Spatial Temporal Graph Neural Networks for Human Activity Recognition and Prediction

Temiloluwa Aina
anxtem001@myuct.ac.za
University of Cape Town

ABSTRACT

Human Activity Recognition (HAR) and Prediction (HAP) in smart homes support applications ranging from automated assistance to healthcare monitoring. While deep learning models such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs) have achieved strong performance, they do not explicitly capture sensor-to-sensor relationships. This study benchmarks two Spatial-Temporal Graph Neural Networks, Graph WaveNet (GWN) and Adaptive Graph Convolutional Recurrent Network (AGCRN), against CNN and LSTM baselines on the CASAS Aruba and Milan datasets. To address the challenges of irregular, event-driven sensor data, we develop temporal feature engineering methods using time-delta representations. Results show that STGNNs achieve competitive accuracy but do not consistently surpass the baselines; in particular, AGCRN matches CNN and LSTM performance but with substantially higher computational cost. On the more difficult HAP task, all models exhibit limited performance, especially on minority classes. Analysis of the learned graph structures reveals semantically meaningful inter-room activity flows, highlighting the interpretability advantages of STGNNs. Overall, STGNNs provide competitive accuracy and valuable insights into activity dynamics but do not yet demonstrate clear superiority over traditional models. Future research should prioritise improved temporal representations tailored to irregular, event-driven data.

KEYWORDS

Human Activity Recognition, Smart Homes, Deep Learning, Graph Neural Networks, Time Series Analysis

1 INTRODUCTION

Human Activity Recognition (HAR) uses sensor data to automatically classify a person's actions [1]. It spans different modalities, including cameras (vision-based HAR) and wearables such as smartwatches (wearable HAR), but this work focuses on smart homes equipped with unobtrusive ambient sensors. These systems interpret activations from passive infrared (PIR) motion detectors, door contacts, and temperature sensors to identify daily activities. For example, motion and door sensor activations in a kitchen may together indicate "Cooking."

Accurate HAR enables intelligent environments with broad applications. In smart homes, HAR can improve comfort and energy efficiency by adjusting lighting, climate control, or powering down unused rooms. In healthcare, it supports independent living by monitoring Activities of Daily Living (ADLs) and detecting anomalies such as prolonged inactivity or unusual routines [2].

Approaches to HAR have progressed from classical machine learning methods like Decision Trees and Bayesian Networks, which required extensive manual feature engineering [3], to deep neural networks (DNNs) such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs). These models automatically extract temporal and contextual features from raw sensor streams and achieve strong performance [4]. However, they generally ignore the spatial layout of sensors, leaving sensor-to-sensor relationships to be inferred implicitly, which may mean missing out on potential performance gains from learning sensor relationships.

Human Activity Prediction (HAP) extends HAR by forecasting future actions. We adopt the activity-transition formulation [5], where models predict the next distinct activity (e.g., forecasting "Eating" after "Cooking") rather than activity occurrence at fixed time intervals. The focus is on learning the sequence of daily activities and the sensor activation patterns associated with transitions. This is valuable because it tests whether models capture the underlying structure of human routines, which is essential for proactive applications such as assistive healthcare and smart home automation.

Spatial-Temporal Graph Neural Networks (STGNNs) have emerged as a promising alternative to CNNs and LSTMs. Unlike traditional DNNs, STGNNs jointly model temporal dynamics and spatial relationships within sensor networks [6]. In smart homes, sensors can be represented as graph nodes with edges capturing spatial proximity (e.g., sensors in the same room) or activity-driven connections (e.g., frequent transitions between kitchen and dining room). By learning these relationships from data, STGNNs can capture complex dependencies in human activity. They have shown strong performance in domains such as traffic forecasting [6], weather prediction [7], and, more recently, HAR [8].

This research investigates whether two STGNN architectures, Graph WaveNet (GWN) [9] and Adaptive Graph Convolutional Recurrent Network (AGCRN) [10], originally developed for traffic forecasting, are applicable to HAR and HAP. We benchmark them against CNN and LSTM baselines, which are well-established and strong performers. The study has three objectives:

(1) Develop temporal feature engineering methods that transform irregular, event-driven smart home sensor data into representations suitable for STGNN learning.
(2) Evaluate whether Graph WaveNet and AGCRN outperform CNN and LSTM baselines on HAR, using weighted average F1 score as the primary metric.
(3) Evaluate whether the STGNNs outperform the baselines on the more complex HAP task, again using weighted average F1 score as the primary metric.

An additional motivation for using STGNNs is that they base their classifications on learned graph structures, which provide partial transparency into how the model makes its classifications and

offer greater interpretability than conventional DNNs. In safety-critical applications such as smart homes, accuracy alone is insufficient, since users must also be able to establish trust in the model's outputs. By examining the learned graph structures, we can evaluate whether the model has captured sensible activity dynamics, which is essential for establishing such trust.

2 BACKGROUND AND RELATED WORKS

2.1 The Supervised Learning Problem

HAR is typically formulated as a supervised, multi-class classification problem. The objective is to train a model, F, that can map a window of sensor readings to a corresponding activity label. The input to the model can be formatted as a matrix of sensor data S, where the columns represent distinct sensors and the rows represent timestamps.

$$S = \begin{pmatrix} S_{0,1} & S_{0,2} & \dots & S_{0,n} \\ \vdots & \vdots & \ddots & \vdots \\ S_{t,1} & S_{t,2} & \dots & S_{t,n} \end{pmatrix}$$

Here, $S_{i,k}$ is the reading at time i from sensor k, for n sensors over t time steps. The model's task is to, for a given window of sensor activations S, produce an activity label $\hat{A} \in A = \{A_1, \dots, A_k\}$ for that window. A is a set of predefined activity labels (e.g., "Sleeping"). The model is trained by minimising a loss function, typically cross-entropy, which quantifies the discrepancy between the model's classification of the activity for the window and the ground truth activity. In the subsections that follow, we will provide an overview of previous work that has been done to find a suitable model F and some newer model architectures that can be applied to HAR.

2.2 Evolution of Deep Models For HAR

2.2.1 Convolutional Neural Networks. Convolutional Neural Networks (CNNs) are widely used in HAR to automatically learn features from raw sensor data [2]. They apply convolution filters to detect local patterns, treating sensor data as temporal sequences or images. CNNs naturally capture nearby dependencies and are robust to time shifts, making them effective for recognising activities [11]. Prior work has shown their versatility: converting sensor sequences into images for 2D CNNs [12, 13], or applying 1D CNNs directly to raw timelines with high accuracy [14]. These approaches highlight CNNs' strong feature extraction capabilities across different input formats.

2.2.2 Long Short-Term Memory Networks. Long Short-Term Memory networks (LSTMs) are a form of Recurrent Neural Network that use gating mechanisms to retain or discard information over time. This enables them to capture long-range dependencies, such as recognising that a fridge door opening followed by stove activation suggests cooking. LSTMs have achieved strong results in HAR, with studies showing that bidirectional LSTMs significantly outperform traditional sequence models like Hidden Markov Models and Conditional Random Fields [15].

2.3 The Spatial Temporal Graph Neural Network

While early deep learning models advanced performance, they largely ignored the topology of sensor networks, forcing them to learn sensor inter-dependencies implicitly. To address this recent research has shifted to STGNNs [8]. These models represent the environment as a graph, $G = (V, E)$, where sensors are nodes (V) and their relationships are edges (E), allowing them to learn patterns across both spatial and temporal dimensions simultaneously. The efficacy of this approach was demonstrated by P et al. [8], who achieved high overall weighted average F1 scores, ranging from 0.783 to 0.924 on several CASAS datasets.

While their model achieves strong performance, its graph construction relies on a critical heuristic. The graph topology is determined by pruning a fully connected graph to retain the top k neighbours for each sensor, where k is a fixed hyperparameter. The authors identify this as a significant limitation, noting the model's sensitivity to this value and suggesting that future work should overcome the reliance on a predefined k. This limitation motivates our investigation of the GWN and AGCRN architectures, which are designed to learn the graph structure adaptively during the training process.

2.3.1 Graph WaveNet. GWN is an architecture that integrates Graph Convolutional Networks (GCNs) for spatial analysis with Gated Temporal Convolution Networks (TCNs) for temporal modelling. A key innovation in its GCN module is a self-adaptive adjacency matrix that is learned directly from the data [9]. While this removes the need for a predefined graph, the model can optionally incorporate such prior structural information. The GCN extracts spatial dependencies by aggregating features from neighbouring nodes. The Gated TCN module leverages one-dimensional convolutions to capture long-term temporal patterns. GWN has demonstrated strong predictive performance across diverse domains, including traffic flow forecasting [9], stock market analysis [16], and weather prediction [7].

2.3.2 Adaptive Graph Convolutional Recurrent Network. The Adaptive Graph Convolutional Recurrent Network integrates Graph Convolutional Networks (GCNs) with Gated Recurrent Units (GRUs) to dynamically model spatiotemporal dependencies without requiring a predefined graph structure [10]. The model leverages two primary components to achieve this. First, a Data Adaptive Graph Generation (DAGG) module is employed to infer spatial patterns by learning latent relational structures and hidden interconnections among nodes. Second, a Node Adaptive Parameter Learning (NAPL) module captures the temporal dynamics by learning unique, node-specific parameters for each time period, thereby accounting for the distinct evolution of each entity in the network. When applied to traffic flow prediction, AGCRN has demonstrated superior performance compared to traditional STGNNs that rely on predefined graph structures [10].

2.4 Segmentation Strategies For HAR

A foundational step in sensor-based HAR involves segmenting continuous data streams into discrete windows for subsequent classification. The choice of segmentation, or windowing, strategy is

particularly critical, as it directly affects how well models can capture activity boundaries and represent meaningful patterns in the data. As surveyed by Bouchabou et al. [11], researchers have investigated several approaches, primarily distinguished by whether the window size is fixed or dynamic. The main techniques include fixed-size Time Windows and Sensor Event Windows, alongside variable-size methods like Explicit Windowing and Dynamic Windows.

Each method presents distinct trade-offs. Fixed-size windows, such as Time Windows (which segment the data stream into intervals of a fixed time duration) and Sensor Event Windows (which create windows containing a fixed number of sensor readings), are simple to implement but face the challenge of selecting an optimal window size. A window that is too small may lack the context needed to make an accurate classification, while one that is too large can obscure the transitions between activities [11]. In contrast, Explicit Windowing, a variable-size method, segments data by grouping all sensor readings for a single activity into one window, making it impractical for real-time applications. This impracticality arises because the method requires the system to know the exact start and end of each activity beforehand, which is not feasible in real-world scenarios. Dynamic Windows attempt to offer more flexibility by using offline-trained rules to determine segment boundaries online, but this approach can be time-consuming to develop, reliant on expert input, and less effective for complex activities [17].

2.5 Human Activity Prediction

Du et al. [5] conceptualised HAP as a time-sequence problem, leveraging LSTMs to forecast an inhabitant's future activities based on past behaviour. The authors' approach was predicated on the idea that daily activities follow relatively fixed patterns. A crucial step in their methodology was a data preprocessing stage where repeated instances of the same activity were merged into a single event. This forced the LSTM to learn the transitions between distinct activities rather than a simple continuation of an ongoing one. The input for the model was a sequence of the three most recent activities, which the authors found to be the optimal length for prediction.

3 EXPERIMENTAL DESIGN

3.1 Datasets

For this study, we utilised two publicly available datasets from the Centre of Advanced Studies in Adaptive Systems (CASAS) smart home project: Aruba [18] and Milan [19]. Both datasets collect data from three sensor types, namely PIR motion, door and temperature sensors. More details about each dataset are given in the table below.

Table 1: Characteristics of the CASAS Datasets

Characteristics	Milan	Aruba
Number of Days	82	219
Participants	Single resident & pet	Single resident
Number of Sensors	33	39
Number of Activities	16	12

The Milan dataset offers a more challenging evaluation scenario than Aruba. Its significantly smaller size and the presence of a pet, which generates confounding sensor activations, create a difficult learning environment. Consequently, Milan is an ideal benchmark for assessing a model's robustness and generalisation capabilities in complex, real-world conditions. Detailed floorplans of both dataset environments are provided in the Appendix.

3.2 Data Preprocessing

3.2.1 Data Cleaning and Structuring. Our data preprocessing began with a cleaning phase, similar to the procedure used by Bouchabou et al. [20], that targeted two primary anomalies in the raw data:

- **Duplicate Data:** Redundant sequences of sensor activations were identified and removed.
- **Chronological Inconsistencies:** Sensor activations that did not appear in chronological order were resorted to be temporally correct.

The next step was to put the data in a format suitable for modelling. The raw dataset is event-driven, meaning that at any given timestamp, a reading is typically recorded from a single sensor, and the intervals between these readings are irregular. A snippet of the data is provided in Figure 15 in the Appendix. This data structure makes preprocessing approaches used in works like Licotti et al. [15], which are suitable for traditional DNNs such as LSTMs and CNNs, inappropriate here. Their method involves a direct numerical encoding of the raw event stream, which is not compatible with STGNNs that require structured inputs with readings from all sensors at regular timestamps. Although P et al. [8] employed an STGNN to the task of HAR, their study does not specify the exact shape of the model's input or the nature of the features used, which prevents a direct replication of their methodology. We thus decided to implement our own preprocessing approach, detailed below, to effectively structure the data for STGNNS.

The data is first processed sequentially to prepare it for modelling. Categorical sensor states (e.g., "ON", "OFF") are converted to numeric values (e.g., -1.0, 1.0). The continuous temperature sensor readings are normalised using Z-score normalisation, which transforms data to have zero mean and unit variance by subtracting the mean and dividing by the standard deviation. Activities in the datasets are demarcated by "begin" and "end" markers. During preprocessing, each sensor reading is assigned the label of the activity that was most recently initiated. Any sensor readings that fall outside of a defined activity block are categorised as "Other".

Following this, the event stream is structured into a feature matrix where rows correspond to event timestamps and columns represent each sensor. To address the issue of missing readings for sensors that were not active at a given timestamp, we apply forward imputation, a technique that follows from the work of P et al. [8], carrying the last known value of a sensor forward.

3.2.2 Temporal Feature Engineering. A unique challenge of smart home data is its **irregular timing**: events occur only when sensors are triggered, leaving highly uneven intervals between readings. We initially attempted to resolve this by taking readings at fixed 1-second intervals, regardless of sensor changes, and carrying observations forward. However, this approach led to poor model performance (see Section 4.3). We therefore decided to explicitly

encode the temporal information through additional node features, which allows the models to understand the irregular progression of time.

For each sensor at every timestamp, three distinct features are engineered:

(1) The numerical value of the sensor itself.
(2) The log-normalised time elapsed since the previous event from any sensor, capturing the overall pace of events.
(3) The log-normalised time elapsed since the last activation of that specific sensor, indicating its individual recency.

Log-normalisation is applied to these time-based features to handle their wide-ranging and skewed distribution. Time gaps between sensor events can vary from less than a second to many hours, and this large variance can make it difficult for the model to learn effectively. By taking the logarithm of these values, we compress this range and reduce the impact of extreme outliers while preserving the relative ordering of time intervals. This transformation also means that for very large time gaps, the model learns that they are large without needing to distinguish between specific magnitudes, resulting in more stable and manageable features for processing.

Worked Example. Consider the following three raw events:

```
2009-10-16 00:01:04   M017   ON
2009-10-16 00:01:06   M009   ON
2009-10-16 00:01:07   M017   OFF
```

After preprocessing into the feature format, with only two sensors shown (M017 and M009), the feature matrix would appear as:

Table 2: Example of temporal feature engineering

Timestamp	M017 Value	Δt_{global}	Δt_{M017}	M009 Value	Δt_{global}	Δt_{M009}
00:01:04	1.0	0	0	0.0	0	∞
00:01:06	1.0	$\log(2)$	$\log(2)$	1.0	$\log(2)$	0
00:01:07	-1.0	$\log(1)$	$\log(3)$	1.0	$\log(1)$	$\log(1)$

Here, Δt_{global} is the time since the last event from any sensor, and Δt_{sensor} is the time since the last activation of that particular sensor.

Final Representation. This feature matrix is then segmented using a sliding sensor event window. Each window is assigned a single, representative activity label by majority vote, with ties resolved by the most recent label in the window (see Appendix, Figure 16).

The outcome of this preprocessing is a feature tensor X, and a corresponding label vector Y. The tensor X has a four-dimensional shape of (windows, steps, sensors, features) where windows is the number of windows generated from the dataset, steps is the length of the sliding window, sensors is the number of sensors (33 for Milan and 39 for Aruba) and features is the number of features per sensor (three). The label vector Y is a one-dimensional tensor of shape (windows), where each element contains the integer-encoded activity label for its corresponding window X.

A full overview of this end-to-end preprocessing and modelling pipeline is provided in Figure 14 in the Appendix.

3.2.3 Modification for HAP. For the activity prediction task, the feature engineering process remains identical, but the method for assigning labels is modified to shift the objective from recognition to forecasting. While the input window X is constructed in the same way, its corresponding label Y is no longer the most frequent activity within that window. Instead, the framework first identifies the window's primary activity using the majority voting rule. It then searches forward in time from the end of the window to find the very first sensor reading associated with a different activity. The label of this next, distinct activity becomes the ground truth target for the current window. This reframes the learning task: the model architecture and input features are unchanged, but the objective is now to predict, given the events in the current window, what the next new activity will be.

3.3 Model Architectures

We implemented four deep learning models for recognising and predicting activities. In all cases, the input is a window of sensor readings arranged as a 4D tensor with shape (windows, steps, sensors, features) as described in the previous section.

Each model processes this input differently and outputs an activity label for the window.

3.3.1 LSTM. In this study, we implement a bidirectional LSTM (bi-LSTM), but for simplicity, we refer to it as LSTM throughout the remainder of the report. The sensor and feature dimensions are flattened so that each timestep becomes a single feature vector, giving an input of shape (batch, steps, features), where features = sensors × number of features per sensor (3). The bi-LSTM processes the sequence in both forward and backward directions to capture long-term dependencies. The final hidden state is then passed to a classification head to classify the activity.

3.3.2 CNN. The input is flattened in the same way (features = sensors × number of features per sensor (3)) and then transposed into shape (batch, features, steps). One-dimensional convolutional layers slide across the time axis to detect short-term patterns. A global pooling step summarises these into a single vector, which is then passed to a classification head.

3.3.3 Graph WaveNet (GWN). This model preserves the full 4D structure (batch, steps, sensors, features) and requires an initial graph describing how sensors are connected. The graph can be randomly initialised, but in our case we passed a graph where sensors in the same room were connected while sensors in different rooms were not. During training, GWN adapts this graph while also learning patterns in the data. Temporal patterns are captured with dilated convolutions, while graph convolutions aggregate information across connected sensors. The resulting representations are then flattened and passed to a classification head.

3.3.4 AGCRN. AGCRN also works directly on the 4D input. It uses recurrent units (GRUs) where the usual internal multiplications are replaced by graph convolutions, meaning each update step considers both temporal and spatial relationships. Additional modules adaptively learn the graph structure and node-specific parameters. The final hidden state is flattened and sent to a classification head.

3.4 Model Evaluation

3.4.1 Walk Forward Validation. For our evaluation, we adopt a rigorous methodology appropriate for time-series data known as

3-fold walk-forward validation, or forward chaining, as detailed by P et al. [8]. This procedure involves splitting the dataset into three consecutive segments. In the first fold, the model is trained on the first quarter of the data and tested on the second quarter. In the second fold, it is trained on the first half and tested on the third quarter. In the third fold, it is trained on the first three quarters and tested on the final quarter. For each fold, 15% of the training set is reserved for validation to select the optimal configuration, after which the model is retrained on the full training portion. A diagram of this methodology is presented in Figure 9 in the Appendix. This ensures that the model is always tested on "future" unseen data, preserving temporal dependencies and preventing data leakage from future events into the training set. To provide a robust estimate of the out-of-sample error, we will report the average performance across all three folds. Additionally, we will report the specific results from Fold 3, as this model is trained on the largest portion of the data and best represents performance in a real-world scenario.

To assess model performance, we employ the F1 score, which is the harmonic mean of precision and recall. In the context of activity recognition, precision is the fraction of correctly identified activities among all classifications/predictions made for that class (i.e., when the model predicts an activity, how often is it correct?). Recall is the fraction of actual activities (for HAR) or future activities (for HAP) that were correctly identified. The F1 score provides a single, robust metric that balances these two, penalising models that perform poorly on one while excelling on the other. Its values range between 0 and 1, with higher scores indicating better performance. For our multi-class evaluation, we calculate two aggregate scores: the macro-average F1, which computes the unweighted mean of the F1 scores for all classes, and the weighted-average F1, which weights each class's F1 score by its support (the number of true instances for that class). Reporting both averages ensures a comprehensive assessment that considers performance on both rare and frequent activities.

3.4.2 Comparing Performance Across Different Window Sizes. To fully understand the conditions under which each model excels or struggles, we evaluated their performance across a wide range of window sizes rather than fixing this parameter to a single value. Window size plays a critical role in determining how much contextual information is available to the model: some architectures benefit from short, recent histories of sensor activations, while others require longer sequences to capture the temporal dynamics of activities. By systematically exploring these variations, we are able to identify not only the scenarios where each model performs best, but also the situations where their performance deteriorates. In addition, we report the scores obtained under each model's optimal configuration, ensuring that our comparisons reflect both their maximum potential and their practical limitations.

3.5 Training and Hyperparameter Tuning

To our knowledge, our study is the first to apply this specific data preprocessing methodology to CNN and LSTM models for HAR and HAP on the CASAS datasets. We also pioneer the application of GWN and AGCRN models to these datasets for HAR and HAP. The novelty of these approaches necessitated a comprehensive hyperparameter tuning process to determine the optimal configuration for each model and dataset.

We conducted hyperparameter optimisation for each model on both datasets and for both HAR and HAP. While the Milan and Aruba datasets share a common data format and purpose, their intrinsic properties diverge significantly. The Aruba dataset was collected over a more extended period, whereas the Milan dataset contains a greater variety of activities, leading to more activity classes and consequently, more complex temporal patterns to model. These fundamental differences mean that the specific combination of hyperparameters that yields the best results for one dataset will not be optimal for the other. Therefore, a distinct and independent hyperparameter tuning process was essential for each dataset to identify the unique configurations required to achieve optimal model performance.

Hyperparameter tuning was conducted with the Optuna framework [21], which automates the search process. Because the search space is large and costly to explore exhaustively, Optuna uses a Bayesian optimisation method, the Tree-structured Parzen Estimator (TPE) [22], to model the objective function (weighted F1-score on a validation set) and prioritise the most promising hyperparameters. This approach is more efficient than a grid search, enabling robust configurations to be found with fewer trials. Tables 7 and 8 in the Appendix summarise the search and list the optimal values for each model and dataset in both HAR and HAP tasks.

3.6 Experiment Infrastructure

All model training and hyperparameter tuning were conducted on the University of Cape Town's High-Performance Computing cluster. For all experiments, we utilised the L40s GPU partition to accelerate the computationally intensive training processes. Our implementations of the CNN and LSTM models were built using the PyTorch framework [23]. For the more complex spatiotemporal graph-based models (AGCRN and GWN), we employed the torch_spatiotemporal library [24], which provides implementations of these architectures.

4 RESULTS AND DISCUSSION

4.1 HAR Performance

4.1.1 Benchmarking Model Performance. To assess whether explicitly learning spatial relationships between sensors improves performance, we benchmarked the STGNN models (GWN and AGCRN) against the traditional baselines (CNN and LSTM). Figures 1 (Aruba) and 2 (Milan) illustrate how each model's weighted average F1-score varies across a range of window sizes. Importantly, these graphs show that evaluating all models at a single window size would not have provided a fair comparison. In particular, GWN achieves its strongest performance at the smallest window size, whereas CNN, LSTM, and AGCRN generally perform their worst under this setting and improve with larger windows. By presenting results across the full range, the figures enable a fair comparison under identical conditions, while Table 3 summarises performance at each model's optimal configuration and is complemented by a visualisation in Figure 7 in the Appendix.

Table 3: HAR Performance at Optimal Model Configurations

Algorithm	Weighted Avg F1 (across folds)		Weighted Avg F1 (Fold 3)		Macro Avg F1 (across folds)		Window Size	
	Milan	Aruba	Milan	Aruba	Milan	Aruba	Milan	Aruba
CNN	**0.54** ± 0.08	0.80 ± 0.01	**0.54**	0.81	0.30 ± 0.05	0.35 ± 0.00	40	40
LSTM	0.53 ± 0.04	**0.82** ± 0.01	0.51	**0.82**	0.28 ± 0.05	0.37 ± 0.01	40	40
GWN	0.53 ± 0.03	0.77 ± 0.00	0.51	0.77	0.32 ± 0.03	0.34 ± 0.01	10	10
AGCRN	0.53 ± 0.05	**0.82** ± 0.00	0.51	**0.82**	**0.33** ± 0.05	**0.38** ± 0.02	90	70

4.1.2 Model-Specific Trends. A distinct trend is visible for the Graph WaveNet (GWN) model on both datasets: its performance is highest at a window size of 10 and consistently degrades as the window size increases. This indicates that GWN performs worse when given more information from the past, meaning it relies primarily on recent sensor activations to make accurate classifications.

On the Aruba dataset, the general trend for the CNN, LSTM, and AGCRN models is a steady improvement in performance as the window size increases, before plateauing. In contrast, on the Milan dataset, there is significantly more variability. While there appears to be a slight increase in performance with larger windows (excluding GWN), the overlapping error bars suggest that this trend may not be statistically significant. These observations highlight a fundamental difference between architectures: GWN depends on very recent context, whereas the other models tend to require longer histories to improve their classifications.

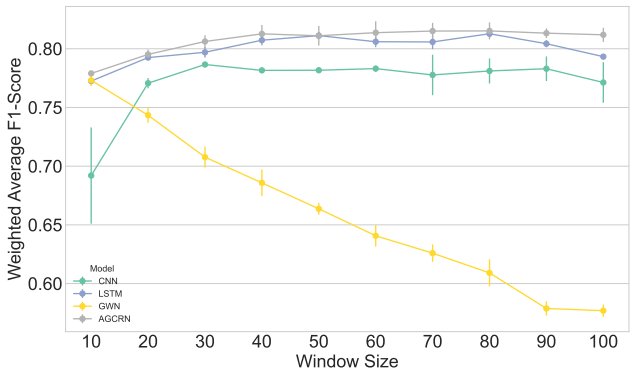

Figure 1: Weighted average F1-scores for HAR across varying window sizes on the Aruba dataset

4.1.3 Performance at Optimal Model Configuration. Table 3 summarises each model's performance at its optimal hyperparameter configuration. On the Aruba dataset, the LSTM and AGCRN both achieved the highest weighted average F1-score for HAR on the Aruba dataset (0.82), but under different conditions: the LSTM reached this score with a window size of 40, while the AGCRN required a larger window of 70. There is no clear performance difference between the two models, but the LSTM's optimal window size for this dataset is smaller, suggesting it can achieve the same accuracy with less contextual information.

On the Milan dataset, the performance differences between models were marginal and unlikely to be statistically significant. The CNN achieved the highest weighted F1-score (0.54) with a window size of 40, but this was almost indistinguishable from the GWN's score of 0.53 at a much smaller window size of 10. Likewise, the LSTM and AGCRN both reached essentially the same score (0.53) at window sizes of 40 and 90, respectively.

These optimal-configuration results complement the trend analysis above: while several models achieve comparable accuracy at their optimal configurations, they require different amounts of temporal context to do so. This underscores the need to consider statistical significance and to report both the performance trends across varying window sizes and the performance at each model's optimal configuration, rather than relying on a single window size that could unfairly favour one architecture over another.

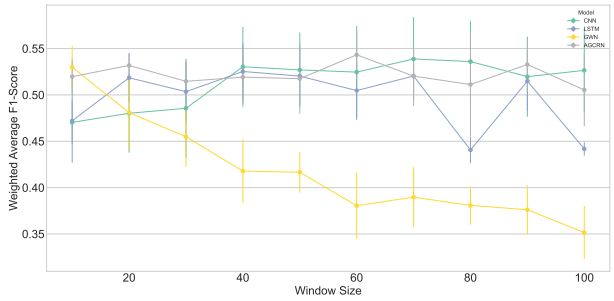

Figure 2: Weighted average F1-scores for HAR across varying window sizes on the Milan dataset

4.1.4 Computational Efficiency. Computational efficiency is a crucial differentiator between the models. The traditional DNNs demonstrated a significant speed advantage over their graph-based counterparts. For instance, on the Aruba dataset, the LSTM model converged in just 3.5 minutes (208.76s), whereas the AGCRN took nearly 2.2 hours (7,779.39s). This dramatic gap in training time is even more significant considering the AGCRN's training was cut short at epoch 50 by early stopping, while the LSTM completed 103 epochs. The disparity extends to inference time (the time the model takes to produce a classification for a given sample), where the AGCRN was over 150 times slower per sample (2.3881ms vs. 0.0154ms). Therefore, despite achieving comparable accuracy, the AGCRN's prohibitive computational cost makes it a far less practical solution for this application.

The Graph WaveNet (GWN) model fared much better than the AGCRN model, converging in approximately 12 minutes (720.59s) on the Aruba dataset. However, it was still significantly slower than the LSTM. The GWN model took over three times as long to train and was more than twice as slow during inference (0.0373ms per sample) compared to the LSTM.

Table 4: Training time (to convergence) and inference time (per sample) for all models on Fold 3 of the Milan and Aruba datasets

Model	Dataset	Training Time (s)	Inference Time (ms)
CNN	**Milan**	10.06	0.0059
	Aruba	18.58	0.0079
LSTM	**Milan**	3.67	0.0045
	Aruba	208.76	0.0154
GWN	**Milan**	255.05	0.1255
	Aruba	720.59	0.0373
AGCRN	**Milan**	1168.79	0.5463
	Aruba	7779.39	2.3881

4.1.5 Performance on Minority Classes. To gain a deeper insight into HAR performance, we analysed the macro-average F1 scores, which, unlike the weighted average, are not skewed by highly frequent activities. The macro F1 scores (presented in Table 3) were considerably lower than their weighted counterparts for all models, indicating that every architecture struggled to correctly classify less frequent activities. This is clearly illustrated by the AGCRN's confusion matrix on the Aruba dataset (Figure 10 in the Appendix). While it correctly classified thousands of instances of dominant activities like "Other" (4080), "Relax" (1777), and "Meal Preparation" (1711), it made zero correct classifications for minority classes like "Enter Home" and "Wash Dishes".

4.1.6 Do The Graph-Based Models Outperform the DNNs? Overall, the graph-based models did not demonstrate a clear advantage over the traditional CNN and LSTM baselines. On Aruba, AGCRN achieved performance comparable to the LSTM at its optimal configuration, with both models tying for the best results on this dataset. On Milan, although the CNN obtained the highest mean score at its optimal configuration, the differences between CNN, GWN, and AGCRN did not suggest a clear winner. These findings indicate that while graph-based models can achieve performance comparable to traditional DNNs (as measured by the weighted average F1 score), they do not consistently exceed them, and this parity is achieved at a substantially higher computational cost, especially in the case of AGCRN.

4.1.7 Comparison With Other Studies. It should be noted that models such as GWN and AGCRN were originally designed for data streams with regular sensor readings at fixed time intervals. In this study, the irregular, event-driven nature of smart home data required adaptations that may not have been fully optimal. Consequently, while GWN and AGCRN did not outperform the traditional baselines here, it remains possible that alternative STGNN architectures, or more suitable adaptations, could yield stronger results. For example, P et al. [8] reported considerably higher weighted average F1 scores of 0.924 on Aruba and 0.804 on Milan using their own STGNN. However, because their work does not specify the exact input features or the method of temporal encoding, the precise reason for this performance gap between their study and ours cannot be determined.

4.2 HAP Performance

To assess the suitability of the four deep learning architectures for the Human Activity Prediction task, we evaluate their performance using the same methodology as the HAR task. Figures 3 and 4 show the performance trends across varying window sizes, while Table 5 summarises the results at each model's optimal configuration, complemented by a visualisation in Figure 8 in the Appendix. The HAP task, which requires predicting the next distinct activity, is inherently more complex than recognising the current one. This increased difficulty is immediately apparent in the overall performance, with the weighted average F1 scores for HAP being substantially lower across all models and datasets compared to the HAR task.

4.2.1 HAP Performance on Aruba. On the Aruba dataset (Figure 3), the models demonstrate a moderate predictive capability. The CNN, LSTM, and AGCRN models exhibit a similar trend: their performance generally improves as the window size increases, plateauing around 40–70 steps. At their respective optimal configurations, these three models achieve nearly identical weighted F1 scores, with negligible differences. In contrast, the Graph WaveNet (GWN) model displays a distinct pattern: its best performance occurs at the smallest window size of 10, after which performance steadily declines as more historical context is included. Overall, no single model shows a decisive advantage on the Aruba dataset.

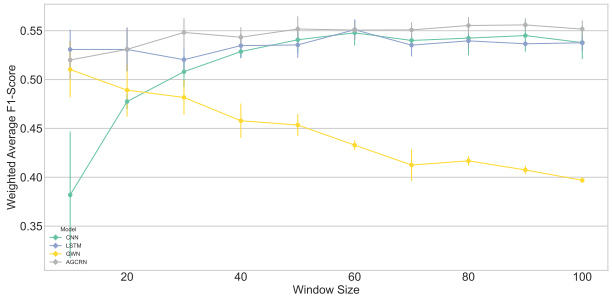

Figure 3: Weighted average F1-scores for HAP across varying window sizes on the Aruba dataset

4.2.2 HAP Performance on Milan. The models' performance on the more challenging Milan dataset (Figure 4) highlights the significant difficulty of the HAP task in a more complex environment. Weighted average F1 scores remain markedly lower, with all models achieving approximately 0.40 at their optimal configurations (Table 5). For the CNN, LSTM, and AGCRN, the performance trends are less consistent: although there is a slight tendency for performance to decrease with larger window sizes, the curves fluctuate substantially. The large error bars also indicate high variance across validation folds, suggesting that minor score differences are unlikely to be

Table 5: HAP Performance at Optimal Model Configurations

Algorithm	Weighted Avg F1 (across folds)		Weighted Avg F1 (Fold 3)		Macro Avg F1 (across folds)		Window Size	
	Milan	Aruba	Milan	Aruba	Milan	Aruba	Milan	Aruba
CNN	0.39 ± 0.05	0.55 ± 0.01	0.32	0.52	0.07 ± 0.02	0.27 ± 0.01	20	70
LSTM	0.40 ± 0.05	0.55 ± 0.02	0.32	**0.55**	0.09 ± 0.02	0.27 ± 0.01	10	70
GWN	**0.41** ± 0.05	0.51 ± 0.04	0.34	0.43	0.11 ± 0.02	0.21 ± 0.03	20	10
AGCRN	**0.41** ± 0.04	**0.56** ± 0.01	**0.36**	**0.55**	**0.12** ± 0.02	**0.28** ± 0.01	20	70

statistically significant. This makes it difficult to identify a clear optimal window size. Consistent with its behaviour on Aruba, GWN again performs best at the smaller window sizes before its predictive power diminishes.

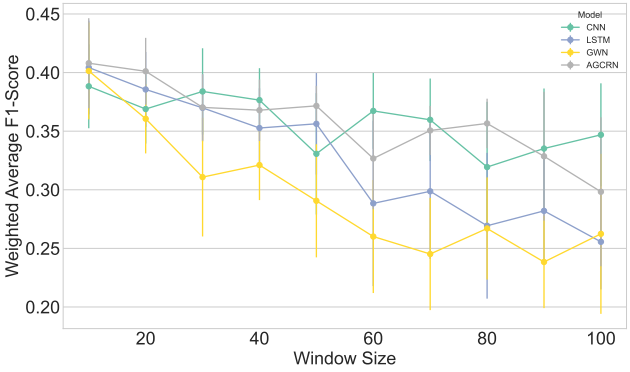

Figure 4: Weighted average F1-scores for HAP across varying window sizes on the Milan dataset

4.2.3 HAP Performance on Minority Classes. The imbalance observed in Section 4.1 becomes even more pronounced for the prediction task. Table 5 shows that macro-average F1 scores remain very low compared to the weighted averages, with the best model (AGCRN on Aruba) reaching only 0.28. This indicates that the models not only struggle with minority activities, but that predictive uncertainty further amplifies their difficulty.

The AGCRN confusion matrix on Aruba (Fold 3) (Figure 11 in the Appendix) illustrates this effect. While majority classes such as "Other", "Relax", and "Meal Preparation" dominate predictions, minority activities are almost entirely overlooked. For example, "Enter Home" and "Respirate" receive no correct predictions, while activities like "Eating" and "Work" are consistently absorbed into more frequent categories such as "Other" or "Meal Preparation". Compared to HAR, this suggests that the additional forecasting step compounds the bias towards frequent activities, further suppressing the visibility of rare but meaningful transitions.

These findings highlight that any practical deployment of HAP systems will require additional strategies to counteract this skew, particularly since minority activities, though rare, may carry disproportionate importance in applications such as health monitoring or anomaly detection.

4.2.4 Predictive Performance Summary. In conclusion, the suitability of these models for HAP is limited. While they show some ability to predict the next activity on the cleaner Aruba dataset, their performance drops significantly on the more complex Milan dataset, where no single architecture proves robust. The spatial-temporal models do not offer a clear performance benefit over the traditional CNN and LSTM baselines for this predictive task. The modest F1 scores (on Aruba) at optimal configurations suggest that while the models are learning patterns beyond random chance, their predictive accuracy may not be sufficient for reliable real-world deployment in safety-critical applications.

4.3 Assessing Temporal Representations in HAR

4.3.1 Impact of Temporal Feature Engineering. In our initial experiments, we addressed the irregular timing of sensor activations by resampling the data at a fixed one-second interval. Although simple to implement, this approach produced suboptimal weighted average F1 scores (see Table 6). To more effectively represent the irregular nature of the data, we subsequently adopted an event-driven approach with engineered time-delta features, as outlined in Section 3.2. The results reported in Section 4.1, obtained using log-normalised time deltas, show a clear improvement over fixed-interval sampling. For example, the Graph WaveNet model's weighted average F1-score on the Aruba dataset increased from 0.65 to 0.77. This improvement highlights the value of the engineered temporal features and motivates further exploration of alternative representations of time.

4.3.2 Alternative Normalisation Strategy: A Hard Cutoff. To further investigate the role of temporal feature engineering, we conducted additional experiments using a simpler normalisation strategy: applying a hard cutoff value. We selected the AGCRN model, as it generally outperformed Graph WaveNet on both datasets and thus provided a stronger basis for testing alternative encodings. For these experiments, we fixed the hyperparameters to the optimal configuration identified for AGCRN on the Milan dataset and evaluated performance on fold 3 to ensure consistency. Both global and sensor-specific time deltas were subjected to varying cutoff thresholds, with any value exceeding the threshold replaced by the cutoff itself. To improve robustness, performance at each cutoff point was averaged across three runs.

For a sequence of sensor events with timestamps $t_1, t_2, \ldots, t_n$, the raw time deltas Δt_i represent the elapsed time between consecutive events (for global deltas) or between repeated activations of the same sensor (for sensor-specific deltas). The hard cutoff

Table 6: HAR Performance Using 1 second Sampling Pipeline

	CNN		LSTM		GWN	
Metric	**Milan**	**Aruba**	**Milan**	**Aruba**	**Milan**	**Aruba**
Weighted Average F1 (across folds)	0.37 ± 0.08	0.56 ± 0.03	0.39 ± 0.04	**0.72** ± 0.03	**0.44** ± 0.01	0.65 ± 0.01
Weighted Average F1 (Fold 3)	0.24	0.59	0.33	**0.66**	**0.45**	0.63

transformation is then defined as

$$\tilde{\Delta t}_i = \begin{cases} \Delta t_i, & \text{if } \Delta t_i \leq \tau, \\ \tau, & \text{if } \Delta t_i > \tau, \end{cases}$$

where τ is the cutoff threshold.

To determine appropriate thresholds, we analysed the empirical distributions of both global and sensor-specific deltas in the Milan dataset only. Milan was chosen because it is smaller, allowing the model to train more quickly during exploratory analysis. From these distributions, we selected the 75th–95th percentile ranges as candidate cutoff values, as they retain meaningful variability in typical time gaps while excluding rare, high-magnitude intervals that could obscure underlying temporal patterns. The specific thresholds tested are shown in Figures 12 and 13, where performance is plotted as a function of τ.

For the sensor-specific time delta plot in Figure 13, a logarithmic scale was used for the x-axis. This was necessary to effectively visualise the model's performance across a very wide range of cutoff values, which span several orders of magnitude, from thousands to tens of thousands of seconds.

4.3.3 Findings from Hard Cutoff Experiments. The results of the cutoff experiments reveal consistent trends in model performance as the thresholds are varied. For the global time delta (Figure 12), the mean weighted average F1-score peaked around 19 seconds, while for the sensor-specific delta (Figure 13), performance was highest near 50,000 seconds (approximately 13.8 hours). However, the error bars, which represent the standard deviation across the three runs, show substantial overlap across all cutoff points. This indicates that the observed peaks are likely not statistically significant.

The lack of significant differences suggests that simpler cutoff values, such as those near the 75th percentile, are just as effective as much higher thresholds. In practice, this means the model does not rely on fine-grained distinctions between very long delays, but only on whether a delay is relatively short or long. Both the cutoff and log-normalisation strategies achieve this by reducing the influence of extreme intervals, albeit in different ways. The cutoff experiments therefore indicate that the precise degree of compression applied to large values is not critical. This validates the use of log-normalisation in the main experiments, as it provides a simple, general-purpose approach that captures the necessary contrast between short and long delays without requiring dataset-specific threshold tuning.

4.4 What Do The Learned Graph Structures Reveal?

To gain deeper insight into the decision-making processes of the graph-based models, we analysed the graph structures they learned during training. This analysis was focused on the Aruba dataset, as our previous results indicated that it contained a more meaningful signal for the models to learn from compared to the Milan dataset. For both GWN and AGCRN, we extracted the learned adjacency matrices from fold 3 of the Aruba data. The corresponding graph visualisations are presented in Figures 5 and 6, while heatmaps of the adjacency matrices are presented in 18 and 19 in the Appendix. To improve interpretability, the visualisations omit self-loops and display only the top 20 strongest connections between distinct nodes (i.e., sensors).

4.4.1 General Trends Across Models. A common insight across both models is that their strongest learned connections are not confined to sensors within the same room. Instead, they frequently span different rooms, indicating that the models are prioritising inter-room pathways of user movement rather than local co-activations. This behaviour aligns with empirical activity transition probabilities observed in the data and suggests that the models are capturing meaningful semantic information about daily routines rather than relying solely on spatial proximity. For the calculation of the empirical tranisition probabilities (Figure 17), the 'Other' label was filtered out to focus attention on transitions between activities of interest.

4.4.2 Patterns Learned by AGCRN. The AGCRN model highlights several key inter-room connections that map directly to frequent activity transitions. Strong edges emerge between the office and the kitchen, and between the office and the living room, reflecting the high empirical probabilities of transitioning from 'Work' to 'Meal Preparation' (0.3) and from 'Work' to 'Relax' (0.4). Another prominent connection links the dining room and the kitchen, consistent with the observed probability of transitioning from 'Eating' to 'Relax' (0.38). AGCRN also captures connections from Bedroom 2 to the living room, reflecting typical intra-household movement patterns.

4.4.3 Patterns Learned by Graph WaveNet. Graph WaveNet identifies a related but distinct set of connections. It learns strong pathways from the office to the living room, from Bedroom 2 to the living room (also identified by AGCRN), and from Bedroom 1 to the living room. One particularly strong edge links the kitchen and the living room, which aligns with the very high transition probability from 'Meal Preparation' to 'Relax' (0.72). Unlike AGCRN, however, Graph WaveNet places less emphasis on the direct office–kitchen relationship, suggesting subtle differences in how each architecture prioritises transitional patterns.

4.4.4 Comparing Learned Patterns. Both models capture routine patterns in daily activity flows, but their priorities differ. Graph WaveNet largely emphasises strong, high-frequency transitions

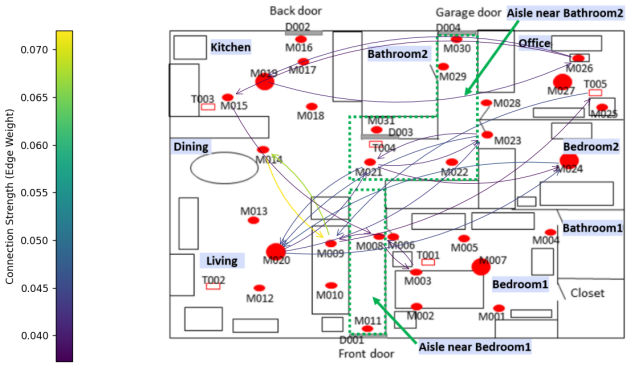

Figure 5: Learned AGCRN graph on Aruba, showing the top 20 strongest inter-sensor connections

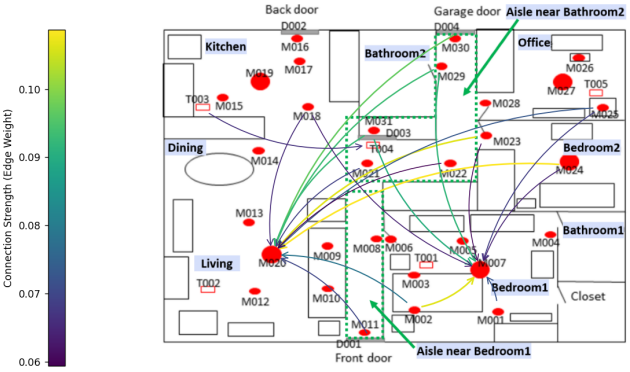

Figure 6: Learned GWN graph on Aruba, showing the top 20 strongest inter-sensor connections

such as kitchen–living room, while AGCRN integrates a broader set of pathways linking work, meal preparation, and relaxation. This broader modelling of cross-room routines appears to underpin AGCRN's stronger performance on the HAR task. The results indicate that by embedding the structure of daily movement more effectively than Graph WaveNet, AGCRN achieves more accurate classifications.

4.4.5 Insights from Learned Graph Structures. The extracted graphs provide valuable insight into what the models have actually learned. By revealing that both architectures encode spatially meaningful patterns of activity flow, we gain confidence that their classifications are grounded in realistic behavioural dynamics rather than spurious correlations. This understanding is essential for assessing whether the models' decisions can be trusted in deployment, particularly in applications like healthcare monitoring the ability to trust model outputs is just as important as accuracy.

5 LIMITATIONS AND FUTURE WORK

A key limitation lies in the temporal encoding for the graph-based models. Although engineered time-delta features improved performance over fixed-interval resampling, it is unclear whether they fully capture the irregular nature of event-driven smart home data. Prior work [8] reports stronger results on the same datasets, suggesting that suboptimal temporal feature design may have limited performance in our study. Future research should explore alternative representations of time, for example, more fine-grained sampling at intervals shorter than one second.

Our analysis of the learned graph structures was qualitative. Although it suggested that the models capture sensible activity patterns, it did not offer a systematic assessment of their reliability. Future work should therefore focus on developing more robust methods to evaluate the stability and consistency of these structures, ensuring that model predictions can be trusted in safety-critical applications.

Finally, for computational feasibility, hyperparameters were optimised once per model, dataset, and task using Optuna, then reused across window-size experiments. Since parameters like learning rate and hidden size may interact with sequence length, this approach could confound comparisons. Future work should conduct per-window tuning or sensitivity analyses to isolate the effects of window size from hyperparameter choice.

6 CONCLUSION

This study investigated the applicability of Spatial-Temporal Graph Neural Networks (STGNNs) to Human Activity Recognition (HAR) and Human Activity Prediction (HAP) in smart home environments, using the CASAS Aruba and Milan datasets. By benchmarking Graph WaveNet (GWN) and Adaptive Graph Convolutional Recurrent Network (AGCRN) against traditional CNN and LSTM baselines, we evaluated both performance and interpretability in a safety-critical context.

Our findings show that while STGNNs can achieve competitive recognition accuracy, their advantages over traditional models are not consistent. Overall, AGCRN outperformed GWN but only matched the performance of the CNN and LSTM baselines, and did so at a higher computational cost. On the more challenging HAP task, none of the architectures demonstrated robust predictive performance, and all models struggled with minority activity classes, highlighting the difficulty of anticipating rare but meaningful behavioural transitions.

A key strength of STGNNs lies in their ability to make their reasoning partially transparent. By inspecting the learned graph structures, we can see that the models capture high-level activity flows consistent with real human routines. This ability to "open the black box" helps us judge whether the models' classifications can be trusted. In safety-critical smart home applications, this is as important as raw predictive performance.

Overall, this work demonstrates that STGNNs are a promising but not yet definitive solution for smart home HAR and HAP. Future research should focus on developing more effective temporal encodings for irregular, event-driven data, as this may allow graph-based models to realise their full potential and consistently outperform traditional DNNs.

ACKNOWLEDGMENTS

This work was supported by the University of Cape Town's High-Performance Computing cluster.

REFERENCES

[1] S. Gupta. 2021. Deep learning based human activity recognition (HAR) using wearable sensor data. *Int. J. Inf. Manag. Data Insights* 1, 100046.

[2] M. Kaseris, I. Kostavelis, and S. Malassiotis. 2024. A comprehensive survey on deep learning methods in human activity recognition. *Machine Learning and Knowledge Extraction* 6, 842–876. https://doi.org/10.3390/make6020040

[3] O. D. Lara and M. A. Labrador. 2013. A survey on human activity recognition using wearable sensors. *IEEE Commun. Surv. Tutor.* 15, 3, 1192–1209.

[4] K. Chen, D. Zhang, L. Yao, B. Guo, Z. Yu, and Y. Liu. 2018. Deep learning for sensor-based human activity recognition: Overview, challenges and opportunities. *J. ACM* 37, 4, Article 111 (Aug. 2018), 40 pages. https://doi.org/10.1145/1122445.1122456

[5] Y. Du, Y. Lim, and Y. Tan. 2019. A novel human activity recognition and prediction in smart home based on interaction. *Sensors* 19, 20, 4474. https://doi.org/10.3390/s19204474

[6] K. H. N. Bui, J. Cho, and H. Yi. 2022. Spatial-temporal graph neural network for traffic forecasting: An overview and open research issues. *Appl. Intell.* 52, 2763–2774. https://doi.org/10.1007/s10489-02102587-w

[7] A. Gaibie, H. Amir, I. Nandutu, and D. Moodley. 2025. Predicting and discovering weather patterns in South Africa using spatial-temporal graph neural networks. *Commun. Comput. Inf. Sci.* 2326, 144–160. https://doi.org/10.1007/978-3-031-78255-8_9

[8] S. P. and T. Plötz. 2024. Using graphs to perform effective sensor-based human activity recognition in smart homes. *Sensors* 24, 12, 3944. https://doi.org/10.3390/s24123944

[9] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang. 2019. Graph WaveNet for deep spatial-temporal graph modeling. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*. 2607–2613.

[10] L. Bai, L. Yao, C. Li, X. Wang, and C. Wang. 2020. Adaptive graph convolutional recurrent network for traffic forecasting. In *Advances in Neural Information Processing Systems* 33, 17804–17815. Curran Associates, Inc.

[11] D. Bouchabou, S. M. Nguyen, C. Lohr, B. Le Duc, and I. Kanellos. 2021. A survey of human activity recognition in smart homes based on IoT sensors: Algorithms, taxonomies, challenges, and opportunities with deep learning. *Electronics* 10, 20, 2498.

[12] M. Gochoo, T. H. Tan, S. H. Liu, F. R. Jean, F. S. Alnajjar, and S. C. Huang. 2018. Unobtrusive activity recognition of elderly people living alone using anonymous binary sensors and DCNN. *IEEE J. Biomed. Health Inform.* 23, 693–702.

[13] T. H. Tan, M. Gochoo, S. C. Huang, Y. H. Liu, S. H. Liu, and Y. F. Huang. 2018. Multi-resident activity recognition in a smart home using RGB activity image and DCNN. *IEEE Sens. J.* 18, 9718–9727.

[14] D. Singh, E. Merdivan, S. Hanke, J. Kropf, M. Geist, and A. Holzinger. 2017. Convolutional and recurrent neural networks for activity recognition in smart environments. In *Proceedings of Towards Integrative Machine Learning and Knowledge Extraction*. Springer, Berlin/Heidelberg, Germany, 194–205.

[15] D. Liciotti, M. Bernardini, L. Romeo, and E. Frontoni. 2020. A sequential deep learning application for recognising human activities in smart homes. *Neurocomputing* 396, 501–513.

[16] K. Pillay and D. Moodley. 2022. Exploring graph neural networks for stock market prediction on the JSE. In E. Jembere, A. J. Gerber, S. Viriri, and A. Pillay (Eds.), *SACAIR 2021*, CCIS, vol. 1551. Springer, Cham, 95–110. https://doi.org/10.1007/978-3-030-95070-57

[17] B. Quigley, M. Donnelly, G. Moore, and L. Galway. 2018. A comparative analysis of windowing approaches in dense sensing environments. *Proceedings* 2, 1245. https://doi.org/10.3390/proceedings2191245

[18] D. Cook. 2011. Learning setting-generalized activity models for smart spaces. *IEEE Intelligent Systems.*

[19] D. Cook and M. Schmitter-Edgecombe. 2009. Assessing the quality of activities in a smart environment. *Methods Inf. Med.* 48, 5, 480–485. https://doi.org/10.3414/ME0592

[20] D. Bouchabou, S. M. Nguyen, C. Lohr, B. LeDuc, and I. Kanellos. 2021. Using language model to bootstrap human activity recognition ambient sensors based in smart homes. *Electronics* 10, 20, 2498.

[21] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*. ACM, New York, NY, USA, 2623–2631. https://doi.org/10.1145/3292500.3330701

[22] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2546–2554.

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, … and S. Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* 32.

[24] A. Cini and I. Marisca. 2022. Torch Spatiotemporal. MIT License. https://github.com/TorchSpatiotemporal/tsl

A APPENDIX

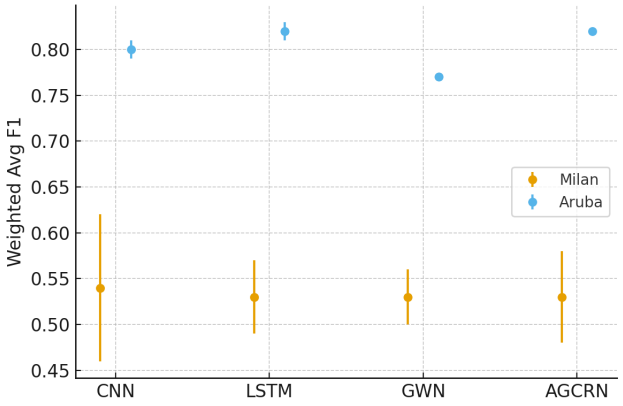

Figure 7: Visual Representation of Table 3 Results.

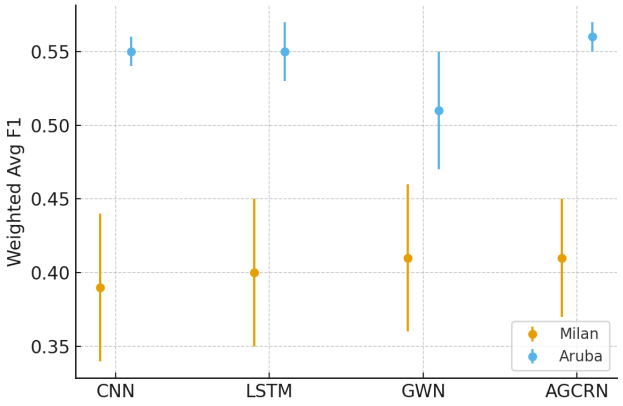

Figure 8: Visual Representation of Table 5 Results.

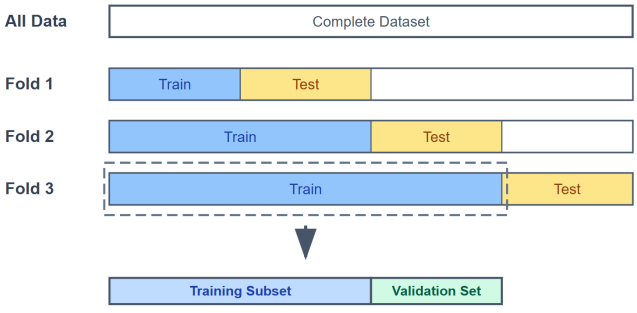

Figure 9: Diagram of the 3-fold forward chaining evaluation methodology. In this scheme, the training set (blue) progressively expands with each fold to include past data, and the model is evaluated on the subsequent, unseen time segment (yellow). A portion of the training data in each fold is held out as a validation set.

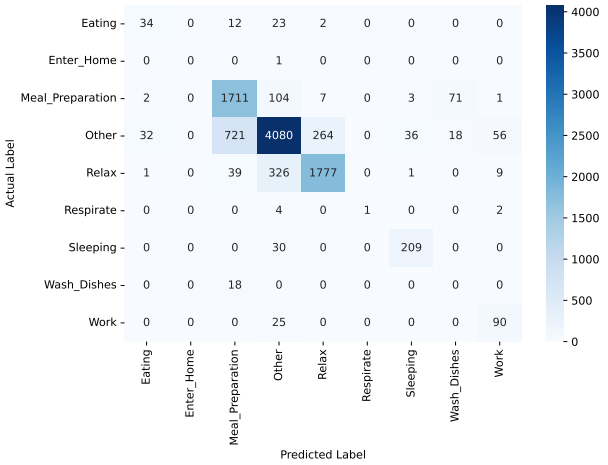

Figure 10: Confusion matrix for the AGCRN model evaluated on the Fold 3 test set of the Aruba dataset for HAR. Classes with zero support in this test set have been excluded for clarity.

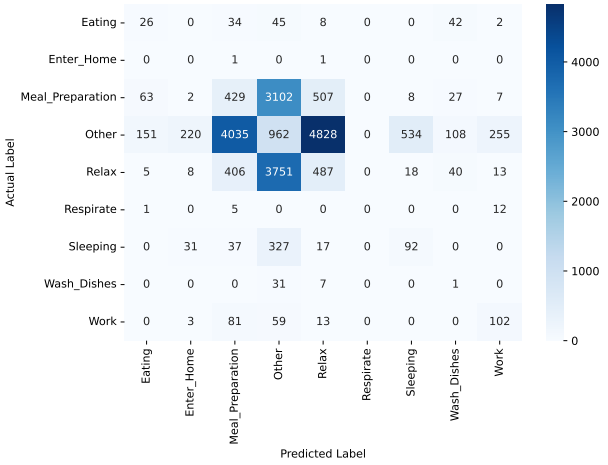

Figure 11: Confusion matrix for the AGCRN model on the Fold 3 test set of the Aruba dataset for HAP. Classes with zero support in this test set have been excluded for clarity.

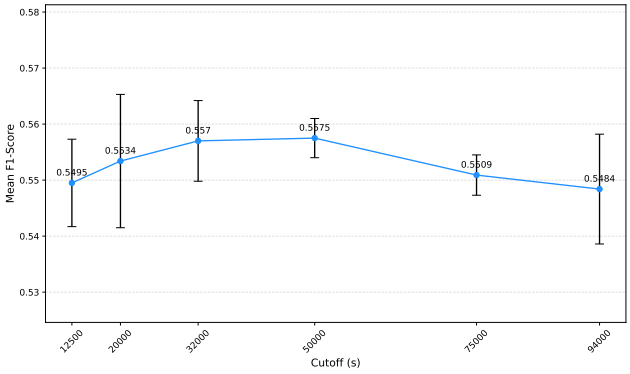

Figure 13: Mean Weighted Average F1-Score vs. Sensor Time Delta Cutoff for AGCRN on Milan (Fold 3). The x-axis is presented on a log scale to accommodate the wide range of cutoff values. Error bars represent one standard deviation over three runs.

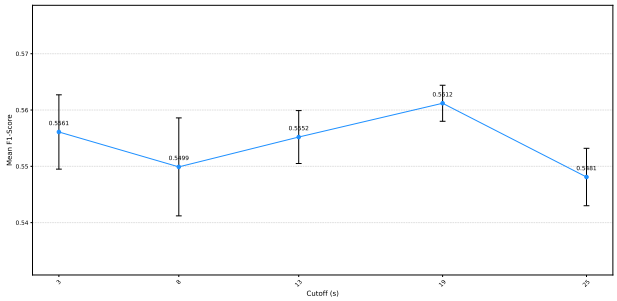

Figure 12: Mean Weighted Average F1-Score vs. Global Time Delta Cutoff for AGCRN on Milan (Fold 3). Error bars represent one standard deviation over three runs.

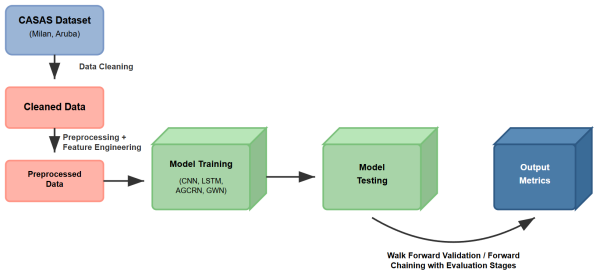

Figure 14: Diagram Showing our ML Pipeline.

```
2009-10-16 00:01:04.000059      M017    ON
2009-10-16 00:01:06.000046      M009    ON
2009-10-16 00:01:07.000064      M017    OFF
2009-10-16 00:01:08.000081      M019    ON
2009-10-16 00:01:09.000028      M009    OFF
2009-10-16 00:01:13.000051      M019    OFF
2009-10-16 00:08:50.000081      M020    ON
2009-10-16 00:08:55.000040      M020    OFF
```

Figure 15: Snippet showing the irregularity of readings from the Milan dataset

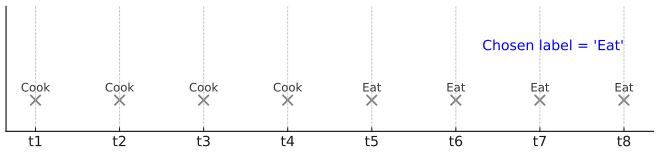

Figure 16: Tie-breaking strategy for labeling of Sensor Event Windows for HAR

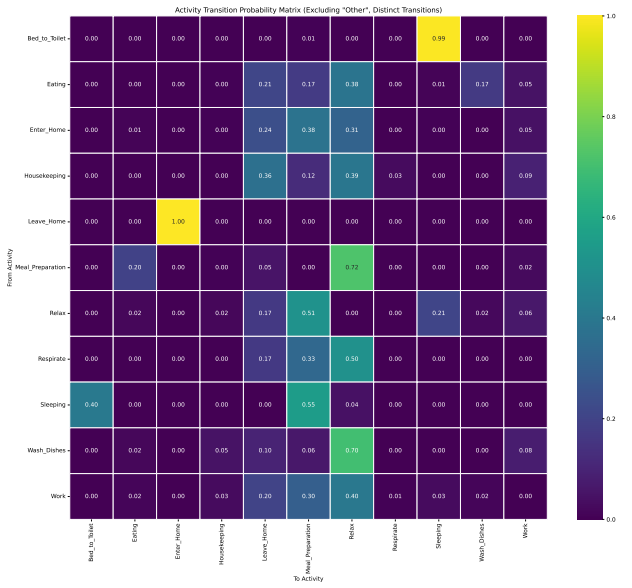

Figure 17: Activity transition probability matrix for the Aruba dataset. The 'Other' label was filtered out to show direct transitions between activities of interest.

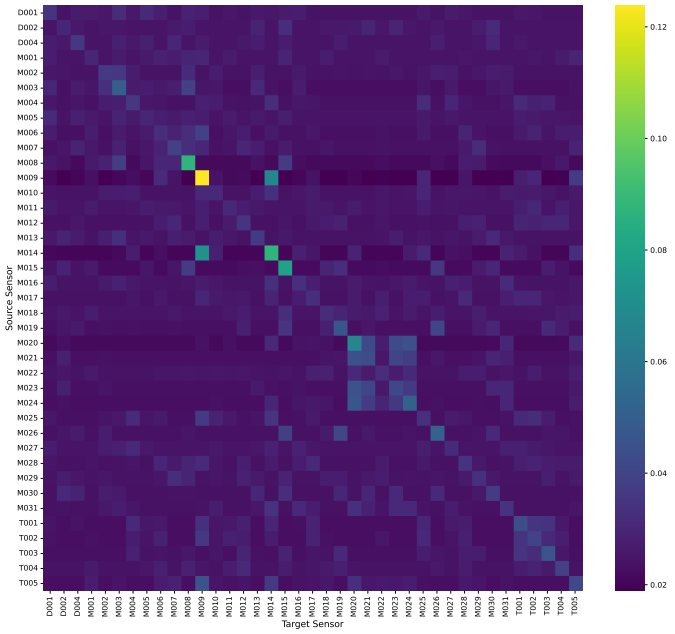

Figure 18: Adjacency heatmap for the AGCRN model on Aruba (Fold 3).

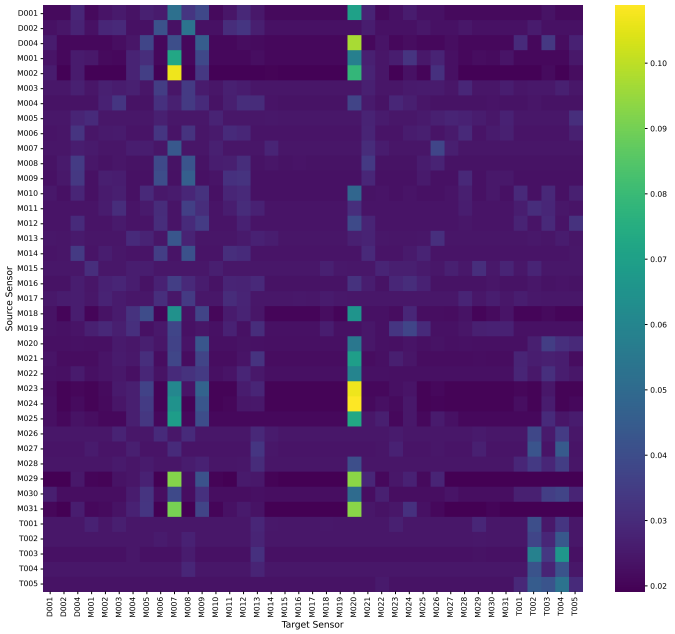

Figure 19: Adjacency heatmap for the GWN model on Aruba (Fold 3).

Table 7: Hyperparameter Search Space and Selected Values for HAR

Model	Dataset	Window Size	Stride %	Batch Size	Learning Rate	L2 Reg.	Early Stop	Hidden Size	Filters 1	Filters 2	Kernel Size	Activation	FF Size	Layers	Emb. Size
Search Space		[10, 100]	[0.2, 1.0]	{32, 64, 128}	[1e-4, 1e-2]	[1e-6, 1e-3]	[15, 25]	[32, 256]	{32, 64, 128}	{64, 128, 256}	{3, 5, 7}	{ReLU, LeakyReLU}	{128, 256, 512}	[1, 5]	{10, 20, 40}
LSTM	Milan	40	0.93	128	3.274e-4	9.645e-5	19	96	–	–	–	–	–	–	–
	Aruba	60	0.58	32	1.656e-3	2.053e-4	19	160	–	–	–	–	–	–	–
CNN	Milan	40	0.31	128	3.138e-4	4.982e-5	16	–	128	256	5	LeakyReLU	–	–	–
	Aruba	40	0.34	32	1.357e-3	2.592e-6	24	–	64	64	3	LeakyReLU	–	–	–
GWN	Milan	10	0.37	32	1.225e-3	1.995e-6	22	32	–	–	–	–	256	4	–
	Aruba	10	0.22	64	8.985e-4	3.852e-4	23	64	–	–	–	–	256	5	–
AGCRN	Milan	90	0.95	128	3.661e-4	4.217e-5	18	32	–	–	–	–	–	2	10
	Aruba	40	0.24	32	1.029e-4	1.812e-4	18	64	–	–	–	–	–	1	40

Table 8: Hyperparameter Search Space and Selected Values for HAP

Model	Dataset	Window Size	Stride %	Batch Size	Learning Rate	L2 Reg.	Early Stop	Hidden Size	Filters 1	Filters 2	Kernel Size	Activation	FF Size	Layers	Emb. Size
Search Space		[10, 100]	[0.2, 1.0]	{32, 64, 128}	[1e-4, 1e-2]	[1e-6, 1e-3]	[15, 25]	[32, 256]	{32, 64, 128}	{64, 128, 256}	{3, 5, 7}	{ReLU, LeakyReLU}	{128, 256, 512}	[1, 5]	{10, 20, 40}
LSTM	Milan	10	0.46	32	1.617e-4	1.594e-5	24	128	–	–	–	–	–	–	–
	Aruba	70	0.20	32	1.693e-4	8.948e-4	19	160	–	–	–	–	–	–	–
CNN	Milan	20	0.28	32	1.793e-4	3.910e-4	24	–	128	256	7	ReLU	–	–	–
	Aruba	70	0.39	128	4.876e-4	4.160e-5	20	–	64	256	5	ReLU	–	–	–
GWN	Milan	20	0.20	32	1.571e-4	2.050e-6	15	48	–	–	–	–	256	5	–
	Aruba	10	0.46	32	1.217e-4	9.704e-4	20	64	–	–	–	–	128	5	–
AGCRN	Milan	20	0.37	32	3.812e-4	9.720e-6	16	96	–	–	–	–	–	3	20
	Aruba	70	0.30	32	9.648e-4	3.398e-4	15	32	–	–	–	–	–	1	10

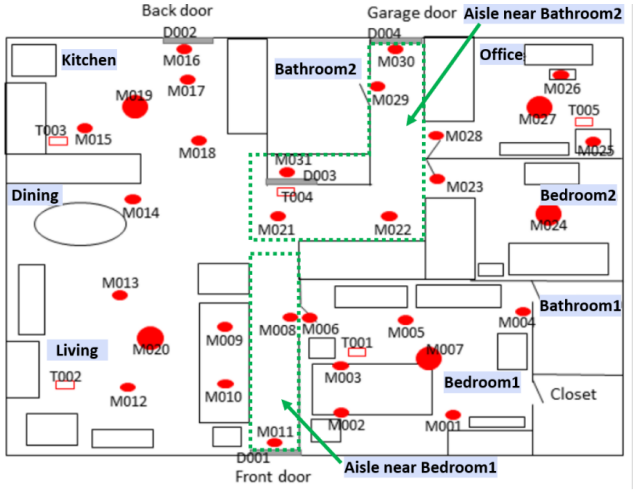

Figure 20: Floorplan of the Aruba dataset

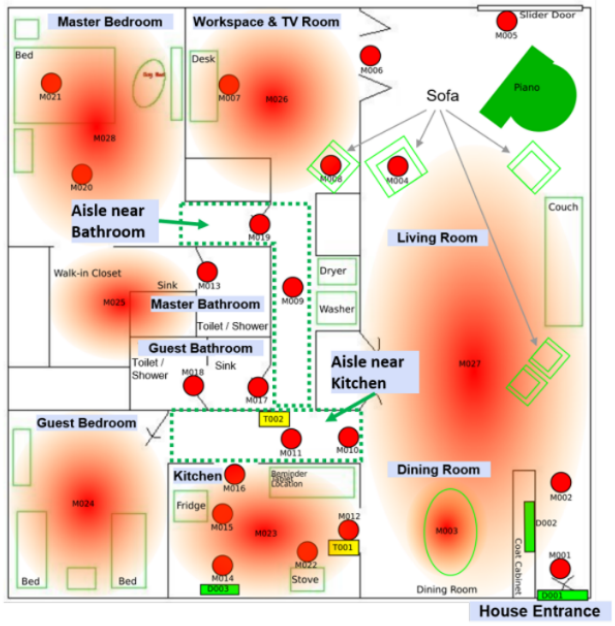

Figure 21: Floorplan of the Milan dataset