Augmenting IFC-Based BIM Models for Graph Neural Network Training of Non-Orthogonal Spatial Typologies:

A Context-Informed Design Methodology

Matthias Hornung[1 [0009−0002−9508−3760]], Markus Renner[1 [0009−0004−4752−7448]], Diellza Elshani[1 [0000−0003−2902−341X]], Mathias Niepert[2 [0000−0002−8401−3751]], and Thomas Wortmann[1 [0000−0002−5604−1624]]

[1] Department for Computing in Architecture, Institute for Computational Design and Construction, University of Stuttgart, Germany
[2] Institute for Artificial Intelligence, University of Stuttgart, Germany

Abstract. *Building Information Modeling* (**BIM**) and *Industry Foundation Classes* (**IFC**) data are valuable resources in architecture, offering detailed and structured information on building elements. However, using such resources to develop *artificial intelligence* (**AI**) models that are capable of learning and generating spatial and geometrical suggestions remains a challenging problem.
This study presents a systematic method for converting real-world BIM/IFC projects into graph representations, suitable for training Graph Neural Networks (**GNNs**), which overcome the limitations inherent to conventional AI techniques based on pixel and 3D voxels in architecture.
First, a custom schema was defined to encode each building element and spatial relationship as triplets, allowing for the generation of heterogeneous graphs that capture geometry, topology and semantic labels. Using models provided by multiple architecture offices, we extracted over 180,000 subgraphs that represent diverse non-orthogonal spatial typologies. Next, each subgraph was annotated with multi-label classifications (element type) and continuous attributes (e.g. dimensions, orientation) to support both classification and regression tasks.
Finally, to demonstrate the utility of the dataset, we trained a custom generative *neural network* (**NN**) to complete partial spatial graphs, producing non-orthogonal room layouts based on the existing building context. The custom NN suggests that the dataset has the capacity to support advanced 3D generative tasks in architecture and could lay the groundwork for future studies on automated design synthesis.

Keywords: Building Information Modeling (BIM) · Industry Foundation Models (IFC) · 3D Building Machine Learning · Graph Neural Networks (GNNs)

Hornung, M., Renner, M., Elshani, D., Niepert, M., & Wortmann, T. (2025). Augmenting IFC-Based BIM Models for Graph Neural Network Training of Non-Orthogonal Spatial Typologies: A Context-Informed Design Methodology. In *CAAD Futures 2025 – Catalytic Interfaces: Conference Proceedings, Vol. II*, pp. 869–883. Hong Kong: University of Hong Kong. DOI: 10.25442/hku.29365832

1 Introduction

As building designs become increasingly intricate, architectural tools require geometric and semantic 3D information to support complex design and analysis tasks [12,31]. *Machine learning* (**ML**) models demand richly annotated datasets that represent diverse element types and their interrelations [9,30]. Although *Industry Foundation Classes* (**IFC**) provide a standardized format for detailed building information, their complexity and the lack of publicly available, comprehensive datasets have posed challenges for applying ML methods[9]. Despite the recognized benefits of digital technologies, their adoption within the *Architecture, Engineering and Construction* (**AEC**) sector remains limited, with the construction industry lagging in digitization compared to other sectors [22,4]. This paper defines and implements a methodology to construct a heterogeneous graph dataset from real-world IFC files. We develop a parsing and augmentation pipeline that (1) extracts multiple node and edge types from IFC models, (2) encodes geometric attributes alongside semantic labels, and (3) generates a dataset suitable for downstream generative ML tasks [2]. We evaluate this pipeline through predictive experiments on residential building components, demonstrating its utility and establishing a foundation for future research in three-dimensional generative architectural ML.

2 Background

The following section presents three themes that are potentially relevant to digital innovation in the AEC sector: (1) autocompletion, (2) graph-based representations, and (3) state-of-the-art generative tools. We then examine how to address industry-specific 3D-ML challenges.

2.1 Autocompletion

Autocompletion extends beyond text and code into areas such as image editing and animation, enhancing both efficiency and user experience by predicting and completing content from partial input. As Lehmann and Buschek define, autocompletion systems generate multiple probabilistic suggestions conditioned on user input, allowing users to select or refine proposals in real time [19]. This iterative feedback loop ensures that the final output closely aligns with the user's intent.

2.2 Utilization of Graphs

Graph structures provide a versatile framework for encoding relationships and connectivity patterns across diverse fields, including proteomics, image analysis, software engineering, and natural language processing. By representing entities as nodes and their interactions as edges, graph-based methods integrate topological information directly into processing and analysis tasks [28]. In architectural theory, Hillier argues in *"Space is the Machine"* that the complexity of the built environment emerges from local configurational rules that evolve over time, a phenomenon that descriptive language alone cannot capture [13].

2.3 State of the Art

Prior work in architectural AI can be grouped into three areas: image-based generative methods, graph-based modeling with GNNs, and language-model-driven generative design.

Image-based Generative AI in Architecture Generative image models such as *Midjourney* and *DALL-E* have been adapted for architectural design, producing specialized tools such as *LookXAi* and *Archicad AI Visualizer*. Non-diffusion approaches employ *Convolutional Neural Networks* (**CNNs**) and *Generative Adversarial Networks* (**GANs**) to learn patterns from architectural imagery: *ArchiGAN* focuses on synthesizing 2D floor plans [6], while others add 3D reconstruction via post-processing steps [33,5]. These techniques operate on pixel data alone and do not encode true 3D geometry. *Neural Radiance Fields* (**NeRF**) reconstruct scene geometry from sparse views [24], but they still face challenges in accurately modeling enclosed interiors.

Graphs and GNNs in Architecture In 2D and 2.5D applications, floor plans are represented as program graphs, where nodes denote rooms or elements and edges denote adjacency, enabling GNNs and optimization algorithms to generate layouts effectively [15,10,25]. For 3D, voxel grids are converted into graph structures and processed with GAN NN models to address scalability and long-range dependencies [18,34]. However, voxelized representations struggle with non-orthogonal and unstructured geometries. More recent work embeds multi-scale building components directly into graph structures, capturing spatial and semantic relationships without heavy geometric approximations [17,16,3,21]. These methods advance beyond voxel limits, but still rely on mostly synthetic data.

Autocompletion in Architectural Design A novel development is the use of Retrieval-augmented generation frameworks, where *large language models* (**LLMs**) are integrated with selected knowledge sources to support architectural tasks [20]. In *Text2BIM*, natural language descriptions are converted into 3D BIM elements and executable code, then validated against building regulations with iterative feedback loops [7]. LLM agents function as co-pilots for design autocompletion, though they depend on text and rule-based checks rather than the direct use of geometric IFC data.

2.4 Research Gaps

The transition from purely voxelized 3D approaches to hybrid methods that embed GNNs within voxel grids improves the capture of overall shape but remains unable to distinguish individual architectural components (walls, doors, windows) without extensive post-processing [18]. Instead, unstructured, graph-based methods map building elements directly into nodes and edges, aligning more naturally with architectural complexity [29]. For example, spatial graph embeddings combined with an autoencoder capture latent relationships among building typologies [3].

Despite these advances, two critical gaps persist. First, heterogeneous graphs, which differentiate between node and edge types to represent both geometric connections and semantic groupings, are scarcely employed in architectural ML. For instance, a heterogeneous representation can model a column that is both physically attached to a slab and semantically assigned to a room on a given building

story (Figure 1). Second, most studies use synthetic or simplified data; real IFC datasets from BIM projects remain largely untapped, due to the complexity of parsing IFC schemas and the lack of a standardized graph-conversion pipeline.

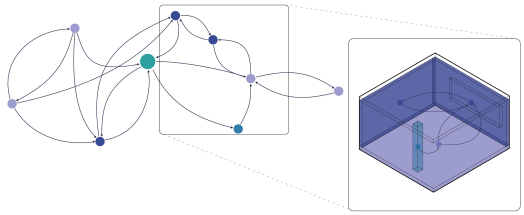


Fig. 1: Building section represented as a directed, heterogeneous graph.

2.5 Problem Statement

As building designs become more intricate, architectural tools must exploit both geometric and semantic 3D information. Therefore, ML models need access to richly annotated datasets that capture diverse element types and their interrelations. While IFC files store detailed building information, transforming IFC files into graphs suitable for GNNs requires a clear schema and robust preprocessing.

This paper's objective is to define and implement a methodology for constructing a heterogeneous graph dataset from real-world IFC data. We will present a parsing and augmentation pipeline that (1) extracts nodes and edges of multiple types from IFC models, (2) encodes both geometric attributes and semantic classifications, and (3) produces a dataset suitable for downstream ML tasks. The effectiveness of this pipeline will be assessed through predictive experiments on residential building components, establishing a foundation for future research in architectural 3D ML.

3 Methodology

This study adopts a three-phase workflow to produce structured graph representations from real building data for architectural ML applications.

1. *Graph Embedding* converts 3D building information from IFC files into heterogeneous graphs, capturing spatial, hierarchical and semantic relationships.
2. *Preprocessing* refines the raw graphs by dividing each building graph into localized subgraphs and forming input–output pairs, which enhances dataset diversity and usability.
3. *Cluster Analysis* uncovers recurrent patterns among buildings and subgraphs, informing the design of training protocols and evaluation metrics.

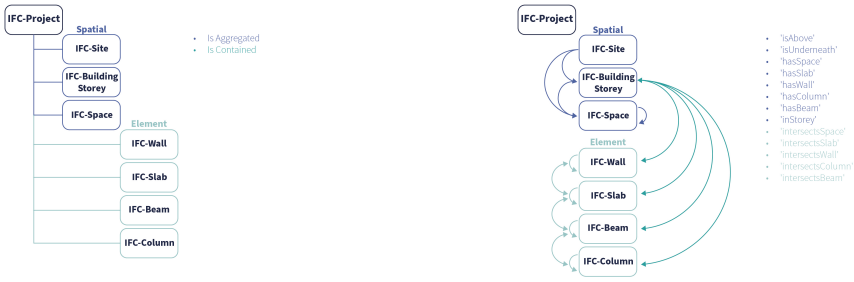
3.1 Dataset

To address the absence of real-world 3D building datasets for architectural ML, we sourced authentic BIM models rather than rely on synthetic data. Despite ex-

tensive searching, no publicly available qualitative dataset was identified. Therefore, we solicited residential BIM projects from architecture offices in Germany and Austria. Of 148 firms contacted, 24 provided substantive feedback, yielding 50 IFC files. After resolving formatting and modeling inconsistencies, 22 files remained suitable for our pipeline. Of the 50 original files, 28 were excluded because they lacked one or more of the six required *IfcElement* classes or were excluded due to corruption or parsing failures in *IfcOpenShell* [14].

3.2 Graph Embedding

The utilization of real-world data presented several challenges that went beyond typical collection efforts. Key issues included the limited number of samples available to train a ML model, as well as the noise and inconsistencies present in the data collected. To address the limited samples, data augmentation techniques, such as rotating and mirroring, were applied, expanding the dataset to approximately 109 buildings. To mitigate inconsistencies derived from the variability in modeling habits and strategies across architectural offices, IFC-standardized conventions for component types were adopted to establish a consistent baseline for all IFC files within the dataset. Due to the lack of standardization, customizable parameters, an advanced feature of BIM, were excluded from the scope of the work. Common issues included incorrect aggregation (e.g. elements modeled in story one but aggregated into story three) and improper labeling of IFC categories (e.g. using *IfcColumn* for façade shading), highlighted the need for rigorous preprocessing to standardize the dataset information. Several *Python* libraries were used to generate graph representations from IFC files. *IfcOpenShell* provided tools for editing IFC data, while *Open3D* [35] enabled bounding-box computation and enriched geometric encoding. Inspired by the *BOTOntology* framework for modeling relationships among building elements [27,8], we developed a custom schema that establishes edges based on bounding-box overlap and IFC containment or aggregation metadata (Figure 2). Each resulting graph is a directed *NetworkX* graph [11], with nodes representing *IfcObject* instances and edges encoding spatial and semantic relations.



(a) Simplified diagram of the IFC structure.

(b) Custom schema used for the graph creation.

Fig. 2: Data schemas.

To simplify graph creation, a total of six basic *IfcElements*, sufficient to describe a simple building, were selected: *IfcBuildingStorey, IfcSpace, IfcSlab, IfcWall, IfcColumn, IfcBeam*. The graph data is expressed as triplets, each comprising a subject, a predicate, and an object. The relationship descriptions listed in (Table 1) were used for the directed edges.

Table 1: Predicates, subjects, and objects for IFC relationships.

Subjects	**Predicates**	**Objects**
IfcBuildingStorey	isAbove	IfcBuildingStorey
IfcBuildingStorey	isUnderneath	IfcBuildingStorey
IfcBuildingStorey	hasSpace	IfcSpace
IfcBuildingStorey	hasSlab	IfcSlab
IfcBuildingStorey	hasWall	IfcWall
IfcBuildingStorey	hasColumn	IfcColumn
IfcBuildingStorey	hasBeam	IfcBeam
IfcSpace/IfcSlab/IfcWall/IfcColumn/IfcBeam	inStorey	IfcBuildingStorey
IfcSpace/IfcSlab/IfcWall/IfcColumn/IfcBeam	intersectsSpace	IfcSpace
IfcSpace/IfcSlab/IfcWall/IfcColumn/IfcBeam	intersectsSlab	IfcSlab
IfcSpace/IfcSlab/IfcWall/IfcColumn/IfcBeam	intersectsWall	IfcWall
IfcSpace/IfcSlab/IfcWall/IfcColumn/IfcBeam	intersectsColumn	IfcColumn
IfcSpace/IfcSlab/IfcWall/IfcColumn/IfcBeam	intersectsBeam	IfcBeam

A custom tool was developed to visualize the generated graphs. To highlight the heterogeneity, a color code was assigned to each node and edge type. Figure 3 shows an example of a housing building from the dataset, presented as a heterogeneous graph. To facilitate ML tasks, a one-hot encoding vector

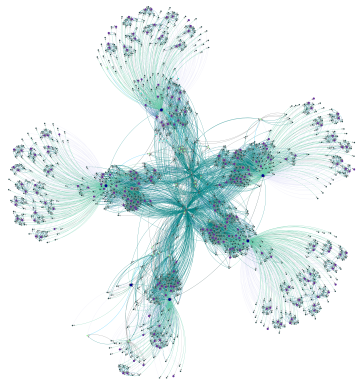
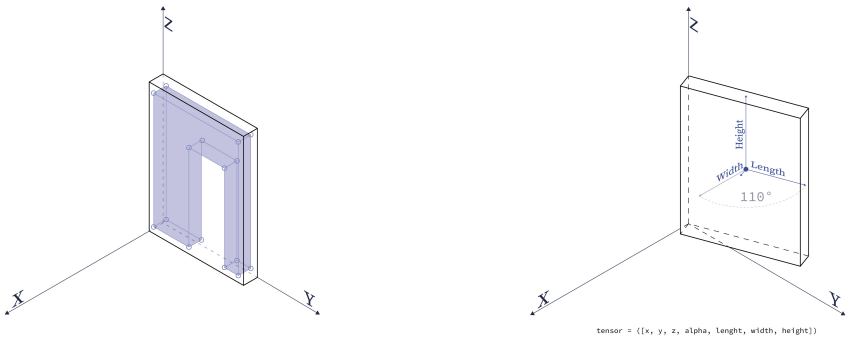


Fig. 3: Example of a building represented as a graph. Colors represent different node and edge types.

representation was employed for the node and edge types, as string representations were inadequate. For node features, a simplified geometry description of each element was encoded by computing a minimal oriented bounding box. The complexity introduced by features such as windows and doors within walls, or openings within slabs, could generate irregular polygonal shapes (Figure 4a). Therefore, the coordinates of the bounding-box center, its width, length, and height, and the normalized vectors for each box dimension were used as node features. Adding the six-dimensional one-hot encoding vector to the node features produced a 15-dimensional vector for each node, ensuring a uniform tensor shape across all nodes.



(a) Minimal oriented bounding box-simplification.

(b) Extraction of geometry description for tensor representation.

Fig. 4: Geometric representation of a wall with a door opening.

To reduce the tensor length, which improves NN learning by decreasing the degrees of freedom in the data, the following vector engineering was proposed:

1. Given the node feature tensor:

$$tensor = [x, y, z, \overline{v}_x, \overline{v}_y, \overline{v}_z, m_x, m_y, m_z] \tag{1}$$

2. The x, y, z values represent the bounding box center point and are normalized over the complete dataset.
3. Where every vector $\overline{v}_x$, $\overline{v}_y$ and $\overline{v}_z$ is a local vector in 3D space, each defined by its three components:

$$\overline{v} = (v_x, v_y, v_z) \tag{2}$$

4. Each local v_x, v_y and v_z has a magnitude equal to 1:

$$\sqrt{v_x^2 + v_y^2 + v_z^2} = 1 \tag{3}$$

5. Where the components m_x, m_y, m_z represent the magnitudes of the local vectors, normalized over the entire dataset.

Testing the original 15-dimensional vectors revealed that their high degrees of freedom produced overly abstract geometries. To support more reliable predictions, we simplified the tensor representation by reducing its parameter count. Since most architectural components align with the global Z-axis, $(0, 0, 1)$, we assumed a local orientation vector $v_z = (0, 0, 1)$ for each element. This constraint removed redundant orientation parameters while retaining essential geometric information. Furthermore, it was determined to project the $\overline{v}_x$ to the global XY-plane and to compute the local $\overline{v}_y$ based on the predicted local $\overline{v}_x$ using cross multiplication:

$$\overline{v}_x \times global\overline{v}_z = \overline{v}_y \tag{4}$$

Restricting the 3D box predictions to have free rotation around the Z-axis (Figure 4b), which aligned the components with the XY-plane. Subsequently, the vector $\overline{v}_x$ was utilized to calculate the angle relative to the global X-axis in degrees, then normalized to the unit interval $[0, 1]$ denoted here as α. The final tensor representation for the geometrical representation was composed as follows:

$$tensor = [x, y, z, \alpha, m_x, m_y, m_z] \tag{5}$$

3.3 Preprocessing

The preprocessing is composed of two phases: graph splitting and graph deconstruction. First, the 109 building graphs were partitioned into smaller subgraphs of uniform sizes to enlarge the dataset. These subgraphs encapsulated a variety of scenarios, including prevalent structural patterns, functional groupings of building elements, and typical spatial arrangements found in the architectural dateset.

Next, a modified *Breadth-First Search* (**BFS**) algorithm was applied to each node, performing a constrained traversal that divided each graph into subgraphs containing each 20 nodes. In total, this procedure yielded over 600.000 heterogeneous subgraphs capturing diverse building components and their local relationships.

Inspired by [23], the second step involves graph deconstruction to produce the ground truth dataset needed for ML training. Each subgraph is iteratively processed by systematically removing nodes and edges. The resulting partially deconstructed graph is stored as the training input along with the removed nodes, which serve as the training ground truth in pairs. The order of node removal is determined by three algorithms: *hierarchical Breadth-first Search* (**hBFS**), *random Breadth-first Search* (**rBFS**) and *Minimum Node Degree* (**MND**). *hBFS* and *rBFS* differ in using a hierarchical prioritized or a random starting point, while *MND* considers the node with the smallest edge degree. These graph traversal algorithms ensured that no decomposed node splits the graph into disjointed

parts. In addition, any resulting homogeneous graph that emerged from the deconstruction phase was deleted to maintain the homogeneity of the dateset.

The employed deconstruction method aimed to create a dataset to be employed in a multi-label classification problem for a simplified *Deep Generative Model of Graphs* (**DGMG**) [23]. The method focused solely on node storage during deconstruction and disregarded edge deletion and termination probability.

Following graph deconstruction, we generated over 1.8 million pairs of subgraphs and their corresponding removed nodes (Table 2). To address the imbalance in removed node types, where walls and spaces were disproportionately overrepresented, we applied over- and undersampling techniques to achieve a balanced distribution. This process produced the *IFC_ 180k* dataset, comprising 180 000 unique samples with no more than 30.000 instances per node type. For evaluation purposes, we also created a smaller dataset, *IFC_ 18k*, containing 18.000 samples and 3.000 elements per type.

Table 2: Table showing the occurrences per type for each deconstruction type employed before under- and oversampling.

	IFC types						
Deconst. Algorithm	IfcBuildingStorey	IfcSpace	IfcSlab	ifcWall	IfcColumn	IfcBeam	**Total**
hBFS	7.774	543.766	365.583	892.761	27.464	31.665	**1.869.013**
rBFS	88.388	527.943	348.089	855.699	25.962	30.079	**1.876.160**
MND	105.221	570.277	322.040	780.784	25.559	29.208	**1.833.089**

The *Deep Graph Library* (**DGL**) [32] was employed to implement graph-specific operations, such as graph convolution and node sampling, during dataset construction and subsequent graph-based ML tasks. For functions beyond core graph processing, we utilized *PyTorch* [1] to implement multi-layer perceptron architectures, pooling mechanisms, and comprehensive training workflows. In this configuration, *DGL* managed the essential graph operations, while *PyTorch* provided the flexibility needed for additional NN components and their integration into the overall model architecture.

3.4 Cluster Analysis

To examine the 109 augmented building graphs and the preprocessed datasets (*IFC_ 18k*), we applied unsupervised clustering using the *K-means* algorithm from *scikit-learn* [26]. In the first step, graph-level embeddings for the 109 building models were then generated using a lightweight GNN architecture with the following hyperparameters:

After obtaining the embeddings at the graph level, a graphical "elbow method" was applied to determine the most representative number of clusters, revealing that $k = 2$ and $k = 4$ were the most expressive, as they represented the points

Hyperparameter	Value
Convolution Method	SAGEConv
Layer Sizes	13-32-64-128
GRU Layer Count	1
Final Layer	AdaptiveMaxPool1d (size = 512)

Table 3: Hyperparameters used in the GNN model.

where the inertia visually decreased the sharpest. Once the number of clusters (k) was determined, dimensionality reduction was performed using *Principal Component Analysis* (**PCA**). After performing *PCA* and *K-means* clustering, the buildings were visualized in a reduced 2D space (Figure 5a), revealing distinct clusters. Each cluster indicates a group of buildings with similar characteristics or features. The same clustering method was applied to three preprocessed datasets, each based on the deconstruction type employed: *IFC_hBFS_180k*, *IFC_rBFS_180k* and *IFC_MND_180k*.

4 Application and Results

This section presents a concise summary of the NN architecture outcomes, emphasizing the dataset creation process to illustrate network training on building components. The results demonstrate the network's ability to learn meaningful spatial and relational patterns. Although the selected graph ML configurations and hyperparameters were not exhaustively optimized, they constitute a viable framework for testing GNN performance in a three-dimensional architectural ML task.

The cluster analysis indicated a skewed distribution, reflecting the limited diversity of the initial building sample. The *IFC_hBFS_180k* dataset (Figure 5b) exhibits pronounced differences compared to the *IFC_rBFS_180k* (Figure 5c) and *IFC_MND_180k* (Figure 5d) variants, likely due to the random sampling inherent in the *rBFS* and *MND* methods. Consequently, NN training on the three datasets resulted in differentiated performance metrics.

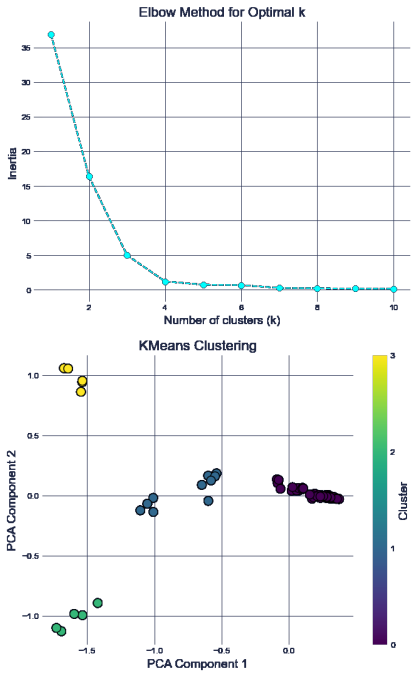
The test training results include a multi-label classification network, which predicts new node labels, and a regression network, which predicts new node features.

4.1 Multi-label Classification Network

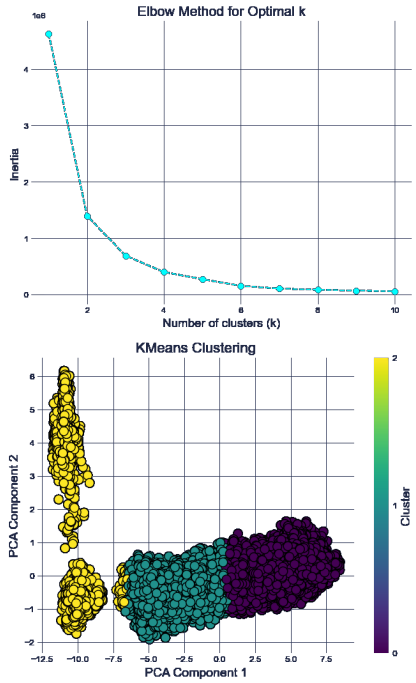
After 20 epochs (approximately 80 hours) of training on the *IFC_hBFS_180k* dataset, the network achieves an overall accuracy of about 79% in predicting node types, with class precisions for building storey, space, slab, wall, column, and beam at 91.2%, 77.3%, 68.9%, 69.0%, 80.6%, and 87.0%, respectively. The multi-label classification network also shows a macro-averaged precision of 79.0% and recall of 79.29%, along with a micro-averaged precision and recall both at 79.26%.

4.2 Regression Network

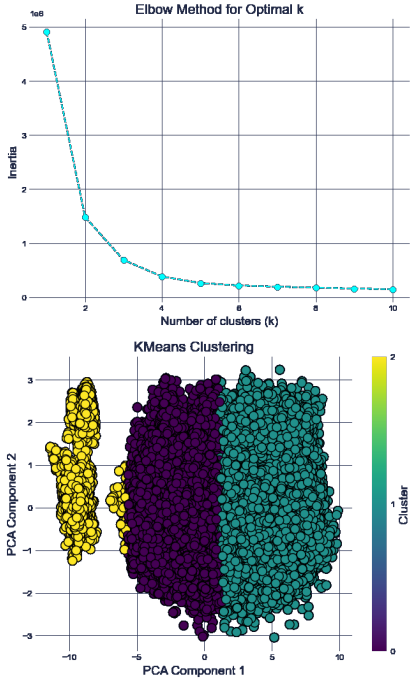
The regression network's performance was evaluated under different hyperparameter settings by comparing the vertices of the reconstructed bounding box



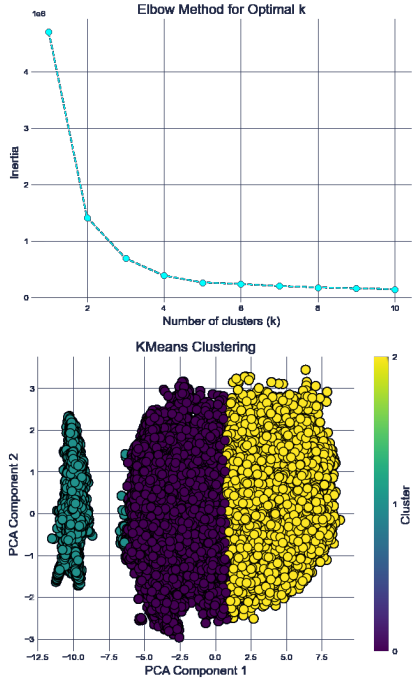
(a) Augmented building dataset.



(b) Hierarchical Breadth First Search.



(c) Random Breadth First Search.



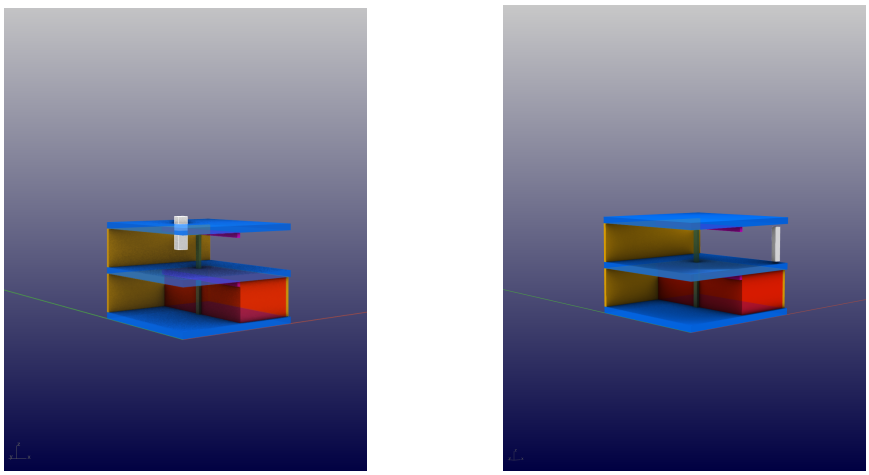
(d) Minimal Node Degree.

Fig. 5: PCA visualization of K-means clustering on the augmented building dataset and the *IFC_180k* dataset for every deconstruction type employed.

with the corresponding ground truth vertices. When provided with known coordinates, the models achieved an average convergence distance of approximately 15 meters, compared to approximately 22 meters when coordinates were unknown. This improvement is expected, since concatenating known coordinates with the node type restricts the prediction to angle and dimensions, reducing the output size to four. Although a 22 meter error is conventionally considered poor, visual inspection shows that the network nevertheless captured the correct spatial relationships between components; for instance, columns were frequently predicted with proportions consistent with their context and correctly positioned between or close to slabs.

4.3 Architectural Implementation

A visual inspection was conducted using an *Hops* script that links external *Python* scripts to *Rhinoceros-3D/Grasshopper* to evaluate the practical utility and interpretability of the co-pilot. This setup allows intuitive loading of trained models and automatic, context-based prediction of architectural components. User-defined 3D components are iteratively transformed into graph representations, which are then processed by NNs to predict node type and features. The predicted outputs are converted back to 3D geometry and displayed within the *Rhinoceros-3D/Grasshopper* interface. Users can refine results by accepting, rejecting, or manually adjusting each component to align with their design intent. Each interaction dynamically updates the underlying graph representation and triggers new node prediction iteration. As shown in Figure 6, the visual evaluation revealed both effective and flawed predictions, underscoring the model's potential and areas for improvement in spatial reasoning.



(a) Poor prediction. (b) Good prediction.

Fig. 6: Visual evaluation of predicted building elements.

5 Conclusion and Discussion

This study introduces a methodology for augmenting and preprocessing IFC data, featuring graph splitting and deconstruction, to create graph-based datasets designed for architectural ML. The proposed approach demonstrates how IFC data can be transformed into a graph-based representation suitable for training generative models. Such a transformation enables the development of tools, similar to an architectural design co-pilot that is capable of suggesting building components based on spatial and functional context.

Working with real-world IFC data revealed challenges, particularly due to inconsistencies from non-standardized modeling practices that complicate preprocessing. Simplifications such as bounding box approximation and tensor engineering were found to potentially lose spatial and relational details. Future research could focus on integrating more features or exploring alternative techniques for simplifying geometric representations to more effectively preserve critical information. Potential solutions, such as vector similarity search or hard-coded dictionaries, could solve the variability in custom properties and naming conventions found in real IFC data. Future work will explore more nuanced filtering strategies, such as selective retention of IFC property sets to preserve beneficial design diversity while excluding only files that truly lack essential elements or exhibit technical errors.

The large, dense nature of heterogeneous graphs, while expressive, posed computational challenges. Addressing these through studies on architectural graph ontologies and their impact on GNN performance could be beneficial. Expanding the ontology to include material properties, spatial programs, and qualitative attributes improves applications such as *Mechanical, Electrical, and Plumbing* (**MEP**) systems.

Overall, this work is an initial step toward using IFC data for generative ML in architecture, significant room for improvement and further exploration. Although preliminary results are promising, additional evaluation is necessary to fully assess the effectiveness of the methodology and its practical relevance. Embedding the geometry-to-graph transformation and AI prediction pipeline directly within BIM platforms via native plugins and platform APIs would enable seamless IFC ingestion and real-time, context-informed design feedback.

Acknowledgments. This research was partially funded by the German Federal Institute for Research on Building, Urban Affairs, and Spatial Development on behalf of the Federal Ministry of Housing, Urban Development and Building with funds from the Zukunft Bau research funding program (Grant Nos. 10.08.18.7-22.45), partially supported by the European Union's Horizon Europe research and innovation programme under the call HORIZON-CL4-2024-DIGITAL-EMERGING-01, project AMALTEA, Grant Agreement No. 101189678, and by the Deutsche Forschungsgemeinschaft (DFG; German Research Foundation) under Germany's Excellence Strategy - EXC 2120/1 - 390831618.

References

1. Ansel, J., Yang, et al.: Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. vol. 2, pp. 929–947 (2024)
2. Austern, G., Bloch, T., et al.: Incorporating context into bim-derived data—leveraging graph neural networks for building element classification. Buildings **14**(2) (2024), https://www.mdpi.com/2075-5309/14/2/527
3. Bauscher, E., Dai, A., et al.: Learning and generating spacial concepts of modernist architecture via graph machine learning. CAADRIA (2024)
4. Büchel, J., Engels, B.: Digitalisierung der wirtschaft in deutschland. https://www.bmwk.de/Redaktion/DE/Publikationen/Digitalisierungsindex/publikation-download-Langfassung-digitalisierungsindex-2021.html, (accessed: 2024-05-16)
5. del Campo, M.: Deep house-datasets, estrangement, and the problem of the new. Architectural Intelligence **1**, 12 (2022)
6. Chaillou, S.: Archigan: Artificial intelligence x architecture. In: Architectural intelligence: Selected papers from the 1st international conference on computational design and robotic fabrication. pp. 117–127. Springer (2020)
7. Du, C., Esser, S., et al.: Text2bim: Generating building models using a large language model-based multi-agent framework (2024), https://arxiv.org/abs/2408.08054
8. Elshani, D., Hernandez, D., et al.: Building information validation and reasoning using semantic web technologies. In: International Conference on Computer-Aided Architectural Design Futures. pp. 470–484. Springer (2023)
9. Emunds, C., Pauen, N., et al.: Ifcnet: A benchmark dataset for ifc entity classification (2021), https://arxiv.org/abs/2106.09712
10. Gavrilov, E., Schneider, S., et al.: Computer-aided approach to public buildings floor plan generation. magnetizing floor plan generator. Procedia Manufacturing **44**, 132–139 (2020)
11. Hagberg, A., Swart, P., et al.: Exploring network structure, dynamics, and function using networkx. Tech. rep., Los Alamos National Laboratory (2008)
12. Heidari, N., Iosifidis, A.: Geometric deep learning for computer-aided design: A survey (2024), https://arxiv.org/abs/2402.17695
13. Hillier, B.: Space is the machine: a configurational theory of architecture. Space Syntax (2007)
14. IfcOpenShell: Ifcopenshell: The open source ifc toolkit and geometry engine. https://ifcopenshell.org, (accessed: 2025-01-08)
15. Imdat, A., Siddharth, P., et al.: Artificial intelligence in architecture: Generating conceptual design via deep learning. International Journal of Architectural Computing **16**, 306–327 (2018)
16. Jabi, W., Alymani, A.: Graph machine learning using 3d topological models. SimAUD (2020)
17. Kai-Hung, C., Chin-Yi, C.: Learning to simulate and design for structural engineering. In: International Conference on Machine Learning. pp. 1426–1436. PMLR (2020)
18. Kai-Hung, C., Chin-Yi, C., et al.: Building-gan: Graph-conditioned architectural volumetric design generation. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 11956–11965 (2021)

19. Lehmann, F., Buschek, D.: Examining autocompletion as a basic concept for interaction with generative ai. i-com **19**, 251–264 (2021)
20. Lewis, P., Perez, E., et al.: Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in Neural Information Processing Systems **33**, 9459–9474 (2020)
21. Liang, J., Zhong, X., et al.: Bridging bim and ai: A graph-bim encoding approach for detailed 3d layout generation using variational graph autoencoder. In: International Conference on Computer-Aided Architectural Design Research in Asia. pp. 221–230. CAADRIA (2024)
22. Manzoor, B., Othman, I., et al.: Digital technologies in the architecture, engineering and construction (aec) industry—a bibliometric—qualitative literature review of research activities. International Journal of Environmental Research and Public Health **18**(11) (2021), https://www.mdpi.com/1660-4601/18/11/6135
23. Mercado, R., Rastemo, T., et al.: Graph networks for molecular design. Machine Learning: Science and Technology **2** (2021). https://doi.org/10.1088/2632-2153/abcf91
24. Mildenhall, B., Srinivasan, P., et al.: Nerf: Representing scenes as neural radiance fields for view synthesis. Communications of the ACM **65**, 99–106 (2021)
25. Nelson, N., Sepidehsadat, H., et al.: House-gan++: Generative adversarial layout refinement network towards intelligent computational agent for professional architects. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 13632–13641 (2021)
26. Pedregosa, F., Varoquaux, G., et al.: Scikit-learn: Machine learning in python (2018), https://arxiv.org/abs/1201.0490
27. Rasmussen, M.H., Lefrançois, M., et al.: Bot: the building topology ontology of the w3c linked building data group. Semantic Web (2020). https://doi.org/10.3233/SW-200385
28. Scarselli, F., Gori, M., et al.: The graph neural network model. IEEE transactions on neural networks **20**, 61–80 (2008)
29. Schlichtkrull, M., Kipf, T., et al.: Modeling relational data with graph convolutional networks. In: The semantic web: 15th international conference. pp. 593–607. Springer (2018)
30. Selvaraju, P., Nabail, M., et al.: Buildingnet: Learning to label 3d buildings (2021), https://arxiv.org/abs/2110.04955
31. W., Z., Sacks, R., et al.: Exploring graph neural networks for semantic enrichment: Room type classification. Automation in Construction **134**, 104039 (2022). https://doi.org/https://doi.org/10.1016/j.autcon.2021.104039, https://www.sciencedirect.com/science/article/pii/S0926580521004908
32. Wang, M., Zheng, Da, e.a.: Deep graph library: A graph-centric, highly-performant package for graph neural networks. arXiv preprint arXiv:1909.01315 (2019)
33. Zhang, H., Huang, Y.: Machine learning aided 2d-3d architectural form finding at high resolution. In: Proceedings of the 2020 DigitalFUTURES: The 2nd International Conference on Computational Design and Robotic Fabrication. pp. 159–168. Springer (2021)
34. Zhong, X., Koh, I., et al.: Building-gnn: Exploring a co-design framework for generating controllable 3d building prototypes by graph and recurrent neural networks. In: International Conference on Education and Research in Computer Aided Architectural Design in Europe. pp. 431–440. eCAADe (2023)
35. Zhou, Q.Y., Park, J., et al.: Open3d: A modern library for 3d data processing. arXiv preprint arXiv:1801.09847 (2018)