An alternative approach to Automated Code Checking – Application of Graph Neural Networks trained on synthetic data for an accessibility check case study

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
[3] Faculty of Built Environment, Eindhoven University of Technology, the Netherlands

Abstract. Automated Code Checking (ACC) can be defined as a classification task aiming to classify building objects as compliant or not compliant to a code provision at hand. While Machine Learning (ML) is a useful tool to perform such classification tasks, it presents several drawbacks and limitations. Buildings are complex compositions of instances that are related to each other by functional and topological relationships. This type of data can be easily supported by property graphs that provide a flexible representation of attributes for every instance as well as the relationships between the instances. This, together with the recent developments in the field of graph-based learning led the authors to explore a novel approach for ACC supported by Graph Neural Networks (GNN). This paper presents a new workflow that implements GNNs for ACC to leverage the advantages of ML but alleviate the limitations. We illustrate the suggested workflow by training a GNN model on a synthetic data set and using the trained classifier to check compliance of a real BIM model to accessibility requirements. The accuracy of the classifier on a test set is 86% and the accuracy of obtained results during the accessibility check is 82%. This suggests that GNNs are applicable to ACC and that classifiers trained on synthetic data can be used to classify building design provided by the industry. While the results are encouraging, they also point to the need for further research to establish the scope and boundary conditions of applying GNNs to ACC.

Keywords: Automated Code Checking (ACC), Machine Learning (ML), Graph Neural Networks (GNN), Building Information Modeling (BIM), accessibility.

1 Introduction

In current practice, Automated Code Checking (ACC) is performed by qualified experts manually and it is a costly, cumbersome, error prone and time consuming process [1]. As our digital design capabilities increase, our buildings become more complex making it even more difficult to check their compliance to all codes, regulations and standards to ensure safety and usability of the designed building. Automating the process can be of great benefit for the construction industry in many aspects. For example, as design review is one of the stages for construction permit approval in many countries [2], automating that stage can lead directly to shortening the time needed for construction permitting. Due to these potential benefits and the ability to represent building information in a computer readable manner using Building Information Modeling (BIM), the subject of ACC received much attention in the scientific community for the past 50 years [3]. Although there has been much progress in the field, even the most advanced commercial platforms for ACC (such as Solibri for example [4]) fail to provide a comprehensive and fully automated tool for code checking.

Majority of research into the subject focus on a rule based approach as described in [1]. This consists of representing the regulations in a computer readable format and enhancing computer readability of the BIM model to be checked, either manually or by automated processes such as semantic enrichment [5]. The design review process is eventually a matching of concepts represented in the regulations to those represented in the BIM model. This consists of mapping between the two and interpretation of meaning and intent, which usually requires considerable amount of manual work.

In this work, we propose to look at ACC in a different manner and define code checking as a classification task, where the goal is to assign the building (or a building element) with a "pass" label if it satisfies all relevant design requirements and a "fail" label if it violates one or more of the requirements. We therefore propose an alternative workflow for ACC that relies on a novel Machine Learning (ML) approach applied directly to a graph representation of the building information. In this paper we illustrate the proposed workflow on a simple test case of accessibility check in single family homes. Through the test case, we are able to illustrate the initial feasibility of applying novel ML techniques to ACC, but also to explore more general issues such as the core differences between the existing approaches for ACC, their advantages, limitations, use of synthetic data and direction for future research.

2 Background

Automated Code Checking is usually considered as a four-stage process that consists of translating the code requirements to logical statements, preprocessing of the BIM model, rule execution and report [1]. The need to translate codes written in natural language to logical statements, and the need to preprocess the BIM model to supplement all the semantically rich information required for checking, are the main challenges that hinder the development of a fully automated ACC platform that covers a wide range of requirements [1], [3], [6]. In this work, we suggest to reevaluate the general approach to ACC. The underlying assumption of this work is that application of novel Machine Learning techniques for design review can overcome some of the existing challenges (such as compiling rules for performance based regulations [7]), thus allowing a breakthrough in the field. Since ML relies on learning from past experience, it does not require to translate the codes and regulations into computable rules. A ML model is trained using a set of labeled examples where the labels are provided by experts in the field. In the case of code checking, these labels express the conformance of a proposed design to a specific code clause [8]. Namely, the regulations are implicitly captured in the set of labeled examples used to train the ML model. Thus, implementing ML as the checking mechanism eliminates the need to engage in the challenging task of converting natural language in to computable rules [9].

Although the idea of using Machine Learning (ML) techniques as the checking regime has been presented before [8], [10], the existing research is focused on very simple test cases and presents several drawbacks of the process. One is the lack of data for training, and the other is the difficulty in representing building information in a structured tabular form. Therefore, in this work we present and illustrate a workflow in which a Graph Neural Network (GNN) is implemented as the checking mechanism for automated code checking. Switching to a checking regime that is based on learning instead of hard-coded rules will eliminate the need to process the written documents. In addition, since graph structures are very suitable for representing building information in a complete and detailed form [11], [12], we expect to be able to overcome some of the drawbacks of using "classic" ML tools for code checking.

2.1 Application of Graph Neural Networks to building information

Buildings are complex structural systems composed of many elements that are related to each other by functional and topological relationships. Buildings, even of the same type, are designed with diverse shapes, functions and other characteristics, making it difficult to identify fixed data structures to represent them, as usually required by the classic ML applications. Graphs, on the other hand, due to their flexibility are extremely useful for describing such complex systems by representing building elements as nodes and the relationships as edges [13]. A Labeled Property Graph (LPG) is able to represent both the geometry of the building elements, through a set of values (features) assigned to each node in the graph, as well as the

spatial relationships amongst them, through the edges connecting the nodes [14]. With the development of graph based learning methods and recent advances in graph data science [15], we are now able to better leverage the capabilities of ML techniques by applying them directly to the graph structures representing the building information.

Graph-based learning is useful for dealing with data that cannot be appropriately structured in a tabular or hierarchical form [16]. Graph Neural Networks (GNN) operate directly on the graph. The goal is to learn a d-dimensional vectorized representation (node embedding) of every node in the graph, that represents the attribute information assigned to each node and preserves the topological information described in the graph [17]. To do so, every node defines a computational graph, which consists of the node's neighbors up until *k* hops away from the node (denoting the number of layers). Each such neighborhood graph is used to propagate the information from all neighboring nodes across all the graph layers to compute a node embedding [18], a process called message passing. The node embeddings are generated based on the local neighborhoods while every node aggregates the information from its neighbors using neural networks.

For example, every node in the input graph illustrated in Fig. 1 defines its own neighborhood graph. Looking at the immediate neighbors of every node is equivalent to a GNN with a single layer (*k*=1). To learn the node embedding of node A in a single GNN layer for example, we transform the representations (messages) of all immediate neighbors of node A and aggregate them. This is parametrized and sent through a Neural Network to introduce non-linearity. The result is a d-dimensional vector that encapsulates information about the attributes assigned to node A, as well as information about its position in the graph.

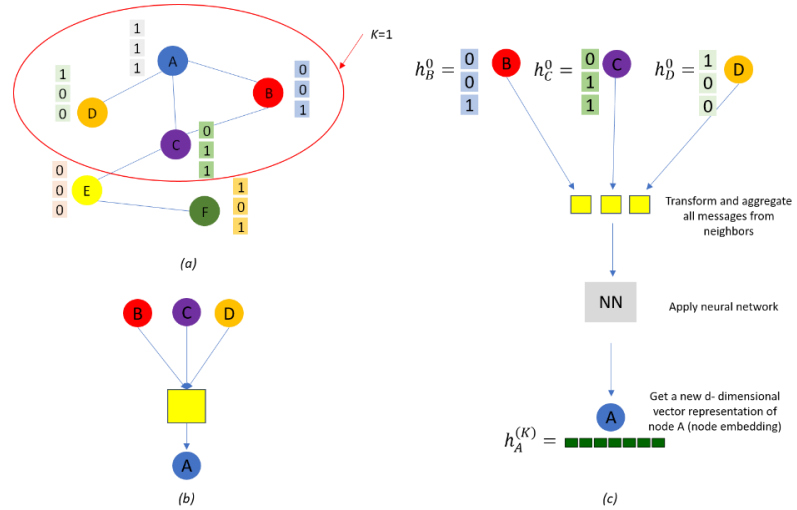


Fig. 1. Message passing in a single GNN layer. (a) illustrates an example input graph (b) is the computational graph defined by node A, and (c) is the message passing process computing the embedding of node A in a single layer GNN.

Many GNN architectures have been developed and demonstrated for various applications over the years [19]. The use of GNNs was recently proved useful for the construction domain as well. For example, in the work of [20] a Graph Convolutional Network (GCN) [21] model was applied for node classification to support point cloud data processing. In the work of [22] a SAGE-E model (an enhancement of the SAGE model [23]) was applied to classify room types in residential buildings for semantic enrichment purposes. In this work we define the task of code compliance checking as a classification task and aim to explore the applicability of GNNs for that task. We implement the Graph Attention Network (GAT) model [24] in the StellarGraph library [19] to perform the classification task. The main difference between GAT to other GNN models is that not all messages propagated from neighboring nodes are considered equally important. The assumption is that information from some nodes might be more significant for computing node embedding than others. Hence, in GAT every message is normalized by an attention factor that is learned for every neighbor separately.

2.2 The use of synthetic data for Machine Learning

Graph-based learning, as any supervised ML algorithm, is reliant on a large data set of examples for training. In this case, the data set should consist of building design information, structured in a form of an LPG, labeled as compliant or not compliant to the specific code requirement at hand. Since such large data set is not available, we explore the possibility of generating a synthetic data set to be used during the training stage of the process.

The use of synthetic data to allow application of machine learning techniques is not a new idea and has been applied as early as 2004 to supplement survey data from non-respondents [26]. Since then, synthetic data generation methods have been developed for various domains, like the healthcare system [27]. The need for synthetic data sets for the construction domain has also been recognized in previous work. For example, [28] enhances a small existing data set with synthetic data to train a computer vision based system for monitoring the movement of construction workers on site. Based on the results of their work, the predictive model trained on the enhanced data set performed better than the model trained on only real data.

Although the construction domain produces vast amount of data, this data is currently compartmentalized and not accessible, or accessible but not complete making it unsuitable for ML applications for specific tasks. Furthermore, when dealing with ACC, the majority of available design documents are of buildings that have already received permit approval thus they are all code compliant and not sufficient to train a supervised ML algorithm. We assume that the lack of data is often a barrier to explore the potential of implementing ML for various purposes. To overcome this barrier, the training stage of the suggested process relies only on a synthetically generated data set. Fully synthetic data sets for training ML models are becoming increasingly popular for dealing with lack of data, especially in domains where privacy and data protection issues are dominant [29]. As data sharing is a problem in

the AEC domain as well, we aim to investigate the capabilities of completely synthetic data sets to serve as starting points for training ML models for the use of the AEC industry. Similarly to the approach for generating synthetic data based on unprocessed "real" data [30], we rely on real floor plans of buildings that are publically available as a baseline for the data generation procedure as described in section 4. We then focus on examining the performance of a GNN model that has been trained on a synthetic data set for classifying BIM models received from the industry as compliant or not compliant to a specific code requirement.

3 Research aims and method

The main purpose of this research is to demonstrate the initial feasibility of applying GNNs to ACC. We do that by illustrating the proposed workflow of ACC supported by application of a Graph Attention Network (GAT) model on a small-scale problem from the world of design review. Within that, the presented test case will also illustrate the applicability of ML models that have been trained on completely synthetic data sets for predictive analytics tasks performed on real design received from the industry. The overall suggested workflow for implementing GNNs trained on synthetic data as the checking mechanism for ACC is illustrated in Fig. 2. The training stage in the proposed workflow is implemented using the synthetically generated data set, which produces a trained classifier to be used for prediction. A "prediction" in this case is the result of code compliance checking of a new "real world" design.

We demonstrate the process through a small scale test case of checking the compliance of single family houses to several accessibility requirements based on the International Building Code [31]. The requirements to be checked are the minimal width of spaces, doors and ramps, the allowed slope of ramps and the general "ability to access". Since the chosen regulations address both geometric and topological aspects of the design, the strength of implementing graph based learning instead of "classic" ML approach can be explored.

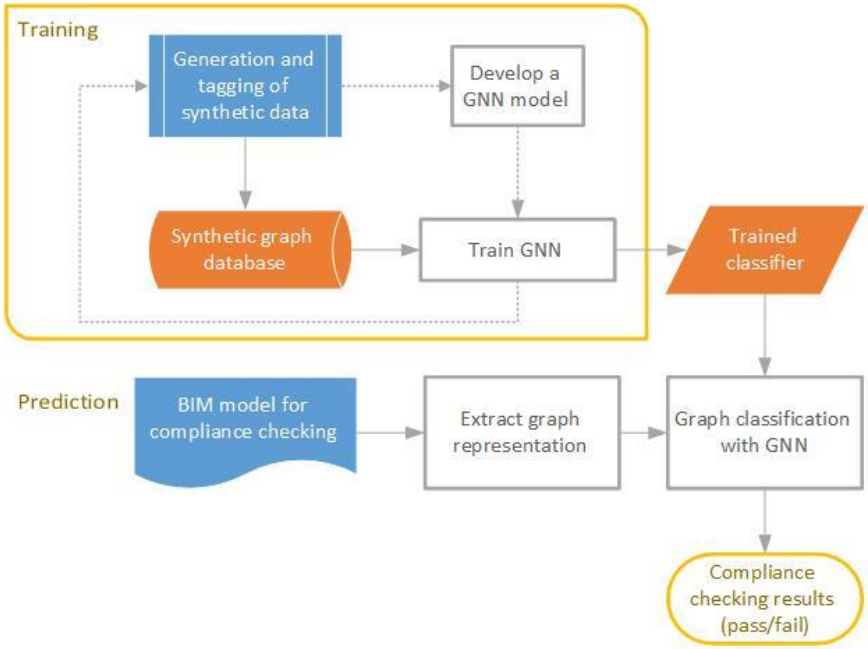


Fig. 2. Suggested process for GNN based ACC

4 GNN for ACC – test case

To illustrate the suggested workflow, we choose a small but representative test case of checking single family houses for some basic accessibility regulations. The checked requirements are the minimum width of doors, corridors, ramps and ramp slope, and the general ability to access each of the spaces. Although residential houses are usually not required to be accessible, unless in some special situations, this test case was chosen due to the simplicity on one hand and the ability to demonstrate the influence of geometry and topology together on the other hand. In addition, since the data for training is synthetically generated, it is important that the task is such that allows the use of a fully automated routine for labeling the entire generated data set.

While some construction regulations deal with simple geometric requirements that are concerned with specific building elements (the size of a window, the slope of a ramp etc.), others describe restrictions based on topologically complex dependencies between various building elements. Accessibility, or "the ability to access" is a requirement that encapsulates both geometric and topological aspects. Namely, for a room to be considered accessible, it is not sufficient that the room complies to all the geometric requirements, as we must also provide the ability to access the space meaning that all the spaces, doors, ramps that lead to that space must be accessible as well. The fact that we must look at the room in the context of the entire building to decide whether it is accessible or not, aligns with graph based learning

models where we look at every node in the context of the graph to learn the class of the node.

4.1 Train, validate and test – synthetic data

Following the suggested workflow, as illustrated in Fig. 2, a set of 1,000 graphs representing single family houses were generated and labeled. As it is often believed that synthetic data sets must be based on real data, we begin the data generation process by collecting 10 floor plans of single family homes that are publically available on the web. The floor plans are manually translated to graph representations and serve as a baseline for generating floor plan variations. Information represented in the graphs includes only objects and attributes that are relevant to accessibility checking, i.e. spaces and their size, doors and their width, ramps and the slope of ramps, stairs. The Labeled Property Graphs therefore contain nodes that represent building elements which are assigned with properties such as element type, size etc. Edges between nodes represent navigable connections between the aforementioned objects, linking a door and its adjacent spaces, for example. By implementing a random number generator in a predefined restricted range for each of the properties, we create floor plan variations based on the baseline. Each baseline floor plan is modified 100 times which leads to 100 graphs that represent geometrically different floor plans. The topologies on the other hand remain unchanged in each of the 10 baseline floor plans, in order to ensure that we maintain topological integrity and generate graphs that represent feasible buildings. Applying the random number generators to each of the baseline floor plans we generate 1,000 graphs each representing one variation of a single-family house.

In order to train a GNN model, we label each of the nodes in the graphs based on their conformance to the chosen code provision. Labeling is performed in two stages, where the first stage is a deterministic check of the geometric requirements for each of the individual objects. For example, based on the code requirements the slopes of ramps must be within the range of 5-8.3% [31]. The results of this first stage are initial labels for each node of "pass" if the geometric requirements are met, and "fail" otherwise. In the second stage, we search for all possible paths leading from the entrance to the house to every space to check the "general ability to access". Namely, a space will be considered accessible only if there is a path leading to it which consists of other geometrically accessible elements. Eventually, the labeling routine aims to classify each node in the graph to three classes:

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.

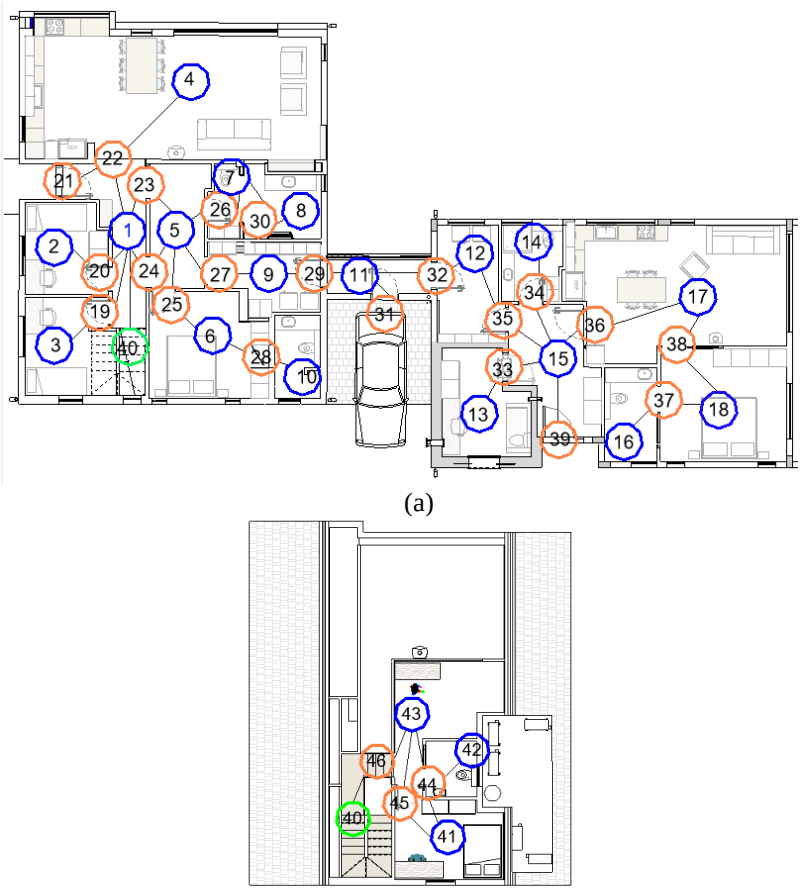
c) Not compliant – for elements that do not satisfy the geometric requirements of the accessibility code.

Once the data set was generated and labeled, Graph Attention Network (GAT) model was trained in a full batch mode using the generated 1,000 graphs containing 28,400 nodes and 27,900 edges. The final model for training contained 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. Evaluating the performance of the model, the data set was randomly split to data for training, data for validation and data for testing. Performance of the model was evaluated using the F1 score calculated based on the test set. The obtained F1 score was 0.86 which indicates the obtained classifier performs well on unseen data. As the validation and testing data sets are portions of the generated synthetic data set, to validate the results we must test the performance of the obtained classifier on "real world" data. The focus of the rest of the paper is the application of the trained classifier to check the compliance of design documents obtained from the industry to the accessibility code requirements.

4.2 Check compliance – real design data

To evaluate the feasibility of the entire workflow (Fig. 2), the obtained trained classifier must be applied to make predictions (classifications) based on design data provided from the industry. The following section describes the application of the trained classifier to check compliance of a BIM model that was obtained from a local architectural firm is Israel. The floor plans of the house, overlaid with their graph representation, are illustrated in Fig. 3. This design contains a main house which has two levels, and it is connected to an independent rental unit, which is very common in Israel. Note that this is a slightly different topology than in most of the houses used in the training set. The entire training set was defined based on the topologies of single-family houses mostly with a single level. While there is a minority of graphs representing houses with more than one level, there is no representation in the training data of houses connected to an independent unit that is also accessible from the main house. ML models are designed to generalize to new entities that are not present in the training data. Using this test case, we can begin to explore the flexibility that graph based learning models provide in terms of being able to generalize and provide classifications for buildings with various topologies.

The graph representation of the house (both levels) is given in Fig. 4. It contains all the rooms, doors and stairs represented as nodes, and the topological relationships between them are represented as edges. The only topological relationships represented in the graph are "access" relationships, meaning there is an edge between two nodes only if they represent elements with direct accessibility between them. The goal of this stage is to classify each building element represented in the graph as 'Not compliant', 'Compliant and accessible', 'Compliant but not accessible'. It is important to note that residential buildings built for the private sector usually do not have to be accessible. Hence, the ground truth contains elements of all three possible labels.



(a)

(b)

Fig. 3. Floor plan of the ground level of a house used as validation for accessibility check (design by Arch. Odelya Bar-Yehuda). (a) the ground level including the main house and the independent dwelling unit, (b) is the second storey of the main house.

The nodes in the graph are assigned with a list of features to describe the elements which they represent, using the same data structure as for the training stage. Overall, nine categorical features are assigned to each node as listed in Table 1. Features F1, F2, F3 and F4 determine the function of the node (space, door, stairs or ramp). F5, F6, F7 determine the minimal width of the component. For example, F5 will be assigned with the value 1 if the minimal width is greater than 170 cm, and the value 0 otherwise. F8 determines whether a space is a functional room such as kitchen,

bathroom, bedroom, etc. or it is part of the circulation area within the house, such as a corridor. F9 determines the slope of ramps, such as that it is assigned with the value 1 if the slope is between 5-8.3%, and 0 otherwise. The list of features was determined based on key values from the accessibility code that were mapped into categories with numeric values. An example of a feature vector assigned to node 11 is illustrated in Fig. 4. The final graph representing this house contains total of 46 nodes and 46 edges.

Table 1. List of features assigned to each node in the graph

Feature	0	1
F1	If the element is a space	For all other elements
F2	If the element is a door	For all other elements
F3	If the element is a stair	For all other elements
F4	If the element is a ramp	For all other elements
F5	If the width of the element is greater than 170 cm	Otherwise
F6	If the width of the element is greater than 91.5 cm	Otherwise
F7	If the width of the element is greater than 81.5 cm	Otherwise
F8	If the element is a space that is part of the circulation path	For all other elements
F9	If the element is a ramp and its slope is within the range of 5-8.3%	Otherwise

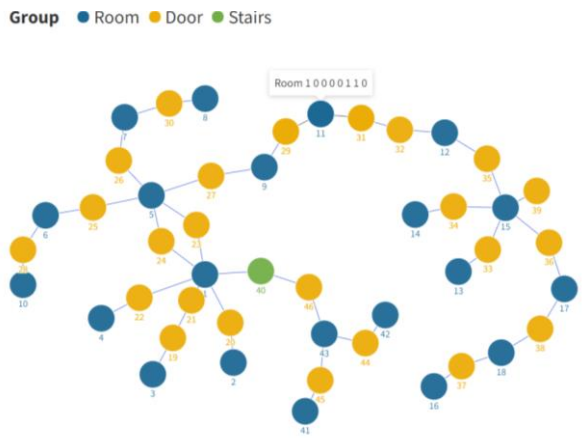


Fig. 4. The graph representation of the house plans used for validation. Each node in the graph is assigned with a vector of features (illustrated only for node 11).

5 Results

In this work we implement the inductive learning setting [32], meaning that during the training only the training data is available (synthetic data) and we apply the trained classifier on a dataset which the model has never encountered before (real design documents). Overall, out of 46 entities in the graph that represents the real test case, 8 entities were misclassified resulting in an accuracy of classification of 82%. Comparing these results to the performance metrics of the trained model, we see a small deviation as the accuracy of the test set during training was 86%. Table 2 presents the misclassified nodes, their location in the floor plan, the predicted label and the result of a manual compliance check (ground truth).

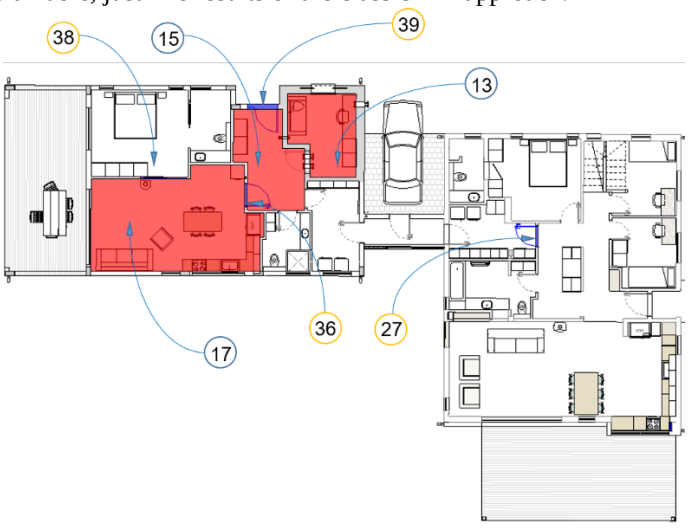
We can clearly see that most of the misclassified nodes in this case represent entities in the rental unit portion of the floor plan and not the main house. We can also see that most misclassified nodes are classified correctly in terms of geometry but not in terms of topology. In other words, spaces or doors that are compliant to the geometric requirements were indeed classified as such, but instead of being labeled "Compliant and accessible" they were classified "Compliant but not accessible", suggesting there is a problem with the path leading to those elements but not the elements themselves. We can also see that 100% of the mistakes are false negatives, meaning that relying on these results would lead to a reevaluation of the floor plan by the designers and not cause problems in later stages of the project.

Table 2. List of misclassified entities as result from using the classifier trained on synthetic data

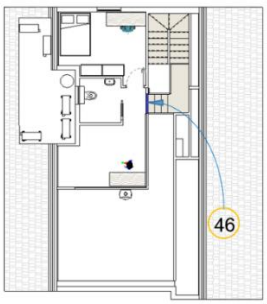
Room/Door	Location	Node number	Predicted label	True label
Door	Main house	27	Compliant but not accessible	Compliant and accessible
Door	Main house – second floor	46	Not compliant	Compliant but not accessible
Room – Security room	Rental unit	13	Compliant but not accessible	Compliant and accessible
Room - Foyer	Rental unit	15	Compliant but not accessible	Compliant and accessible
Room – Living room	Rental unit	17	Compliant but not accessible	Compliant and accessible

Door- exit door	Rental unit	39	Compliant but not accessible	Compliant and accessible
Door	Rental unit	36	Compliant but not accessible	Compliant and accessible
Door	Rental unit	38	Compliant but not accessible	Compliant and accessible

The misclassified elements are marked in Fig 5 a and b below. Although we can assume that the major cause for the misclassification is the fact that this type of topology is not well represented in the training set, the results obtained with GNN are unexplainable, just like results of the classic ML approach.



a) Ground level



b) Second level

Fig. 5 Floor plans of the checked building with marked misclassifications

6 Discussion

Automated Code Checking has been a subject of interest for many researchers over the years. Moving to BIM technology, we were able to make significant progress in the field, and lead to the development of advanced and sophisticated platforms with the ability to automatically check the compliance of a given design to several regulatory documents or user requirements. Despite the sophistication of the existing tools and workflows, an automated platform that provides a checking routine for a wide range of regulations in a completely automated manner remains a distant goal.

As the majority of work on the subject is focused on further development of the ACC following the well-established rule-based approach, we are constantly making progress but do not reach major breakthroughs. The broad applicability of ML techniques led to various breakthroughs in many different domains. The potential of using ML and leveraging data in the construction domain has been long recognized as well. However, the idea of applying ML techniques for ACC has not been sufficiently examined. We cannot expect for ML based ACC to reach the same accuracy as rule-based checking since results obtained with ML are probabilistic. However, while the rule-based approach provides very reliable results, it is limited in scope and requires much manual processing for rule compilation and for building information extraction.

There are some major differences between the rule-based approaches for ACC to the ML based approach for ACC. One of them is that the rule-based approach requires to process the regulations and the building design to bring them to a common environment as depicted in the left side of Fig. 6. Still, usually that representation of the regulatory documents and the design concepts do not overlap sufficiently, which causes difficulties in development of ACC platforms that cover a wide range of regulations. As described in [5], checking of a given design requires the user to "correct" the model to match the requirements of the checking routine in a process commonly called 'normalization'.

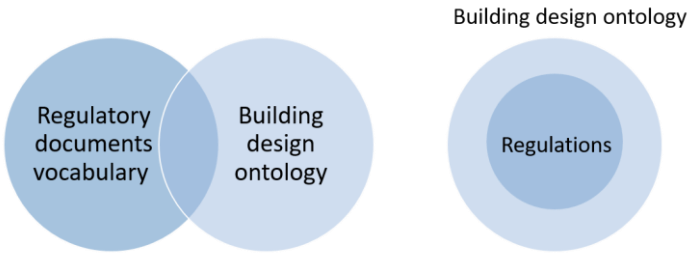


Fig. 6. Two approaches for ACC: on the left-hand side representation of the regulations and the design as separate ontologies. On the right-hand side representation of design and regulation using the same data structure

In the ML based approach, we still look for a common data representation, in the case of GNN it is LPG, but the regulations are encapsulated within that representation by the labels assigned to the nodes during the training (right side of Fig 6). In other words, there are no two separate ontologies or vocabularies, instead both the design and the regulations are represented on the same graph which can be a great benefit of the approach.

On the other hand, representing the regulations and the design on a single data structure can also be a drawback. Regulations are often revised and changed, although the changes are not usually drastic, they have to be integrated correctly with the training process. This means that changes in the regulations will require re training, and possibly relabeling of the training set to obtain new classifiers. The complexity of a process for relabeling and retraining has not been evaluated yet. Similarly, the ability to group regulations into a single classifier is also an issue that needs to be investigated. In the presented work, several geometric requirements for doors, spaces and ramps were checked simultaneously in addition to the "general ability to access". This suggests that not every code clause will require developments of its own trained classifier. A wider analysis of the regulations is needed to identify the clauses that can benefit from the GNN approach and to develop strategies for grouping code requirements that can be dealt with by the same classifier.

One of the barriers for implementing classic ML techniques or graph-based learning techniques is that both require large data sets for training. Although the construction industry produces vast amounts of data in all stages of construction projects, most of this data is not public. In addition, data that is available is not always complete, represented in a compatible format and labeled. Although the idea of generating synthetic data for training of ML models is not a new concept, this work illustrates that it can be useful for ACC. Furthermore, this work demonstrates that a graph-based learning model that was trained solely on synthetic data is applicable for checking the compliance of design documents obtained from the industry to check several accessibility requirements.

This work is focused on validating the suggested process of a GNN based ACC. Demonstration of the process for accessibility checking in a residential building leads to very encouraging results, but also reveals numerous directions for needed research. The GAT model developed in this case, and the routine for data generation and labeling are goal driven. This means that it might be difficult to generalize the created data set to be used for other purposes. In this case, we only represent the building elements relevant to the specific code requirement we want to check, but we assume it is possible to use more detailed graph representations of buildings to be able to check a larger set of regulations. This will of course influence the labeling method as well since we aim at providing labels that indicate what the design issue is instead of simple 'compliant' or 'not compliant'. We can also assume that variations in the GNN model architecture, the data representation and the attributes assigned to every node will significantly influence the performance of the model. Although a rule-based approach can reach 100% accuracy in checking the same requirements, we cannot expect the same performance from a ML based approach. However, the obtained results demonstrate that GNNs can be applied to problems

from the code checking domain, and they have the potential to provide a possible solution for regulations that cannot be checked deterministically. For example, vaguely written regulations or performance-based regulations that are difficult to represent as rigid rules. In conclusion, this work illustrates that GNNs are applicable for ACC, and that determining their scope and boundary conditions is a valid and important direction for future research.

7 Conclusions

This paper demonstrates a suggested workflow for implementing GNNs for ACC while relying on a synthetic data set for training and making prediction on BIM models received from the industry. The workflow is illustrated through a small-scale problem of checking compliance to accessibility requirements. The accuracy of the trained model applied to a test set achieved 86%, suggesting the classifier performs well on unseen data. Using the trained model to classify building elements presented in a BIM model received from the industry achieved accuracy of 82%.

This work has two main contributions, one is the feasibility check for using GNNs to automate compliance checking of code requirements that incapsulate both geometric aspects and topological aspects of the design. The other is a demonstration that in fact, synthetic data sets can be useful for training models that will later be used for classification of real design information.

The possible potential of using ML (whether classic ML algorithms or graph-based algorithms) has been long recognized. However, one of the main drawbacks is usually the unavailable data set for training. The construction industry produces large amounts of data in every construction project, but it is unfortunately not always available for researchers. Relying on synthetic data, we are able to illustrate the potential use and benefit of data driven approaches for ACC.

References

[1] C. Eastman, J. Lee, Y. Jeong, and J. Lee, "Automatic rule-based checking of building designs," *Automation in construction*, vol. 18, no. 8, pp. 1011–1033, 2009, doi: https://doi.org/10.1016/j.autcon.2009.07.002.

[2] J. Fauth and L. Soibelman, "Conceptual Framework for Building Permit Process Modeling: Lessons Learned from a Comparison between Germany and the United States regarding the As-Is Building Permit Processes," *Buildings*, vol. 12, no. 5, p. 638, May 2022, doi: 10.3390/buildings12050638.

[3] R. Amor and J. Dimyadi, "The promise of automated compliance checking," *Developments in the Built Environment*, vol. 5, p. 100039, Mar. 2021, doi: 10.1016/j.dibe.2020.100039.

[4] Solibri, "Solibri Model Checker (SMC)," Mar. 13, 2017. https://www.solibri.com/ (accessed Mar. 13, 2017).

[5] T. Bloch and R. Sacks, “Clustering Information Types for Semantic Enrichment of Building Information Models to Support Automated Code Compliance Checking,” *Journal of Computing in Civil Engineering*, vol. 34, no. 6, p. 04020040, 2020.

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

[7] S. Malsane, J. Matthews, S. Lockley, P. E. D. Love, and D. Greenwood, “Development of an object model for automated compliance checking,” *Automation in Construction*, vol. 49, pp. 51–58, Jan. 2015, doi: 10.1016/j.autcon.2014.10.004.

[8] T. Bloch, M. Katz, and R. Sacks, “Machine learning approach for automated code compliance checking,” presented at the 17th International Conference on Computing in Civil and Building Engineering, Tampere, 7/6 2018.

[9] R. Zhang and N. El-Gohary, “Hierarchical Representation and Deep Learning–Based Method for Automatically Transforming Textual Building Codes into Semantic Computable Requirements,” *J. Comput. Civ. Eng.*, vol. 36, no. 5, p. 04022022, Sep. 2022, doi: 10.1061/(ASCE)CP.1943-5487.0001014.

[10] R. Sacks, T. Bloch, M. Katz, and R. Yosef, “Automating Design Review with Artificial Intelligence and BIM: State of the Art and Research Framework,” in *Computing in Civil Engineering 2019*, Atlanta, Georgia, Jun. 2019, pp. 353–360. doi: 10.1061/9780784482421.045.

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

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

[13] J. Zhou *et al.*, “Graph neural networks: A review of methods and applications,” *arXiv preprint arXiv:1812.08434*, 2018.

[14] R. Angles, “The Property Graph Database Model.,” in *AMW*, 2018.

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

[16] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric deep learning: going beyond euclidean data,” *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[17] W. L. Hamilton, R. Ying, and J. Leskovec, “Representation Learning on Graphs: Methods and Applications,” *CoRR*, vol. abs/1709.05584, 2017, [Online]. Available: http://arxiv.org/abs/1709.05584

[18] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, “A comprehensive survey on graph neural networks,” *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[19] J. Zhou *et al.*, “Graph neural networks: A review of methods and applications,” *AI Open*, vol. 1, pp. 57–81, 2020, doi: 10.1016/j.aiopen.2021.01.001.

[20] F. Collins, “Encoding of geometric shapes from Building Information Modeling (BIM) using graph neural networks,” 2020.
[21] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.
[22] Z. Wang, R. Sacks, and T. Yeung, “Exploring graph neural networks for semantic enrichment: Room type classification,” *Automation in Construction*, vol. 134, p. 104039, Feb. 2022, doi: 10.1016/j.autcon.2021.104039.
[23] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *Advances in neural information processing systems*, vol. 30, 2017.
[24] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.
[25] C. Data61, “StellarGraph Machine Learning Library,” *GitHub Repository*. GitHub, 2018. [Online]. Available: https://github.com/stellargraph/stellargraph
[26] D. B. Rubin, *Multiple imputation for nonresponse in surveys*, vol. 81. John Wiley & Sons, 2004.
[27] J. Dahmen and D. Cook, “SynSys: A Synthetic Data Generation System for Healthcare Applications,” *Sensors*, vol. 19, no. 5, p. 1181, Mar. 2019, doi: 10.3390/s19051181.
[28] M. Neuhausen, P. Herbers, and M. König, “Using Synthetic Data to Improve and Evaluate the Tracking Performance of Construction Workers on Site,” *Applied Sciences*, vol. 10, no. 14, p. 4948, Jul. 2020, doi: 10.3390/app10144948.
[29] F. K. Dankar and M. Ibrahim, “Fake It Till You Make It: Guidelines for Effective Synthetic Data Generation,” *Applied Sciences*, vol. 11, no. 5, p. 2158, Feb. 2021, doi: 10.3390/app11052158.
[30] N. Patki, R. Wedge, and K. Veeramachaneni, “The Synthetic Data Vault,” in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Montreal, QC, Canada, Oct. 2016, pp. 399–410. doi: 10.1109/DSAA.2016.49.
[31] International Code Council and American National Standards Institute, Eds., *Accessible and usable buildings and facilities: ICC A117.1-2009: American National Standard*. Washington, DC: International Code Council, 2010.
[32] G. Ciano, A. Rossi, M. Bianchini, and F. Scarselli, “On Inductive–Transductive Learning With Graph Neural Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 2, pp. 758–769, Feb. 2022, doi: 10.1109/TPAMI.2021.3054304.