Another Brick in the Wall: Leveraging Feature Extraction and Ensemble Learning for Building Data Classification

Bram Steenwinckel
bram.steenwinckel@ugent.be
Ghent University - imec
Ghent, Belgium

Sofie Van Hoecke
sofie.vanhoecke@ugent.be
Ghent University - imec
Ghent, Belgium

Femke Ongenae
femke.ongenae@ugent.be
Ghent University - imec
Ghent, Belgium

Abstract

Accurately classifying building time series data enables the identification of patterns, detecting anomalies, and the understanding of how people behave within a building and eventually leads to the automated control, predictive maintenance, and efficient energy management of smart buildings. The 2024 Brick by Brick competition aimed to create machine learning models that classify devices, which output building time series data according to the standardized Brick schema. Since manually labeling all these devices can require significant effort, the competition's goal was to automate this process. In this work, the competition's multi-label classification problem is reframed as a multiclass task in order to apply a classical multiclass classification approach. Our solution involves a structured pipeline where we extract statistical, temporal and frequency-domain features from various time intervals and apply feature selection techniques to enhance the model's efficiency. An extra-trees classifier, optimized using stratified k-fold cross-validation, forms the core of our model. Our method achieves a macro F1-score of 0.5767 on the Brick by Brick's public leaderboard test set, resulting in the 4th place and demonstrating its effectiveness in automating building device classification. By reducing manual effort and improving classification accuracy, our approach contributes to more scalable and intelligent building management systems.

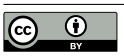
CCS Concepts

• **Computing methodologies** → **Supervised learning by classification**; • **Information systems** → **Data analytics**; • **Applied computing** → Architecture (buildings).

Keywords

Time Series Classification, Building Sensor Data, Smart Buildings, Multiclass Classification, Ensemble Learning

ACM Reference Format:
Bram Steenwinckel, Sofie Van Hoecke, and Femke Ongenae. 2025. Another Brick in the Wall: Leveraging Feature Extraction and Ensemble Learning for Building Data 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, 5 pages. https://doi.org/10.1145/3701716.3718480



This work is licensed under a Creative Commons Attribution 4.0 International License.
WWW Companion '25, Sydney, NSW, Australia
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1331-6/2025/04
https://doi.org/10.1145/3701716.3718480

1 Introduction

Buildings consume about 40% of global energy, making efficient management essential for sustainability [13]. Smart technologies, such as Internet Of Things (IoT) enabled devices, AI-driven analytics, and automated building management systems, can optimize the energy usage by dynamically adjusting lighting, heating, ventilation, and air conditioning based on real-time occupancy and environmental conditions [12]. Solutions today leverage predictive algorithms and machine learning to anticipate the energy demand, reduce waste, and eventually enhance the overall efficiency [4]. However, buildings consist of a large amount of different IoT devices with inconsistent data formats although they offer the same functionality. This problem hinders the effectiveness of current AI solutions [5]. Integrating diverse building devices in one system without standardization is challenging, as inconsistency hinders interoperability, which limits the automation, increasing operational costs, and eventually reduces the effectiveness of energy-saving strategies [15]. Classifying IoT building devices onto a given standardized schema is, therefore, an important first step to ensure that the energy-saving decisions can be made across diverse platforms and technologies [15].

A promising ontology to make building IoT devices and their output interoperable is Brick: a uniform metadata schema for buildings [1]. This metadata standard provides a structured way to represent and categorise building sensors and equipment data. However, manually annotating and classifying building IoT devices according to the provided Brick schema can be labor-intensive and costly. This is one of the limitations that slowed down the widespread adoption of smart building technologies, restricting their potential to enhance sustainability and energy efficiency [7]. To address this challenge, the Brick by Brick competition[1] aimed to develop automated solutions for classifying building IoT devices into the Brick schema using time series data. It focused on machine learning models to categorize diverse data streams like sensor readings, setpoints, and alarms within the hierarchical Brick schema [10]. Such a model could potentially revolutionize facility management and enhance real-time monitoring, fault detection, or predictive maintenance in general [11].

In this paper, we transformed the competition's multi-label classification problem into a multiclass problem, assigning each time series sample to one mutually exclusive label. We did this to enhance the model performance and to achieve a robust solution. We employed an ensemble machine learning approach, integrating feature extraction, feature selection, and model optimization. Details of our pipeline are in Section 3, results in Section 4, unsuccessful attempts in Section 5, and the conclusion in Section 6.

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

2 Competition Background

The Brick by Brick competition is a multi-label time series classification challenge designed to automate building devices classification using IoT data [10].

The dataset consists of time series data collected from three anonymized buildings in Australia, capturing a range of sensor readings, equipment statuses, and environmental parameters such as temperature, setpoints, and alarms. The data is stored as timestamp-value pairs, where the timestamps are provided in relative terms. One of the primary challenges is the irregular sampling rates, meaning that data points do not occur at fixed intervals, requiring models to adapt to inconsistencies in time series observations.

All data from the three buildings are combined into a single dataset and then segmented into the following distinct sets:

- Training set: This portion of the dataset is available for participants to develop and fine-tune their models.
- Leaderboard set: This subset is used to assess the model's performance during the competition, providing intermediate rankings.
- Secret competition set: This final dataset is used to determine competition winners, ensuring unbiased evaluation.

Approximately 20% of the time series were used for the training set, 45% for the leaderboard testing set, and 35% for the secret competition set. The length of each time series varied. Therefore, the series in each set were further divided into shorter segments or chunks with durations ranging from minimum 2 to maximum 8 weeks. This approach allows for assessing model performance across different observation windows, testing its adaptability and consistency. The resulting training set contains 31.840 samples, the combined leaderboard set and secret set contains 315.721 samples.

2.1 Labeling Structure

The competition provides a hierarchical labeling system based on a modified version of the Brick schema (v1.2.1)[2], limited to the 94 Point subclasses. Label indicators were categorized as follows:

- Positive labels: These consist of the true label indicators for each sample, including both the fine-grained label and its parent classes within the hierarchy. This means that a sample's label spans multiple levels using these label indicators.
- Zero labels: These are subclasses of the fine-grained true label that are hidden during evaluation. Masking these subclasses ensures that the model is not penalized when it predicts a more specific subclass than the one provided in the ground truth, especially when the most specific label within the hierarchy is unknown.
- Negative labels: These represent unrelated classes that the model should avoid predicting. The purpose of these labels is to guide the model to not make incorrect predictions by excluding irrelevant classes.

This hierarchical approach encourages classification models to use the relationships between the hierarchical classes and enhance the model's ability to generalize and learn more effectively. For example, if the model confuses an Air Temperature Sensor with a Water Temperature Sensor, the hierarchical approach allows it to correctly classify both as the more general Temperature Sensor, leveraging their shared parent class to enhance generalization and avoid misclassifying them as unrelated sensors, like a Temperature Setpoint.

[2]https://github.com/BrickSchema/Brick/releases/tag/v1.2.1

2.2 Evaluation Metrics

The competition is evaluated according to the macro F1 score, which ensures balanced performance across all labels and it prevents bias toward the more frequent classes. The evaluation follows these key steps:

(1) Macro F1 calculation: Precision and recall are computed for each label class.
(2) Label masking: Predictions more specific than the ground truth are accepted, avoiding Macro F1 penalties for over-specification.
(3) Final scoring: The final ranking is based on the average macro F1 score across all labels.

Additionally, the mean Average Precision (mAP) is calculated as a secondary metric, which evaluates model performance by considering multiple confidence thresholds to generate precision-recall curves and aggregate scores. This metric is particularly useful when machine learning models output probabilities for each label class as it provides a more nuanced assessment of the model's ability to rank the predictions accurately.

3 Methods

Our solution to the competition follows a classical machine learning pipeline structured into multiple steps as illustrated in Figure 1. Each of these steps will be examined in greater detail within this section. Ultimately, our pipeline resulted in a trained model that, when feature extraction techniques are appropriately applied, can classify unseen time series.

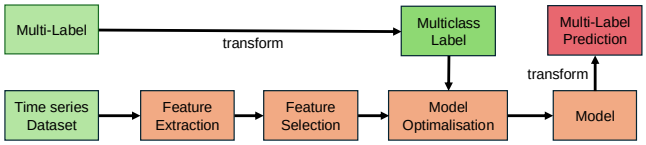


Figure 1: Pipeline with feature extraction, feature selection, and model optimization steps and multi-labels transformed into a multiclass setting for improved performance.

3.1 Multiclass Label

As indicated within Section 2, each time series sample within our dataset has multiple labels, indicated by the hierarchy of the Brick schema. For example, when a sample has the indoor temperature sensor label, it also has the temperature sensor and sensor label to specify this hierarchical aspect.

This multi-label approach means, theoretically, that each sample will belong to more than one class. Given the fact that this competition had to deal with 94 possible classes, this would result in 2^{94} possible outcomes (either labeled as true or false per class). However, in practice, when dealing with the Brick hierarchical structure, the amount of possibilities is much lower. The Brick schema hinders

a sample to, for example, have both the temperature sensor label and the alarm label.

Therefore, we identified all the unique label combinations based on the training set. We did this by simply concatenating all the label names according to a sample together. By doing this, we get unique labels like: Sensor$Temperature_Sensor$Air_Temperature_Sensor. In total, only 91 such unique label combinations are available in the training set, which is much lower than lower than 2^{94}. Hence, the idea to transform the original problem into a 91-class multiclass classification problem. By doing this transformation, our approach is, however, limited to detecting only these 91 label combinations and will not be able to predict any other combination.

Predictions and evaluations are still performed in a multi-label manner. This means that the prediction of a multiclass model will have to be transformed back to the individual labels. The algorithm in Listing 1 performs this transformation.

Listing 1: Transforming multiclass labels back to multi-labels

```
y_pred = pd.DataFrame(np.zeros((#rows, 94)))
for r in range(len(predictions)):
    for value in (predictions[r].split('$')):
        if value != '':
            y_pred.loc[r, value] = 1
```

3.2 Feature Extraction

For the feature extraction in our pipeline, we employ a variety of domain-specific techniques to extract meaningful features from time series data.

3.2.1 Full time series feature extraction. Our pipeline first processes the whole time series samples to calculate statistical features like skewness and kurtosis using the scipy stats module [14]. Additionally, frequency domain features, such as spectral entropy and dominant frequency, are derived via the welch method from scipy signal. We further calculate shape-based features like autocorrelation, peak and valley counts, and trend analysis. Next, we defined domain-specific functions for the different device types: sensor, alarm, setpoint, status, and parameter time series. Unique features such as noise level, event density, and outlier ratio were computed based on the expected behavior of these sensors, based on the data from the training set. Temporal features, such as the variance and standard deviation of time differences, inter-arrival times, and periodicity were also extracted. We also included entropy-based features, such as the permutation entropy and sample entropy [3]. Additionally, fractal dimension and event-based features (burstiness and inter-event times) were computed to capture the time series' underlying complexity.

This multi-faceted approach ensures that the extracted features provide a comprehensive representation of the time series, enhancing the predictive power of the subsequent machine learning models. In total, this resulted in 67 unique features. More details about all the features can be found within the feature extractor files in our repository[3].

[3]https://github.com/predict-idlab/brick-by-brick-pipeline

3.2.2 Interval-based time series feature extraction. As indicated, all of the above features were extracted from the available values within a time series sample. As the different time series had a different amount of values and the above feature extraction techniques did not take into account the time aspect within these time series, we created a second group of features that more closely aligned the different time. For each sample, we used the timestamps to resample the values in averages of 5 minutes, 1 hour, 1 day, and 1 week data and calculated the same set of features as in Section 3.2.1 on those resample series. This resulted in $4x67$ additional features.

3.2.3 Timestamp feature extraction. In addition to the primary feature extraction steps, another interesting aspect for analyzing the data involves the consistency and regularity of the timestamps between consecutive values. An alarm time series will provide values at different time intervals compared with a temperature sensor, which can have a fixed sensor rate. To account for these differences, we calculated the time differences between consecutive timestamps. This is achieved by using the Pandas [8] diff() method on the timestamp column and converting the result to seconds. Next, we assess the most common time difference ((mode, the statistical function that identifies the value appearing most frequently in a dataset) across these differences, which serves as the expected interval. By comparing each timestamp interval with the expected value, we can identify the differences or irregularities in the time sequence. These deviations are captured by flagging the time differences that differ from the expected interval. The total number of these deviations, together with the expected interval value are provided as two additional features. We hoped with these two features that sudden gaps in data or other anomalies in the timestamp distribution could be detected.

3.3 Feature Selection

In total, the previous feature extraction module created 337 ($5 \times 67 + 2$) features. As they were generated in bulk at once, it might be the case they are not all as relevant for the multiclass problem as expected. Hence, the idea to use feature selection to reduce the amount of features. We utilized the SelectFromModel feature selector from scikit-learn [9] because it works with any scikit-learn estimator that provides feature importances, allowing us to easily adapt our pipeline to possible different machine learning models. This method works by fitting a model that uses all the features and then evaluating their importance based on the model's internal feature ranking. Specifically, it calculates feature importance scores and selects the most significant features that contribute the most to the model's predictions. Through experimental evaluation and in combination with an optimized model, we were able to retain only the top 40 features ensuring that we focused on those that had the greatest impact on the model's macro F1 score. We did this to reduce the possible risk of overfitting, enhancing computational efficiency, and improving the interpretability of the results. Figure 2 provides an overview of these top 40 features, ranked by their importance.

3.4 Model Optimization

As machine learning model choice, we used the extra-trees classifier as our final model after multiple experimentation with various

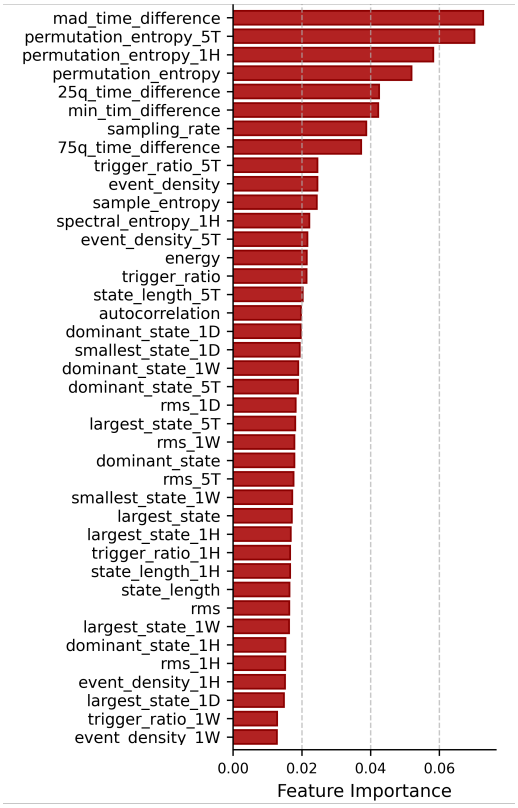


Figure 2: Feature importance for the top 40 features.

ensemble methods, including random forests. Extra-trees, or extremely randomized trees, is an ensemble learning method that builds multiple decision trees and averages their predictions to enhance accuracy and reduce overfitting [2]. Unlike random forests, which selects optimal split points based on feature values, extra-trees introduces additional randomness by choosing split points entirely at random within a feature's range. We used the scikit-learn implementation in all our evaluations [9].

Our choice of extra-trees was guided by our internal evaluations, where it consistently outperformed other ensemble methods in terms of predictive performance and stability. Our internal evaluation existed out of a stratified k-fold cross-validation approach with $k = 5$, ensuring that each fold preserved the distribution of the imbalanced classes. This stratification was important to effectively capture variations in the data and prevent bias toward dominant classes. By tracking the predictive performance across multiple iterations, we assessed the impact of additional features, feature selection strategies, and model optimization techniques. We evaluated the macro F1 score for the multi-labels on the holdout validation sets by reverting the multiclass labels, as explained in Section 3.1. The average macro F1 scores of these holdout validation sets provided us with strong evidence of which configurations led to the most interesting results. This approach also aligned with the leaderboard results on the test set (meaning that an increase in average validation also led to an increased score on the leaderboard). Hence, with a minimal number of submissions, we could already achieve interesting results on the unseen test set.

Several parameters of the extra-trees classifier were tuned within this cross-validation loop using an additional randomized grid search cross validation of 30 iterations (with 3 cross-validation splits). While the outer cross-validation loop assesses the model's overall performance and generalization, the inner grid search fine-tunes the model's parameters using these splits, ensuring that the chosen hyperparameters perform well on unseen data and aren't overfitted to any specific training set from the outer loop. Table 1 shows the tuned parameters, together with the best obtained parameters for our model.

Table 1: Extra-trees classifier hyperparameters

Parameter	Range	Best Parameter
n_estimators	randint(100, 1000)	314
max_depth	randint(10, 100)	30
min_samples_leaf	[2, 4, 8]	2
min_samples_split	[2, 4, 8]	4
criterion	['entropy', 'gini']	entropy
class_weight	[None, 'balanced']	None
max_features	[None, 10, 50, 100, 250]	None

Our final extra-trees model was then trained on the entire training dataset using the best obtained parameters and the 40 highest-performing features.

4 Results

The results of our approach and configurations on the public leaderboard test set are provided in Table 2. These results confirm the effectiveness of our incremental strategy. The results are first presented for the full features (described in Section 3.2.1) and an initial interesting solution was obtained using a random forest classifier. Later on, this model was replaced by the extra-trees classifier, as mentioned earlier. In subsequent iterations, we first integrated the interval features (Section 3.2.2) on top of the full features, then added the time features (Section 3.2.3), and finally reduced the feature set by using the feature selector to retain the 40 best-performing features.

5 Unsuccessful Attempts

During the construction of our pipeline, we explored several additional optimization strategies to enhance our model's performance. However, these efforts did not improve the score in our internal validation, and therefore, they were not submitted for evaluation on the public leaderboard.

One strategy we explored involved applying class and sample weighting based on the hierarchical depth of the label. The goal was to assign more weight to certain classes based on their position in the hierarchy, potentially improving accuracy for more nuanced categories. Despite careful calibration, this approach did not lead to any measurable performance improvement.

We also explored a multi-stage classification approach. In this method, we initially trained a classifier to distinguish between the broader, high-level category classes within the Brick schema. After this initial classification, a more specialized classifier was used to differentiate between the finer-grained classes within each broader category. Although this hierarchical breakdown aimed to

Table 2: Results on the Public Leaderboard Set. The 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
Random forests (Features full)	0.9869	0.5108	0.4811	0.4692	0.4061
Extra-trees (Features full)	0.9879	0.6200	0.5132	0.5216	0.4884
Extra-trees (Features full+intervals)	0.9883	0.6432	0.5567	0.5616	0.5201
Extra-trees (Features full+intervals+time)	0.9884	0.6477	0.5587	0.5650	0.5244
Extra-trees (Features top 40)	**0.9886**	**0.6521**	**0.5677**	**0.5767**	**0.5310**

streamline classification and improve performance, it did not result in better validation scores.

Finally, we experimented with a hierarchical classification approach using the hiclass Python package, designed specifically for hierarchical classification problems [6]. By leveraging the label hierarchy, we aimed to better capture relationships between parent and child classes. However, once again, our internal results did not improve.

6 Conclusion

In this work, we presented an automated approach for classifying building devices using their time series output data by transforming the original multi-label classification task into a multiclass problem. By employing a standardized machine learning pipeline that integrates feature extraction, feature selection, and model optimization, we developed a potential robust solution for smart building data classification. Our approach demonstrates the effectiveness of ensemble methods, particularly the extra-trees classifier, to improve classification performance while maintaining computational efficiency. All code to replicate our results is available at our Github repository: https://github.com/predict-idlab/brick-by-brick-pipeline

Beyond the provided Brick-by-Brick competition context, our work contributes to the broader field of smart building management by offering a robust and standardized data classification method. Accurate automated classification of building time series data enables improved facility management, real-time monitoring that can ultimately support further energy efficiency and sustainability efforts. Future research could explore other strategies for handling rare and unseen labels to enhance this model's generalization capabilities. The insights gained from this work provide a solid foundation for advancing intelligent building automation. Our current findings encourage further innovation in automated building data classification.

Acknowledgments

This work has been (partially) funded by the Flanders AI Research programme.

References

[1] Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, Jason Koh, Joern Ploennigs, Yuvraj Agarwal, et al. 2018. Brick: Metadata schema for portable smart building applications. *Applied energy* 226 (2018), 1273–1292.

[2] Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine learning* 63 (2006), 3–42.

[3] Bill Kay, Audun Myers, Thad Boydston, Emily Ellwein, Cameron Mackenzie, Iliana Alvarez, and Erik Lentz. 2024. Permutation Entropy for Signal Analysis. *Discrete Mathematics & Theoretical Computer Science* 26, Special issues (2024).

[4] Sangkeum Lee, Sarvar Hussain Nengroo, Hojun Jin, Yoonmee Doh, Chungho Lee, Taewook Heo, and Dongsoo Har. 2023. Power management in smart residential building with deep learning model for occupancy detection by usage pattern of electric appliances. In *Proceedings of the 2023 5th International Electronics Communication Conference*. 84–92.

[5] Xiachong Lin, Arian Prabowo, Imran Razzak, Hao Xue, Matthew Amos, Sam Behrens, Stephen White, and Flora D Salim. 2024. A Gap in Time: The Challenge of Processing Heterogeneous IoT Point Data in Buildings. *arXiv preprint arXiv:2405.14267* (2024).

[6] Fábio Miranda, Niklas Köhnecke, and Bernhard Y Renard. 2023. HiClass: A Python Library for Local Hierarchical Classification Compatible with Scikit-Learn." arXiv (Cornell University), December.

[7] Sakshi Mishra, Andrew Glaws, Dylan Cutler, Stephen Frank, Muhammad Azam, Farzam Mohammadi, and Jean-Simon Venne. 2020. Unified architecture for data-driven metadata tagging of building automation systems. *Automation in Construction* 120 (2020), 103411.

[8] The pandas development team. 2020. *pandas-dev/pandas: Pandas.* doi:10.5281/zenodo.3509134

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[10] 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).

[11] Mashud Rana, Ashfaqur Rahman, Mahathir Almashor, John McCulloch, and Subbu Sethuvenkatraman. 2023. Automatic Classification of Sensors in Buildings: Learning from Time Series Data. In *Australasian Joint Conference on Artificial Intelligence*. Springer, 367–378.

[12] Aya Sayed, Yassine Himeur, Faycal Bensaali, and Abbes Amira. 2022. Artificial intelligence with iot for energy efficiency in buildings. In *Emerging Real-World Applications of Internet of Things*. CRC Press, 233–252.

[13] Vibhu Sharma. 2024. Integrating renewable energy with building management systems: Pathways to sustainable infrastructure. *Journal of Waste Management & Recycling Technology* 2, 1 (2024).

[14] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. doi:10.1038/s41592-019-0686-2

[15] Filippo Vittori, Chuan Fu Tan, Anna Laura Pisello, Adrian Chong, and Clayton Miller. 2023. BIM-to-BRICK: Using graph modeling for IoT/BMS and spatial semantic data interoperability within digital data models of buildings. *arXiv preprint arXiv:2307.13197* (2023).