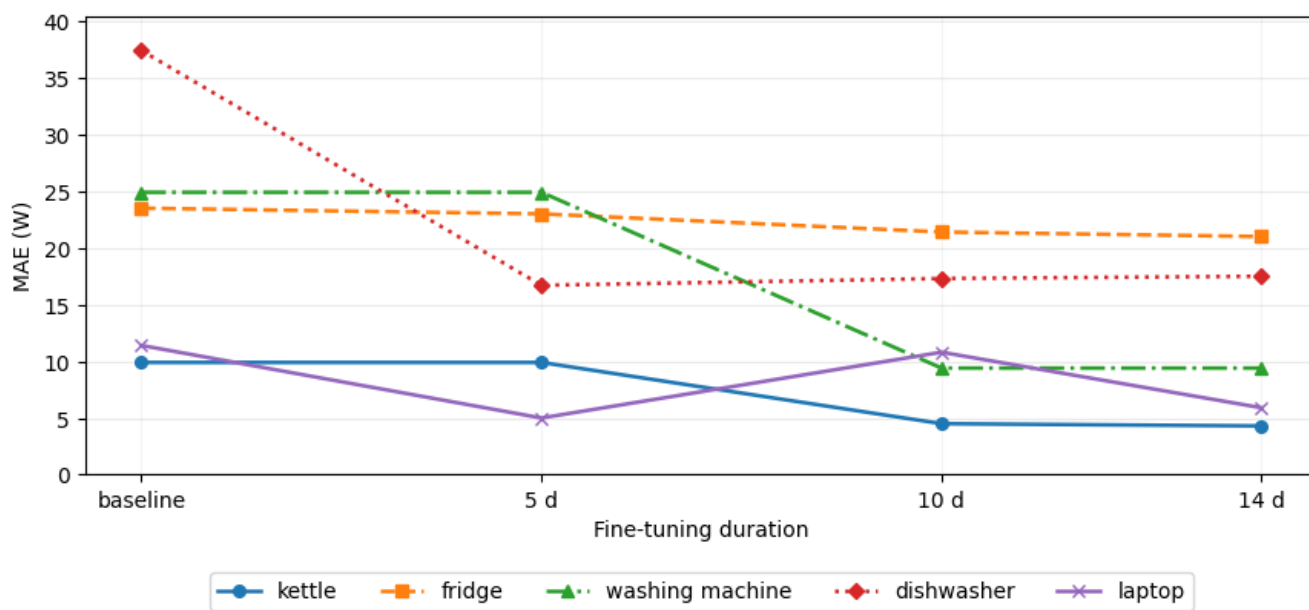


Figure 6.10: MAE comparison across fine-tuning durations for transfer from UK-DALE House 1 to House 2

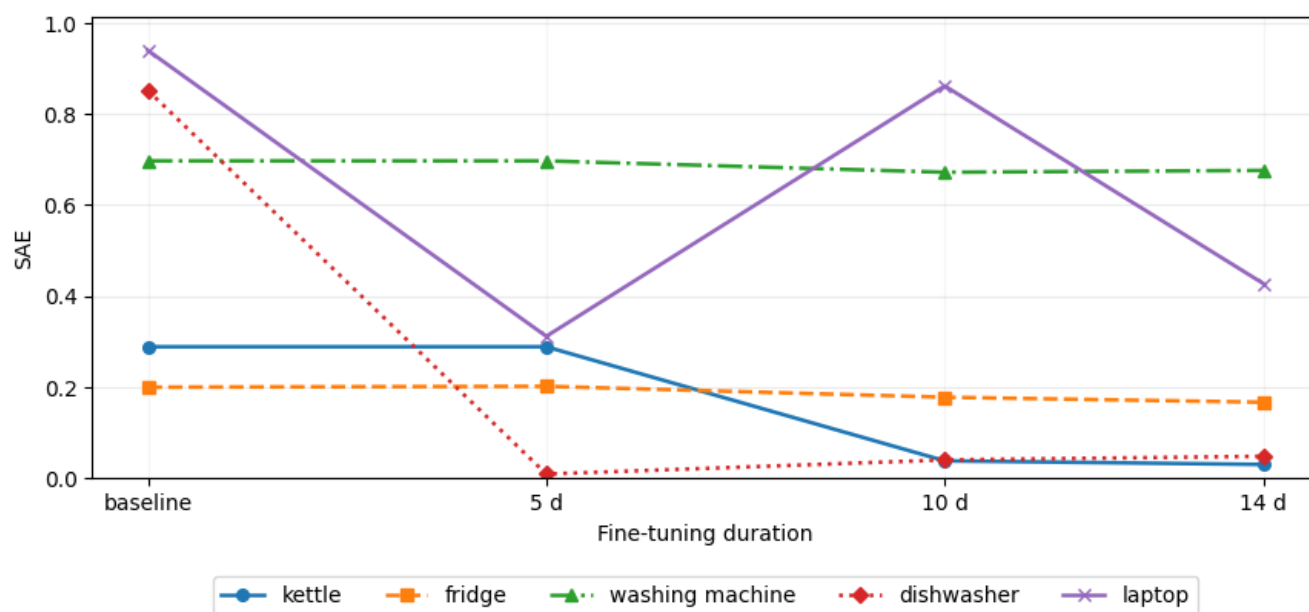


Figure 6.11: SAE comparison across fine-tuning durations for transfer from UK-DALE House 1 to House 2

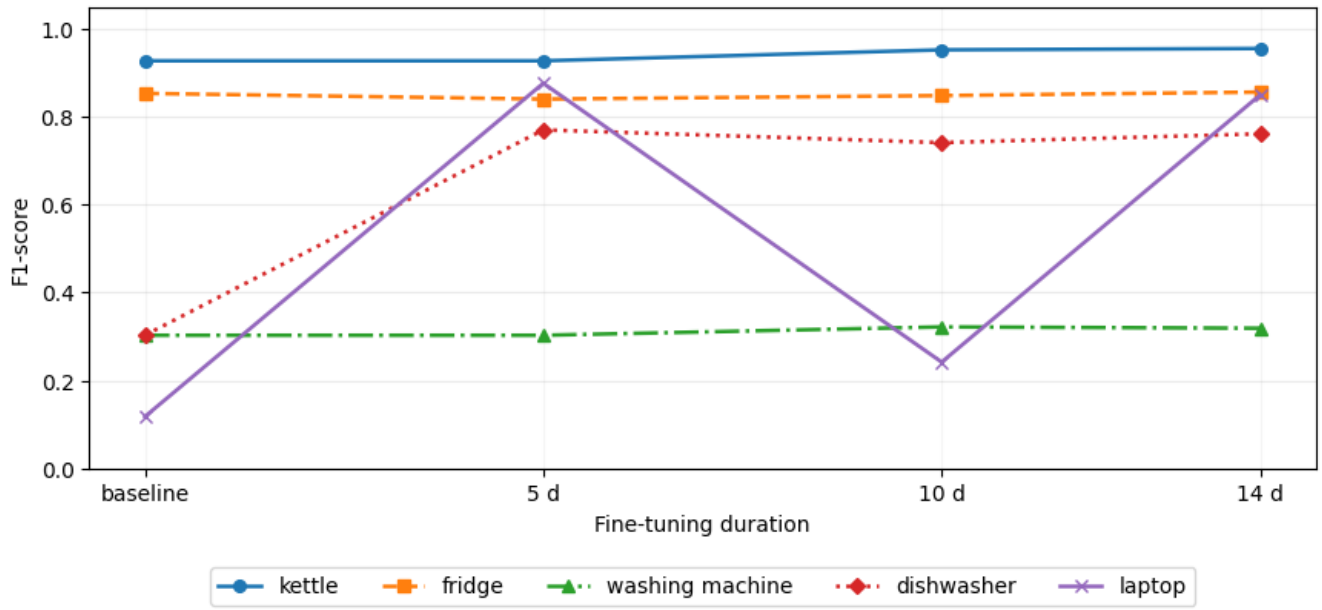


Figure 6.12: F1-score comparison across fine-tuning durations for transfer from UK-DALE House 1 to House 2

Table 6.8: Cross-house fine-tuning performance when a model trained on UK-DALE House 2 is adapted to UK-DALE House 1. The baseline corresponds to direct transfer without target-house fine-tuning. Appliance-meter delays are fixed to 5 s. *Events* denotes the number of activation events available in the fine-tuning window. Rows marked with – indicate appliances for which fine-tuning was skipped and the reported values therefore correspond to the previous model state. Laptop with 10 d of finetune was also skipped due to 0 activation in validation. MAE and RMSE are expressed in watts.

Appliance	Setting	Events	MAE	RMSE	SAE	F1	P	R
kettle	Baseline	–	12.7	141.5	0.084	0.826	0.793	0.863
	Fine-tuned 5 d	29	11.7	127.0	0.145	0.819	0.787	0.853
	Fine-tuned 10 d	52	12.0	127.8	0.102	0.812	0.763	0.868
	Fine-tuned 14 d	73	12.0	127.0	0.150	0.814	0.774	0.858
fridge	Baseline	–	28.0	52.5	0.252	0.718	0.715	0.721
	Fine-tuned 5 d	132	20.0	44.7	0.107	0.809	0.777	0.844
	Fine-tuned 10 d	265	19.6	44.2	0.131	0.818	0.792	0.845
	Fine-tuned 14 d	367	19.6	44.1	0.142	0.820	0.800	0.841
washing machine	Baseline	–	38.4	238.8	0.875	0.012	0.062	0.006
	Fine-tuned 5 d	574	39.7	184.5	0.235	0.496	0.449	0.553
	Fine-tuned 10 d	815	35.7	173.3	0.042	0.486	0.468	0.505
	Fine-tuned 14 d	1262	33.8	166.7	0.073	0.484	0.490	0.479
dishwasher	Baseline	–	44.9	270.0	2.201	0.217	0.170	0.298
	Fine-tuned 5 d	15	–	–	–	–	–	–
	Fine-tuned 10 d	30	–	–	–	–	–	–
	Fine-tuned 14 d	46	9.7	84.9	0.121	0.415	0.578	0.324
laptop	Baseline	–	12.7	16.3	1.099	0.449	0.323	0.734
	Fine-tuned 5 d	10	9.5	12.5	0.109	0.435	0.351	0.571
	Fine-tuned 10 d (skip)	16	–	–	–	–	–	–
	Fine-tuned 14 d	18	7.2	12.8	0.579	0.342	0.446	0.278

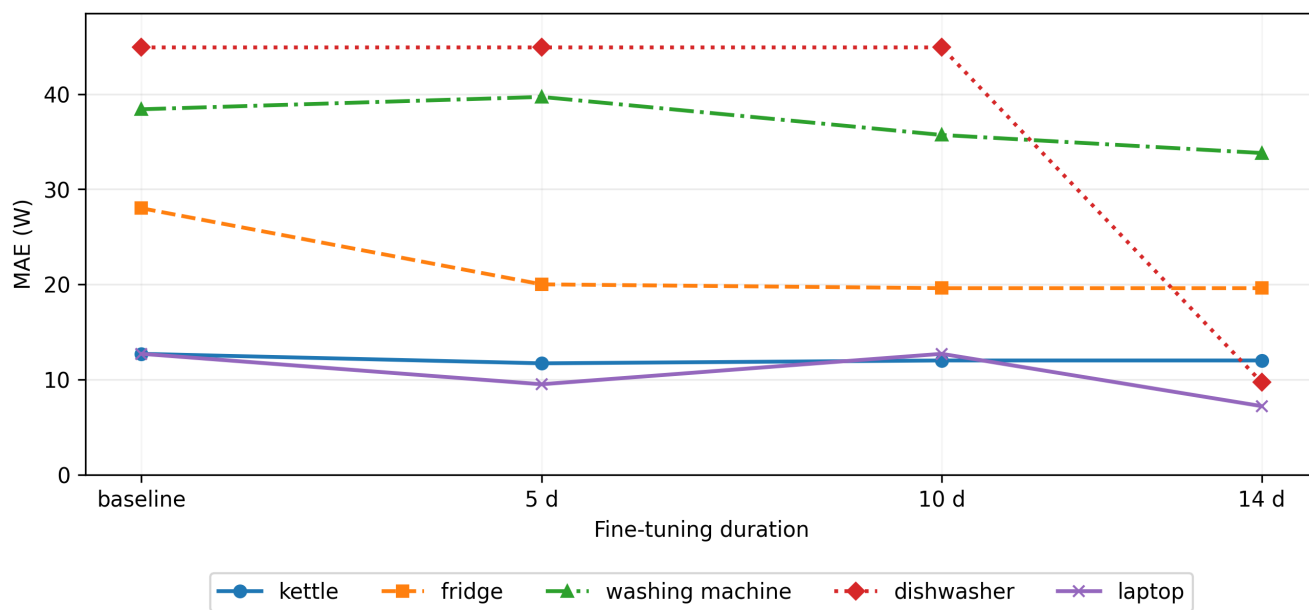


Figure 6.13: MAE comparison across fine-tuning durations for transfer from UK-DALE House 2 to House 1

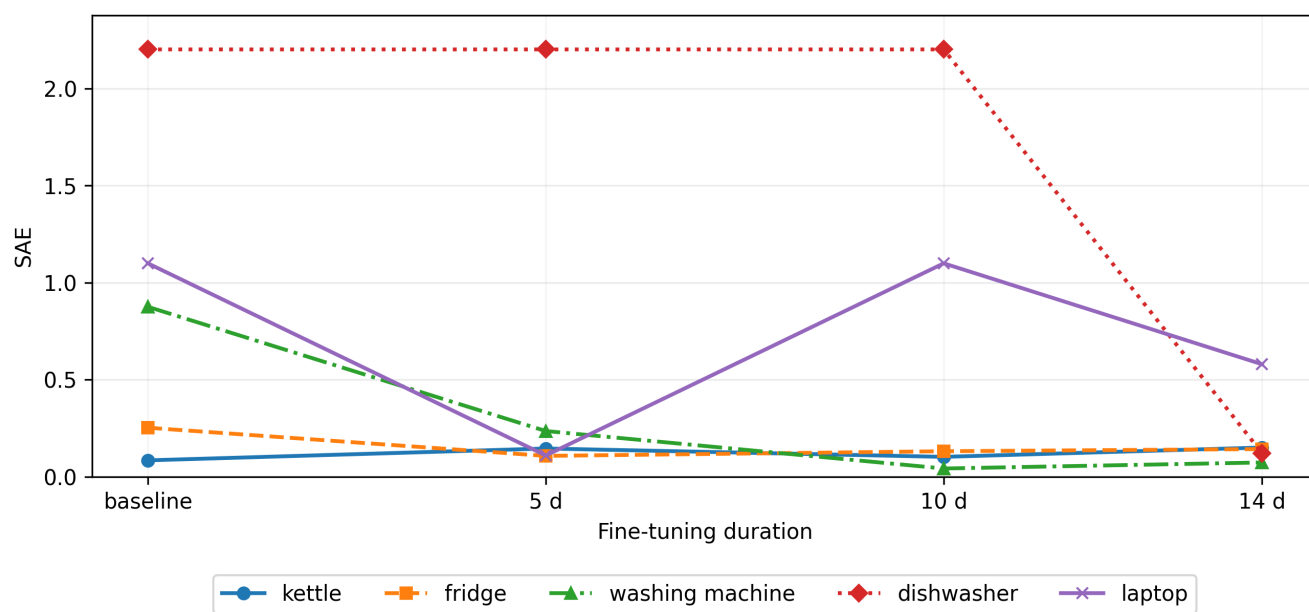


Figure 6.14: SAE comparison across fine-tuning durations for transfer from UK-DALE House 2 to House 1

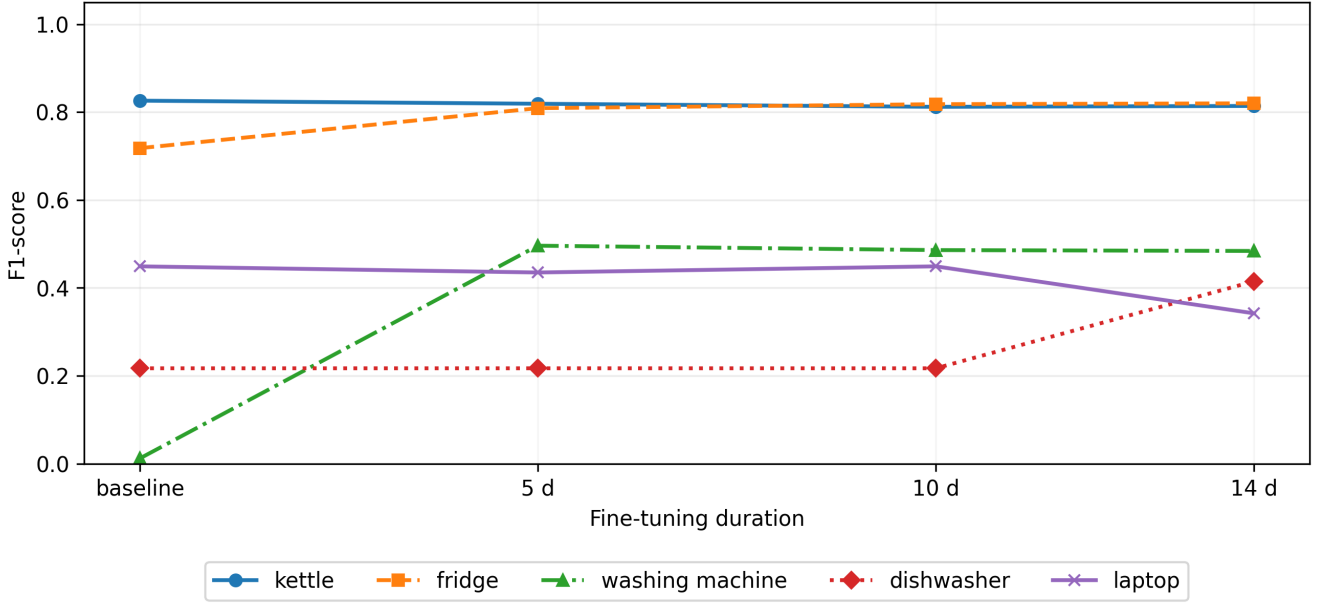


Figure 6.15: F1-score comparison across fine-tuning durations for transfer from UK-DALE House 2 to House 1

Fine-tuning rescues poorly-transferred appliances and saturates quickly.

Across both transfer directions, fine-tuning produces the largest gains on appliances whose cross-house baseline is weakest, and the benefit largely saturates within 5–10 days of target data (Figure 6.10). On H1→H2, MAE falls by 57% on the kettle (9.9 → 4.3 W), 53% on the dishwasher (37.4 → 17.5 W), and 48% on the laptop (11.4 → 5.9 W); the fridge, whose baseline MAE was already acceptable, improves marginally. The same asymmetry holds on H2→H1: the fridge improves by 30% (28.0 → 19.6 W), the dishwasher by 78% once the event threshold is reached, and the laptop by 43% (12.7 → 7.2 W), while the kettle (already at 12.7 W cross-house) improves only marginally. F1 follows the same overall pattern (Figures 6.12, 6.15) with two exceptions discussed below.

The washing machine shows opposite responses in the two transfer directions. On H1→H2, the F1 stays in the 0.30–0.32 range regardless of fine-tuning duration, suggesting fine-tuning is ineffective. The MAE tells a different story: it drops from 24.9 to 9.4 W, the precision rises from 0.234 to 0.895, and the recall falls from 0.431 to 0.197. Fine-tuning therefore shifts the model toward a conservative regime: it fires less often, but is almost always correct when it does. The F1 conceals this qualitative change because precision and recall move in opposite directions. The pattern is consistent with the limited number of H2

washing machine activations (221 at 14 d in the fine-tuning pool, 152 in the 7-day test window), which prevents the model from learning the full multi-phase distribution; fine-tuning over-corrects toward the most distinctive sub-events and ignores the rest. The H1→H2 baseline F1 of 0.303 is also markedly lower than the 0.818 reported in the cross-house evaluation (Table 6.1); the gap reflects the different evaluation window used by the two protocols (the fine-tuning protocol evaluates on the late-July segment which contains denser activations under a higher mains background, both of which depress detection accuracy) and the different appliance-meter delay (5 s here against 0 s in cross-house), not a regression of the model itself.

The H2→H1 direction shows the opposite behaviour and constitutes the most dramatic fine-tuning partial recalibration in the entire evaluation. The baseline F1 is essentially zero (0.012, $R = 0.006$) : the H2-trained model is functionally blind to the H1 washing machine. Five days of fine-tuning with 574 events lifts F1 to 0.496 and recall to 0.553, with negligible additional gain at 10 and 14 days (0.486 and 0.484). The MAE follows a similar trajectory (38.4 → 33.8 W). The contrast with H1→H2 reflects the underlying event asymmetry between houses: H1 exposes the fine-tuner to a far richer event repertoire (1,262 vs. 221 events at 14 d), which is sufficient to recover detection capability from a catastrophic baseline but still falls short of in-house performance ($F1 = 0.826$, Table 6.3). Fine-tuning therefore rescues the model from total failure, but the residual gap to in-house training confirms that no amount of cross-house adaptation fully substitutes for matched training data when the activation count is the binding constraint.

The laptop V-shape at 10 d is explained by the validation window in H1 → H2. The laptop shows a non-monotonic V-shape across all three metrics at the 10-day fine-tuning window: MAE jumps from 5.0 to 10.8 and back to 5.9 W, SAE from 0.311 to 0.862 to 0.427, and F1 from 0.876 to 0.242 to 0.850 (Figure 6.12). With `val_ratio = 0.2`, the 10-day fine-tuning window places its validation tail on 2013-07-19–20, a two-day period in which the labelled laptop (#1) is active only 7.7% of the time while a second, unmetered laptop (#2) occupies the aggregate at 49.6%. Early stopping therefore selects a checkpoint that minimises a contradictory validation loss (aggregate power consistent with a laptop ON, label almost always zero), producing a near-always-OFF predictor ($P = 0.981$, $R = 0.138$) qualitatively identical to the baseline. The 5-day and 14-day windows avoid this period; their validation tails contain adequate labelled ON events and yield well-calibrated checkpoints. This also highlights a fundamental limitation when two near-identical appliances coexist in the same house : the unmetered second laptop appears as label noise from the model’s perspective.

The minimum-event gate delays fine-tuning for infrequent appliances.

The $H2 \rightarrow H1$ dishwasher is skipped at 5 and 10 days (15 and 30 events respectively) because the activation count falls below the training floor, and the 14-day model is the first to update (46 events, $F1 = 0.415$). The kettle on $H1 \rightarrow H2$ is similarly skipped at 5 days. For infrequent appliances, the adaptation window should be sized by event count rather than calendar duration; a fixed 14-day schedule may still be insufficient for a dishwasher used fewer than three times per week.

SAE recovers faster than F1 under fine-tuning. Unlike the artifactual SAE shifts observed elsewhere in the evaluation (Sections 6.1, 6.2.1, 6.2.2), fine-tuning on target-house data produces genuine amplitude recalibration : SAE consistently improves with the first available fine-tuning step even when F1 remains flat. The kettle on $H1 \rightarrow H2$ illustrates this most clearly: F1 rises by only 0.025 at 10 days, yet SAE drops from 0.288 to 0.037. Amplitude recalibration on target-house data is faster to learn than accurate event timing, making SAE a sensitive early indicator of fine-tuning progress.

Practical implication for Home Assistant deployment. The cross-house fine-tuning protocol evaluated here is an upper bound on adaptation difficulty : it combines device-level and household-level shift simultaneously. The more frequent real-world scenario in long-term deployment is intra-house drift driven by seasonal patterns, occupant habit changes, or appliance ageing, where the household and the appliance remain the same and only the usage distribution shifts. The five-to-fourteen-day fine-tuning window measured above is therefore pessimistic for the typical case ; less data should suffice when only the usage pattern has drifted. A practical heuristic for triggering fine-tuning is to monitor the residuals between disaggregated predictions and smart-plug ground truth, and adapt when these exceed a calibrated threshold, with the largest gains concentrating on appliances whose current model handles recent data poorly.

6.4 Computational Cost

This section evaluates whether the proposed system meets its edge-deployment constraints.

6.4.1 Deployment Topologies

Two deployment topologies are relevant for self-hosted Home Assistant users. In the *co-located* topology, the NILM backend runs on the same Raspberry Pi as Home Assistant, minimising hardware cost at the price of CPU contention. In the *split* topology, the backend is deployed on a more capable always-on machine on the local network (NAS, mini-PC, or homelab server), while Home Assistant remains on the Pi and communicates with the backend over LAN. The split topology is the standard configuration in Home Assistant deployments and is the recommended setup for this system, since it removes the training-cost constraint identified in Section 6.4.2 and frees the Pi from the inference workload. The measurements reported in this section are conducted on the co-located topology, which constitutes the worst-case compute envelope: the split topology dominates it at every percentile and is therefore not separately benchmarked.

6.4.2 Training Cost

Table 6.9 reports training stats on three different platforms.

Table 6.9: Training cost for the kettle model across three execution platforms. The reported time is the average wall-clock time per epoch, including training and validation. Peak RAM denotes the peak resident set size of the training process.

Platform	Accelerator	Epochs	Time/epoch (s)	Peak RAM (MB)
Raspberry Pi	CPU	5	1829.4	2805.5
Desktop	CPU	5	468.1	4716.5
Desktop	GPU	5	13.6	1098.6

Training cost spans more than two orders of magnitude across platforms.

The Raspberry Pi 4 trains the kettle model in 1829 s per epoch, against 468 s on the desktop CPU and 13.6 s on the desktop GPU, a ratio of approximately 134.5 between the Pi 4 and the GPU. As is standard for self-hosted deployments, training is not expected to be performed on the deployment device itself; the Pi 4 figures are reported here for completeness. Inference latency on the same Pi 4 (Section 6.4.3) remains well below the real-time disaggregation budget, confirming that the platform’s compute constraints affect training but not live operation.

6.4.3 Inference Latency

Inference is benchmarked on the Raspberry Pi 4 across four deployment configurations crossing two factors: local (loopback) versus LAN access to the inference endpoint, and isolated versus concurrent Home Assistant (HA) workload. Each row of Table 6.10 reports the latency distribution and throughput over 1000 inference calls preceded by 50 discarded warmup calls. A single inference call loads the five appliance models sequentially and returns one disaggregated power estimate per appliance for the input window.

Table 6.10 reports the latency distribution and throughput for the four configurations.

Table 6.10: Inference latency and throughput on a Raspberry Pi 4 (8 GB) for the five-appliance NILM model under four deployment configurations. Each row reports statistics over 1000 inference calls (preceded by 50 warmup calls), with random input windows of length 518 samples.

Configuration	Latency (ms)						Throughput
	median	p95	p99	p99.5	max	std ($w/s = \text{windows per s}$)	
Local, isolated	50.77	52.99	55.60	57.52	96.47	2.25	19.5
Local, with HA	51.64	107.52	132.70	140.47	284.04	19.18	17.0
LAN, isolated	48.26	50.81	52.47	54.31	1088.42	45.82	19.7
LAN, with HA	50.54	109.59	297.87	1071.09	1145.44	103.52	14.6

Local isolated baseline. The local isolated configuration provides the hardware baseline. The distribution is tight (median 50.77 ms, std 2.25 ms, p99 only 4.83 ms above the median), as expected for a deterministic compute workload running on an unloaded CPU. Throughput reaches 19.5 windows per second, well above the 0.2 Hz operating rate of the resampled aggregate signal (one new window every 5 s).

Concurrent Home Assistant workload expands the upper percentiles. Activating HA on the same Pi increases the median latency by less than 1 ms, but expands the upper percentiles substantially: p95 grows from 52.99 ms to 107.52 ms, and p99 from 55.60 ms to 132.70 ms. The standard deviation increases by an order of magnitude. This behaviour can be attributed to the asynchronous nature of the HA workload (recorder writes, integration polling, automation triggers) which competes for CPU cycles with the inference process. Throughput drops by 13 % ($19.5 \rightarrow 17.0$ w/s).

LAN access introduces a one-second tail consistent with network delay

Up to p99.5, the LAN latency stays close to the corresponding local values (LAN isolated p99.5 = 54.31 ms vs. local isolated p99.5 = 57.52 ms). At the highest percentiles, however, a sharp cliff appears: the maximum in the LAN isolated run reaches 1088.42 ms, and the same cliff is visible in the LAN-with-HA run. This bimodal behaviour, present only in the network configurations and centred close to one second, is consistent with a network-layer delay (TCP retransmission being the most likely candidate). These events do not move the median or p99 meaningfully, but they inflate the standard deviation and the maximum.

The system meets the real-time budget under all measured conditions.

Even in the worst-case co-located configuration (LAN access with concurrent HA), all percentiles up to p99 remain well below the 5 s budget imposed by the resampled aggregate grid (Section 3.10). The recommended split topology is strictly dominated by these measurements and therefore meets the constraint by a wider margin. Approximately one request in two hundred experiences a TCP-related delay of around one second, which remains acceptable given that aggregate samples arrive every 5 s.

6.4.4 CPU and RAM Utilisation

Table 6.11 summarises the resource consumption of the NILM service in the local isolated configuration.

Table 6.11: Resource utilisation of the NILM service on the Raspberry Pi 4, local isolated configuration.

Metric	Value
CPU mean utilisation	93.4 %
CPU peak utilisation	95.3 %
Peak resident set size	113.5 MB
Model size (per appliance, on disk)	380 KB

The inference workload is CPU-bound on the Pi 4. The aggregate mean CPU utilisation of 93.4% indicates that the inference workload is fully CPU-bound on the Pi 4: there is essentially no compute slack on the platform. This explains the disproportionate expansion of the upper latency percentiles when HA runs concurrently, since any contention immediately translates into queueing delay for the inference process. The peak resident set size of 113.5 MB represents

approximately 1.4 % of the Raspberry Pi 4 8 GB platform, leaving ample headroom for HA and its addons. The model itself contributes negligibly to the memory footprint (380 KB per appliance, 1.9 MB for the five models combined), confirming that the architectural choice of a compact TCN is well suited to edge deployment.

6.5 General Discussion

The experimental results should be interpreted as evidence of deployment feasibility rather than as a ranked benchmark against the full NILM literature. The contribution lies in demonstrating that a compact causal model can be embedded into a complete local NILM workflow, not in establishing architectural superiority over recent transformer-based or multi-household models.

The experiments in this chapter show that disaggregation performance is driven primarily by appliance activation events rather than by recording duration alone. Increasing the calendar window is useful only as it exposes the model to more diverse events. This explains why some cross-house models can outperform in-house baselines when the source house provides a richer event distribution (Section 6.1), and why fine-tuning is effective even with a small number of target-house activations (Section 6.3).

Appliance behaviour also depends on signature type: salient short pulses (kettles) and stable cyclical loads (fridges) generalise better than multi-phase appliances (washing machine, dishwasher) which require more events, and than low-amplitude or confounded loads (laptop H1, kettle ECO H1) which need target-house supervision.

A recurring caveat is that SAE alone is misleading. Across the cross-house, duration, and delay ablations, low SAE values may coincide with poor MAE and F1 through two distinct mechanisms. First, an under-detecting model whose predictions collapse toward zero produces a small cumulative error simply because it predicts almost nothing. Second, an over-estimating model whose positive and negative per-sample errors cancel in the cumulative sum produces a small SAE without reflecting any per-event accuracy.

These findings qualify the assumption that in-house training is always the effective upper bound for NILM: local data can be insufficient when few events are available or when the aggregate carries strong confounders. The recommended deployment strategy in a practical self-hosted setting follows a simple decision logic. When smart-plug coverage is available on the target appliance, in-house training remains the preferred path, although its performance depends on the number and diversity of activation events available in the local recording window. When it

is not, a pre-trained model from a reference dataset can be deployed with input recalibration on the local aggregate to limit domain shift; cross-dataset transfer remains unreliable when source and target operate on incompatible electrical or measurement regimes (Section 6.1). Fine-tuning serves both as a faster cold-start when a source model is available and as an adaptation mechanism for intra-house drift caused by appliance replacement, occupant habit changes, or seasonal regime shifts (Section 6.3), without requiring a full retraining cycle.

6.6 Answers to Research Questions

Research Question 1. *Can a Temporal Convolutional Network of compact parameter count, trained on the data of a single household, reach a level of disaggregation performance compatible with residential deployment, while remaining executable on consumer-grade hardware?*

The results support a qualified positive answer. The compact causal TCN, with approximately 92 000 parameters, remains well within the Raspberry Pi 4 resource envelope: median inference latency is 50.8 ms, p95 latency reaches 110 ms under realistic LAN access with concurrent Home Assistant load, peak resident memory remains around 113 MB, and total model storage stays below 2 MB (Section 6.4). These values are negligible relative to the 5 s sampling period used in this work.

In terms of accuracy, the model reaches deployment-compatible performance in favourable in-house conditions. On UK-DALE House 2, 28 days of training allow four of five appliances to reach F1 above 0.72, with MAE below 12 W (Section 6.2.1). However, this conclusion is not universal: performance degrades when the training window is not representative of the test usage regime, as observed for the laptop on House 1, or when source and target appliances have incompatible power signatures, as observed for the kettle in cross-dataset transfer to ECO H1 (Sections 6.1 and 6.3).

Research Question 2. *How does the disaggregation accuracy of an in-house-trained model vary with the duration of the recording period and with the synchronisation delay between smart plugs and the aggregate meter, and what data-collection budget can be recommended for residential deployment?*

Two to four weeks of recording, with synchronisation tolerances of a few seconds acceptable for most loads. Disaggregation accuracy scales primarily with the number of activation events rather than calendar duration (Section 6.2.1): where 14 days captures enough events, MAE changes by less than 2 W with additional data; where it does not (washing machine and dishwasher on House 1), yields large

MAE reductions. Synchronisation delays up to 10 s produce bounded degradation on all appliances except the washing machine, whose multi-phase signature is alignment-sensitive (MAE +49% at 10 s on House 1, Section 6.2.2).

Research Question 3. *When a pre-trained appliance model is deployed in a target environment whose appliance signature differs from the source, whether due to a different household or a replaced appliance instance, can fine-tuning on a small sample of target-environment data recover the disaggregation performance of a model trained from scratch on that target?*

Partially. Fine-tuning on a small sample of target-environment data substantially improves poorly transferred models, with most of the gain captured within the first available fine-tuning window (Section 6.3). The effect is strongest when the source model fails under domain shift: on H1→H2, dishwasher MAE drops from 37.4 to 16.7 W and F1 rises from 0.30 to 0.77 with only 22 target activation events. Similar recovery is observed for the kettle and laptop.

However, fine-tuning does not systematically recover the performance of a model trained from scratch on the target household. The washing machine remains a partial exception: fine-tuning can reduce MAE by shifting the model toward a higher-precision, lower-recall regime, but F1 may remain stable or below the in-house baseline. Fine-tuning is therefore best interpreted as an effective adaptation and cold-start mechanism, not as a complete substitute for representative target-house training data.

6.7 Limitations

End-to-end validation restricted to two simple appliances. A limited deployment validation has been carried out on a real installation, restricted to a fan and a lamp : two simple single-phase appliances with steady ON/OFF signatures. This exercise confirms the operational viability of the full software stack (signal acquisition, training, inference, Home Assistant integration), but does not exercise the more challenging appliance categories on which the rest of this work is evaluated, in particular the multi-phase signatures (washing machine, dishwasher) and the low-amplitude appliances exposed to confounders (laptop). Several aspects of full deployment therefore remain unverified : model behaviour on multi-phase appliances under residential conditions, multi-month operation as appliances age and routines shift, and robustness to the heterogeneous Home Assistant configurations encountered in practice.

Operational gaps in the deployment stack. Two operational concerns are deferred to future engineering work. First, the prototype exposes the training age of each model in the management interface but does not automatically detect drift : silent model degradation caused by appliance replacement or seasonal regime changes is left to the user’s attention. Second, the backend API does not implement authentication between the Home Assistant integration and the local service, relying on the assumption that both run within a trusted local network : an assumption increasingly questioned by zero-trust security models given the prevalence of compromised IoT devices on residential networks. Both are tractable engineering extensions but should be addressed before any production deployment of a system that handles consumption-pattern data.

No sum constraint across appliances. The per-appliance seq2point paradigm adopted in this work does not enforce $\sum_i \hat{P}_i(t) \leq P(t)$ at inference time. Per-sample appliance predictions can therefore exceed the measured aggregate. Source-separation alternatives (e.g., Conv-NILM-Net) enforce this constraint but at the cost of joint training and reduced modularity, which conflicts with the per-appliance independence requirements of the present system.

6.8 Threats to Validity

Internal validity. Three factors bound the internal validity of the reported results. First, the temporal segments used for training, fine-tuning and evaluation (Table 5.2) were selected to ensure a minimum number of activations per appliance, which biases the reported metrics towards activation-rich periods rather than arbitrary calendar windows. Second, all experiments are conducted with a single random seed. Third, the cross-house fine-tuning protocol used as a proxy for intra-house brand change introduces a confound, namely the simultaneous shift of household context and appliance instance, which prevents the isolated measurement of brand-change adaptation.

External validity. Three factors limit the generalisation of the reported results beyond the experimental setting. First, the evaluation is conducted on public datasets (UK-DALE especially and ECO for cross house only) representing UK and Swiss residential contexts; results may not directly transfer to households on different electrical infrastructures, with markedly different appliance populations, or in non-European usage patterns. Second, the appliance set is restricted to five categories representing three of the four signature types defined in Section 2.2; Type-

IV always-on appliances (e.g. routers, smoke detectors, set-top boxes in standby) are deliberately excluded as they exhibit no informative temporal structure, but their cumulative contribution to household consumption is not addressed by the proposed system. Third, the held-out evaluation windows are seven days long, which captures short-term performance but does not characterise stability over months of continuous operation; appliance signatures may drift over time as devices age, and this drift is not measured here.

Construct validity. The metrics adopted in this work (MAE, RMSE, SAE, F1 with fixed per-appliance thresholds) are standard in the NILM literature and each captures a distinct failure mode of the disaggregation task. However, two construct-level limitations should be acknowledged. First, none of the regression metrics directly measures the end-user-perceived quality of the system: a household interested in identifying which appliance is responsible for a consumption peak may tolerate a high MAE if the temporal localisation of the activation is correct, while a household interested in monthly energy attribution cares primarily about SAE. The relative importance of these metrics for residential adoption is not addressed in this work. Second, the binary on/off state derived from a fixed power threshold is a simplification of the actual operating modes of multi-state appliances such as washing machines and dishwashers, where intermediate phases (rinsing, idle pauses) fall ambiguously between the on and off categories.

Conclusion validity. The conclusions drawn from the experimental results are subject to two statistical limitations. First, all reported figures correspond to a single experimental run per condition; no confidence interval or significance test is reported, which precludes formal conclusions about whether observed differences between conditions exceed run-to-run variance. This is most consequential when one condition marginally outperforms another, where the direction of the effect is reported but its significance is not established. Second, the comparison with external baselines (Table 5.4) is qualitative rather than quantitative, due to protocol differences documented in Section 5.5; statements such as “competitive with” or “exceeds” should be read as order-of-magnitude observations, not as ranked benchmarks.

Chapter 7

Conclusion

This thesis addressed the gap between the strong performance of NILM methods in academic settings and their limited adoption in real residential environments. Rather than focusing on benchmark accuracy alone, the work investigated the practical obstacles that prevent deployment in ordinary households: domain shift across homes, the scarcity of appliance-level labelled data, and the complexity of installing and operating existing NILM solutions. This thesis should therefore be read as a deployment-oriented NILM system contribution rather than as a proposal for a new disaggregation architecture. The TCN model is used as a compact and causal backbone suitable for the constraints of local residential deployment; the main contribution lies in the integration of training, inference, correction, and fine-tuning into a self-hosted Home Assistant workflow.

To address these limitations, the thesis proposed a complete deployment-oriented NILM system integrated with Home Assistant and supported by a Dockerised local backend. The system was designed to run entirely on consumer-grade hardware, keeping all fine-grained energy data on local hardware rather than on third-party cloud services, thereby reducing the privacy risks associated with cloud-based processing. The remaining security assumptions of the local deployment are discussed in the limitations and perspectives.

At the modelling level, the work adapted a Temporal Convolutional Network-based seq2point architecture for low-frequency active power signals compatible with widely available smart plugs. The proposed workflow enables automatic household-specific data collection and supervised training through smart plugs, allowing users to progressively build appliance models in their own home without manual annotation. In addition, a fine-tuning mechanism was introduced to adapt existing models when an appliance is replaced by another instance with a different power signature.

The experimental evaluation on UK-DALE and ECO established several practical results. On UK-DALE House 2 with 28 days of in-house training, the system reaches deployment-grade accuracy on the kettle, fridge, dishwasher and laptop, with MAE ranging from 2.6 W on the kettle to 11.4 W on the fridge. The washing machine remains the weak case on this house (0.360 F1, 8.8 W MAE) due to its low duty cycle and small training event count. Inference latency on the Raspberry Pi 4 remains below 300 ms at p_{99} under realistic concurrent Home Assistant workload, well within the 5-second budget imposed by the aggregate sampling rate. Cross-house transfer can serve as a cold-start fallback when no appliance meter is available, but the results show that it remains unreliable under strong appliance or dataset shift. Fine-tuning on a small target-house sample, typically within the first available 5–14 day window, recovers a substantial fraction of the cross-house MAE gap on appliances where the source model is weakest, with gains often saturating quickly thereafter. A recurring methodological observation is that event count, not calendar duration, drives disaggregation quality: 14 days of recording is sufficient when the appliance activates frequently, while infrequent appliances such as dishwashers require proportionally longer collection windows.

From this work, a defensible position emerges: in addition to model accuracy, real-world NILM adoption depends critically on the existence of an end-to-end system designed for local, user-accessible deployment. The present work is one practical contribution toward that direction.

7.1 Perspectives

Several directions could extend the work presented in this thesis.

Community model-sharing platform. A first natural extension would be a community repository where users publish appliance models trained in their own households, annotated with appliance brand, model, and relevant household characteristics (country, wiring type, sampling frequency). Other users could then retrieve and deploy a pre-trained model for a matching appliance configuration, bypassing the local training phase entirely. Such a platform would directly address the cold-start limitation of the proposed system and transform individually collected household data into a collective resource.

Context-aware disaggregation. A second avenue concerns the integration of contextual signals available within the home automation infrastructure. In Home Assistant deployments, auxiliary data streams are readily accessible alongside

the aggregate power signal: room occupancy from presence sensors, time-of-day and day-of-week encodings, outdoor temperature from weather integrations, and user-triggered events such as manual appliance activations. Incorporating such contextual features as additional input channels to the TCN could resolve ambiguities that are inherently unresolvable from the power signal alone. This direction aligns with recent work on context-aware NILM [62], where auxiliary signals have been shown to improve disaggregation accuracy particularly for appliances with variable usage patterns.

Backend authentication and zero-trust deployment. A third direction concerns the security boundary between the Home Assistant integration and the local inference backend. The current prototype relies on the assumption that both components run within a trusted local network, which is increasingly difficult to justify given the prevalence of compromised IoT devices on residential networks. Adding mutual authentication between the front-end integration and the backend service, for instance through a locally-generated token exchanged at first pairing and rotated periodically, would close this gap without introducing external dependencies. This is a tractable engineering extension that should be addressed before any production deployment of a system handling fine-grained consumption-pattern data.

NILM-driven energy feedback. A fourth direction would build on the per-appliance estimates produced by the system to deliver actionable energy feedback directly within Home Assistant. Whereas aggregate monitoring only reports how much energy is consumed, the disaggregated output identifies which appliances drive that consumption, opening the way to targeted recommendations: shifting deferrable loads (washing machine, dishwasher) toward off-peak tariff windows, flagging abnormal consumption that may signal a degrading appliance (for instance a fridge whose compressor draws increasing power over time), or attributing cost to individual usage habits. Such feedback could be surfaced through the existing Home Assistant dashboard and automation engine, turning the inference backend from a passive measurement tool into a decision-support layer. This direction is primarily an application of the present work rather than a methodological contribution, but it represents the most direct path from appliance-level inference to a tangible reduction in household consumption.

Use of Generative AI Tools

Claude was used to improve English prose and to translate or rephrase passages originally written in French. GitHub Copilot and Claude were used as AI-assisted development tools for code completion, generation of small code fragments, and code review support. All suggestions were reviewed and validated by the author. The intellectual contributions, system design, software architecture, methodology, experimental design, results, interpretation, and conclusions are entirely the author's own.

Bibliography

- [1] Eurostat, “Energy consumption in households,” 2024. [Online]. Available: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Energy_consumption_in_households
- [2] S. Darby, “The Effectiveness of Feedback on Energy Consumption,” vol. 486.
- [3] C. Fischer, “Feedback on household electricity consumption: A tool for saving energy?” vol. 1, no. 1, pp. 79–104. [Online]. Available: <https://doi.org/10.1007/s12053-008-9009-7>
- [4] G. Hart, “Nonintrusive appliance load monitoring,” *Proceedings of the IEEE*, vol. 80, no. 12, pp. 1870–1891, Dec. 1992.
- [5] J. Breyer, S. Jauhari, R. Glebke, M. H. Alizai, M. Stroot, and K. Wehrle, “Investigating Domain Bias in NILM,” in *Proceedings of the 11th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. ACM, pp. 333–336. [Online]. Available: <https://dl.acm.org/doi/10.1145/3671127.3699532>
- [6] X. Chang, W. Li, Y. Shi, and A. Y. Zomaya, “Taming the Domain Shift in Multi-source Learning for Energy Disaggregation,” in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD ’23. Association for Computing Machinery, pp. 3805–3816. [Online]. Available: <https://dl.acm.org/doi/10.1145/3580305.3599910>
- [7] M. DIncecco, S. Squartini, and M. Zhong. Transfer Learning for Non-Intrusive Load Monitoring. [Online]. Available: <http://arxiv.org/abs/1902.08835>
- [8] S. Chouchene, “Domain Adaptation Methods for Sparse Coding Based Non-Intrusive Load Monitoring.” [Online]. Available: <https://spectrum.library.concordia.ca/id/eprint/993576/>

- [9] J. Xue, Y. Zhang, X. Wang, Y. Wang, and G. Tang. Towards Real-world Deployment of NILM Systems: Challenges and Practices. [Online]. Available: <http://arxiv.org/abs/2409.14821>
- [10] S. Sykiotis, S. Athanasoulas, M. Kaselimi, A. Doulamis, N. Doulamis, L. Stankovic, and V. Stankovic, “Performance-Aware NILM Model Optimization for Edge Deployment,” vol. 7, no. 3, pp. 1434–1446. [Online]. Available: <https://ieeexplore.ieee.org/document/10042478/>
- [11] C. Zhang, M. Zhong, Z. Wang, N. Goddard, and C. Sutton. Sequence-to-point learning with neural networks for nonintrusive load monitoring. [Online]. Available: <https://dl.acm.org/doi/abs/10.5555/3504035.3504353>
- [12] A. Zoha, A. Gluhak, M. Imran, and S. Rajasegarar, “Non-Intrusive Load Monitoring Approaches for Disaggregated Energy Sensing: A Survey,” vol. 12, no. 12, pp. 16 838–16 866. [Online]. Available: <https://www.mdpi.com/1424-8220/12/12/16838>
- [13] M. Kaselimi, E. Protopapadakis, A. Voulodimos, N. Doulamis, and A. Doulamis, “Towards Trustworthy Energy Disaggregation: A Review of Challenges, Methods, and Perspectives for Non-Intrusive Load Monitoring,” vol. 22, no. 15, p. 5872. [Online]. Available: <https://pmc.ncbi.nlm.nih.gov/articles/PMC9371074/>
- [14] J. Z. Kolter and M. J. Johnson, “REDD: A Public Data Set for Energy Disaggregation Research.” [Online]. Available: https://www.researchgate.net/publication/266597071_REDD_A_Public_Data_Set_for_Energy_Disaggregation_Research
- [15] J. Kelly and W. Knottenbelt, “The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes,” vol. 2, no. 1, p. 150007. [Online]. Available: <https://www.nature.com/articles/sdata20157>
- [16] D. Murray, L. Stankovic, and V. Stankovic, “An electrical load measurements dataset of United Kingdom households from a two-year longitudinal study,” vol. 4, no. 1, p. 160122. [Online]. Available: <https://www.nature.com/articles/sdata2016122>
- [17] C. Beckel, W. Kleiminger, R. Cicchetti, T. Staake, and S. Santini, “The ECO data set and the performance of non-intrusive load monitoring algorithms,” in *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, ser. BuildSys ’14. Association for Computing Machinery, pp. 80–89. [Online]. Available: <https://dl.acm.org/doi/10.1145/2674061.2674064>

- [18] A. Monacchi, D. Egarter, W. Elmenreich, S. D'Alessandro, and A. M. Tonello, "GREEND: An energy consumption dataset of households in Italy and Austria," in *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 511–516. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7007698>
- [19] C. Shin, S. Joo, J. Yim, H. Lee, T. Moon, and W. Rhee, "Subtask Gated Networks for Non-Intrusive Load Monitoring," vol. 33, no. 01, pp. 1150–1157. [Online]. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/3908>
- [20] W. Yang, C. Pang, J. Huang, and X. Zeng, "Sequence-to-Point Learning Based on Temporal Convolutional Networks for Nonintrusive Load Monitoring," vol. 70, pp. 1–10. [Online]. Available: <https://ieeexplore.ieee.org/document/9520412/>
- [21] H. Rafiq, P. Manandhar, E. Rodriguez-Ubinas, O. Ahmed Qureshi, and T. Palpanas, "A review of current methods and challenges of advanced deep learning-based non-intrusive load monitoring (NILM) in residential context," vol. 305, p. 113890. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0378778824000069>
- [22] J. Kelly and W. Knottenbelt, "Neural NILM: Deep Neural Networks Applied to Energy Disaggregation," in *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*. ACM, pp. 55–64. [Online]. Available: <https://dl.acm.org/doi/10.1145/2821650.2821672>
- [23] O. Krystalakos, C. Nalmpantis, and D. Vrakas, "Sliding Window Approach for Online Energy Disaggregation Using Artificial Neural Networks," in *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*. ACM, pp. 1–6. [Online]. Available: <https://dl.acm.org/doi/10.1145/3200947.3201011>
- [24] A. Harell, S. Makonin, and I. V. Bajić. Wavenilm: A causal neural network for power disaggregation from the complex power signal. [Online]. Available: <http://arxiv.org/abs/1902.08736>
- [25] Y. Qian, Q. Yang, D. Li, D. An, and S. Zhou, "An Improved Temporal Convolutional Network for Non-intrusive Load Monitoring," in *2021 33rd Chinese Control and Decision Conference (CCDC)*, pp. 2557–2562. [Online]. Available: <https://ieeexplore.ieee.org/document/9601611/>
- [26] M. Alami Chehboune, J. Decock, R. Kaddah, and J. Read, "Conv-NILM-Net, a causal and multi-appliance model for energy source separation," vol. 1753, pp. 207–222. [Online]. Available: <http://arxiv.org/abs/2208.02173>

- [27] S. Bai, J. Z. Kolter, and V. Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. [Online]. Available: <http://arxiv.org/abs/1803.01271>
- [28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. [Online]. Available: <http://arxiv.org/abs/1706.03762>
- [29] Z. Yue, C. R. Witzig, D. Jorde, and H.-A. Jacobsen, “BERT4NILM: A Bidirectional Transformer Model for Non-Intrusive Load Monitoring,” in *Proceedings of the 5th International Workshop on Non-Intrusive Load Monitoring*, ser. NILM’20. Association for Computing Machinery, pp. 89–93. [Online]. Available: <https://dl.acm.org/doi/10.1145/3427771.3429390>
- [30] S. Sykiotis, M. Kaselimi, A. Doulamis, and N. Doulamis, “ELECTRICity: An Efficient Transformer for Non-Intrusive Load Monitoring,” vol. 22, no. 8. [Online]. Available: <https://www.mdpi.com/1424-8220/22/8/2926>
- [31] A. Petralia, P. Charpentier, Y. Kadhi, and T. Palpanas, “NILMFormer: Non-Intrusive Load Monitoring that Accounts for Non-Stationarity,” in *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 4761–4772. [Online]. Available: <http://arxiv.org/abs/2506.05880>
- [32] S. van den Oord, Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. [Online]. Available: <http://arxiv.org/abs/1609.03499>
- [33] F. Yu and V. Koltun. Multi-Scale Context Aggregation by Dilated Convolutions. [Online]. Available: <http://arxiv.org/abs/1511.07122>
- [34] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. [Online]. Available: <https://ieeexplore.ieee.org/document/7780459/>
- [35] D. Hendrycks and K. Gimpel. Gaussian Error Linear Units (GELUs). [Online]. Available: <http://arxiv.org/abs/1606.08415>
- [36] B. Giresha, L. N. S. Varanasi, V. M. Krishna, and S. Anakal, “A deep learning approach to energy disaggregation using TCN layers for appliance and load insights,” vol. 45, no. 2, pp. 102–113. [Online]. Available: <https://hrcak.srce.hr/335594>

- [37] C. Klemenjak, A. Faustine, S. Makonin, and W. Elmenreich. On Metrics to Assess the Transferability of Machine Learning Models in Non-Intrusive Load Monitoring. [Online]. Available: <http://arxiv.org/abs/1912.06200>
- [38] L. Torgo and R. Ribeiro, “Utility-Based Regression,” pp. 597–604.
- [39] P. Branco, L. Torgo, and R. Ribeiro. A Survey of Predictive Modelling under Imbalanced Distributions. [Online]. Available: <http://arxiv.org/abs/1505.01658>
- [40] Y. Shang, X. Gao, and X. Zhao, “Imbalanced data In Energy Disaggregation,” in *CIBDA 2022; 3rd International Conference on Computer Information and Big Data Applications*, pp. 1–4. [Online]. Available: <https://ieeexplore.ieee.org/document/9899096/>
- [41] D. Precioso and D. Gómez-Ullate, “NILM as a regression versus classification problem: The importance of thresholding,” vol. 79, no. 13, pp. 14 039–14 062. [Online]. Available: <http://arxiv.org/abs/2010.16050>
- [42] T. Bogaerts, S. Jacobs, S. Ghane, F. Van Riet, W. Casteels, S. Mercelis, I. Verhaert, and P. Hellinckx, “Disaggregation of digital meter data for synthetic load profile generation,” ser. Building Simulation, vol. 17. IBPSA, pp. 1271–1278. [Online]. Available: https://publications.ibpsa.org/conference/paper/?id=bs2021_30236
- [43] N. Batra, J. Kelly, O. Parson, H. Dutta, W. Knottenbelt, A. Rogers, A. Singh, and M. Srivastava, “NILMTK: An Open Source Toolkit for Non-intrusive Load Monitoring,” in *Proceedings of the 5th International Conference on Future Energy Systems*, pp. 265–276. [Online]. Available: <http://arxiv.org/abs/1404.3878>
- [44] N. Batra, R. Kukunuri, A. Pandey, R. Malakar, R. Kumar, O. Krystalakos, M. Zhong, P. Meira, and O. Parson, “Towards reproducible state-of-the-art energy disaggregation.” New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: <https://doi.org/10.1145/3360322.3360844>
- [45] J. Winkler, H. Bousbiat, S. Jost, and W. Elmenreich, “Energy Disaggregation with NILM on a Raspberry Pi with Smart-Metering Extension,” in *2023 2nd International Conference on Power Systems and Electrical Technology (PSET)*. IEEE, pp. 191–195. [Online]. Available: <https://ieeexplore.ieee.org/document/10346408/>
- [46] C. Klemenjak, S. Jost, and W. Elmenreich, “YoMoPie: A User-Oriented Energy Monitor to Enhance Energy Efficiency in Households,” in *2018 IEEE*

- Conference on Technologies for Sustainability (SusTech)*. IEEE, pp. 1–7. [Online]. Available: <https://ieeexplore.ieee.org/document/8671331/>
- [47] D. Serna, C. Arias, T. Manrique, A. Guerrero, and J. Sierra, “Assessing Computational Resources and Performance of Non-Intrusive Load Monitoring (NILM) Algorithms on Edge Computing Devices,” vol. 18, no. 22. [Online]. Available: <https://www.mdpi.com/1996-1073/18/22/5991>
 - [48] E. Tabanelli, D. Brunelli, A. Acquaviva, and L. Benini, “Trimming Feature Extraction and Inference for MCU-based Edge NILM: A Systematic Approach,” vol. 18, no. 2, pp. 943–952. [Online]. Available: <http://arxiv.org/abs/2105.10302>
 - [49] Sense - Control your home energy use. [Online]. Available: <https://sense.com/homes/>
 - [50] Installing the Sense Home Monitor. [Online]. Available: <https://help.sense.com/hc/en-us/articles/206400938-Installing-the-Sense-Home-Monitor>
 - [51] Privacy - French. [Online]. Available: <https://sense.com/privacy-fr/>
 - [52] Looking Ahead: The Next Chapter for Sense. [Online]. Available: <https://sense.com/consumer-blog/looking-ahead-the-next-chapter-for-sense/>
 - [53] J.-M. Verhalle. How does Smappee’s NILM technology work? [Online]. Available: <https://www.smappee.com/blog/smappee-appliance-recognition/>
 - [54] Designed to Empower. [Online]. Available: <https://www.smappee.com/>
 - [55] H. Assistant. Smappee. [Online]. Available: <https://www.home-assistant.io/integrations/smappee/>
 - [56] Home Assistant. [Online]. Available: <https://www.home-assistant.io/>
 - [57] Integration architecture | Home Assistant Developer Docs. [Online]. Available: https://developers.home-assistant.io/docs/architecture_components
 - [58] Entities: Integrating devices & services | Home Assistant Developer Docs. [Online]. Available: <https://developers.home-assistant.io/docs/architecture/devices-and-services>
 - [59] Sensor entity | Home Assistant Developer Docs. [Online]. Available: <https://developers.home-assistant.io/docs/core/entity/sensor/>
 - [60] Fetching data | Home Assistant Developer Docs. [Online]. Available: https://developers.home-assistant.io/docs/integration_fetching_data/

- [61] History - Home Assistant. [Online]. Available: <https://www.home-assistant.io/integrations/history/>
- [62] K. Chen, Y. Zhang, Q. Wang, J. Hu, H. Fan, and J. He, "Scale- and Context-Aware Convolutional Non-intrusive Load Monitoring," vol. 35, no. 3, pp. 2362–2373. [Online]. Available: <http://arxiv.org/abs/1911.07183>

Chapter A

Loss comparison

Table A.1: Ablation of $\mathcal{L}_{\text{power}}$ on UK-DALE House 1 (28-day training window, followed by 7day test window, delay of 5 s). *Eq. 8* refers to Shin’s strict formulation $\mathcal{L}_{\text{gated}} + \mathcal{L}_{\text{state}}$ ($\lambda_{\text{power}} = 0$). *Three-term* refers to the full objective described in Section 3.7.2. All hyperparameters other than λ_{power} are identical (window 518, batch size, learning rate, dropout 0.2, seed 42). MAE and RMSE in watts; SAE normalised. Best result per metric per appliance in **bold**.

Appliance	Loss	Events	MAE	RMSE	SAE	F1	P	R
Kettle	Eq. 8	177	11.1	136.5	0.200	0.828	0.864	0.795
	Three-term	177	11.4	121.9	0.111	0.826	0.764	0.900
Fridge	Eq. 8	716	22.5	43.0	0.037	0.801	0.750	0.861
	Three-term	716	15.8	38.3	0.059	0.864	0.837	0.894
Washing machine	Eq. 8	2510	11.9	89.6	0.072	0.880	0.891	0.869
	Three-term	2510	9.9	78.0	0.015	0.923	0.904	0.942
Dishwasher	Eq. 8	87	9.0	102.7	0.021	0.407	0.817	0.271
	Three-term	87	6.7	61.3	0.150	0.610	0.853	0.475

Chapter B

Home Assistant Integration: User Workflow

This appendix details the end-user workflow of the NILM integration described in Chapter 4. It documents the configuration flow, the appliance representation, the training and fine-tuning controls, and the appliance parameter management, with screenshots of the deployed component. The content is intended as a usage reference for readers wishing to understand the day-to-day operation of the system, and is not required for following the architectural arguments of Chapter 4.

B.1 First Configuration

The integration is installed like any other Home Assistant custom component, by placing the component directory in the `custom_components` folder and restarting Home Assistant. Once detected, the user initiates the configuration through the standard integration setup interface, which launches the config flow described in Section 4.1.1.

×

NILM Backend Connection

?

Connect to your NILM Docker container running the FastAPI backend.

Host*

localhost

IP address or hostname of the NILM Docker container.

Port*

8000

API port of the NILM backend (default 8000).

Power sensor entity

Select an entity *

▼

Home Assistant entity providing aggregate power in Watts.

Scan interval (seconds)*

30

s

Seconds between prediction cycles.

data_push_interval*

24

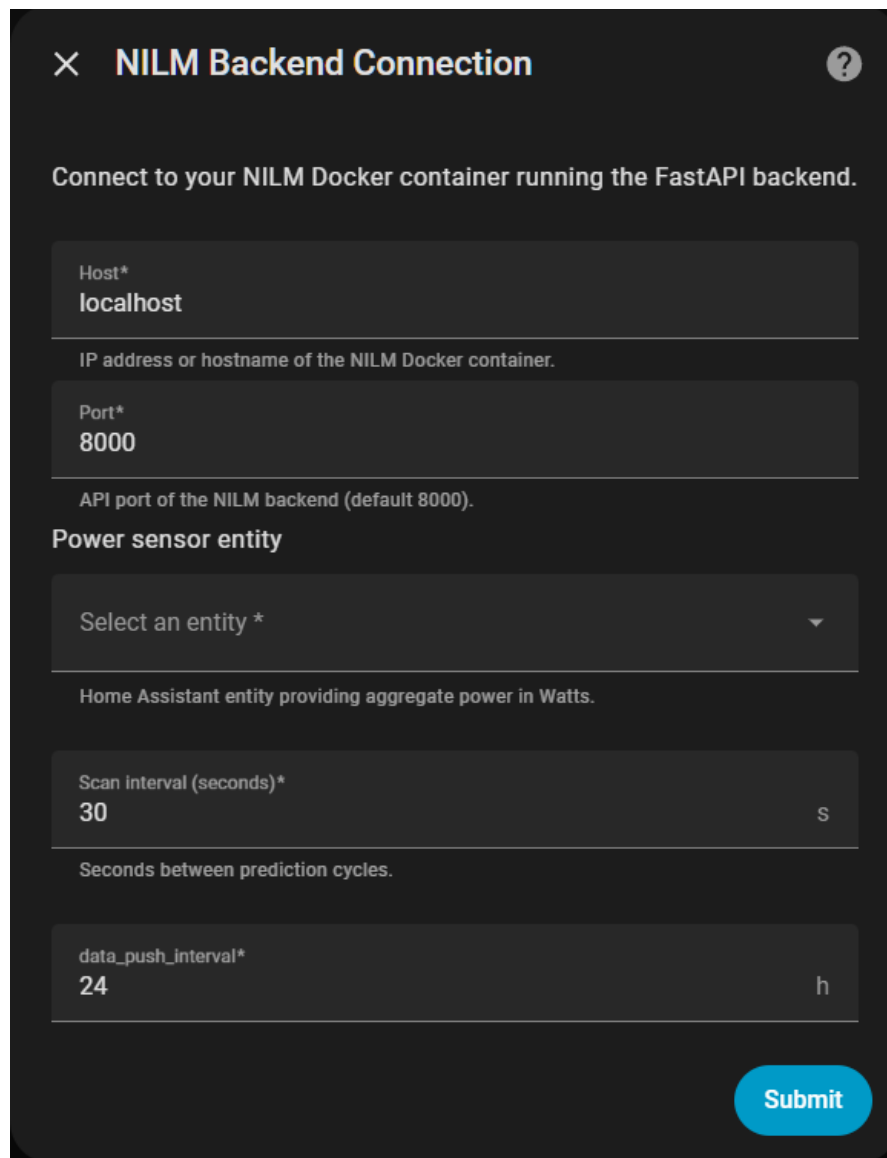
h

Submit

Figure B.1: First configuration step : backend connection parameters.

The first step, illustrated in Figure B.1, collects the backend connection parameters. The user must provide the host address and port of the NILM Docker container, select the Home Assistant entity that reports the aggregate power consumption, and set the scan interval (in seconds) and the data push interval (in hours). After submission, the flow attempts a connection to the backend ; if it succeeds, the sampling rate and expected window size are retrieved and stored in the config entry.

The second step prompts the user to select an existing model from the backend or to create a new one by providing a name, as shown in Figure B.2. The third and final step, shown in Figure B.3, allows the user to select which appliances from the chosen model should be monitored. Only appliances present in the target household need to be selected ; the others can be deselected and excluded from the integration. Once submitted, the config entry is created and the integration becomes fully operational.



The screenshot shows a configuration window titled "NILM Backend Connection" with a close button (X) and a help icon (?). The instruction reads: "Connect to your NILM Docker container running the FastAPI backend." The form contains the following fields:

- Host***: A text input field containing "localhost". Below it is the label "IP address or hostname of the NILM Docker container."
- Port***: A text input field containing "8000". Below it is the label "API port of the NILM backend (default 8000)."
- Power sensor entity**: A section header followed by a dropdown menu labeled "Select an entity *". Below the dropdown is the label "Home Assistant entity providing aggregate power in Watts."
- Scan interval (seconds)***: A text input field containing "30" with a unit "s" on the right. Below it is the label "Seconds between prediction cycles."
- data_push_interval***: A text input field containing "24" with a unit "h" on the right.

A blue "Submit" button is located at the bottom right of the form.

Figure B.2: Second configuration step : model selection.

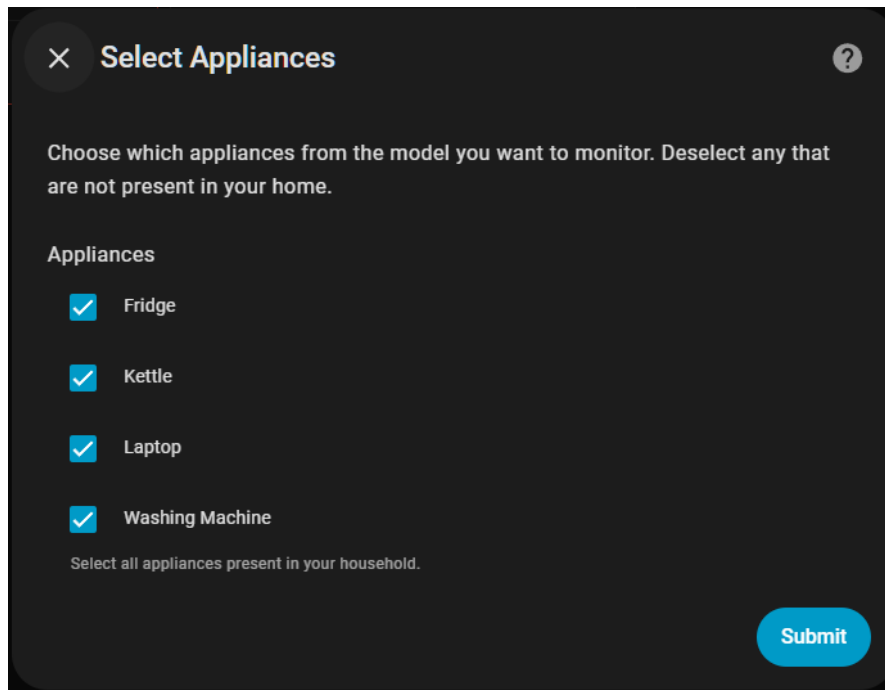


Figure B.3: Third configuration step : appliances selection.

B.2 How Appliances Are Represented

Once configured, each selected appliance appears in Home Assistant as a dedicated device grouping two entities : a power sensor and a binary sensor as described in section 4.1.2. Figure B.4 shows an example for a lamp.

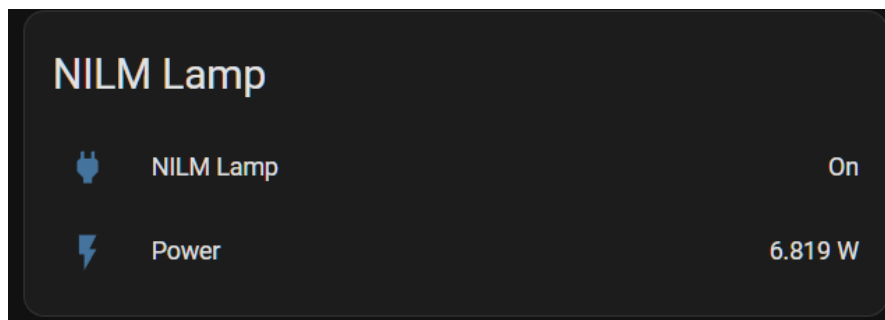


Figure B.4: Appliance card : Binary Sensor and Power Sensor.

All appliance devices, along with the NILM Hub device, are listed under a

single integration entry in the Home Assistant integrations panel, as shown in Figure B.5. This provides a unified view of all monitored appliances and their associated entities, without any manual entity registration required from the user.

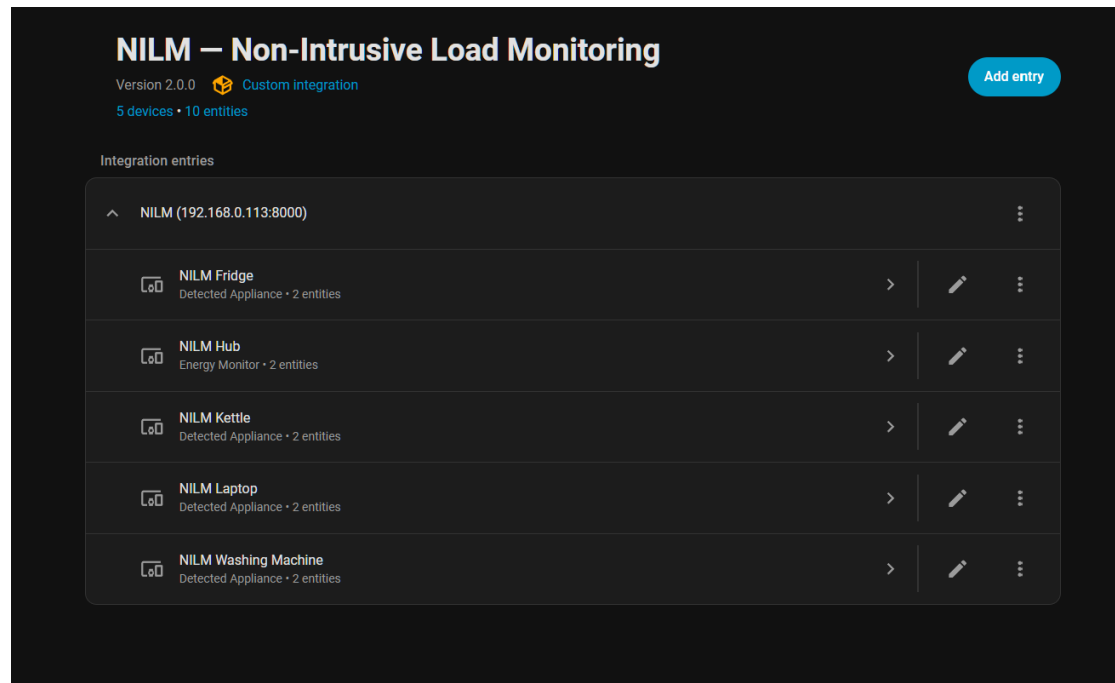


Figure B.5: Appliances listed under the NILM Hub Device.

The complete list of active appliances is also accessible directly from the management panel, through the Appliances tab shown in Figure B.6. For each appliance, the panel displays the current estimated power consumption and the on/off state derived from the configured threshold. A summary bar at the top of the panel reports the total number of monitored appliances, the sum of all predicted powers, and the current real aggregate power, allowing the user to quickly assess how well the model accounts for the measured consumption.

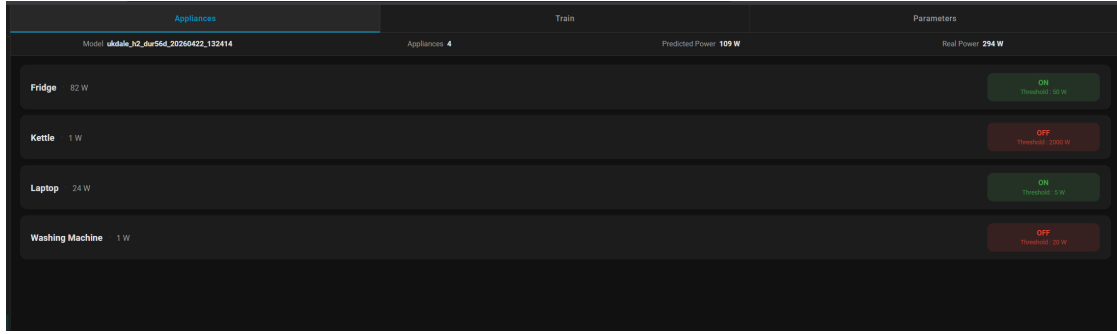


Figure B.6: Appliances listed in the Dashboard.

B.3 Training

The Train tab of the management panel, shown in Figure B.7, exposes the controls needed to initiate training and fine-tuning operations. At the top, the panel reports the total number of samples currently available on the backend, along with the date range they cover. The user can manually trigger a data push for a custom time interval using the *Send Data* button, or clear all backend data using *Remove All Data*.

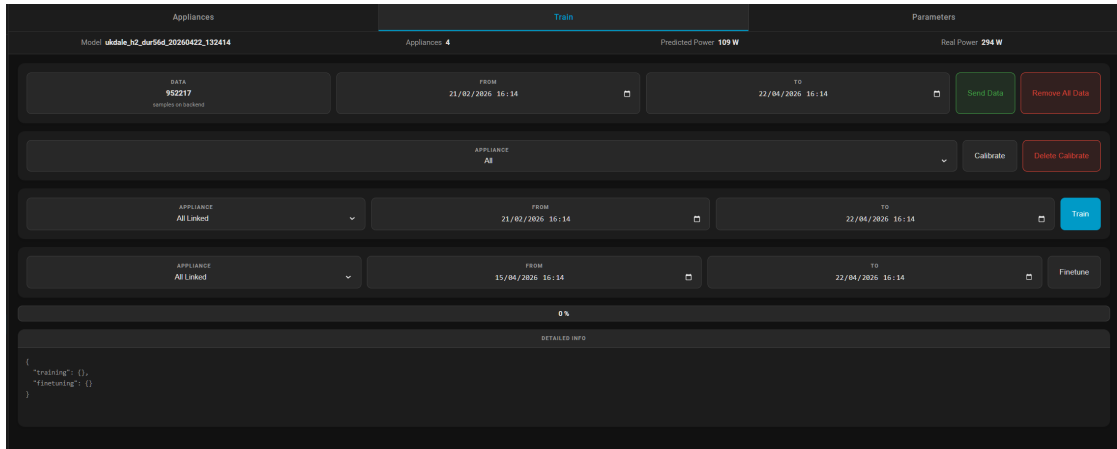


Figure B.7: Train tab of the Dashboard.

The calibration row recomputes the input-normalisation statistics (agg_mean, agg_std) from the aggregate-power samples already stored on the backend. This adapts a pretrained appliance model to the household's specific power distribution without retraining; thresholds are unaffected and remain editable in the Parameters

tab. It works either for a single appliance or for all existing appliances at once. Training and fine-tuning are each triggered independently, with a configurable appliance scope and date range. A progress bar and a detailed JSON status panel provide feedback during long-running operations.

In practice, the recommended workflow is to first send data covering a sufficient period (or wait some days), set the thresholds, and then launch training. Fine-tuning can be applied subsequently to adapt a pretrained model to the specific consumption patterns of the target household, or to adapt to an appliance brand change.

B.4 Appliance Parameters

The Parameters tab, shown in Figure B.8, provides per-appliance management controls. At the top, a form allows the user to add a new appliance to the model by specifying its name and an initial on/off threshold in watts.

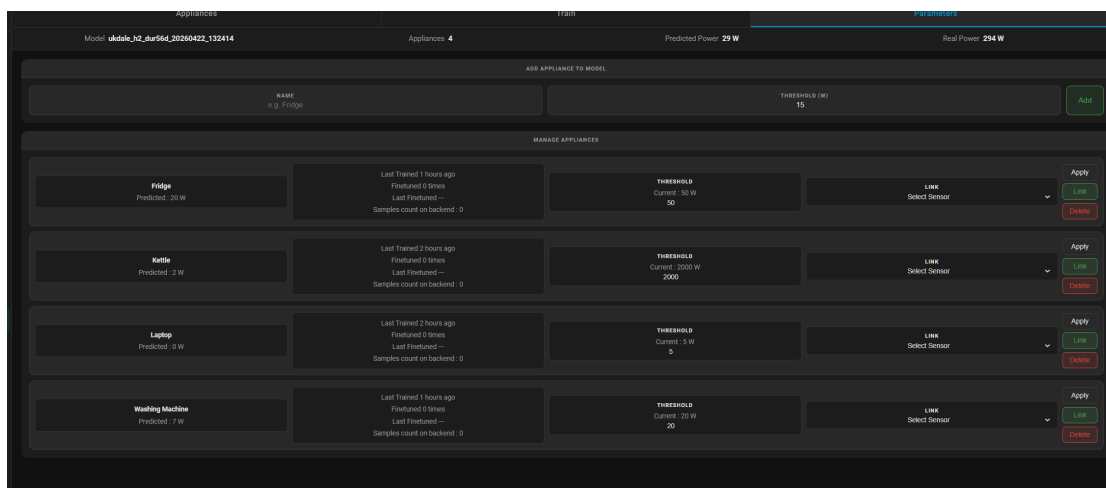


Figure B.8: Parameters tab of the Dashboard.

Below, each existing appliance is listed with its current predicted power, training metadata (last trained, number of fine-tuning cycles, last fine-tuned date, and sample count on the backend), and two editable fields. The *Threshold* field controls the watt value above which the appliance is considered active ; it can be updated and applied independently for each appliance. The *Link* field allows the user to associate the appliance with a real Home Assistant sensor entity, which is necessary for the data export pipeline to include labeled measurements for that appliance.

Each row also exposes a *Delete* button to permanently remove the appliance from the model.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl