BrickMIR: A Minimal, Imbalance-tuned, and Ratio-based Framework for Brick Metadata Classification

Jun Hao Chan
chan.jun.hao.e0421461.01@u.nus.edu
National University of Singapore
Singapore

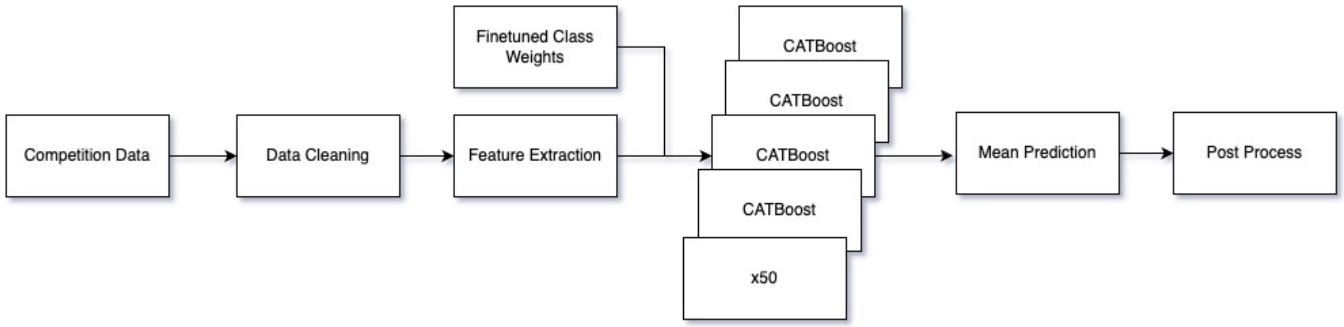

Figure 1: Overview of the Pipeline Architecture

Abstract

We propose a multi-label time-series classification framework for building metadata labeling. Rather than relying on absolute values (which vary widely across different buildings), we focus on ratio- and correlation-based features that capture each sensor's relative behavior. Along with minimal data cleaning and label-tuned class weights, this approach generalizes well across buildings with differing baselines, as demonstrated on the Brick by Brick 2024 dataset [8]. Source code is publicly available at https://github.com/chanjunha0/BrickMIR.

CCS Concepts

• **Software and its engineering**; • **Applied computing** → *Physical sciences and engineering*;

Keywords

multi-label classification, time-series classification, building metadata, Brick schema

ACM Reference Format:
Jun Hao Chan. 2025. BrickMIR: A Minimal, Imbalance-tuned, and Ratio-based Framework for Brick Metadata Classification. 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, 4 pages. https://doi.org/10.1145/3701716.3718481

1 Introduction

The *Brick by Brick 2024* challenge[1] aims to standardize building metadata classification via a multi-label time-series task. The dataset [8] includes timestamped signals (e.g., temperature, setpoints, alarms) from three anonymized Australian buildings. Each time series is labeled based on a modified Brick schema (v1.2.1). Automatically assigning these labels benefits sustainable building management by reducing manual annotation effort.

2 Methods

2.1 3 Key Strategies

We build our framework around three primary strategies:

2.1.1 Minimal Data Cleaning to Preserve Signals. Since missing or outlier readings can themselves be indicative of sensor downtime or power loss, we adopt a minimal cleaning approach. We remove only abrupt drops to zero if they fall beyond a certain Z-score threshold. Rather than imputing missing values, we keep them as a feature, reflecting offline periods that may characterize certain device classes.

2.1.2 Label-Tuned Class Weights. Class imbalance is prevalent in building metadata, where some labels (e.g., specialized sensors) are rare compared to common ones. Instead of default class-weight implementations, we apply a custom weight scaling function that slightly boosts minority labels without overly penalizing major classes. This helps maintain both precision and recall.

2.1.3 Ratio-Based Features. Buildings differ in absolute setpoints and typical temperature ranges. Hence, absolute-value features risk poor generalization. We instead favor ratio-based and correlation-based features that capture relative behavior. For instance, using the

[1]https://www.aicrowd.com/challenges/brick-by-brick-2024

5th-to-95th percentile ratio captures a signal's distribution without anchoring to building-specific baselines.

2.2 Data Cleaning

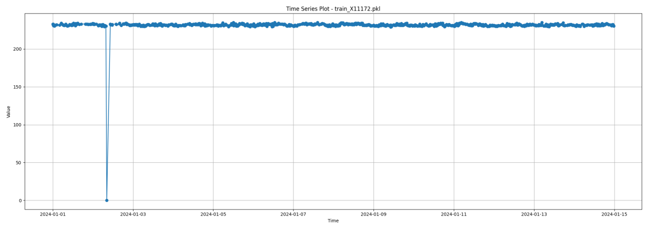

Figure 2: Sensor "0" reading

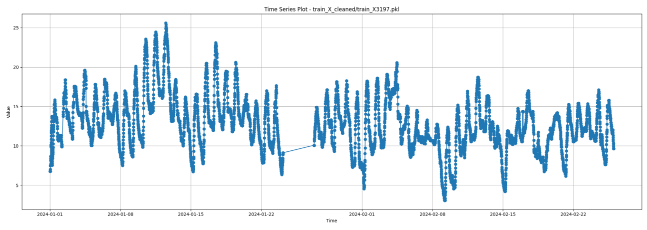

Figure 3: Missing Timesteps

Following our first strategy (minimal cleaning), we exclude only extreme zero-valued outliers that exceed a user-defined Z-score (default=3) (Fig.2). Large positive spikes are retained to avoid discarding genuine device behavior. We do not fill missing timesteps, as they often imply sensor downtime (Fig.3). Binary or near-binary signals (e.g., alarms, certain setpoints) are exempt from this filtering to preserve valid on/off patterns.

One of the strongest pieces of evidence that minimal data cleaning is effective is that `missing_runs_ratio` is among the top-ranked features, and by a large margin. This indicates that the presence and pattern of missing data provide valuable information, reinforcing our decision to avoid imputation.

2.3 Feature Engineering

Consistent with our ratio-based approach, we compute 52 features that highlight patterns relatively invariant to a building's baseline:

- **Statistical and Time-Series Features:** Captures trends, seasonality, and periodic patterns in the time series. These include autocorrelation, frequency domain transformations, entropy measures, peak characteristics, and rolling statistics. The Tsfresh [3] time-series library was referenced. Some absolute values were retained due to it showing a high permutation importance on the cross validation setup. However, the number of absolute value features were kept to a minimum.
- **Ratio-Based Features:** Encode relationships between statistical properties by normalizing percentiles, unique values, and recurrence patterns, making features robust to scale variations.
- **Missing Value Features:** Measure missing timesteps and zero-runs, helping to identify operational downtimes and data sparsity issues.
- **Setpoint Flags:** Binary flags representing different control setpoints (e.g., temperature, humidity, air flow) in HVAC and environmental control systems. These flags were engineered from observing the unique values found in the training data for setpoints.

Table 1: Feature Categories and Their Purpose

Category	Number of Features
Time-Series Features	16
Ratio-Based Features	11
Missing Value Features	3
Setpoint Flags	22
Total	**52**

Figure 4 shows the mean permutation importance of the features.

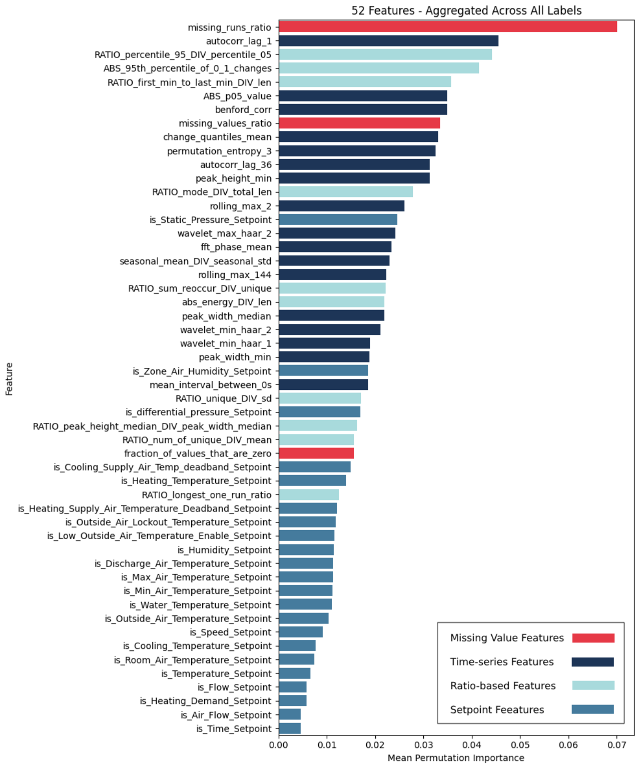

Figure 4

2.4 Model Training

We train one binary classifier per label, a standard One-vs-All approach [9] for multi-label tasks. We tested three gradient-boosted tree models: XGBoost [2], CatBoost [5], and LightGBM [6] with default hyperparameters. CatBoost achieved the best overall balance between precision and recall, so we adopted it for our final submission.

2.5 Class Imbalance

Our second key strategy is to mitigate class imbalance using label-tuned weights. For each label ℓ, we define the positive-class weight pos_weight_ℓ as

$$\text{pos\_weight}_\ell = \max\left(1.0,\ \alpha \times \frac{N_{\ell,\text{neg}}}{N_{\ell,\text{pos}}}\right), \tag{1}$$

where $N_{\ell,\text{neg}}$ and $N_{\ell,\text{pos}}$ are the number of negative and positive samples for label ℓ, respectively, and α ($0 < \alpha < 1$) is a tuned scaling factor. We clamp pos_weight_ℓ from at 1.0 so that we do not downweight the positive label even if it appears more often in some subset of data.

We experimented with methods such as SMOTE [1] and undersampling [7], but found that carefully tuning and clamping the positive-class weight yielded better performance. This approach boosts underrepresented labels (where $\alpha \times \frac{N_{\ell,\text{neg}}}{N_{\ell,\text{pos}}} > 1$) while avoiding unintended penalties on labels that happen not to be minority.

Table 2: Comparison of class imbalance strategies.

Method	F1	Precision	Recall
Baseline (none)	0.513	0.711	0.466
Auto balanced	0.539	0.542	0.625
Auto sqrt balanced	0.552	0.552	0.545
Finetuned scaling factor	**0.559**	**0.665**	**0.553**

2.6 Class Imbalance

Our second key strategy is to mitigate class imbalance using label-tuned weights. For each label ℓ, we define the positive-class weight pos_weight_ℓ as

$$\text{pos\_weight}_\ell = \max\left(1.0,\ \alpha \times \frac{N_{\ell,\text{neg}}}{N_{\ell,\text{pos}}}\right), \tag{2}$$

where $N_{\ell,\text{neg}}$ and $N_{\ell,\text{pos}}$ are the number of negative and positive samples for label ℓ, respectively, and α ($0 < \alpha < 1$) is a tuned scaling factor. We clamp pos_weight_ℓ at 1.0 so that we do not downweight the positive label even if it appears more often in some subset of data.

To further address class imbalance, we explored several additional strategies:

- **SMOTE:** Synthetic Minority Over-sampling Technique that generates synthetic samples for minority classes [1].
- **Undersampling:** Reduces the number of samples in majority classes to balance the dataset [7].
- **Auto balanced:** Automatically calculates class weights inversely proportional to class frequencies, assigning higher weights to minority classes [4].
- **Auto sqrt balanced:** Similar to Auto balanced but uses the square root of class frequencies for weight calculation, providing a more moderate adjustment [4].

Our experiments indicated that fine-tuning and clamping the positive-class weight yielded superior performance. This approach amplifies underrepresented labels (where $\alpha \times \frac{N_{\ell,\text{neg}}}{N_{\ell,\text{pos}}} > 1$) while avoiding undue penalties on labels that are not true minorities.

Table 3: Comparison of class imbalance strategies.

Method	F1	Precision	Recall
Baseline	0.513	0.711	0.466
SMOTE [1]	0.342	0.308	0.300
Undersampling [7]	0.180	0.164	0.129
Auto balanced [4]	0.539	0.542	0.625
Auto sqrt balanced [4]	0.552	0.552	0.545
Fine-tuned scaling factor	**0.559**	**0.665**	**0.553**

2.7 Cross Validation

We use stratified 3-fold cross validation (`StratifiedKFold`) to preserve label distribution. Because some labels are especially rare, $k = 3$ balances minority-sample coverage with multiple folds. To reduce variance, we train with multiple random seeds.

2.8 Brick Schema Post-Processing (Optional)

While our One-vs-All classifiers provide strong independent label predictions, the Brick schema is inherently hierarchical. We experimented with a two-pass approach to enforce consistency:

(1) *Depth-First Pruning:* If a parent's probability is below a threshold, we set all descendants to zero.
(2) *Sibling-Wise Selection:* Among siblings, only the subtree of the highest-probability child is retained.

Though it enforces hierarchy, this process slightly lowered our overall F1 score. **Consequently, our final submission does not apply these constraints.**

3 Results

3.1 Comparison with Baselines

Table 4 shows that our BrickMIR approach outperforms challenge baselines [8] in precision and recall.

3.2 Local CV vs. Public Leaderboard

We observed a gap between local cross validation results and the public leaderboard (Table 5), which stems from the distribution mismatch between training and unseen test buildings. Despite this, our method maintains robust performance.

3.3 Training and Inference Time

Table 6 outlines approximate time costs for each step. Data cleaning and feature extraction were run locally on an M3 MacBook Air with 24GB RAM, while model training and inference were performed on Google Colab.

3.4 Visualization of Best Submission

Figure 5 shows our best public leaderboard submission, visualizing macro-averaged F1, precision, and recall for each label set.

Table 4: Comparison with Challenge Baselines

Method	Precision	Recall	F1 Score
Zero	0.000	0.000	-
Random Uniform	0.152	0.500	-
Random Proportional	0.151	0.147	-
Mode	0.099	0.187	-
LR	0.000	0.000	-
XGBoost	0.099	0.187	-
Transformer	0.151	0.866	-
Informer	0.151	0.362	-
DLinear	0.152	0.752	-
PatchTST	0.151	0.728	-
BrickMIR-XGB	0.576	0.501	0.498
BrickMIR-LBM	0.415	0.556	0.416
BrickMIR-CAT-PostProcess	0.635	0.456	0.492
BrickMIR-CAT	**0.665**	**0.553**	**0.559**

Table 5: Local CV vs. Public Leaderboard.

Data	F1	Precision	Recall
Local CV	0.707	0.716	0.720
Public Leaderboard	0.559	0.665	0.553

Table 6: Approximate Time Costs (8 Core Parallelized)

Process	Time (H:M)	Platform
Data Cleaning	00:25	Local (M3)
Feature Extraction	00:30	Local (M3)
Model Training	00:24	Google Colab (High-RAM CPU)
Inference	00:02	Google Colab (High-RAM CPU)
Total	01:21	-

4 Conclusion

We presented a minimal, imbalance-tuned, ratio-based classification pipeline for building metadata. Key choices include limiting data cleaning so that anomalies remain informative, using ratio/correlation features to avoid building-specific baselines, and scaling class weights. Our model outperforms baselines on the Brick by Brick 2024 dataset, demonstrating better generalization across diverse buildings. Future work will investigate hierarchical loss functions that impose stricter Brick schema constraints without reducing overall F1.

Acknowledgments

We thank Flora, Arian, Matthew, and the University of New South Wales Digital Infrastructure Energy Flexibility team for providing this dataset and organizing the competition. Their work has made this real-world problem both practical and insightful.

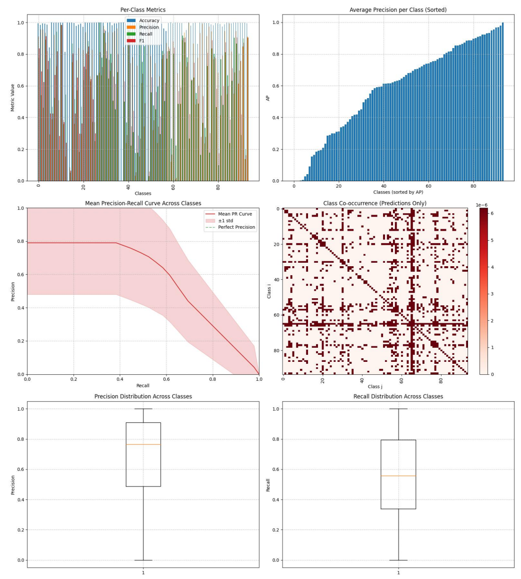

Figure 5: Performance breakdown of our best public leaderboard submission.

References

[1] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* 16 (2002), 321–357. doi:10.1613/jair.953

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

[3] Maximilian Christ, Nicholas Braun, Julius Neuffer, and Andreas W Kempa-Liehr. 2018. Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh–A Python package). *Neurocomputing* 307 (2018), 72–77.

[4] CatBoost Developers. 2025. CatBoost Training Parameters. https://catboost.ai/docs/en/references/training-parameters/common. Accessed: 2025-02-16.

[5] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. 2018. CatBoost: unbiased boosting with categorical features. *Advances in Neural Information Processing Systems* 31 (2018). https://arxiv.org/abs/1810.11363

[6] Guolin Ke, Qiwei Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, Vol. 30. https://papers.nips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html

[7] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. 2017. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *Journal of Machine Learning Research* 18 (2017), 1–5. https://arxiv.org/abs/1609.06570

[8] 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).

[9] Ryan Rifkin and Aldebaro Klautau. 2004. In defense of one-vs-all classification. *Journal of Machine Learning Research* 5 (2004), 101–141.