Graph Neural Networks for Node Classification and Attribute Allocation in Architectural BIM

Wassim Jabi[1], *Yang Li*[2]
[1,2]*Cardiff University*
[1,2]*{jabiw|liy327}@cardiff.ac.uk*

Building Information Modelling (BIM) marks a notable shift in architectural design, extending beyond simple digital reproductions by capturing the spatial, physical, and operational characteristics of structures. Unfortunately, these representations are often complex in nature and difficult to inspect, analyze, and understand which can lead to errors and omissions during model construction. This research aims to leverage graph machine learning systems, utilizing learned datasets, to detect and rectify these issues, improving model quality and minimizing costly mistakes. To illustrate the application of graph neural networks in this domain, this paper applied a graph-based geometric and topological editor coupled with a graph neural network to a real-world dataset of residential building complexes. The developed workflow operates by converting traditional architectural floor plans into graph-structured data, enabling precise node classification predictions. The paper details the overall workflow, data preparation and conversion, hyperparameter optimization and experimental results. Comparing the performance of various graph neural network models has validated the efficiency of the chosen prediction model in processing and analyzing architectural floor plans, achieving an overall accuracy rate of approximately 95%. The paper concludes with a discussion of the potential and limitations of graph-based machine learning methodologies within the architectural domain and an outline of future work plans.

Keywords: *Topology, Artificial Intelligence, Machine Learning, Graph Neural Network, Node Classification, Floor Plans.*

INTRODUCTION

In contemporary architectural design, Building Information Modelling (BIM) has become an indispensable digital three-dimensional tool to design and specify a project. BIM models encode the relationships among the various components of the project. However, the creation and maintenance of BIM models are resource-intensive tasks, fraught with the potential for human error. Architects and engineers spend countless hours manually inputting data into digital BIM models, risking inconsistencies and inefficiencies. A possible solution to this issue takes advantage of the graph-like structure of BIM models which paves the way for the application of a field of artificial intelligence called graph-based machine learning (GML). If we can harness the power of GML, we can not only visualize and analyze these networks but also predict and automatically assign attributes to

increase efficiency and reduce errors. This is the main aim of and motivation for this research.

Graphs and Graph Machine Learning

Zhou et al (2020) describe a graph as a fundamental data structure for illustrating relationships between entities, where nodes denote entities and edges signify relationships. In the field of urban analysis and space syntax, Batty (2004) relied on graphs to represent relationships between streets – where each street is represented as a node in the graph. Early in the field of computational methods in architecture, March and Earl (1977) analysed the the problem of counting various classes of architectural plans and their adjacency structures using the graphs of trivalent 3-polytopes. Unlike traditional Euclidean structures such as matrices and vectors, graph data exhibits dynamic and variable numbers and orders of node neighbors. This poses a challenge for applying conventional convolutional neural networks to unstructured graphs due to the difficulty in defining regular convolutional filters.

To overcome this hurdle, researchers have adapted machine learning (ML) techniques to learn from graphs. The field of graph-based machine learning (GML) is credited to many researchers who have contributed to its development. However, one seminal paper often credited with laying the foundation for this field is "Semi-Supervised Classification with Graph Convolutional Networks" by Kipf and Welling (2017). This paper introduced the concept of Graph Convolutional Networks (GCNs), which are a type of graph neural network designed for semi-supervised learning tasks on graph-structured data. The paper demonstrated the effectiveness of GCNs in classifying nodes in graph data and sparked significant interest and research in the field of graph-based machine learning.

Through continuous advancements, GCNs have emerged as a predominant method for processing graph data, delivering notable achievements across diverse applications.

RESEARCH METHODOLOGY

The research methodology in this paper focuses on the construction and testing of graph neural network models to automate the classification of room types within architectural floor plans. This involves data preprocessing to enable the conversion of floor plans, derived from an existing public dataset, into graph representations. Feature engineering techniques are then applied to extract relevant features from the architectural data. The connectivity between architectural elements, such as doors, windows, and walls, is carefully considered and encoded to capture the spatial relationships within the graph. Furthermore, feature fusion strategies are employed to integrate zone and connectivity attributes. Finally, graph construction techniques, leveraging a 3D geometrical and topological editor are utilized to construct the architectural graphs, ensuring a structured representation of the building layout for efficient processing by the graph neural network models. Once the dataset is ready, a rigorous experiment was designed to test the classification models against it.

Data Preprocessing

Residential floor plans often categorize areas into distinct zones like living, dynamic, static, and functional zones, each comprising various room types. For example, a dynamic zone may include living rooms, kitchens, and corridors. Visualizing the layout, each room is represented as a node, forming the planar structure of a residence, as depicted in Figure 1.

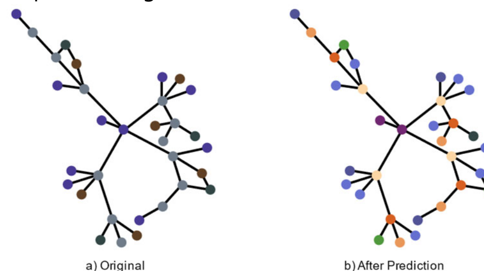

Figure 1 Floor Plan Structure Schematic

Edges in the diagram possess a characteristic called connectivity, indicating architectural elements linking two rooms, notably doors, windows, and walls. Figure 1a showcases nodes colored according to four distinct zone types, while Figure 1b uses nine colors to depict various room types. This study aims to predict room types for each node based on zone and connectivity attributes, leveraging information from Figure 1a to infer Figure 1b.

A residential floor diagram comprises three main parts: the graph, nodes, and edges. The graph includes information about the number of nodes, represented by [num_nodes]. Each node within the graph holds two attributes: zone, representing its area, and room type, the prediction target, forming [zone, room_type] pairs. Edges in the graph contain three pieces of information: source node number, destination node number, and connectivity attribute, denoted as [src, dst, connectivity]. Thus, each edge represents the connection between nodes, crucial for understanding spatial relationships within the floor plan.

Feature Engineering

Feature engineering is the process of selecting, transforming, and creating new features from raw data to improve the performance of machine learning models. It involves extracting relevant information, identifying patterns, and crafting input variables that enhance the model's ability to learn and make accurate predictions. In this paper, feature engineering involved extracting key attributes from the architectural floor plans dataset. These features included zone types, room classifications, and connectivity attributes such as doors, windows, and walls.

When handling different zone types, numerical representation and one-hot encoding proved efficient for feature processing. This method converts each zone into a binary vector with dimensions equal to the total zone types. For example, assuming the categories are (0,1,2,3), if a node's zone type is 2, then its zone feature vector becomes [0,0,1,0]. One-hot encoding prevents categorical variables from being misinterpreted as ordinal, ensuring model accuracy and reliability.

When distinguishing between different room types, understanding the number and types of connectivity within a room is important. For example, areas like stairs or corridors typically exhibit the highest connectivity count, while rooms like living rooms and bedrooms linked to windows may have varying connectivity quantities. Since connectivity varies across room types, accurately tallying each type aids in prediction. However, due to variations in residence size, connectivity counts also fluctuate. Simply counting connectivity counts at each node can increase prediction error. To mitigate this, calculating the proportion of each node's connectivity within the residential graph is crucial. For instance, using one-hot encoding might initially yield an array like [5,0,0] for three connectivity types (0,1,2) where the first type has 5 connections. If connectivity type 0 in the graph total 30 connections, then the processed result becomes [5/30,0,0] for this node. This method reduces prediction errors, enhancing accuracy and robustness.

Feature Fusion

Feature fusion involves concatenating different feature vectors along a specific dimension. This entails combining the zone feature and connectivity feature of each node in every graph to create a new feature vector. For instance, if a node's zone feature and connectivity feature are represented as one-dimensional arrays, they are concatenated into a longer array, thus creating a feature vector with enhanced information. This fusion strategy enables the model to simultaneously process and learn multiple feature information from both zone and connectivity sources.

Graph Construction

Constructing the graph is a fundamental step in graph neural networks, for which the topologicpy library is chosen as the primary tool as described in Jabi and Chatzivasileiadi (2021). Topologicpy is an open-source Python 3 adaptation of Topologic, a robust spatial design and analysis tool in architecture, engineering, and construction based on the concept of non-manifold topology as explained in Jabi and Aish (2018). It integrates Graph Machine Learning (GML) capabilities using the DGL library by Wang et al (2019). DGL is a high-performance Python package for deep learning that enables accurate processing and analysis of interconnected data. The combination of Topologicpy and DGL provide the needed algorithms for graph and node classification, building type identification, association prediction, and completion of missing information in building information models. The library has already been used for graph classification as shown in Alymani et al (2023).

Drawing from these technologies, the graph structure is primarily defined by node count (num_nodes), edge start nodes (src), and end nodes (dst). Src and dst determine the edge direction, while num_nodes sets the total node count. Once the graph structure is defined, predicted node labels (e.g., room types) and features from feature engineering are integrated into the DGL graph's node data. This provides a comprehensive graph structure and node attributes for the graph neural network, optimizing performance across tasks. This approach ensures the model effectively utilizes graph information, resulting in more accurate parsing of room types and associations.

Model Framework Construction

This paper uses a Graph Neural Network model method based on Graph SAGE for node prediction created by Hamilton et al (2017). Graph SAGE leverages neighborhood information for feature extraction and node prediction, allowing the model to operate without the entire graph structure. It aggregates node and neighbor information through multiple layers of aggregation functions, producing feature vectors for subsequent layers. Both node sampling and structural sampling are achieved via random sampling in the Graph SAGE model, ensuring a direct connection between sampled nodes and those aggregated in the graph structure.

The proposed model method differs significantly from traditional Convolutional Neural Networks (CNNs). While CNNs focus on learning from regular grid-like data such as images, Graph SAGE is tailored for learning from graph data with irregular structures. This flexibility enables better understanding of complex structures and relationships within graph data, resulting in higher predictive accuracy and model performance.

Graph SAGE's principle process comprises four steps: Sampling, Aggregation, Updating Target Node Representation, and Classification/Prediction.

Sampling: Graph SAGE conducts neighbor sampling for each node, organizing graph data from inside out into k layers, centered on each target node. For nodes at layer x ($1 \leq x < k$), the model samples Sx neighbor nodes. If a node has fewer than Sx neighbors, all its neighbors are sampled in this step. As illustrated in Figure 2, dashed lines delineate layers where neighbor nodes reside. Central nodes are marked with a cross symbol, first-order neighbor nodes with one horizontal line, and second-order neighbor nodes with two horizontal lines. Blank nodes denote unsampled nodes, and arrows indicate the direction of sampling along edges between nodes.

Aggregation: The aggregation operation gathers and integrates information from a node's neighbors, ensuring a fixed-dimensional feature representation for each node regardless of the varying number of neighbors. Graph SAGE utilizes predefined aggregation functions to integrate

information from all neighbor nodes. For example, in a graph *G*(*V*,*E*) where *V* is the number of nodes and *E* is the number of edges, the average aggregation function can be mathematically represented as:

$$h_{N(v)}^{k} = \frac{1}{|N(v)|} \sum u \in N(v) h_{u}^{k-1} \qquad 1)$$

Where:

- $N(v)$ is the set of neighbor nodes for node v.
- $h_{N(v)}^{k}$ is the aggregated representation of the neighbor nodes at layer k.
- h_{u}^{k-1} is the representation of node u at layer $k-1$.

Updating Target Node Representation: Following aggregation, the model combines obtained neighbor information with the target node's features, typically through concatenation. Subsequently, a fully connected layer and activation function yield the updated node representation, expressed as:

$$\left[h_{v}^{k} = \sigma\left(W^{k} \cdot CONCAT\left(h_{v}^{k-1}, h_{N(v)}^{k}\right)\right)\right] \qquad 2)$$

Where:

- σ is the activation function, for example ReLU or Tanh.
- W^{k} is the trainable weight matrix for layer k.
- $CONCAT$ represents the concatenation operation.

Figure 3 depicts a schematic diagram of node aggregation, where aggregation occurs from second-order nodes to corresponding first-order nodes and then to the central node. Classification is subsequently performed based on the aggregated features of the central node.

Classification/Prediction: The model utilizes the resulting embedding representations from the aggregation stage for classification or prediction tasks. This typically involves employing standard machine learning classifiers like SoftMax to predict node labels.

The Graph Neural Network model employed in this paper consists of four layers, each utilizing Graph SAGE networks and employing the pool aggregation method.

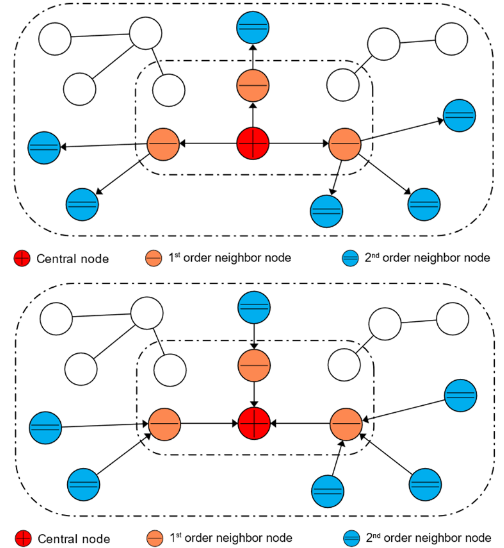

Figure 2
Node Sampling Schematic

Figure 3
Node Aggregation Schematic

First Layer: The model transforms input features into hidden node representations through sampling and pool aggregation operations. Batch normalization is applied to standardize inputs to a mean of 0 and a variance of 1, reducing internal covariate shift and enhancing stability. Subsequently, the tanh activation function is applied for non-linear transformation, compressing input values between -1 and 1 to enable the learning of complex representations.

Second, Third, and Fourth Layers: These layers repeat the process of the first layer, including resampling, aggregation, batch normalization, and tanh activation on updated features.

The output dimension of the fourth layer matches the number of classes for classification tasks. Through progressive learning and integration across these layers, the model effectively captures structural information in the

graph and maps original input features to classification output, ensuring robust learning and generalization capabilities.

EXPERIMENT DESIGN

This section provides an overview of the configuration and design of the experiment in this study, including hardware and software configurations, dataset, hyperparameters, training and validation methods, and performance evaluation metrics. In addition, this section includes a description of a comparative experiment to validate the performance of the chosen experiment design.

Hardware and Software Configuration

The specifications of the computer hardware and operating system, and software libraries used in this experiment are as follows:

- Processor: 11th Gen Intel® Core™ i7-1165G7 @ 2.80GHz, 1690 MHz, 4 Core(s), 8 Logical Processor(s)
- Installed Physical Memory (RAM): 16.0 GB
- GPU: Intel(R) Iris(R) Xe Graphics
- Operating System: Microsoft Windows 10 Education (x64)
- Software Libraries: Python (version 3.10.4), topologicpy (version 0.4.32), pytorch (version 2.0.1), DGL (version 0.9.1).

Dataset

The data utilized in this study is sourced from a Kaggle project available at “Modified Swiss Dwellings” (n.d.). The dataset is derived from the Swiss Dwellings database (v3.0.0) and contains highly detailed floor plans of single and multi-unit buildings from across Switzerland. Figure 4 visually depicts typical floor plans found in this dataset.

The dataset comprises two main file types: .npy and .pickle, totaling 4,167 samples. For this research, pickle data was employed. Here is detailed information about the input and output datasets:

Input Dataset: .pickle file: This file represents a NetworkX graph. Nodes within this graph possess an attribute named "zoning," while edges have an attribute named "connectivity," categorizing various types of access such as "door," "entrance door," and "passage."

Output Dataset: .pickle file: This file also represents a NetworkX graph, where nodes have attributes including:

- roomtype: Categorization of room types such as "Bathroom," "Livingroom," and "Bedroom."
- centroid: The centroid (middle point) of each room.
- geometry: The shape of each room is represented as a polygon using shapely.geometry.Polygon().

Edges in this graph possess an attribute named "connectivity," indicating access types.

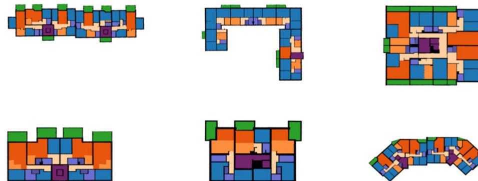

Figure 4
Visual samples from the dataset

Hyperparameters and Model Training

The Graph SAGE model's hyperparameters include the number of hidden layers, the number of units per layer, the activation function, the learning rate, and the batch size. To find the optimal model parameters, a grid search was conducted. Grid search operates by exhaustively testing all possible combinations of hyperparameter values within predefined ranges, creating a grid-like structure, and evaluating each combination using cross-validation to determine the model's performance. The combination with the highest performance metric is then selected

as the optimal set of hyperparameters for the model.

The model was trained using the Adam optimizer, with the learning rate selected from the parameter grid. The Adam optimizer is an adaptive learning rate optimization algorithm that combines techniques such as momentum and adaptive learning rates to efficiently update model parameters during training. Each model underwent training for 20 epochs, utilizing the cross-entropy loss function, commonly used for classification problems. The chosen parameter ranges were:

- Number of Hidden Layers: [2, 3, 4, 5]
- Number of Units per Layer: [8,16, 32, 64]
- Activation Function: ['relu', 'tanh']
- Learning Rate: [0.01, 0.0005, 0.001, 0.0001]
- Batch Size: [16, 32, 64]

Validation

In the experiment, the hold-out validation method is used to evaluate the model. For each graph, all nodes are randomly divided into a training set, a validation set, and a test set, with the respective ratios of 80%, 10%, and 10%. The training set is used for training the model; the validation set is used for model tuning and selection; the test set is used for assessing the final performance of the model. Through this method, the generalization ability of the model can be judged by its performance on unseen data.

Performance Evaluation Metrics

Since this paper investigates classification problems using graph neural networks, four scenarios will arise during the experiment based on the combination of actual classes of sample data and classes predicted by the graph network. Taking binary classification as an example, the four situations are:

1. True Positive (TP): Positive samples correctly predicted as positive by the graph network model.
2. False Positive (FP): Negative samples incorrectly predicted as positive by the graph network model.
3. True Negative (TN): Negative samples correctly predicted as negative by the graph network model.
4. False Negative (FN): Positive samples incorrectly predicted as negative by the graph network model.

To evaluate the model's performance, four main metrics will be employed:

1. Accuracy: The proportion of correctly classified samples to the total number of samples.
2. Precision: The proportion of true positive samples to all samples predicted as positive.
3. Recall: The proportion of true positive samples to all actual positive samples.
4. F1 Score: The harmonic mean of precision and recall, providing a balanced measure between the two.

To further validate the superiority of the Graph SAGE model, commonly used models for node prediction, GCN (Graph Convolutional Networks), and GIN (Graph Isomorphism Network), were utilized as baselines for comparison with Graph SAGE. Performance was compared based on average accuracy, precision, F1 score, and recall.

RESULTS

This section reports the results of the analysis of several hyperparameters with a focus on the aggregation method, number of layers, number of units, and learning rate. The section concludes with a comparison of the performance of the Graph SAGE model with other models frequently used for graph machine learning.

Hyperparameter Optimization

In the experiment, the primary goal is to achieve optimal model performance. To this end, in the first stage all variables are kept constant, with only the aggregation method of Graph SAGE being varied. This approach aims to identify the most suitable aggregation method for the given data. In the initial setup, the following hyperparameters were kept constant:

- Number of hidden layers: 3
- Hidden units: 16
- Activation function: tanh
- Learning rate: 0.005
- Batch size: 32
- Maximum number of epochs: 50

Three types of aggregation methods were tested: mean, pool, and LSTM. The results are summarized in Table 1.

These results suggest that the GraphSAGE-Pool aggregation method yields the highest performance in terms of accuracy while the GraphSAGE-LSTM aggregation method demonstrates the highest precision, recall, and F1-score.

Figure 5a illustrates the change in loss function curves for the Graph SAGE models with different aggregation methods over increasing epochs, while Figure 5b depicts the accuracy trend on the validation set. Additionally, Figure 5c presents the training duration of these models at each epoch.

Observing Figure 5c, it is evident that the training duration of GraphSAGE-LSTM significantly exceeds that of GraphSAGE-Mean and GraphSAGE-Pool. Considering all factors, particularly time efficiency, the GraphSAGE-Pool model appears to be a more suitable choice as it maintains a good balance between performance and training efficiency.

In the next stage, further parameter tuning and in-depth experimental analysis were conducted. While keeping other settings constant, the learning rate was set to 0.0005, batch size to 32, activation function to tanh, and hidden units to 16. Experiments varying the number of layers were performed, specifically setting 2, 3, 4, and 5 layers respectively. The results obtained are presented in Table 2.

These results indicate that the model with 4 hidden layers consistently achieves the highest performance across all evaluation metrics. Further analysis of loss, validation accuracy, and runtime, as shown in Figure 6, also shows that the 4-layer model is the most balanced across all aspects.

Table 1
Performance of models with different aggregation types

	Accuracy	Precision	Recall	F1 Score
GraphSAGE-Mean	93.05	76.00	74.08	74.02
GraphSAGE-Pool	93.70	77.29	75.03	74.96
GraphSAGE-LSTM	93.56	84.58	76.66	77.22

Figure 5
Comparison of models with different aggregation methods

Table 2
Prediction performance of models with different number of layers

	Accuracy	Precision	Recall	F1 Score
2-layer	92.61	78.39	75.17	75.08
3-layer	92.60	78.12	75.32	75.47
4-layer	93.12	78.25	75.51	76.85
5-layer	92.57	78.21	74.99	75.12

Figure 6
Comparison of models with different number of layers

Next, the impact of different numbers of hidden units on model performance was compared. In the experiment, the learning rate was set at 0.0005, batch size at 32, the activation function as

tanh, and the number of hidden layers was fixed at 4. The number of hidden units was varied, specifically set at 8, 16, 32, and 64. The experimental results are summarized in Table 3. The model with 64 hidden units exhibits the best combination of high accuracy, precision, recall, and F1 score.

Finally, with other parameters fixed, the performance of the Graph SAGE model under different learning rates was compared. In these settings, the batch size is set to 32, the activation function is set to tanh, the number of hidden layers is set to 4, and the number of hidden units is set to 64. The learning rates were varied, specifically set at 0.01, 0.005, 0.001, and 0.0005. The experimental results obtained are summarized in Table 4.

Referring to Table 4 and Figures 7a and 7b, the model performs best when the learning rate is set to 0.001. Meanwhile, Figure 7c shows that the training times for models under all four learning rates fluctuate around 3 seconds, so these four models are essentially equivalent in terms of time consumption. Therefore, all things considered, the model with a learning rate of 0.001 is the optimal choice

To verify the superiority of the Graph SAGE model, its performance was compared to three other models typically used for graph machine learning: GCN, GIN, and TAG. The experimental setups for these models were defined with specific configurations tailored to their architectures. The GCN model was configured with 3 layers, each containing a GraphConv layer, followed by BatchNorm1d and ReLU activation functions. The GIN model also had 3 layers, each featuring a GINConv layer, BatchNorm1d, ReLU activation, and a Linear layer. The TAG model comprised an input layer, two hidden layers, and an output layer, all utilizing k=2 TAGConv layers and ReLU activation functions, with dropout applied after each hidden layer. The number of input features for all models was 7, with varying numbers of hidden units and output classes. Training settings for all models included a learning rate of 0.001, a batch size of 16, and 50 epochs. The obtained results are summarized in Table 5.

The results clearly demonstrate that the Graph SAGE model outperforms the other three models across all performance metrics with a final accuracy of 94.74%.

Table 3 Performance of models with different number of units

	Accuracy	Precision	Recall	F1 Score
Units = 8	92.56	73.90	71.82	70.86
Units = 16	92.53	77.90	75.16	75.17
Units = 32	92.57	80.81	74.74	74.69
Units = 64	92.67	84.16	77.62	78.32

Table 4 Performance of models with different learning rates

	Accuracy	Precision	Recall	F1 Score
lr = 0.01	92.66	85.19	81.95	82.98
lr = 0.005	92.71	84.25	77.07	77.49
lr = 0.001	92.64	88.07	83.16	84.80
lr = 0.0005	92.72	85.41	79.36	80.57

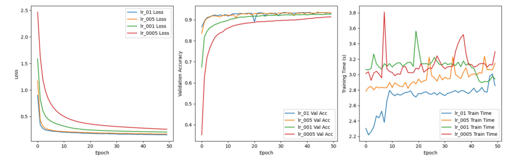

Figure 7 Comparison of models with different learning rates

Table 5 Performance of different graph neural network models

	Accuracy	Precision	Recall	F1 Score
GCN	62.05	57.14	49.27	50.35
GIN	89.75	76.86	74.31	74.70
TAG	88.45	76.02	69.45	69.26
SAGE	94.74	86.55	79.90	81.48

CONCLUSIONS

The application of the Graph SAGE model in classifying nodes within architectural graphs represents a significant and novel advancement in the field of architecture. By converting traditional architectural floor plans into graph-structured data and employing graph neural network models, precise node classification predictions were achieved which enables the automation of architectural information extraction and analysis.

The comparison of various graph neural network models underscored the superiority of the Graph SAGE model on the dataset, achieving a significantly higher accuracy rate of 94.74% when compared to other models. This validation highlights the effectiveness of the chosen approach in handling architectural graphs.

While the research demonstrates notable achievements, there remain areas for improvement. Although the topologicpy library can convert 3D models into graphs, the dataset contains 2D floor plans with a relatively limited number of nodes. Exploring large-scale architectural graphs containing numerous nodes within individual data points could provide valuable insights. Such efforts could enable the model to better handle complex architectural information and extend node classification capabilities from 2D to 3D spaces.

By leveraging graph-based models, architects can enhance the accuracy in BIM creation, but also uncover nuanced spatial relationships and insights that would otherwise be difficult to access. This methodology promises to streamline architectural workflows, optimize resource allocation, and lead to a more sustainable built environment.

NOTES

This paper is based on a masters research dissertation conducted by Yang Li under the supervision of Professor Wassim Jabi at Cardiff University. All code and data are open-source and available at: https://shorturl.at/emnQR

REFERENCES

Alymani, A., Jabi, W., Corcoran, P., 2023. Graph machine learning classification using architectural 3D topological models. Simulation 99. https://doi.org/10.1177/00375497221105894

Batty, M., 2004. A new theory of space syntax.

Hamilton, W.L., Ying, R., Leskovec, J., 2017. Inductive Representation Learning on Large Graphs, in: NIPS.

Jabi, W., Aish, R., 2018. Non-manifold Topology for Architectural and Engineering Modelling, in: Proceedings of the International Conference on Education and Research in Computer Aided Architectural Design in Europe.

Jabi, W., Chatzivasileiadi, A., 2021. Topologic: Exploring Spatial Reasoning Through Geometry, Topology, and Semantics, Advances in Science, Technology and Innovation. https://doi.org/10.1007/978-3-030-57509-0_25

Kipf, T.N., Welling, M., 2017. Semi-Supervised Classification with Graph Convolutional Networks.

March, L., Earl, C.F., 1977. On Counting Architectural Plans. Environ Plann B Plann Des 4, 57–80. https://doi.org/10.1068/b040057

Modified Swiss Dwellings [WWW Document], n.d. URL https://www.kaggle.com/datasets/caspervanengelenburg/modified-swiss-dwellings (accessed 3.27.24).

Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., Zhang, Z., 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M., 2020. Graph neural networks: A review of methods and applications. AI Open 1, 57–81. https://doi.org/https://doi.org/10.1016/j.aiopen.2021.01.001