Hierarchical Multi-Label Classification of Building Management System Time-Series Data

Haokai Zhao[*]
haokai.zhao@student.unsw.edu.au
UNSW Sydney
Sydney, NSW, Australia

Jonas Macken[*]
j.macken@student.unsw.edu.au
UNSW Sydney
Sydney, NSW, Australia

Leo Dinendra[*]
l.dinendra@student.unsw.edu.au
UNSW Sydney
Sydney, NSW, Australia

Yaqing He[*]
yaqing.he@student.unsw.edu.au
UNSW Sydney
Sydney, NSW, Australia

Ruiyuan Yang[*]
ruiyuan.yang1@student.unsw.edu.au
UNSW Sydney
Sydney, NSW, Australia

Abstract

The Brick by Brick (BBB) challenge focuses on automating the classification of building management system (BMS) time-series data to enhance energy efficiency and operational insights. This dataset presents a multi-label hierarchical classification problem, requiring models to capture dependencies among labels.

Our solution involves extracting statistical and spectral features from raw time-series data and expanding the training set using a splitting-based augmentation technique, increasing its size 28-fold. Labels are organised into five hierarchical tiers, and we train an ensemble of Random Forest (RF) and XGBoost (XGB) models using 10-fold cross-validation and sequential tiered predictions. Our submission achieves a **macro F1-score of 0.528** on the public + private data leaderboard, highlighting the effectiveness of hierarchical ensemble learning and data augmentation for multi-label classification in smart building applications.

CCS Concepts

• **Computing methodologies** → **Ensemble methods**; • **Applied computing** → *Engineering*.

Keywords

Time-series classification, Multi-label classification, Hierarchical classification, Building management systems (BMS), Feature extraction, Data augmentation, Smart buildings, Machine learning

ACM Reference Format:
Haokai Zhao[*], Jonas Macken[*], Leo Dinendra[*], Yaqing He[*], and Ruiyuan Yang[*]. 2025. Hierarchical Multi-Label Classification of Building Management System Time-Series Data. In *Companion Proceedings of the ACM Web Conference 2025 (WWW Companion '25), April 28-May 2, 2025, Sydney, NSW, Australia.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3701716.3718484

[*]All authors contributed equally to this research.

1 Introduction

The **Brick by Brick (BBB) competition** is a multi-label classification challenge using time-series data from IoT sensors in three Australian buildings. The goal is to develop methods that accurately classify samples while accounting for hierarchical relationships in the dataset.

The dataset, derived from *Building TimeSeries (BTS)* [4], spans three years and contains over 10,000 time-series data points. It is standardised using the Brick schema. The main challenges include high variance in signal values, the limited performance of native multi-label models due to category similarity, and extreme class imbalance affecting minority labels.

To tackle these challenges, our approach involves:

(1) abundant hand-crafted feature extraction to perform classification based on statistical, temporal, and spectral domain features of the signals;
(2) data augmentation for both training and testing by splitting the time-series into multiple chunks;
(3) hierarchical label structuring and classification to simplify the multi-label task to a multi-class one and utilise the relationship between labels; and
(4) an ensemble of machine learning classifiers, including Random Forest (RF) [2] and XGBoost (XGB) [3].

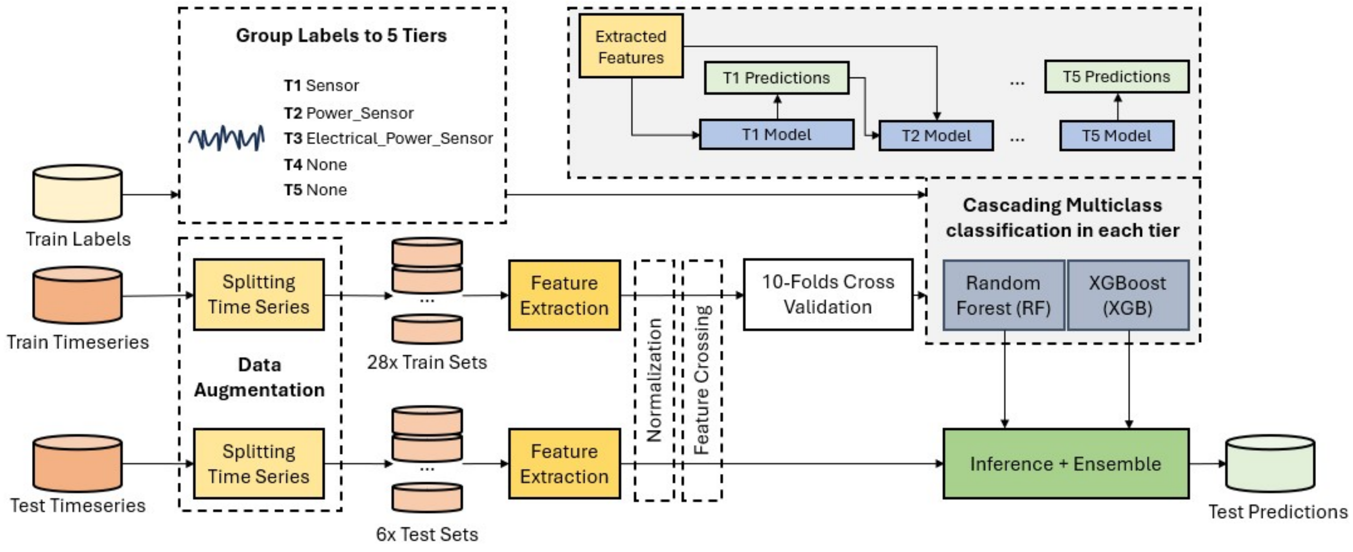
2 Methods

In the following sections we describe our methodology, including data pre-processing, model training, and post-processing strategies. An overview of the complete pipeline is shown in Figure 1.

2.1 Dataset and Task Definition

The Brick by Brick dataset contains 94 unique labels, each representing different aspects of the building management system, with some labels related to others through a hierarchy. Given a univariate time-series data sample, the objective is to classify it with the appropriate label(s).

Figure 1: Overview of Our Team's Solution Pipeline for the Brick-by-Brick Challenge.



2.2 Feature Extraction

We applied a feature extraction process that combines statistical, temporal, and spectral domains to extract possible insights from raw time-series data. Most of our features were computed using the TSFEL library [1], such as statistical features mean, variance, skewness, kurtosis, entropy, and various first-order and second-order time-based differences. Spectral domain features included spectral entropy, spectral centroid, fundamental frequency, and power bandwidth, among others, offering insights into the signal's frequency distribution and energy characteristics. Additionally, the continuous wavelet transform (CWT) was applied to compute features such as wavelet entropy, wavelet energy, and wavelet standard deviation, capturing possible time-frequency patterns. In addition to using TSFEL, we also computed time-based features such as timestamp's slope, burstiness, event density, and time entropy to quantify temporal patterns in the time recordings, independent of the recorded values. These features provided a fundamental representation of the dataset's distribution and variation over time. A total of 70 features were extracted, and the complete list can be found in Table 2.

After feature extraction, we applied normalisation to both the training and test samples to minimise the domain gap. Given the large variance in some features, classical normalisation methods like min-max or z-score would suppress differences for samples with small feature values. Therefore, we implemented a custom bin-normalisation approach. For each feature, we ranked the samples in ascending order and used the p-th percentile as the splitting point, with $p \in \{1, 1000\}$. Additionally, we applied feature crossing to capture interactions between features. We identified pairs of features with correlation less than 0.2 and combined them into new features by adding, resulting in 19 new features.

2.3 Data Augmentation

To increase the diversity and volume of the training dataset, we implemented a **splitting-based augmentation** technique. Each time-series sample was split into multiple segments, and features were extracted from each segment independently. Specifically, we applied splits of $n = 2, 3, 4, 5, 6$ and 7, meaning each sample was divided into halves, thirds, quarters, fifths, sixths, and sevenths, respectively. Including the original samples, this augmentation expanded the training dataset to **28 times** its original size and significantly improved model generalisation. Additionally, we applied this technique to the test set with splits of $n = 2$ and 3 and expanded the test dataset to **6 times** its original size.

2.4 Hierarchical Label Structure

The competition dataset is derived from a condensed label version of the Brick Schema. Given the hierarchical nature of the labels, we structured the classification process into five hierarchical tiers:

- **Tier 1** contains the most general parent labels: Alarm, Command, Parameter, Status, Sensor, and Setpoint.
- **Tiers 2-5** include increasingly specific subcategories of parent labels.

The hierarchical tiers were constructed through an iterative refinement process based on label co-occurrence. Initially, all labels were assigned to Tier 1, with Tiers 2–5 left empty. Starting from Tier 1, labels that only ever appeared in the presence of another label—without occurring independently—were identified as subsets and moved to the next lower tier. This process was then repeated for Tier 2, where labels that exclusively co-occurred with another Tier 2 label were moved to Tier 3, and so on. The procedure continued until all labels were distributed across five tiers, ensuring that the hierarchical dependencies inherent in the dataset were accurately captured.

2.5 Ensemble Learning with Tiered Classifiers

We trained an ensemble of Random Forest (RF) [2] and XGBoost (XGB) [3] classifiers to make predictions at each hierarchical tier. The ensemble training process consisted of:

- **K-Fold Cross-Validation:** We applied 10-fold cross-validation, training a separate classifier on each fold. This resulted in 10 classifiers per model and tier, totaling 50 Random Forest and 50 XGBoost classifiers across all tiers.
- **Handling "None" Labels:** Since many samples contained no meaningful labels in later tiers, we implemented a maximum ratio threshold for "None" labels in each tier during training. Training folds were downsampled to ensure that "None" labels did not dominate the training distribution.
- **Sequential Training:** Predictions from Tier 1 were used as additional features for Tier 2, with this process continuing iteratively for subsequent tiers. This ensured that hierarchical dependencies were effectively captured and leveraged during training.
- **Hyperparameter Tuning:** Each classifier was fine-tuned using grid search and validation metrics to optimise performance.
- **Class Weights**: Initial training was conducted with equal class weights for all labels. The training process was then repeated using class weights derived from the macro F1-score of each label on the validation set.

2.6 Prediction and Post-Processing

For test set inference, we extracted the same statistical and spectral features as in training. The trained ensemble classifiers then made predictions for each tier:

- **Soft Voting Ensemble:** Classifiers within the ensemble were weighted equally, and soft voting was used to aggregate predictions.
- **Hierarchical Predictions:** Predictions at each tier were either a label or "None." If a "None" label was predicted at a tier, all subsequent tiers for that sample were also set to "None."
- **Final Label Assignment:** The predictions across all five tiers were combined into a single multi-label classification output, representing the final labels for each test sample.

2.7 Implementation Details

Hyperparameters: We conducted a grid search for both Random Forest and XGBoost. Since tuning the Random Forest did not yield improvements, we retained its default parameters. For XGBoost, we set n_estimators=400, subsample=0.8, colsample_bytree=0.8, and kept the remaining parameters at their default values. Additionally, all random seeds were fixed to 42 to ensure reproducible results.

Hardware & Software: The experiments were conducted using the hardware and software specifications listed in Table 3. The full implementation code and complete list of packages can be found in https://gitlab.aicrowd.com/leocd/brickbybrick2024_naivebaes.

3 Results

Our model secured third place on the public + private data leaderboard, significantly outperforming the provided baselines. Specifically, our method achieved 0.528 macro f1-score, 0.593 precision, 0.546 recall, and 0.477 mean average precision, as shown in Table 1.

3.1 Performance and Comparison with Baselines

The performance analysis, visualised in Figure 2, shows a high variance in per-class precision and recall due to the strong class imbalance in the training set. The mean precision-recall curve shows the model's trade-off between precision and recall, with an overall tendency toward precision. As shown in Table 1, our model significantly outperforms all baselines. Our hierarchical multi-label approach enables structured predictions, achieving a strong balance between precision and recall.

3.2 Time and Memory Requirements

Preprocessing the training dataset takes 11 hours for 28 sets (25 minutes per set), while the test dataset takes 22 hours for 6 sets (240 minutes per set). Each XGBoost model trains on each fold of all 28 training sets in 4 minutes. With 10 folds and 5 prediction tiers, this results in a total runtime of 2.3 hours. Meanwhile, each Random Forest model trains on each fold in 41.3 seconds, totaling 25 minutes. Prediction, followed by post-processing, requires an additional 30 minutes. Each full experiment generates 100 GB of data, with 20 GB of memory required for model inference.

4 Conclusion

We presented a hierarchical multi-label classification approach that achieved third place in the competition, using a splitting-based data augmentation technique to expand the dataset, a hierarchical label structure to capture class dependencies, and an ensemble learning strategy with sequential tiered predictions.

Future work could explore the n-split data augmentation technique to determine the optimal value of n for maximising performance and further improving the model. Additionally, other methods such as the sliding window or shuffling window techniques could be investigated.

References

[1] Marília Barandas, Duarte Folgado, et al. 2020. TSFEL: Time Series Feature Extraction Library. *SoftwareX* 11 (2020). doi:10.1016/j.softx.2020.100456 GitHub repository: https://github.com/fraunhoferportugal/tsfel.

[2] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32. doi:10.1023/A:1010933404324

[3] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, 785–794. doi:10.1145/2939672.2939785

[4] Arian Prabowo, Xiachong Lin, Imran Razzak, Hao Xue, Emily W Yap, Matthew Amos, and Flora D Salim. 2024. BTS: Building Timeseries Dataset: Empowering Large-Scale Building Analytics. *Thirty-Eighth Annual Conference on Neural Information Processing Systems* (2024).

Table 1: Results on the Public + Private Dataset (Main Metric is the Macro F1-Score).

Method	Accuracy	Precision	Recall	**F1-score**	mAP
Zero	0.9769	0.0000	0.0000	0.0000	0.0000
Random Uniform	0.5005	0.0231	0.4993	0.0356	0.0233
Random Proportional	0.9563	0.0239	0.0224	0.0097	0.0234
Mode	0.9664	0.0000	0.0106	0.0001	0.0000
One	0.0231	0.0231	**1.0000**	0.0396	0.0231
Our Submission	**0.987**	**0.593**	0.546	**0.528**	**0.477**

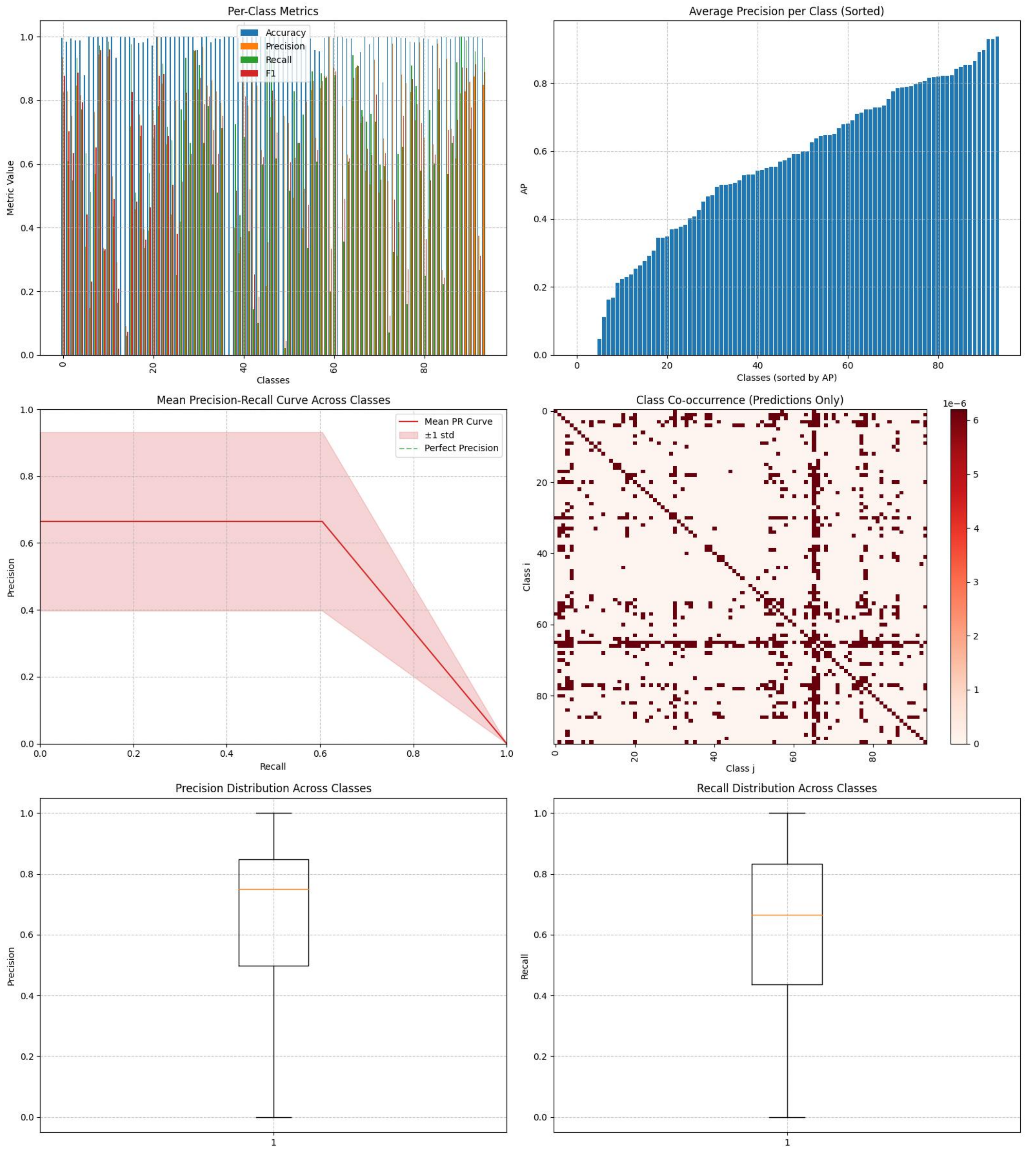
Table 2: Spectral (🟦), Statistical (🟩), and Temporal (🟥) Features Used

Feature Name	Description	Feature Name	Description
Fundamental frequency	Computes the fundamental frequency of the signal. (fundamental_frequency)	**Human range energy**	Computes the human range energy ratio. (human_range_energy)
Max power spectrum	Computes the maximum power spectrum density of the signal. (max_power_spectrum)	**Maximum frequency**	Computes the maximum frequency of the signal. (max_frequency)
Median frequency	Computes the median frequency of the signal. (median_frequency)	**Power bandwidth**	Computes power spectrum density bandwidth of the signal. (power_bandwidth)
Wavelet entropy	Computes the CWT entropy of the signal. (wavelet_entropy)		
Absolute energy	Computes the absolute energy of the signal. (abs_energy)	**Area under the curve**	Computes the area under the curve of the signal (trapezoid rule). (auc)
Autocorrelation	Calculates the first 1/e crossing of the autocorrelation function (ACF). (autocorr)	**Average power**	Computes the average power of the signal. (average_power)
Centroid	Computes the centroid along the time axis. (calc_centroid)	**ECDF Percentile Count_0**	Computes the cumulative sum of samples that are less than the given percentile. (ecdf_percentile_count)
ECDF Percentile Count_1	Same as above but with a different percentile. (ecdf_percentile_count)	**ECDF Percentile_0**	Computes the percentile value of the ECDF. (ecdf_percentile)
ECDF Percentile_1	Same as above but with a different percentile. (ecdf_percentile)	**Entropy**	Computes the Shannon entropy of the signal. (entropy)
Histogram mode	Computes the mode from a histogram of the signal. (hist_mode)	**Interquartile range**	Computes the interquartile range of the signal. (interq_range)
Kurtosis	Computes the kurtosis of the signal. (kurtosis)	**Max**	Computes the maximum value of the signal. (calc_max)
Mean	Computes the mean value of the signal. (calc_mean)	**Mean absolute deviation**	Computes the mean absolute deviation of the signal. (mean_abs_deviation)
Mean absolute diff	Computes mean absolute differences of the signal. (mean_abs_diff)	**Mean diff**	Computes the mean of consecutive differences of the signal. (mean_diff)
Median	Computes the median of the signal. (calc_median)	**Median absolute deviation**	Computes the median absolute deviation of the signal. (median_abs_deviation)
Median absolute diff	Computes median absolute differences of the signal. (median_abs_diff)	**Median diff**	Computes the median of consecutive differences of the signal. (median_diff)
Min	Computes the minimum value of the signal. (calc_min)	**Negative turning points**	Computes number of negative turning points. (negative_turning)
Neighbourhood peaks	Computes the number of peaks in a defined neighborhood. (neighbourhood_peaks)	**Peak to peak distance**	Computes the peak to peak distance. (pk_pk_distance)
Positive turning points	Computes number of positive turning points. (positive_turning)	**Root mean square**	Computes the root mean square of the signal. (rms)
Signal distance	Computes the signal traveled distance. (distance)	**Skewness**	Computes the skewness of the signal. (skewness)
Slope	Computes the slope of the signal (linear regression). (slope)	**Standard deviation**	Computes the standard deviation of the signal. (calc_std)
Sum absolute diff	Computes the sum of absolute differences of the signal. (sum_abs_diff)	**Variance**	Computes the variance of the signal. (calc_var)
Zero crossing rate	Computes the zero-crossing rate of the signal. (zero_cross)	**value_median**	Median of the values (not from reference).
value_mean	Mean of the values (not from reference).	**value_qmean**	Interquartile mean of the values (not from reference).
value_max	Maximum of the values (not from reference).	**value_min**	Minimum of the values (not from reference).
value_maxmin	Difference between max and min of the values (not from reference).	**value_diffmax**	Max of consecutive differences (not from reference).
value_diffmin	Min of consecutive differences (not from reference).	**value_diffmean**	Mean of consecutive differences (not from reference).
value_diffqmean	Interquartile mean of consecutive differences (not from reference).	**value_diffmedian**	Median of consecutive differences (not from reference).
value_diffmaxmin	Difference between max and min of consecutive diffs (not from reference).	**value_std**	Standard deviation of the values (not from reference).
value_var	Variance of the values (not from reference).	**value_diffstd**	Std of consecutive differences (not from reference).
value_diffvar	Variance of consecutive differences (not from reference).		
time_diffmean	Mean of timestamp differences (not from reference).	**time_diffqmean**	Interquartile mean of timestamp differences (not from reference).
time_diffmax	Maximum of timestamp differences (not from reference).	**time_diffmin**	Minimum of timestamp differences (not from reference).
time_diffmedian	Median of timestamp differences (not from reference).	**time_diffstd**	Std of timestamp differences (not from reference).
time_diffvar	Variance of timestamp differences (not from reference).	**time_burstiness**	$(\text{std\_diff} - \text{mean\_diff})/(\text{std\_diff} + \text{mean\_diff})$ (not from reference).
time_total	Total time span (not from reference).	**time_event_density**	Events per unit time (not from reference).
time_entropy	Entropy of timestamp differences (not from reference).	**time_slope**	Slope of timestamps (not from reference).

Table 3: Hardware and Software Specifications for the Experiments

Hardware		**Software**
Component	**Specification**	**(Python 3.9)**
Processor	Intel® Core™ i7-12700H (24 MB cache, 16 cores, up to 4.70 GHz)	scikit-learn (1.6.1)
RAM	32 GB	xgboost (2.1.3)
GPU	Nvidia GTX 1060 (6 GB VRAM)	pandas (2.2.3)
Disk Space	100 GB free space required per experiment	numpy (1.26.4)
		scipy (1.13.1)
		tsfel (0.1.9)

Figure 2: Visualisation of Our Submission's Performance on the Public Dataset.


Per-Class Metrics
Accuracy
Precision
Recall
F1
Metric Value
Classes
Average Precision per Class (Sorted)
AP
Classes (sorted by AP)
Mean Precision-Recall Curve Across Classes
Mean PR Curve
±1 std
Perfect Precision
Precision
Recall
Class Co-occurrence (Predictions Only)
Class i
Class j
1e−6
Precision Distribution Across Classes
Recall Distribution Across Classes