Graph-based Learning for Automated Code Checking – Exploring the application of Graph Neural Networks for design review[1]

Tanya Bloch[1] , André Borrmann[2] ,Pieter Pauwels[3]

[1]Faculty of Civil and Environmental Engineering, Technion Israel Institute of Technology, Israel *bloch@technion.ac.il*

[2] Chair of Computational Modeling and Simulation, Technical University of Munich, Munich, Germany *andre.borrmann@tum.de*

[3] Faculty of Built Environment, Eindhoven University of Technology, the Netherlands *p.pauwels@tue.nl*

Corresponding author – Tanya Bloch *bloch@technion.ac.il*

ABSTRACT

Although automated code checking (ACC) has been a subject of interest for many years, we have not yet seen significant breakthroughs in the field that may lead to the development of generic, comprehensive tools for ACC. Hard-coded rules are the backbone of all emerging platforms for ACC. These rules require a significant amount of engineering, which often requires manual labor; and the resulting rule sets are strict and difficult to scale to other building models. On the other hand, approaches relying purely on classic machine learning (e.g. SVM) are too coarse and unable to accurately express building information. In our hope to come up with a more scalable solution, we investigate here a novel workflow that relies on graph-based learning algorithms instead of processing rule sets. We illustrate the suggested workflow by checking accessibility requirements in residential houses, which we believe is one of the more promising rule sets that can be checked using graph-based learning methods. The high accuracy of the obtained results is encouraging to continue exploring Graph Neural Networks (GNN) for this type of ACC, yet rule-based and classic ML-based approaches show other advantages as well (rigor and speed, respectively). The main contribution of this work is therefore its identification of meaningful limitations and directions for future research, including alternative graph structures and GNN architectures.

[1] Published as / Please cite as: Bloch, T., Borrmann, A., & Pauwels, P. (2023). Graph-based learning for automated code checking – Exploring the application of graph neural networks for design review. Advanced Engineering Informatics, 58, 102137. https://doi.org/10.1016/j.aei.2023.102137

Keywords: Automated Code Compliance, Graph Neural Network (GNN), Machine Learning (ML), Rule-based checking, Building Information Modeling (BIM)

1. INTRODUCTION

1.1. Challenges in Code Compliance Checking

Any new building is designed to fulfill user requirements in a way that ensures the functionality of the building, public safety and welfare of the occupants. A variety of requirements and constraints are important to consider during the design phase, and designers and engineers are now concerned not only with safety requirements but also with constraints regarding accessibility, durability, sustainability, security, energetic efficiency and much more (Meijer et al. 2002). The result is a large number of laws, codes and regulations that any new design must comply with before construction can begin. Checking whether a proposed design conforms to all relevant regulatory requirements is a difficult task that demands a lot of knowledge and expertise and it is therefore performed by qualified and experienced engineers or architects. Although automating the design review process has been a subject of research for several decades (Amor and Dimyadi 2021), a fully automated checking platform that covers a wide range of regulations remains a distant goal.

The most important breakthrough in the field of Automated Code Checking (ACC) has been the development of Building Information Modeling (BIM). As the checking process is concerned with comparing a proposed design to the relevant regulations, BIM provides one part of the equation (the design) in a computer readable format. The second part of this equation, computer readable representation of the regulations, remains a bottleneck in the further development of ACC (Nawari 2012b). Codes and regulations are text documents written in natural language for the understanding, interpretation and use of human experts. Enabling an automated assessment of the design based on the regulations involves the conversion of hundreds of such text documents to a computable form. Furthermore, many of these documents have subjective statements, statements that are open to interpretation, and relatively complex geometric computations (e.g. “the hallway shall be wide enough for 3 persons to pass”).

A lot of research has focused on processing regulatory documents in terms of classification of rules, representation of rules and organization of rules. However, the rule interpretation process remains mostly manual as it requires contextual knowledge and comprehensive understanding of the regulations (Zhang and El-Gohary 2017; Zhang et al. 2022). Although the engineering domain is governed by well-defined principles and accurate calculations, many of the building regulations are ambiguous, subjective or vague, and therefore not suitable to be computerized (Nawari 2019) . When considering the creation of computer-readable rules specifically, a few approaches are available. Many ACC approaches rely on manually processing the natural language used in regulations, namely reading text and writing rules manually. This can also be augmented by using automatic Natural Language Processing (NLP) tools (Zhang and El-Gohary 2016, 2017). Machine Learning (ML) based approaches on the other hand rely on input models, and the eventually obtained tags or classifications (compliant; non-compliant), and hence do not require elaborate processing of human-readable text.

The hard-coded rule sets, which are the core of all existing efforts for ACC, are often too strict to facilitate the flexibility required to capture building regulations. As a result, the existing platforms for ACC are limited in the scope of regulations that can be checked. As stated by (Nawari 2012a) "*The computable model for code representation must possess enough elasticity and expressiveness to capture most of the provisions…*". This suggests a necessary shift to a more flexible form of Artificial Intelligence (AI) than the symbolic AI which is usually practiced in the existing applications for ACC. Indeed, classic ML approaches have been considered, aiming to train a model (e.g. neural network model) that can immediately evaluate whether a building model is compliant to certain regulations or not. The performance of these ML-based checkers is highly reliant on the quality of data they have been trained with. Full accuracy is not available in this case, and the level of reliability then becomes uncertain, which is often highly needed in the case of regulation compliance checking. Furthermore, the black-box nature of these classic ML approaches takes away the proof and explainability behind a certain regulation check. These models can however provide indicative results for clauses that cannot be automated using rules.

For some of the prescriptive regulatory requirements, both the regulations and the design can be presented in a computer-readable form (rule-based ACC), usually involving some extent of manual work. Still, the comparison between the two is not straightforward. Matching of building concepts' representations (e.g. width, accessible route) in regulatory documents to those represented in the BIM model (e.g. IfcSpace) remains a challenging task. This typically requires difficult alignment and mapping processes between two different ontologies or vocabularies, which is nearly always incomplete because their semantics simply do not overlap sufficiently well (*Figure 1*).

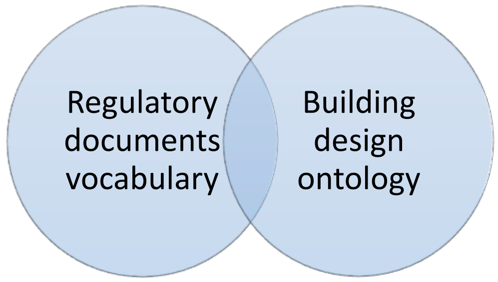


Figure 1 The semantics within the regulatory documents and the BIM model do not sufficiently overlap, hence a difficult and incomplete mapping process is required to match both to a satisfactory degree.

Moreover, because of this significant semantic mismatch of domains and models, a considerable mapping needs to occur, which involves plenty of interpretation of meanings and intentions. Such a mapping nearly always requires a human coder to make this interpretation and mapping step. As a result, the rule-based ACC process is a semi-automatic workflow at best, not only because of the building code that needs to be transposed into machine-readable rules, but also because the mapping of the building model with the hard-coded rules needs to be manually created.

1.2. Graph-based learning as a semi-flexible solution

Based on all the above, this research aims to look at alternative approaches for ACC that have the needed flexibility, yet also achieve sufficiently fine-grained and reliable results. In this search, it is necessary to adopt a holistic perspective on ACC which is concerned with the entire checking process instead of separately dealing with the regulations and the design model. In this perspective, we aim to capture and leverage expert knowledge and past experience, instead of interoperating regulations and forcing them to rigidly defined constructs (flexibility as a requirement). Specifically, we aim to implement a Machine Leaning routine for the entire checking process, in a way that allows us to represent the codes and regulations implicitly through the data set used for training, while also still having a means to trace proofs and explain ACC outcomes (explainability as a requirement). Training an ML model to classify design models as “compliant” or “not compliant” to a specific code provision, has the potential to overcome the barriers described above and thus lead to a wider range of regulations that can be checked automatically.

Previous work that implemented a similar perspective to code checking (Bloch et al. 2019) illustrates an ML-based checking process where indeed an in-depth code analysis becomes more superfluous. However, their work also demonstrates the limited ability of ML to deal with the complex relationships between building elements. Code requirements that involve restrictions on the geometry of building elements as well as restrictions on the possible topological relationships between them are difficult to represent for the “classic” ML models (SVM, decision trees, neural networks, etc.). With the development of Graph Neural Networks (GNN), which are ML models that operate directly on graphs, we hypothesize that this data representation limitation can be overcome as well: topological relationships can be represented in graphs, and flexible as well as explainable training procedures are still available. As GNNs are a relatively new development that has rarely been used in the AEC domain, and since there are only few studies on implementing ML to the entire ACC process, an initial investigation of the feasibility and practicality of the suggested approach is needed in order to establish the application of GNNs to ACC as a viable research direction. Furthermore, its applicability in relation to rule-based ACC as well as classic ML-based ACC needs to be clarified.

Therefore, this exploratory work aims to investigate the applicability of GNNs to code checking and its potential to alleviate the need for explicitly compiling rule sets. Since graphs have the expressive power to deal with the complex topologies represented in building design and regulations, we see a potential to overcome the data representation limitation previously identified during the application of "classic" ML to code checking. Nevertheless, we do not suggest to replace the existing achievements in rule-based code checking, nor any of the NLP procedures to process regulations into computable rules.

There are two underlying hypotheses behind this work. First, we hypothesize that graph-based learning is applicable to problems from the ACC domain, and that it is particularly useful for dealing with regulations that address not only geometric aspects of the design but relational aspects as well. The second hypothesis is that a GNN model trained on a completely synthetic data set can produce a well performing classifier that can provide accurate checking results for real design. This work is designed to illustrate an initial proof of concept for the proposed workflow and thus serve as the basis for further development of GNN-based ACC. In addition, we expect to begin investigating the differences between the various approaches to ACC.

The rest of the paper is structured as follows: Section 2 provides background on current research and state of the art in ACC, and presents the logic behind applying graph-based learning to BIM models. The aims of this work and research methodology are described in Section 3. Section 4 provides a detailed description of the proposed GNN-based workflow. This includes more details about the way in which the synthetic data is generated, what its quality is, and how this training data is labelled. Section 5 illustrates the suggested workflow on a specific problem. Discussion and conclusions are provided in Sections 6 and 7 respectively.

2. BACKGROUND

A visionary paper written by Eastman (1975) illustrates a future Building Description System (BDS) and its applications, one of them being automated code compliance checking. As a comprehensive generic system for Automated Code Checking (ACC) that covers the full range of regulations in the AEC industry has not been developed yet, this remains a relevant and active research area today (Amor and Dimyadi 2021). In this research area, the rule-based approach is most commonly investigated and used to build the backbone of ACC platforms. However, this approach has a number of inherent limitations. Namely, a lot of manual engineering work is needed that consists of (1) interpreting building codes, (2) writing machine-readable rules, (3) defining semantic mappings between rules and building models. This situation is explained in more detail in Section 2.1, including relevant literature references. Alternatively, supervised ML-based methods for ACC can be deployed, and those approaches have been previously demonstrated. Earlier research on such ML-based methods is documented in Section 2.2, including few examples of implemented procedures. The logic behind representing a building model as a graph is explained in section 2.3 which also explains what kinds of graphs can be made available to graph-based learning techniques, including Resource Description Framework graphs (RDF), graph data model graphs (GDM), straightforward topology graphs, and labeled property graphs (LPG). Section 2.4 finally illustrates how this graph-based learning procedure works.

2.1. Automated Code Compliance Checking

Amongst the earliest efforts for ACC is the development of decision tables (Fenves 1966) and mechanisms for representing design constraints so that they can later be used to determine whether those constraints are satisfied by a given design or not (Fenves and Rasdorf 1982). In 1997, Han et al. (1997) describe the use of automated design checking tools, emphasizing that the use of such tools in the industry is feasible only after the development of a standard model that provides more information than a collection of drawings. This 'standard model' for building data has by now been achieved, to a reasonable extent, in the form of Building Information Models (BIM) and its associated data serialization standards. BIM, together with the introduction of the Industry Foundation Classes data model (IFC) for information sharing, holds the potential to provide the required information thus enabling the automation of compliance checking (Dimyadi and Amor 2013). The work of Pauwels et al. (2017) identifies three critical components in a rule-based checking system: a schema, a set of instances, and a set of rules. This is a continuation of the earlier work in Pauwels et al. (2011), where the semantic mapping between building model and regulation was suggested to happen using dedicated conversion rule sets. This is of course only possible with stable, standard and complete enough schemas, both for the building model and regulation.

With IFC as an industry-wide standard, such stable schema is seemingly available (right side in *Figure 1*). However, even implementing open BIM standards that provide better building data sharing and collaboration opportunities (Amor and Dimyadi 2021) is not sufficient for ACC, as data quality and completeness need to be high enough. ACC processes require high quality and complete information stored in the BIM models, which is usually not achieved. A BIM model created in the design phase of a project may contain inaccurate or false information provided by the user, or may lack the information required for ACC (Borrmann et al. 2018; Preidel and Borrmann 2015). Therefore, a model that is not pre-processed before the checking can often lead to inaccurate or false checking results. Such insufficient data quality currently prevents ACC from reaching its full potential, as it leads to plenty of manual pre-processing steps and therefore insufficient scalability.

Systems designed for ACC generally include four stages: interpretation of rules, pre-processing of BIM models, rule execution and reporting (Eastman et al. 2009). Reviews of previous work in the field (Amor and Dimyadi 2021; Dimyadi and Amor 2013; Eastman et al. 2009) indicate that the first two stages (generating proper rule sets and obtaining the required representation of BIM elements) remain major challenges of the process, which aligns with what is indicated above. Translating the massive amount of written codes and regulatory documents into logical statements is considered to be one of the main barriers to comprehensive automation of ACC systems, in particular because human-written regulations are often ambiguous and require contextual knowledge and interpretation (Zhang and El-Gohary 2017). The use of Natural Language Processing (NLP) does not solve that problem. Even manually processing the human-written regulations into machine-readable code seldom deals with such ambiguities, except for simply making an approximation of the original building code text with a lot of implicit assumptions. In this paper, we suggest a radically different and holistic way of addressing ACC by applying ML methods that leverage experts' knowledge and previous experience instead of compiling regulations as rules.

2.2. ML-based approaches for ACC

Previous research suggests that using a Machine Learning (ML) approach instead of relying on hard -coded rules for code checking might lead to a greater degree of automation in the process. In addition to eliminating the need to compile rule sets, it is assumed that ML models for code checking can be implemented even if some information remains in an implicit form(Sacks et al. 2019). An experiment described in (Bloch et al. 2018) is focused on code provisions that restrict the geometrical features of security rooms like minimal wall thicknesses, window size and location etc. This experiment illustrates a checking routine that does not require rule compilation. The data set for the experiment was synthesized through a random number generator that populates 11 parameters with values within reasonable ranges. The result of this process is 10,000 models of security rooms that were evaluated based on the relevant code and labelled 'pass' or 'fail' accordingly. A binary classifier was trained using the created data set as input, which resulted in 99.8% precision and 100% recall on the validation set (subset of the 10,000 models). This indicates that the machine learning approach holds great promise as a solution for code-compliance checking and that there is much value in continuing to explore the capabilities of ML for code-compliance checking.

The major benefit of the ML approach is that the regulations are implicitly represented within the computer readable representation of the design. As the features selected for representing each building element in the data set are chosen based on key values from the codes, they are an implicit expression of the regulations themselves. For example, a code requirement for a minimal space area can be represented by a binary feature indicating if the space is greater or smaller from the value stated in the code. Most importantly, the labels that are assigned to each instance in the data set are also an expression of the regulations. Given a data set that consists of previously checked design, the labels actually express the experts' interpretation and understanding of the regulations. In other words, training a ML model to distinguish between design that is compliant to a specific code and design that is not compliant, is possible without compiling rule sets.

Exploring the range of required preprocessing of BIM models by semantic enrichment to enable ACC, the work of (Bloch et al. 2019) is focused on a single code requirement for security rooms to be stacked one on top of the other in such a way that at least 70% of any given security room's walls are continuously supported through the height of the building and reach the structural foundations (Home Front Command 2010). The code requirement at hand involves information about the topological relationships between various building elements, as do many regulatory requirements. This points to one of the major limitations of using ML with building data. Relationships between entities are difficult to represent for ML learning algorithms as they require rigidly structured data for learning. For example, a description of the most primitive functional building element, a space, in a fixed rigid structure is not straightforward. Spaces can be defined by any number of walls. However, in some cases, not all space boundaries are defined by physical walls at all but by some virtual boundary lines. A single space has relationships to all elements that define the space but also to other spaces next to it, above and below it. Much like in the security rooms experiment, we cannot pre-define a single data structure that is able to express information about all spaces in a model. To learn from building data, we must apply learning algorithms to a more flexible data representation.

2.3. The building as a graph

A building model describes a complex physical system composed of large amounts of instances that are related to each other by different types of topological or functional relationships. This type of geometric data is irregular and randomly distributed, making it difficult to identify patterns and fixed structures. A graph representation can support different information structures providing flexible representations of attributes for every instance. Graphs are extremely useful for describing physical systems by representing objects as nodes and relationships as edges (Zhou et al. 2018).

It has been previously demonstrated that graphs are a suitable way for describing information represented by BIM models, including representation of complex geometries and relationships (Gan 2022; Ismail et al. 2017, 2018; Pauwels and Terkaj 2016; Skandhakumar et al. 2016; Zhi et al. 2003). Information stored in a BIM model can be captured in various types of graphs, thus it is important to obtain the representation most suitable for the problem at hand. In fact, the IFC data model, with its plethora of interconnections and inverse relationships in the EXPRESS information modelling language can easily be expressed as or understood as a graph. The RDF graph (Pauwels and Terkaj 2016) is a more common example of a graph structure able to capture BIM data. While the RDF graph data model enables representation of any data in a web-based graph, ACC does need standardization and stability in those data models. Therefore, several OWL ontologies were developed for describing the concepts of a building, such as the Building Topology Ontology (BOT) (Rasmussen et al. 2021), BRICK (Balaji et al. 2016), Real Estate Core (Erikoskarwallin et al. 2019), etc. For rule-based ACC to work and potentially scale, it is critical that these vocabularies are stable and reliable.

Another example of a graph representation is the graph data model (GDM) developed in (Khalili and Chua 2015), which is a semantically enhanced, 3D topological data model, that represents the topological relationships among 3D objects in buildings. Their method exploits the IFC geometric and topological representation of building elements and transforms the relationships between the elements to the node-edge structure of the graph. The semantic information is then added as weights to nodes and edges. Many other proposals for representing BIM models as graphs exist, several of which consider the use of the simpler and more comprehensive labelled property graphs (LPGs, e.g. Neo4J – see (Donkers et al. 2021) for a comparison).

While the above examples show the merit of a semantic graph, several other graph types exist as well, the most important one being here the topology graph. In fact, graph theory is a widely used approach for indoor and outdoor navigation applications. Spaces and the connections between them are the core elements needed for path finding and can easily be translated from a BIM model to nodes and edges in a graph (de Koning et al. 2021; Skandhakumar et al. 2016). To obtain the graph representation, the IFC file is parsed (XML, JSON, SPF format) to identify all spatial elements (rooms) and all the portals or interfaces between them (like doors) so they can later be translated to nodes and edges in a graph. Then, the relevant attributes are associated with the nodes and edges based on each element's property set. In the work of Skandhakumar et al. (2016), several applications of BIM graphs were mathematically defined. One of the presented algorithms is the "path finding" algorithm. In De Koning et al. (2021), the above approach was used to generate a BOT-based topology graph that can then be queried using the A* algorithm for robot path-finding within a building.

Jin et al. (2018) exploit the fact that building spaces interact with each other according to their function. Thus, feature extraction is based on the relationship between spaces, specifically accessibility and adjacency. In this work, two separate graph representations are constructed; one is the accessibility graph which represents all the spaces one can access from every space in the model. The other is the adjacency graph that takes into account only direct neighbors of every space. In both dimensions the nodes of the graph represent the spaces and their properties are propagated through the edges. The properties assigned to the edges are space area, space circumference, space height and floor level. In addition to those simple features, some complex features are calculated that represent the number of boundaries between the spaces. Through an experiment, they explored the typical spatial functions in an office building.

A graph representation of a building model was used by (Ismail et al. 2018) as a basis for querying the model to find the escape routes from a building. An IFC file was converted to a LPG database (Neo4J) by using an automatic workflow as suggested in Ismail et al. (2017). The graph does not contain information about the geometry of the objects, but it does represent the spatial relationship of a space to other objects in the model. The graph consists of connected entities (nodes) that can hold any number of attributes, and the edges convey the type of relationship between the nodes. In a labelled property graph (LPG) dataset, the edges have a direction, meaning that there is a start node and an end node, and they can also hold any number of attributes. In this work, nodes represent spaces, and the edges convey the relations between them, in terms of accessibility and adjacency, which is similar to the above outlined examples that rely on other graph technologies.

Using this approach, it is possible(Ismail et al. 2017) to query the graph database to retrieve the emergency escape routes for a two-storey office building (Ismail et al. 2017).

2.4. Graph-based learning

Graph-based learning is one way of dealing with data that cannot appropriately be structured in a tabular or hierarchical form (Bronstein et al. 2017), such as the data in the BIM domain. This type of learning is highly specific to the structure of graphs, and gains its merit primarily by finding specific patterns in graphs and using these for computations, either statistically (e.g. ML-oriented graph-based learning) or semantically (e.g. rule and query languages). The use of the latter (semantics-based) is documented in Section 2.3, and in this section and the remainder of this paper, we will instead focus on the first type of graph-based learning: statistical graph-based learning (ML-based).

The basic goal of graph-based learning is to learn a vectorized representation of every node (node embedding) that encapsulates the attribute-based information available for nodes and edges, combined with topological information represented in the graph. In other words, this vectorized representation embeds all spatially relevant data for each node separately into multiple node vectors (right in *Figure 2*). So nodes are encoded as vectors that reflect their position in the graph and the structure of their local graph neighborhood (Hamilton et al. 2017). This approach relies on a function that maps a graph *G* to a *d*-dimensional space. Given a graph with *m* nodes, this results in a R^{mxd} matrix, where each row is the embedding of the node (*Figure 2*).

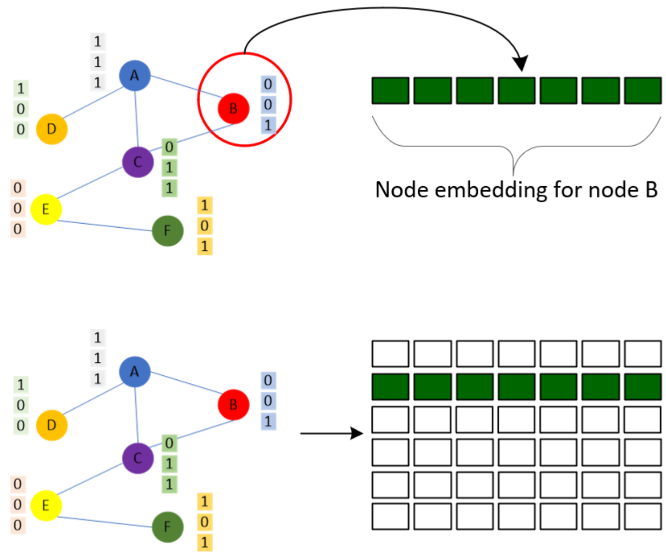


Figure 2 Mapping the input graph to d dimensional embedding space

To learn the mappings, these approaches operate directly on the graph by sampling a fixed size neighborhood for each node. The neighborhood graph consists of the node's neighbors up until k hops away from the node (denoting the number of GNN layers). The basic idea of Graph Neural Networks is that we use the computational graph for every node to propagate the information from all its neighboring nodes across all the graph layers and compute a node embedding. Propagating the information across all layers in a neighborhood graph is a process that is referred to as message passing, which yields new vectorized node representations that should preserve information about the graph's topology (Wu et al. 2020). This approach is useful for tasks such as node classification, graph classification or link prediction(Wu et al. 2020).

Different GNN architectures are defined, amongst others, by different aggregation operators for propagating messages from all neighboring nodes in a single layer. *Figure 3* illustrates an input graph and the single-layer neighborhood graph for node A. In this example, the attribute information from all three neighbors is transformed and aggregated into a single message to pass it to the target node A. The aggregator and transformation operators are parametrized and passed through a small neural network to introduce non-linearity.

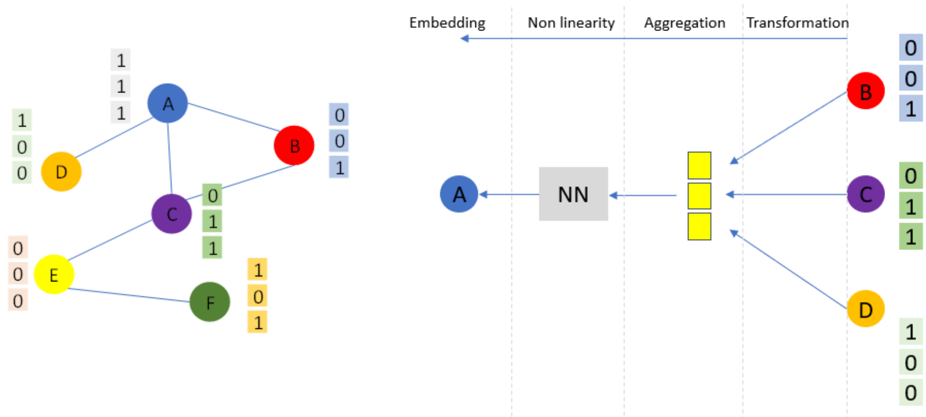


Figure 3 Representation of the local neighborhood of node A and the message passing process in that local neighborhood. Corresponds to a single layer in a Graph Neural Network

Averaging the neighbor messages is the most basic aggregation approach as illustrated in the set of equations below. Given a vector of features assigned to node A, denoted as x_A , the initial embedding of node A in the layer k=0 is simply the vector of feature assigned to that node $h_A^0 = x_A$. In every subsequent layer, the following process is performed (Eq. 1 based on (Hamilton 2020)): (1) compute the messages from the neighboring nodes by sending them through a linear transformation (W_k); (2) aggregate the messages across all the neighbors by averaging the neighbor's previous layer embeddings $\frac{h_u^{(k)}}{|N(A)|}$; (3) add the computed messages to the embedding of node A in the previous layer with a bias B_k. This is then sent through a neural network to introduce non-linearity (σ). The final embedding of node A after K layers is given by Eq. 2. The goal is to use these embeddings to learn the best weight matrices W_k and B_k which are the trainable parameters in a GNN. The fact that these parameters are shared across all nodes makes it possible to generalize to unseen nodes, thus enabling classifications of new instances.

$$1.\quad h_A^{(k+1)} = \sigma\left(W_k \sum \frac{h_u^{(k)}}{|N(A)|} + B_k h_A^{(k)}\right)$$

$$2.\quad Z^A = h_A^{(K)}$$

The aggregation operator can be any order-invariant operator like mean or sum. In Graph Convolutional Networks (GCN) for example (Kipf and Welling 2016), an element-wise mean operation is performed in the aggregation stage. As illustrated in Eq. 2, all the messages from the neighboring nodes are normalized by the degree of the target node, namely all messages are equally important. By contrast, in the Graph Attention Network (GAT), the normalization factor is learned for every neighbor separately. GAT (Veličković et al. 2017) introduces the attention mechanism which assumes that not all messages are equally important. The attention mechanism is popular for sequence-based tasks, such as learning sentence representations (Lin et al. 2017), since it identifies the most relevant parts of the inputs to make decisions. When the attention mechanism is applied for graph learning, an attention coefficient α_{ij} is computed for every pair of connected nodes. This coefficient is an indication of the importance of every node's features for the message passed to the target node *i*. The attention coefficient is used at the transformation stage of the message passing (see *Figure 3*). Considering several neighbors for a target node, the coefficient is normalized across all the neighbors.

In general, different GNN architectures have been suggested and demonstrated for various applications (Zhou et al. 2020), and the expressive power of different GNNs has been explored (Xu et al. 2018). Recently, GNNs were applied for point cloud data processing by performing node classification on induced graph structures (Collins 2020). In addition to node classification, GNNs can be applied for graph classification problems, link prediction etc. For example, graphs can be used to predict molecular properties, classify diseases, predict drug side effects, perform text or image classification etc. (Zhou et al. 2018). In the construction domain, node classification with GNN algorithms was performed for room type classification in residential buildings (Wang et al. 2022).

2.5. Summary

Based on the above, we believe that GNNs are applicable to ACC, and in particular algorithms for graph classification and node classification (e.g. GCNs, GATs) can be powerful tools for classifying elements in BIM models. Given recent developments in the field of graph data science and graph learning (Cao et al. 2020), we conclude that the representation of building information as a graph (Section 2.3) can contribute to the development of an automated process for design review. While RDF graphs may be deployed for this purpose as well, the remainder of this article will primarily consider the use of labelled property graphs (LPGs), which are more compact and more closely aligned to available ML techniques (e.g. representation of object properties using feature vectors).

3. AIMS AND METHODOLOGY

This research suggests shifting the focus from the individual challenges that hinder further developments of the ACC process to the overall approach applied for automated code checking. We propose a novel workflow for automated code checking illustrated in *Figure 4*, supported by the application of graph-based ML techniques as an alternative to the commonly used hard-coded rules. We hypothesize that graph-based learning techniques are applicable as the checking mechanisms for problems from the ACC domain. We assume that it is possible to train a GNN model by a large number of positive/negative examples such that it is capable to correctly classify an unknown building design into pass / fail results regarding the building code investigated.

3.1. Aims and Research Scope

The major difference between the suggested approach and classic ACC, is that we are not concerned with translating the regulations to a computer-readable format. In fact, the LPG represents both the design and the regulations using the same data structure (implicitly, represented by the provided labels). As a result, we no longer need to look for the overlap between the regulatory document's vocabulary and the building design ontology (as depicted in *Figure 1*), since the regulations are embedded within the graph representation of the buildings. Using a graph representation of the building information as input for a learning model presents an opportunity to leverage the benefits of applying ML for code checking (as explained in section 2.2), while overcoming the major limitation of addressing regulations that are concerned with the relationships between the building elements represented in the design. To validate our approach, we aim to show that (1) we have a sufficient amount of data available of sufficient quality, and (2) the GNN algorithms lead to sufficiently reliable results in a scalable manner.

Data availability: Representation of a BIM model as a training set for a machine learning algorithm is a difficult task. Recent developments in graph data science and the possibility to perform learning directly on graph-based data provides an opportunity to overcome the existing problems in the ACC process. In our work, we therefore indicate how such graph-based data can be made available sufficiently abundantly.

Applicability and scalability of GNN algorithms: Since GNNs have not been applied for ACC before, there is no pre-existing knowledge on this subject. Hence, we suggest revisiting the well-established pipeline for ACC and explore a GNN-based workflow, based on the recommended practice that is documented already to some extent in Section 2.4. The scope of this paper is limited to an initial feasibility check for a small-scale problem in the domain that expresses regulations that address geometric and topological aspects of the design. Through this small-scale problem, we aim to demonstrate the initial feasibility and explore the performance of a novel GNN-based procedure for ACC.

3.2. Methodology

Graph-based learning, as any supervised ML algorithm, is reliant on a large data set of examples for training. In this case, the input for the ML algorithm is a set of models represented as graphs, where every BIM object is labeled as compliant or not compliant with a specific code requirement ("pass" or "fail"). Since a large set of labeled models is not available, we propose to implement the training stage on a synthetic data set and explore its applicability to make predictions on real BIM models. The underlying hypothesis in this work is therefore that a GNN trained and validated on a synthetic data set can be used for classification of models obtained from the industry. To confirm this assumption, we must first generate a synthetic training set, test the performance of trained GNNs for compliance checking, and validate the results using design documents obtained from the industry.

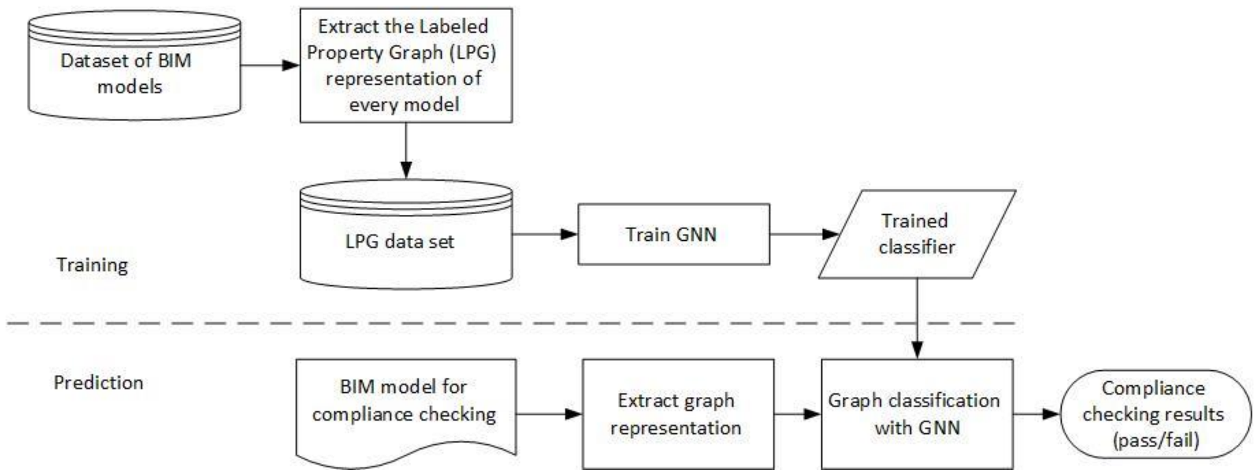


Figure 4 Suggested GNN-based workflow for ACC

To achieve the set goal, we follow the 4-step procedure illustrated in *Figure 5*. We begin by selecting a code provision to define a test case for application of GNN. There are two main criteria to selecting the test case for demonstrating the applicability of GNNs to ACC:

a) The chosen test case should involve both geometric aspects as well as topological aspects represented in the code requirements. We assume that GNNs will not be beneficial for checking simple prescriptive clauses that involve only geometric restrictions, and that the strength of GNNs is in checking clauses that combine topological and geometric requirements.
b) To ensure that we are able to generate and label a large data set, we aim to find a test case for which the data set can be automatically labeled using other procedures.

We therefore choose a test case based on the requirements in the American National Standard for Accessible and Usable Buildings and Facilities (International Code Council and American National Standards Institute 2010). We define a small problem from the domain of accessibility requirements that is the basis for generation and labeling of a data set for training using an initial graph structure. Since we aim to represent BIM models as graphs, we do not generate the 3D geometry, but instead directly a data set of graphs. A detailed description of the test case, data generation and labeling process is provided in the next section of this paper. In short, data is generated (step 1 in *Figure 5*) by creating LPG graphs from scratch, instead of relying on input BIM models and extracting the graph representation from them. Labelling of these graphs (Step 2 of *Figure 5*) is performed using the procedure explained in Section 4.3., namely the creation of feature vectors for the individual nodes of the graph combined with path finding algorithms. It is important to note that the chosen test case is not designed to examine the performance of GNNs in comparison with the existing methods. This would be meaningful only after proving that GNNs are even a feasible solution to problems from the code checking domain, which is the main goal of this work. Thus, we purposely choose a test case where an automated solution for checking is available so that it can be used for labeling of a large synthetic data set. A demonstration of the suggested workflow for this simple problem with satisfactory results will be the foundation for further implementation of the same workflow for regulations that cannot be checked by other means, which will require manual checking and labeling of a data set by experts.

Once the data is labeled, we develop a GNN model architecture using the StellarGraph machine learning library for graphs and networks (Data61 2018), which is trained, validated and tested using the synthetic data set generated in the previous stage (Step 3 in our 4-step procedure). The accuracy of classification results on the test set (which at this point is a portion of the generated data set) are an indication of the overall initial feasibility of applying GNN to ACC. The fourth and last stage of the workflow as illustrated in the lower part of *Figure 5* consists of checking the ability of a GNN trained on the synthetic data set to classify real BIM models. To do so, we present the classification results of three building designs obtained from local Israeli architecture firms. A more detailed performance evaluation of the obtained classifier on “real-world” data is the subject of ongoing work and will be reported at a later time. As explained above, this paper is focused on a demonstration and feasibility proof of the proposed ACC workflow. Hence, revision of the initial graph structure and the chosen GNN algorithm and architecture is outside the scope of this work.

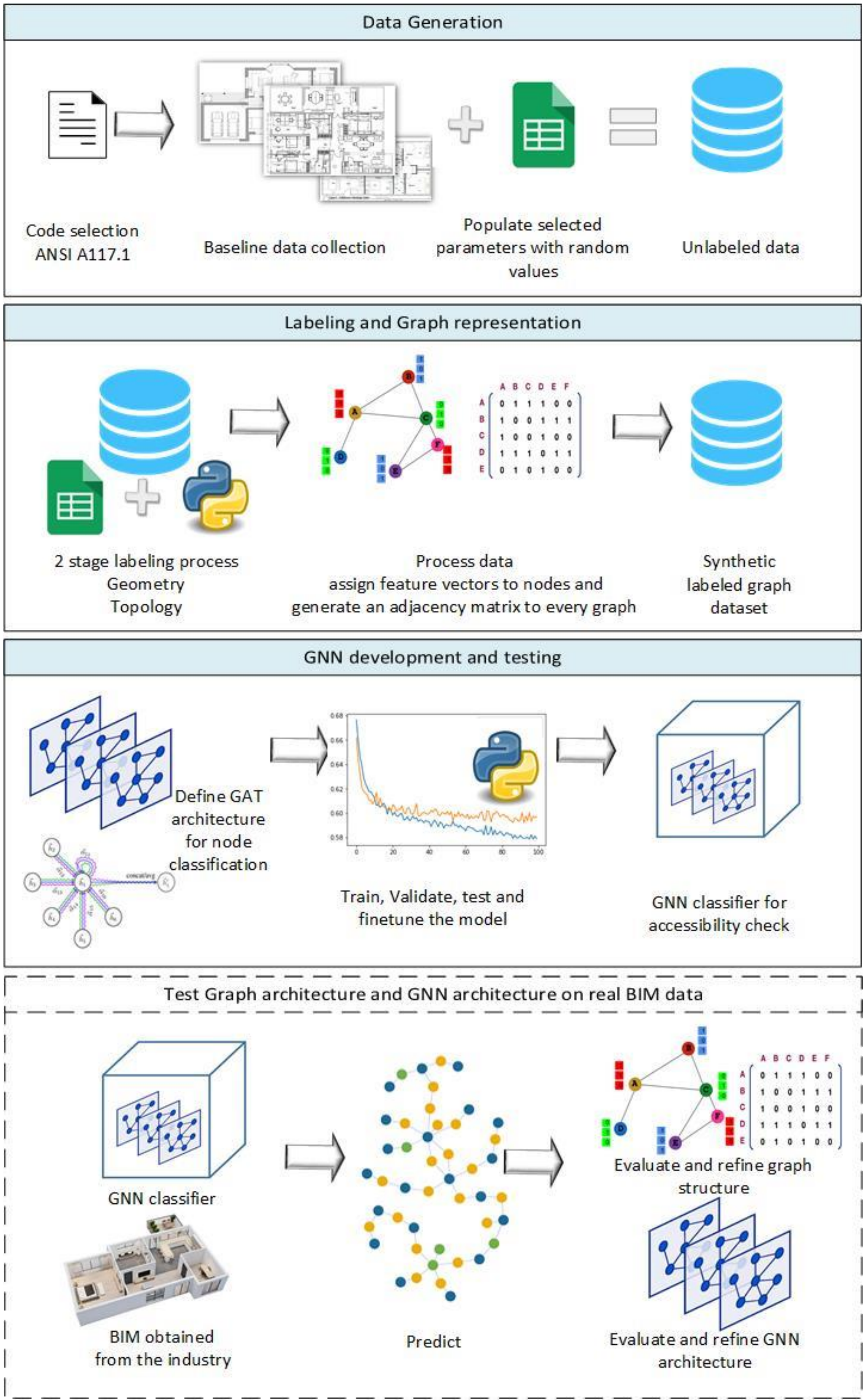


Figure 5 Research method

4. DATA GENERATION AND LABELING

As a first case study, this paper focuses on the geometric requirements for accessible spaces as defined in the American National Standard for Accessible and Usable Buildings and Facilities (International Code Council and American National Standards Institute 2010). As stated in the previous section, we aim to explore the applicability of GNNs to problems from the ACC domain. We do that by defining a small-scale problem of accessibility requirements check in single-family houses. In the sense of a feasibility study, we hypothesize that if the small-scale problem is adequately handled, we see the fundamental chance to successfully apply the GNN-approach also to large-scale problems. We demonstrate the ability to train a GNN model for code checking based on a synthetically generated data set. This section provides a detailed description of the process for generating a synthetic, labeled, training set that consists of graph representations of single-family houses.

4.1. Challenges in generating synthetic data

Obtaining a large enough set of building models of a specific type (in this case residential houses), is a difficult task. In addition to the fact that not all design and construction companies adopt BIM technology, those who do are rarely willing to make their models public. As GNNs are applied directly to graphs, collecting simple drawings will not be sufficient as they would have to be manually translated to their corresponding graph representation. A BIM model on the other hand, is a structured database where every represented building element is assigned with a set of attributes. These can be automatically extracted from BIM software, for example by using computational design tools such as Dynamo ("Dynamo" 2022)or plain C# (e.g. Revit plug-in), and arranged in the form of a graph.

Since a suitable set of models is not available to the authors, we suggest an approach for creating directly the graph representation of buildings. The line of thought is similar to that of generative design, except we define parameters that allow us to generate graphs and not 3D geometry. This will also allow us to generate a much bigger dataset, which is needed for a reliable GNN method. That is, we generate a list of elements and assign each element with a list of attributes. Each entity in the list represents a building element. Labeled property graphs consist of a set of nodes and edges, the labels usually represent the node types. In this case, the list of elements translates to graph nodes. For the edges, we generate a list of connections that represents the topological relationships between the elements. In this case, since we are concerned with accessibility requirements, the only type of relationship represented by the edges is the ability to access one element from another. The labels would be the result of compliance checking, in the most general case that is a "pass" or "fail" for every node. A detailed description of the procedure to label the generated graphs is presented in Section 4.3 of this paper.

The main challenge of the presented approach is that the resulting graphs cannot be arbitrary and must represent feasible buildings since the end goal is to be able to check the compliance of real buildings to specific code requirements. Hence, in order to maintain topological integrity, we randomly modify the geometry of real, publicly available, floor plans while keeping the topology of every floor plan unchanged. The modifications are also restricted to a certain range to maintain feasible geometry of different building elements and avoid contradictions, as explained further in this section. The procedure of generating a single variation of one basic floor plan is illustrated in *Figure 6*.

Theoretically all building elements can be represented as nodes in a property graph and the relationship between the elements can be represented as edges (Ismail et al. 2018). However, this would result in a complex graph with a large number of nodes and edges. Since every building element has various types of relationships to multiple other building elements, the nodes in the graph would have a high degree, making the neighborhood graph of every node large and complex and the GNN computationally expensive. Also, since we limit this work to a small-scale problem, many of the entities are irrelevant as they do not carry relevant information for the code provision we focus on. Hence, we focus here on the elements and the corresponding graphs that are relevant to the learning problem at hand. In the case of a building accessibility check, these are mainly spaces, doors or doorways, stairs and ramps. As described in Table 1, each element in the list is defined by two parameters that reflect some geometric restrictions (restrictions on the sizing of the elements) specified in the code.

For this initial feasibility test, we focus on simple requirements such as the minimum width of doors (as defined in section 404 of the code), corridors, ramps and ramps slope (as defined in section 405 of the code). Hence, for spaces we consider the minimal width of the space and we differentiate between two main space types, one that requires an available turning space (such as bedrooms and bathrooms and other functional areas) and another that defines the circulation path and requires a minimal width (such as a corridor). For doors we also consider the door width, and differentiate between door types like hinged doors, sliding doors and doorways. For stairs we consider the width and the number of stairs to get a complete representation of the building. Since only the existence of stairs influences accessibility to the adjacent spaces, these geometric parameters are meaningless for checking accessibility in the given design. They were used merely as “place holders” in the data generation stage, for keeping a uniform data structure for all elements, but they were not introduced to the GNN model (see section 4.2). At this point, other permitted changes in level (such as thresholds) are not considered. Ramps are restricted both in the minimal required width and in the range of ramp slope. Note that the variations are applied to Parameter 1 for all node types, whereas the only value we change in Parameter 2 is the slope of the ramps.

Since this is an initial feasibility check, we examine the code requirements with several simplifications to ease the data preparation stage. We do not check for available turning spaces in rooms, but instead require that the narrowest part of any room is at least as wide as the required turning space. Explicitly checking for available free turning spaces requires information about fixtures and furniture which are currently not represented in the explored graph structure.

Assign a unique identifier to every space, door, stair and ramp in the basic floor plan

Arrange the elements in a list and assign each element with their corresponding geometric attributes (parameter 1 and parameter 2)

List of elements (nodes)			
Node ID	Enum. Node type	Parameter 1	Parameter 2
1	0	170	1
2	0	110	3
3	0	230	2
4	0	160	2
5	0	160	3
6	0	95	0
7	0	120	3
8	0	120	3
9	0	240	2
10	0	200	2
11	0	300	1
12	0	110	1
13	0	400	1
14	1	90	0
15	0	85	0
16	0	85	0
17	0	85	0
18	0	85	0
19	0	85	0
20	0	85	0
21	0	85	0
22	0	85	0
23	0	85	0
24	0	85	0
25	0	85	0
26	1	85	0
27	2	80	1

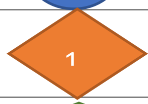
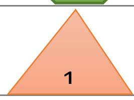
List of connections (edges)	
Source	Destination
14	1
1	16
16	6
6	15
15	2
6	17
17	3
6	18
18	9
6	27
27	11
6	19
19	7
11	26
11	22
22	10
10	21
21	4
4	20
20	5
11	23
23	8
8	24
8	25
25	13
24	12

Generate a list of connections based on the topology of the basic floor plan

Populate the parameters of every element with random values within a predefined range to generate a floor plan variation

Figure 6 The procedure for generating a single variation of a basic floor plan.

Table 1 Parameters defined for every building element described in the graph

Symbol	Node type		Parameter 1	Parameter 2
	Space	0	Min Width	Class
	Door	1	Clear Width	Type
	Stairs	2	Width	Num. of stairs
	Ramp	3	Width	Slope %

As stated before, we aim to obtain graph representations of feasible buildings to serve as the "ground truth" data for training, hence the variations for the geometry of every element are restricted to values from a predefined range. For example, the parameter that represents the slope of a ramp is a random value within the range 3-12%. The parameter that represents the width of functional spaces that are of the class ‘corridor’ is a random value within the range 80-110 cm. The width values of other functional spaces are not fixed to a specific range, the exact range of values is determined based on the examination of every individual basic floor plan to ensure that the variations will not cause any contradictions in topology. In general, the goal is to define a range of feasible values for the parameters for every type of element. It is unlikely for example for corridors to be 60 cm wide or less. On the other hand, the range has been defined so that it contains some values that do not satisfy the accessibility code requirements and some that do. This will lead to a training set ("ground truth" cases) with both examples of elements labeled “pass”, and elements that do not satisfy code requirements that would be labeled “fail”.

To sum up, we aim to create a synthetic "ground truth" data set for training the ML model. During this process we define a specific graph structure to represent this "ground truth" (at the training stage), and the exact same structure needs to be kept when extracting data from BIM models to be classified (at the prediction stage). This means that some processing of the BIM models still needs to happen during the prediction stage in order to retrieve the same representation of building information as used for training. Investigating the extent of required processing of the BIM models to be checked is outside the scope of this work.

4.2. Graph representation and features extraction

The lists of elements and lists of connections are transformed into undirected, unweighted graphs, and used as the training set for GNN. The quality of the generated graphs will be evaluated by the results of training and later by the ability of the trained GNN to make predictions on real design. Therefore, as stated before, although we do not directly generate 3D geometry, we aim to generate graph representations of feasible buildings. To ensure that, data generation begins with a random collection of floor plans from the publicly available floor plans on the internet. For this work, we collect 10 floor plans of single-family houses. Each basic floor plan is modified 100 times, which results in 100 graphs with the same topology but different geometry. Namely, the 100 graphs are represented by the same list of connections, but different parameters are assigned to every node. Overall, the obtained data set contains 1,000 graphs, each representing a single residential house.

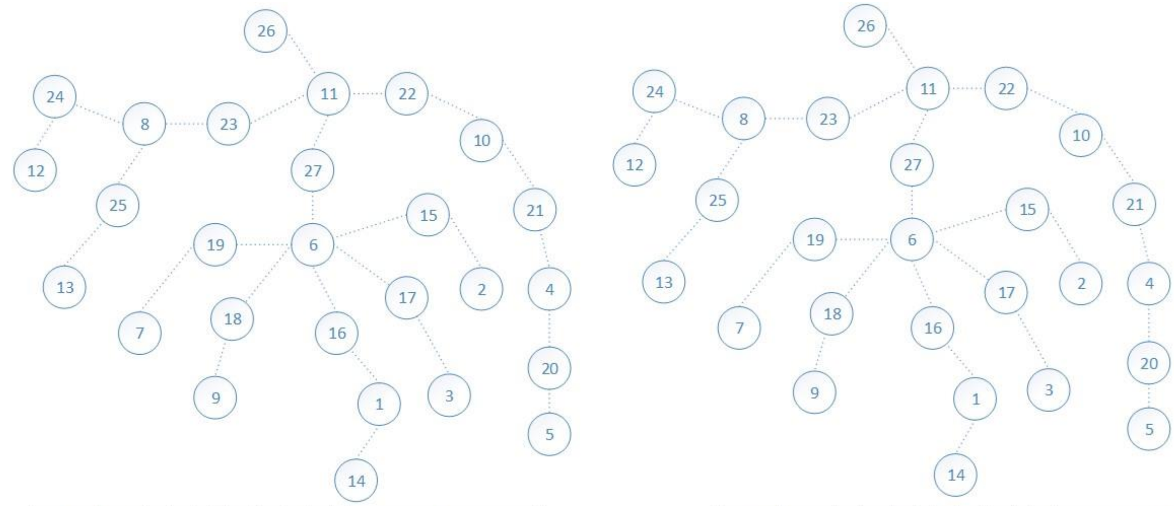
We then further process the parameters to define feature vectors for every node. All parameters created in the element list are transformed to numeric features by mapping the categories to numeric values using predefined unique values in accordance with key values from the accessibility code. For example, the "space" feature is populated with value 1 for every element that represents a space in the building and 0 for other element types. Instead of using the specific dimensions of spaces, we extract the key values from the code requirements and use them as categorical features. For example, corridors are required to be at least 91.5 cm wide to be accessible. Hence the value of the corresponding feature is set to 1 if the width of the element is greater than 91.5 cm, and 0 otherwise. This results in a feature vector of length nine assigned to every node. The final list of features contains the categorical values described in *Table 2* below.

Table 2 List of features and their possible values

#	Feature	Possible values

1	Space	1- If the element is a space 0- For all other elements
2	Door	1- If the element is a door 0- For all other elements
3	Stairs	1- If the element is a stair 0- For all other elements
4	Ramp	1- If the element is a ramp 0- For all other elements
5	Is width greater than 170	1- If the width of the element is greater than 170 cm 0- Otherwise
6	Is width greater than 91.5	0- If the width of the element is greater than 91.5 cm 1- Otherwise
7	Is width greater than 81.5	0- If the width of the element is greater than 81.5 cm 1- Otherwise
8	Accessible route	1- If the element is a space that is part of the circulation path (such as a corridor) 0- For all other elements
9	Slope	1- If the element is a ramp and its slope is within the range of 5-8.3% 0- Otherwise

An example of a basic floor plan and a single variation of that floor plan is illustrated in *Figure 6*. The corresponding graph for the basic floor plan is represented in *Figure 7* (a) and the variation is represented in *Figure 7* (b). Note that the topology of both floor plans is the same. However, in every variation the set of feature vectors assigned to each node is different. As illustrated in *Figure 7*, slight differences in the features may also affect the true labels of each node. We can see for example that the change in the dimensions of space 8 changed the true label of the node from "not compliant" to "compliant but not accessible". The meaning of the given node labels is explained in detail in the following section.



Graph ID	Node ID	S	D	St	R	170	92	82	Co	RS	Label
101	1	1	0	0	0	0	1	1	0	0	Compliant and accessible
101	2	1	0	0	0	0	1	1	1	0	Not compliant
101	3	1	0	0	0	1	1	1	0	0	Compliant and accessible
101	4	1	0	0	0	0	1	1	0	0	Not compliant
101	5	1	0	0	0	0	1	1	1	0	Not compliant
101	6	1	0	0	0	0	1	1	1	0	Compliant and accessible
101	7	1	0	0	0	0	1	1	1	0	Not compliant
101	8	1	0	0	0	0	1	1	1	0	Not compliant
101	9	1	0	0	0	1	1	1	0	0	Compliant and accessible
101	10	1	0	0	0	1	1	1	0	0	Compliant but not accessible
101	11	1	0	0	0	1	1	1	0	0	Compliant but not accessible
101	12	1	0	0	0	0	1	1	0	0	Not compliant
101	13	1	0	0	0	1	1	1	0	0	Compliant but not accessible
101	14	0	1	0	0	0	0	1	0	0	Compliant and accessible
101	15	0	1	0	0	0	0	1	0	0	Compliant and accessible
101	16	0	1	0	0	0	0	1	0	0	Compliant and accessible
101	17	0	1	0	0	0	0	1	0	0	Compliant and accessible
101	18	0	1	0	0	0	0	1	0	0	Compliant and accessible
101	19	0	1	0	0	0	0	1	0	0	Compliant and accessible
101	20	0	1	0	0	0	0	1	0	0	Compliant but not accessible
101	21	0	1	0	0	0	0	1	0	0	Compliant but not accessible
101	22	0	1	0	0	0	0	1	0	0	Compliant but not accessible
101	23	0	1	0	0	0	0	1	0	0	Compliant but not accessible
101	24	0	1	0	0	0	0	1	0	0	Compliant but not accessible
101	25	0	1	0	0	0	0	1	0	0	Compliant but not accessible
101	26	0	1	0	0	0	0	1	0	0	Compliant but not accessible
101	27	0	0	1	0	0	0	0	0	0	Not compliant

Graph ID	Node ID	S	D	St	R	170	92	82	Co	RS	Label
141	1	1	0	0	0	1	1	1	0	0	Compliant and accessible
141	2	1	0	0	0	0	1	1	1	0	Not compliant
141	3	1	0	0	0	1	1	1	0	0	Compliant but not accessible
141	4	1	0	0	0	1	1	1	0	0	Compliant but not accessible
141	5	1	0	0	0	1	0	1	1	0	Not compliant
141	6	1	0	0	0	1	1	1	1	0	Compliant and accessible
141	7	1	0	0	0	0	1	1	1	0	Not compliant
141	8	1	0	0	0	1	1	1	1	0	Compliant but not accessible
141	9	1	0	0	0	1	1	1	0	0	Compliant and accessible
141	10	1	0	0	0	1	1	1	0	0	Compliant but not accessible
141	11	1	0	0	0	1	1	1	0	0	Compliant but not accessible
141	12	1	0	0	0	1	1	1	0	0	Compliant but not accessible
141	13	1	0	0	0	1	1	1	0	0	Compliant but not accessible
141	14	0	1	0	0	0	0	1	0	0	Compliant and accessible
141	15	0	1	0	0	0	0	1	0	0	Compliant and accessible
141	16	0	1	0	0	0	0	1	0	0	Compliant and accessible
141	17	0	1	0	0	0	0	0	0	0	Not compliant
141	18	0	1	0	0	0	1	1	0	0	Compliant and accessible
141	19	0	1	0	0	0	0	0	0	0	Not compliant
141	20	0	1	0	0	0	0	0	0	0	Not compliant
141	21	0	1	0	0	0	1	1	0	0	Compliant but not accessible
141	22	0	1	0	0	0	0	1	0	0	Compliant but not accessible
141	23	0	1	0	0	0	1	1	0	0	Compliant but not accessible
141	24	0	1	0	0	0	0	1	0	0	Compliant but not accessible
141	25	0	1	0	0	0	0	0	0	0	Not compliant
141	26	0	1	0	0	0	0	0	0	0	Not compliant
141	27	0	0	1	0	0	1	1	0	0	Not compliant

(a) *(b)*

Figure 7 Graph representation of a basic floor plan and a single variation of that floor plan (a) represents the topology and the node features of the original base floor plan, (b) represents the topology and the node features of a floor plan variation

4.3. Labeling the data

The "ground truth" labels for the created data are the targets for the ML model. Developing this labeled "ground truth" dataset often requires manual labeling by experts. Through these labels we are able to leverage the knowledge of human experts in the field without trying to hard code it. Graph-based learning provides algorithms for graph classification as well as node classification.

Since the goal here is to check compliance of residential houses to the requirements of the accessibility code, both algorithms may be useful. However, graph classification will provide only a broad indication of a problem without specifying what the problem is or where it occurs. That is, if the graph is classified as “pass” that means that the entire house corresponds to the code requirements making it accessible. However, if the graph is classified as “fail”, we know that at least one space in the house is not accessible, but we have no indication which space and what design requirement are not satisfied. Therefore, the chosen approach in this case is node classification.

Since the graph structure includes nodes that represent spaces, doors, stairs and ramps, each of those elements will be classified as compliant or not compliant to the code. Hence, we will have an indication of where the problem occurs. To also receive an indication of what the problem is, we extend the problem to a multi-class classification problem where the possible labels are:

a) compliant and accessible – for elements that satisfy the geometric requirements of the accessibility code and can be reached through a path that consists of other compliant elements.

b) compliant but not accessible – for elements that satisfy the geometric requirements of the accessibility code but cannot be reached through a path that consists of other compliant elements.

c) not compliant – for elements that do not satisfy the geometric requirements of the accessibility code.

The possible labels indicate that the labeling process needs to be performed in two stages. First, we must check all individual elements’ compliance against the geometric requirements from the accessibility code. This includes simple geometric requirements such as the minimum width of a door, minimum width of a corridor, restricted range of ramp slope, etc. This stage is performed with a set of IF – THEN statements. Once all the individual elements are classified and assigned with a temporary "pass" or "fail" label, we search for all possible paths from the entrance to the house to every space to determine if it is accessible. For example, to determine the final label of the bedroom on the South East corner of the floor plan presented in *Figure 6*, it is not enough to check if the space itself is compliant with the geometric requirements from the code; we must also check the compliance of all the elements that generate a path to that space. To access this bedroom, we must enter the house (element 14), go through the foyer (element 1), then another door (element 16) to the corridor (element 6), through another door (element 17) and finally to the bedroom (element 3). In this case the only possible path from the source (entrance to the house), to the target is as follows: 14,1,16,6,17,3. To determine the final label of element 3 we look at the initial labels assigned to every element in the path. If the initial label assigned to the target node (space 3 in this case) is "fail", then the final label of node 3 would be "not compliant". Otherwise, we look at the rest of the elements in the path. If all of them are initially assigned with a "pass" label, then node 3 would be "compliant and accessible". If any of the element in the path are assigned with an initial "fail", then node 3 would be labeled "compliant but not accessible".

The second stage of labeling is entirely based on the topology of the floor plan. This emphasizes the weakness of "classic" ML learning algorithms and the strength of graph-based algorithms. ML algorithms are limited in expressing the topological relationships between entities; a GNN is expected to overcome this limitation.

5. EXPERIMENT AND RESULTS

The small scale problem designed for proof of concept of a GNN-based code checking consists of checking several design requirements presented in the accessibility code (International Code Council and American National Standards Institute 2010). The test case is focused on checking single family residential houses for compliance with the basic geometric requirements, such as minimum width, defined for an accessible space. We also consider the existence of accessible paths in individual buildings. We use the synthetic data set described in the previous section to train, validate and test a GNN model.

Specifically, a Graph Attention Network (GAT) model was trained in a full batch mode containing 28,400 nodes and 27,900 edges, as illustrated in *Figure 8* which consists of 1,000 unconnected subgraphs, each representing a single-family house. The data is randomly split to three parts: training, validation and testing. In this experiment, 60% of the data was used as a training set, the remaining data was split to 30% as the validation set and the remaining 10% were used for testing. Training and validation sets are iteratively used to optimize the model's hyper-parameters. The test set (a portion of the synthetic data) is kept out until the model is finalized and used to check the final performance of the model.

```
[28400 rows x 9 columns]
StellarGraph: Undirected multigraph
 Nodes: 28400, Edges: 27900

 Node types:
  nodes: [28400]
    Features: float32 vector, length 9
    Edge types: nodes-edges->nodes

 Edge types:
    nodes-edges->nodes: [27900]
        Weights: all 1 (default)
        Features: none
```

Figure 8 Graph generation of the synthetic data set as an undirected graph with 28,400 nodes and 27,900 edges

When dealing with accessibility requirements, the geometric representation of the spaces, doors and ramps is as significant for compliance checking as the topology of the building. Namely, when propagating the messages to a target node, we need to keep in mind that if a neighboring node is not accessible based on the geometric features assigned to it (such as width, slope, etc.), it might directly influence the target node making it not accessible even if the target node is compliant to the basic geometric requirements. We assume that the GAT model has the expressive ability in terms of propagating node features, therefore we implement a GAT model for node classification and compare the models' performance to the performance of the basic GCN model. Based on the work of (Veličković et al. 2017), it is beneficial to extend the attention mechanism and employ a multi-head attention mechanism for every node. Namely, the attention mechanism is performed several times for every pair of nodes. To obtain the new feature representation of the target node all the attention coefficients can be averaged.

The final GNN model architecture consists of four layers and 5 attention heads implemented in each layer. The rectified linear function (Relu) was used as the activation function for all hidden layers. Learning rate was set to 0.01 and the dropout value to 0.1. The training was performed in 300 epochs and took 12.58 minutes on a personal computer with Intel(R) Core(TM) i7-4790 CPU (3.60GHz) and 8.00 GB RAM.

Accuracy of predictions made on the test set while using the best trained model was 0.868, namely 86.8% of the nodes were classified correctly. Nevertheless, the accuracy score is not always a good indication of the predictive power of a GNN model. In particular, when the training set is not well balanced, high accuracy scores may be obtained for poor quality classifiers. One of the limitations of the generated data set is that it is not balanced. *Table 3* presents the label counts for the overall synthetic data set.

Table 3 Label count for the overall synthetic data set and for the portion of the synthetic data set used for training the model. The rest of the data was used for validation and testing.

Label	Total label count	Training set label count
Compliant and accessible	13,991	8,395
Compliant but not accessible	7,972	4,783
Not compliant	6,437	3,862

To evaluate the predictive power of the obtained model, we extract the confusion matrix and calculate the F1 score based on the precision and recall of the test results. The confusion matrix is illustrated in *Figure 9* (a). The F1 score is calculated as the harmonic mean of precision and recall and reaches its optimum 1 only if precision and recall are both at 100%. The obtained F1 score in this case is 0.86 which indicates the obtained classifier performs well on unseen data.

The performance was also compared to a Graph Convolutional Network (GCN) to begin exploring the influence of a chosen GNN model on the results. The architecture of the GCN is similar to the architecture of the GAT mode, it contains four layers with a Relu activation function and a learning rate of 0.01. *Figure 9* presents the confusion matrix for predictions made on the test set based on a GAT model (a) and on the GCN model (b). The F1 score obtained from the GCN model is 0.734 which indicates that the GAT model performs significantly better.

Predicted (GAT)			
True Label	Com_Acc	Com_not_Acc	Not_Com
Com_Acc	3735	67	115
Com_not_Acc	599	1531	102
Not_Com	167	19	1617

(a)

Predicted (GCN)			
True Label	Com_Acc	Com_not_Acc	Not_Com
Com_Acc	3717	83	117
Com_not_Acc	1114	1007	111
Not_Com	257	181	1365

(b)

Figure 9 Confusion matrix based on predictions on test data. (a) is the confusion matrix obtained from the GAT model and (b) is the confusion matrix obtained from the GCN model

To validate the applicability of the proposed approach to "real world" data, the obtained classifier was used to check accessibility of three floor plans obtained from local architectural firms in Israel. The obtained floor plans were translated to a graph representation manually, following the same structure as the training data set. Table 4 provides a general description of the floor plans. *Figure 10* illustrates floor plan 1 and its representative graph topology as an example. Overall, 88% of nodes (across all three floor plans) were classified correctly using the obtained classifiers. The presented results are an indication that the suggested workflow for implementing GNNs trained on synthetic data for ACC has great potential to overcome the existing challenges in the ACC process. We can clearly see that this is a valid research direction that can greatly contribute to further develop the ACC field. A deeper analysis of the performance capabilities and limitations of the obtained classifier for “real-world” data is now a subject of ongoing work.

Table 4 Description of floor plans obtained from local architectural firms

Floor plan	General description	Number of levels	Number of Bedrooms	Number of graph nodes	Number of correctly classified nodes
1	Private single family house	1	3	23	20
2	Private house with a connected independent dwelling unit	2	4 in main house +2 in dwelling unit	46	38
3	Private single family house	1	4	24	22
Total percentage of correctly classified nodes in all three floor plans				88%	
Total percentage of correctly classified nodes in the test case which is a portion of the synthetic data				86.8%	

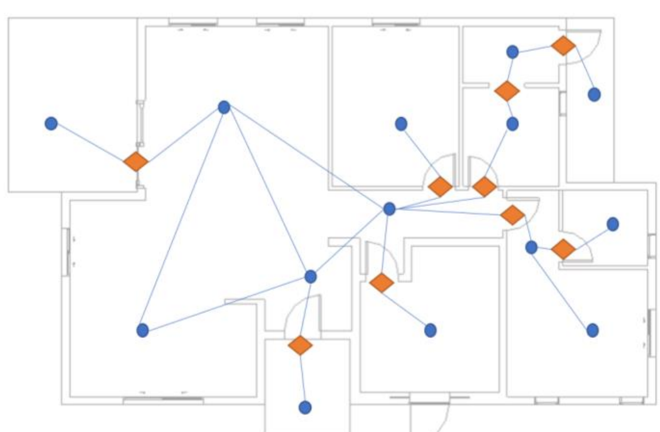


Figure 10 Floor plan 1 obtained from the industry overlaid with its representative topology graph

Although these are promising results, we assume that they can be further improved through a more thorough and detailed examination of the data, the graph structure and the model. A good indication of that is the difference in performance of GAT and GCN models. It has been previously shown that different architectures of popular GNNs vary in their expressive power (Xu et al. 2018). To obtain the best results for the problem at hand, the combination between the graph representation of a building and the GNN model needs to be further explored.

For example, a different possible graph structure for this work can consist of nodes that represent only spaces. Information about the doors, stairs or ramps leading to every space can be assigned to the nodes as attributes. This will lead to a lower number of data points but longer feature vectors. Adding other code requirements to the classification can also lead to more node attributes and more defined classes, yet this likely increases complexity and is expected to lower the values of precision and recall. Nevertheless, adding more code requirements can also be beneficial in terms of the code checking algorithm since it will enable a simulation check of many code requirements.

Every option discussed above can have a significant effect on the performance of the GNN model and eventually on the accuracy and efficiency of the code compliance checking. This work illustrates a new approach for the code checking problem and demonstrates promising initial implementation results. This points to a valid future research direction of GNN-based ACC.

6. DISCUSSION

Constructing safe and efficient buildings is of high societal interest. The regulatory codes and standards in effect ensure safety and usability of the buildings. However, their manual checking costs valuable time and labor resources, especially in the field of highly skilled building engineers. Although commercial applications for code checking are available, they provide a limited solution and are not widely adopted in the industry. Moreover, because they tend to rely on hard-coded rules, either declarative logic statements or procedural code, the development of such applications is a long and costly process that requires much effort and relies on a large amount of manual operations.

This research applies a novel approach to automated code checking and has the potential to make a significant breakthrough in the field. For a proof of concept, we demonstrate the proposed approach through a small-scale problem of compliance checking of single family houses to several requirements from the code for accessible and usable buildings and facilities (International Code Council and American National Standards Institute 2010). The chosen regulations concern both geometric restrictions as well as topological constraints. While the considered regulations are relatively simple, simplifying the process for generating and labeling a large synthetic data set, they allow us to demonstrate that GNNs can indeed be useful to leverage topological information for ACC and thus can contribute to automated code checking of regulations that involve both geometric and topological requirements. We do not suggest to replace the workflow of code checking for simple prescriptive regulations that can be translated to logical statements. Instead, we propose to expand the scope of regulations that can be addressed automatically with ML, and to supplement the abilities of ML approach for ACC with graph-based learning to target regulations that involve relational concerns (either topological or any other type of relationships). The ability to deal with different types of regulations needs further investigation. The results indicate that the application of GNNs to automated code checking is a valid direction for further research in hopes of achieving highly automated ACC systems that cover a wider range of code requirements. However, as all ML algorithms, GNNs provide probabilistic results that may not always be correct. A further investigation into the performance of GNNs under different constraints is required to fully understand the strengths, weaknesses and boundary conditions of the investigated approach.

6.1. Comparison with existing ACC approaches

Previous research on ACC indicates that, while the rule-based approach provides reliable results, it is limited in scope and requires a lot of manual processing both for rule compilation and for building information extraction. As many of the existing regulations are performance-based, they are difficult to express by rigid rules. In those cases, training with examples of design models that are assessed by human plan checkers allows the ML models to learn and mimic the way that human checkers assess the design, without relying on explicitly defined rigid conditions. The ML approach can overcome the problem of rule compilation, but the representation of building information as input to classic ML algorithms remains problematic. Furthermore, the results of compliance checking by ML cannot reach the same accuracy as rule-based checking since results obtained with ML are probabilistic. Nevertheless, previous research on the subject demonstrates good performance of classic ML for ACC in terms of accuracy.

Application of GNN to ACC can overcome some of the limitations of the classic ML algorithms. Graphs are a suitable way to represent building information and they have been numerously used for various applications in the construction domain. Since GNNs are implemented directly on graphs, application of learning algorithms to complex problems such as code compliance checking becomes possible. This work illustrates such an implementation and demonstrates that GNNs perform well in this domain. In addition, the presented results can be further improved by an in-depth investigation into the most suitable graph structures and GNN model architectures. We can conclude that although applications of GNN to ACC can contribute to widening the scope of regulations that can be checked automatically, the knowledge base on the subject is mostly lacking. Based on the presented work, a comparison between the different approaches to ACC, and the existing body of knowledge required for further development of each approach is presented in *Table 5* below.

Table 5 Comparison between rule-based ACC, ML-based ACC and GNN-based ACC based on existing work

Comparison criteria	**Rule-based ACC**	**Classic ML for ACC**	**GNN-based ACC**
Rule compilation	Required	Not required	Not Required
Data collection	Does not require data besides the design to be checked.	Needs to be collected, arranged and managed.	Needs to be collected, arranged and managed.
Data extraction	Mostly automated, requires some data to be manually supplemented during the checking.	Manual in all existing work. Automation is possible but has not been demonstrated in previous work.	Graph generation is manual in this work. Automation is possible but has not been demonstrated yet.
Data representation	Data is acquired from the commonly used data formats	Requires feature extraction. A structured feature representation is	All building data can be represented.

	such as the IFC. Additional data is manually supplemented through a user interface.	not suitable for complex topologies.	
Accuracy	Very high.	Very high based on existing test cases.	High based on an initial test.
Scope	Limited by the ability to compile hard-coded rule sets.	Limited, difficult to deal with topologically complex requirements.	Applicable to regulations that address both the geometry and the topology of the design.
Simultaneous check of several code clauses and report details	Works with individual code requirements. Able to report the specific building elements that violate the code clause at hand.	Can check compliance to several code requirements simultaneously. There is no indication of what code requirement is violated unless it is expresses as a possible label in the classification problem.	Can check compliance to several code requirements simultaneously. There is no indication of what code requirement is violated unless it is expresses as a possible label in the classification problem.
Existing knowledge in the field	Very high. Has been a subject of research for over 50 years.	Limited. ML has been applied and explored as the checking mechanism only a few times before.	Very limited.

6.2. Limitations

This is an exploratory work designed to test the hypothesis that graph-based learning can be implemented as the checking mechanism for ACC. As such, the obtained results are limited to an initial proof of concept for the application of GNN to automated accessibility checking in residential buildings. Based on the results, we can conclude that the application of GNNs for ACC is a valid research direction that needs to be further explored. However, we cannot claim that the proposed approach is feasible for all problems from the ACC domain. Nor can we claim that GNNs outperform other approaches for ACC.

Since all learning models, including GNNs, provide probabilistic results, they are less reliable than results obtained by the application of rule sets. Dealing with design review, we must consider the issue of liability in case the obtained results are false. False positives obtained in the checking process may carry significant safety issues in the designed buildings that will be ignored. Hence, it is possible that the final conformance should be left to the human checker. Nevertheless, additional research and development of the graphs, learning models and training sets can lead to well-performing classifiers that provide sufficiently accurate results. Considering the large volume of work that plan checkers deal with, often not the entire design is processed and random spot checks are performed instead. In addition, making mistakes is an inevitable part of being human, we can assume that domain experts make mistakes too. Therefore, extensive further development of ML-based workflows for ACC can result in reliable models that are able to quickly and efficiently process the full design with a similar accuracy to that of a domain expert.

One of the drawbacks of this research is that it was performed on a synthetically generated data set. The workflow for preparing the data set is designed to generate graphs that represent realistic layouts of residential buildings. To be able to label the generated data, modifications of the basic floor plans were introduced to the geometric features of the relevant building elements, however no modifications to the topology were introduced. It is possible that using a more diverse data set for training can lead to a different performance of the GNN.

6.3. Further work

Currently, there is no pre-existing knowledge on application of graph-based learning to ACC. This work is designed to explore the possibility of using GNNs as the checking mechanism for automated code checking. The obtained results from this study are encouraging to establish this as a valid research direction in the domain of ACC. The demonstrated initial feasibility of applying a GAT model for an accessibility check raises several directions for needed further research. First, the hypothesis that a classifier trained on synthetic data can provide sufficiently precise predictions of “real world” data needs to be further tested. Hence, validation of the proposed approach using several case studies from the industry is a subject of ongoing work.

Additionally, more complex application scenarios need to be explored in order to understand the capabilities and limitations of GNNs in the ACC domain. Specifically, test cases that deal with regulations that cannot be properly translated to rules should be investigated either by collecting a labeled data set, or by involving experts to label a synthetic data set. Currently, many regulations can be checked automatically using existing, rule-based applications. Although full automation is rarely achieved, there is a great benefit to the existing method as it provides accurate and reliable results. We believe that not all code clauses should be translated to classification tasks as there may be regulations that would not benefit from the application of learning methods. Prescriptive requirements written without ambiguities are better solved deterministically. ML is useful to deal with other requirements that maybe vaguely defined, within those the application of graph-based learning should be considered when complex relationships between a large set of building elements need to be examined. The combination of different approaches to code checking can contribute to widening the scope of regulations that can be checked automatically. Investigation into the different types of regulations to match them with the best approach for a solution is a valid direction for future research.

As described in Section 4 of the paper, the considered graph structure only contains representations of elements that are relevant to the regulation being checked. We can refer to these graphs as sub-graphs of the graphs that represent complete BIM models. This means that every regulation will define its own requirements for graph representations. The workflows for extracting such graphs need to be developed. The possibility to combine larger sets of requirements while using different graph structures to represent the buildings needs to be investigated as well.

6.4. Data in the ACC domain

In the world of data that we are currently living in, we need to find ways to leverage the available data in the construction domain and strive to enable data-driven decision-making. Every construction project produces vast amounts of data through its life cycle, beginning with programming documents, design documents, digitalized models, data collected during construction and during the buildings' operation. With rapidly advancing technologies such as sensors and IoT and the adoption of such technologies in the AEC industry, the amount of available data is expected to increase even more. Yet, currently this data is not properly collected, organized and analyzed and we miss out on opportunities to harness the existing data for various applications. This work presents such a possible application by relying on a synthetically generated data set.

Code compliance checking has been a subject of interest for many researchers for over 50 years. This work suggests adopting a new perspective on the subject and revisiting this well-established process as a whole to explore the possibility of bigger and more meaningful progress.

7. CONCLUSIONS

Exploiting building information stored in a graph for various purposes is not a new field of research. This work suggests taking one step forward and extending the previously explored graph-based algorithms to learning algorithms. This work aims to explore two hypotheses; one is that GNNs are applicable to problems from the code-checking domain; another is that models trained on a completely synthetic data set can be implemented for code compliance checking of design obtained from the industry. We demonstrate the application of GNN to check code compliance of single-family houses to the requirements of an accessibility code. Since obtaining a large set of BIM models of specific building types is a difficult task, we suggest exploring the possibility of training the GNN on synthetic data, assuming a GNN trained on synthetic data will perform well on BIM models obtained from the industry.

In this work, we provide a detailed description of synthetic graph data generation where every graph represents a single-family house. The graph representation of every house includes spaces, doors, stairs and ramps as these are the main elements to determine accessibility. To maintain topological integrity and ensure that the generated graphs represent feasible buildings, the data is generated based on random variations of floor plans collected from the internet. Each node in a graph is labeled based on the code requirements to be checked. Since accessibility is determined both by the geometry of every individual element and by the existence of an accessible path leading to that element, the nodes are divided to three classes: compliant and accessible, compliant but not accessible and not compliant. The generated training set is one of the contributions of this work, as we expect the created knowledge on generating and labeling synthetic data sets to be exploited for applications other than code checking in future research.

A GAT model is trained and evaluated based on the generated data. It is then tested using a portion of the data that is held out from the training process. The accuracy of testing results is 86.8% and the obtained F1 score is 0.86 indicating that the trained model performs well on unseen data. The trained classifier was further tested for classifying design obtained from the industry. Applying the trained classifier to three different floor plans, 88% of building elements across the three floor plans were classified correctly. Based on the obtained results, we conclude that the application of GNN to ACC is a valid direction for future. We can see a trade-off between the accuracy obtained by the different approaches to ACC to the range of regulations that can be checked automatically. While the rule-based approach provides accurate results, it is limited to regulations that can be computerized. ML on the other hand is probabilistic and therefore cannot reach 100% accuracy. Nevertheless, it is much more flexible and can be applied to a wide range of regulations, even those that cannot be directly computerized. GNNs contribute by expanding the ability of "classic" ML to regulations that address the relational aspect between the building elements.

The ability to deal with the checking process as a whole, without dividing it to "processing regulations" and then "processing the design" is a benefit of implementing ML for ACC. Furthermore, we do not need to be concerned with translating the regulations to rules, but instead develop representations that encapsulate both the design and the regulations (implicitly) using the same data structure. However, as the existing knowledge on the subject is limited, further research is needed to understand the abilities, strengths and weaknesses of ML models in the ACC domain (including both "classic" models and graph based models). Further investigation of possible graph structures, possible GNN architectures, and the combination between the two is needed to fully understand the abilities, strengths, weaknesses and boundary conditions of applying GNN to ACC. As this work is focused on a simplified problem for an initial feasibility test, the application of the workflow for more complicated regulations, specifically regulations that cannot be directly computerized (either because of complexity or ambiguity) needs to be tested. Also, a hybrid approach, combination between the different approaches to ACC (rules, classic learning, graph based learning) can be explored to cover a wider range of regulations with sufficient accuracy. These can determine future research directions in the field.

References

Amor, R., and J. Dimyadi. 2021. “The promise of automated compliance checking.” *Developments in the Built Environment*, 5: 100039. https://doi.org/10.1016/j.dibe.2020.100039.

Balaji, B., A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, and others. 2016. “Brick: Towards a unified metadata schema for buildings.” *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments,* 41–50.

Bloch, T., M. Katz, and R. Sacks. 2018. “Machine learning approach for automated code compliance checking.” Tampere.

Bloch, T., M. Katz, R. Yosef, and R. Sacks. 2019. “Automated model checking for topologically complex code requirements – security room case study.” *Proceedings of the 2019 European Conference for Computing in Construction*. University College Dublin.

Borrmann, A., M. König, C. Koch, and J. Beetz. 2018. *Building information modeling: technology foundations and industry practice*. New York, NY: Springer Berlin Heidelberg.

Bronstein, M. M., J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. 2017. “Geometric deep learning: going beyond euclidean data.” *IEEE Signal Processing Magazine*, 34 (4): 18–42. IEEE.

Cao, W., Z. Yan, Z. He, and Z. He. 2020. “A Comprehensive Survey on Geometric Deep Learning.” *IEEE Access,* 1–1. https://doi.org/10.1109/ACCESS.2020.2975067.

Collins, F. 2020. “Encoding of geometric shapes from Building Information Modeling (BIM) using graph neural networks.”

Data61, C. 2018. “StellarGraph Machine Learning Library.” *GitHub Repository*. GitHub.

Dimyadi, J., and R. Amor. 2013. “Automated Building Code Compliance Checking–Where is it at.” *Proceedings of CIB WBC,* 172–185.

Donkers, A., D. Yang, B. de Vries, and N. Baken. 2021. “Real-Time Building Performance Monitoring using Semantic Digital Twins.” *Proceedings of the 9th Linked Data in Architecture and Construction Workshop, CEUR Workshop Proceedings, Luxembourg, Luxembourg*.

“Dynamo.” 2022. Accessed February 17, 2022. http://dynamobim.org/.

Eastman, C. 1975. “The use of computers instead of drawings in building design.” *AIA Journal*, 63 (3): 46–50.

Eastman, C., J. Lee, Y. Jeong, and J. Lee. 2009. “Automatic rule-based checking of building designs.” *Automation in construction,* 18 (8): 1011–1033. https://doi.org/10.1016/j.autcon.2009.07.002.

Erikoskarwallin, K. Hammar, Perkarlberg, Leifsundbom, and Wotifi. 2019. “RealEstateCore/rec: V3.0 -- Usability and maintainability refactoring.” Zenodo.

Fenves, S. J. 1966. “Tabular Decision Logic for Structural Design.” *J. Struct. Div.*, 92 (6): 473–490. https://doi.org/10.1061/JSDEAG.0001567.

Fenves, S. J., and W. J. Rasdorf. 1982. “Treatment of engineering design constraints in a relational database.” Carnegie Mellon University.

Gan, V. J. L. 2022. “BIM-based graph data model for automatic generative design of modular buildings.” *Automation in Construction*, 134: 104062. https://doi.org/10.1016/j.autcon.2021.104062.

Hamilton, W. L. 2020. “The Graph Neural Network Model.” *Graph Representation Learning*, 51–70. Cham: Springer International Publishing.

Hamilton, W. L., R. Ying, and J. Leskovec. 2017. “Representation Learning on Graphs: Methods and Applications.” *CoRR*, abs/1709.05584.

Han, C. S., J. Kunz, and K. H. Law. 1997. “Making automated building code checking a reality.” *Facility Management Journal*, 22–28.

Home Front Command. 2010. *Specifications for Building Shelters*. Ramle, Israel: Protective structures department, Home Front Command.

International Code Council, and American National Standards Institute (Eds.). 2010. *Accessible and usable buildings and facilities: ICC A117.1-2009: American National Standard*. Washington, DC: International Code Council.

Ismail, A., A. Nahar, and R. Scherer. 2017. “Application of graph databases and graph theory concepts for advanced analysing of BIM models based on IFC standard.” *Proceedings of EGICE*.

Ismail, A., B. Strug, and G. zyna Ślusarczyk. 2018. “Building Knowledge Extraction from BIM/IFC Data for Analysis in Graph Databases.” *Artificial Intelligence and Soft Computing*, L. Rutkowski, R. Scherer, M. Korytkowski, W. Pedrycz, R. Tadeusiewicz, and J. M. Zurada, eds., 652–664. Cham: Springer International Publishing.

Jin, C., M. Xu, L. Lin, and X. Zhou. 2018. “Exploring BIM Data by Graph-based Unsupervised Learning.” *ICPRAM*, 582–589.

Khalili, A., and D. K. H. Chua. 2015. “IFC-Based Graph Data Model for Topological Queries on Building Elements.” *J. Comput. Civ. Eng.*, 29 (3): 04014046. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000331.

Kipf, T. N., and M. Welling. 2016. “Semi-supervised classification with graph convolutional networks.” *arXiv preprint arXiv:1609.02907*.

de Koning, R., E. Torta, P. Pauwels, R. W. M. Hendrikx, and M. J. G. van de Molengraft. 2021. “Queries on Semantic Building Digital Twins for Robot Navigation.” *9th Linked Data in*

Architecture and Construction Workshop, CEUR Workshop Proceedings, 32–42. CEUR-WS.org.

Lin, Z., M. Feng, C. N. dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio. 2017. "A structured self-attentive sentence embedding." *arXiv preprint arXiv:1703.03130*.

Meijer, F., H. Visscher, and L. Sheridan. 2002. *Building regulations in Europe*. Housing and urban policy studies. Delft: DUP Science.

Nawari. 2019. "A Generalized Adaptive Framework (GAF) for Automating Code Compliance Checking." *Buildings*, 9 (4): 86. https://doi.org/10.3390/buildings9040086.

Nawari, N. 2012a. "The Challenge of Computerizing Building Codes in a BIM Environment." *Computing in Civil Engineering (2012)*, 285–292. Clearwater Beach, Florida, United States: American Society of Civil Engineers.

Nawari, N. O. 2012b. "BIM-Model Checking in Building Design." *Structures Congress 2012*, 941–952. Chicago, Illinois, United States: American Society of Civil Engineers.

Pauwels, P., T. M. de Farias, C. Zhang, A. Roxin, J. Beetz, J. De Roo, and C. Nicolle. 2017. "A performance benchmark over semantic rule checking approaches in construction industry." *Advanced Engineering Informatics*, 33: 68–88. https://doi.org/10.1016/j.aei.2017.05.001.

Pauwels, P., and W. Terkaj. 2016. "EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology." *Automation in Construction*, 63: 100–133. Elsevier.

Preidel, C., and A. Borrmann. 2015. "Automated Code Compliance Checking Based on a Visual Language and Building Information Modeling." *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, 1. Vilnius Gediminas Technical University, Department of Construction Economics & Property.

Rasmussen, M. H., M. Lefrançois, G. F. Schneider, and P. Pauwels. 2021. "BOT: the building topology ontology of the W3C linked building data group." *Semantic Web*, 12 (1): 143–161. IOS Press.

Sacks, R., T. Bloch, M. Katz, and R. Yosef. 2019. "Automating Design Review with Artificial Intelligence and BIM: State of the Art and Research Framework." *Computing in Civil Engineering 2019*, 353–360. Atlanta, Georgia: American Society of Civil Engineers.

Skandhakumar, N., F. Salim, J. Reid, R. Drogemuller, and E. Dawson. 2016. "Graph theory based representation of building information models for access control applications." *Automation in Construction*, 68: 44–51. https://doi.org/10.1016/j.autcon.2016.04.001.

Veličković, P., G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. 2017. "Graph attention networks." *arXiv preprint arXiv:1710.10903*.

Wang, Z., R. Sacks, and T. Yeung. 2022. “Exploring graph neural networks for semantic enrichment: Room type classification.” *Automation in Construction*, 134: 104039. https://doi.org/10.1016/j.autcon.2021.104039.

Wu, Z., S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. 2020. “A comprehensive survey on graph neural networks.” *IEEE transactions on neural networks and learning systems*, 32 (1): 4–24. IEEE.

Xu, K., W. Hu, J. Leskovec, and S. Jegelka. 2018. “How Powerful are Graph Neural Networks?” *CoRR*, abs/1810.00826.

Zhang, J., and N. M. El-Gohary. 2016. “Semantic NLP-Based Information Extraction from Construction Regulatory Documents for Automated Compliance Checking.” *J. Comput. Civ. Eng.*, 30 (2): 04015014. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000346.

Zhang, J., and N. M. El-Gohary. 2017. “Integrating semantic NLP and logic reasoning into a unified system for fully-automated code checking.” *Automation in Construction*, 73: 45–57. https://doi.org/10.1016/j.autcon.2016.08.027.

Zhang, Z., L. Ma, and T. Broyd. 2022. “Towards fully-automated code compliance checking of building regulations: challenges for rule interpretation and representation.” EC^3 (European Conference on Computing in Construction).

Zhi, G. S., S. M. Lo, and Z. Fang. 2003. “A graph-based algorithm for extracting units and loops from architectural floor plans for a building evacuation model.” *Computer-Aided Design*, 35 (1): 1–14. https://doi.org/10.1016/S0010-4485(01)00171-3.

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

Zhou, J., G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. 2018. “Graph neural networks: A review of methods and applications.” *arXiv preprint arXiv:1812.08434*.