Technische Universität München

TUM School of Engineering and Design

Lehrstuhl für Computergestützte Modellierung und Simulation

Automated BIM Model Annotation via Graph Neural Networks: Bridging the Gap Between Design and Documentation

Master Thesis

for the Master of Science Program in Civil Engineering

Author: Server Çeter

Matrikelnummer:

Supervisor: Prof. Dr.-Ing. André Borrmann

Dr. Stavros Nousias

M.Sc. Changyu Du

Submission date: 01. May 2024

Acknowledgements

I would like to express my deepest gratitude to Dr. Stavros Nousias and Changyu Du for their guidance and mentorship throughout my thesis. Their expertise has been instrumental in helping me navigate unfamiliar topics and harness my potential in areas where I previously lacked experience. I am immensely grateful for their patience in answering all my questions.

I would also like to extend my heartfelt thanks to my father, Abdulhakim Çeter, and my mother, Habibe Çeter, for their boundless love and sacrifices they have made to provide everything necessary for me and my siblings to pursue and achieve our dreams. Their constant belief in my abilities has been the driving force behind my academic journey, and I am forever grateful for their unwavering support.

Lastly, I am indebted to my sisters, brother and my friends, whose encouragement and understanding have been a source of strength and motivation throughout this journey.

Abstract

The production of annotated technical drawing in the Architecture, Engineering, and Construction (AEC) industry requires much work, necessitating the adoption of tools that can limit the workload especially with the advent of artificial intelligence (AI). This thesis sought to revolutionize architectural documentation by using Graph Neural Networks (GNNs) as a step toward automating this process. The overarching goal is to automatically generate annotations for Building Information Modeling (BIM) designs, sparing architects from labor-intensive tasks.

This thesis proposes a method to represent BIM models as graphs and classify nodes using GNNs thus predicting interconnectivity of elements. For this purpose, a method has been developed in the BIM authoring software ArchiCAD by introducing a plugin, named ServCAD utilizing C++ API. The information gathered from BIM models is streamlined and is essential for the next steps. The extracted data are transformed into detailed graphs that capture architectural elements' semantic and spatial aspects. These graphs serve as the GNN model's foundational data set. The objective is to evaluate the efficacy and accuracy of GNN in automating the annotation process, specifically in label type prediction based on node classification, thorough testing and evaluation.

In summary, the prediction results for different types of elements (walls, zones, doors) were provided via Graph Convolutional Network (GCN) and Graph Attention Network (GAT). Both models demonstrated exceptional performance in classifying certain label types. Notably, GAT model achieved accuracy rates of over 90% for labels '0', '2', and '3' (GCN was also above 80 %), with label '2' achieving a perfect accuracy of 100%. However, there were discrepancies in the accuracy rates for labels '1' and '4'. The GCN outperformed the GAT model in accurately predicting these labels. Later, the prediction results were examined and mapped on the example projects having different complexities to complete the automatic annotation process by using ServCAD.

Keywords: Architectural design, ArchiCAD, Artificial intelligence, Automated annotation, BIM, C++ API, Graph neural netwoks, Machine learning, PyTorch, Spatial relationships

Zusammenfassung

Die Erstellung kommentierter technischer Zeichnungen in der Architektur-, Ingenieur- und Baubranche (AEC) erfordert viel Arbeit und erfordert den Einsatz von Tools, die den Arbeitsaufwand insbesondere mit dem Aufkommen künstlicher Intelligenz (KI) begrenzen können. Ziel dieser Arbeit war es, die Architekturdokumentation durch den Einsatz von Graph Neural Networks (GNNs) als Schritt zur Automatisierung dieses Prozesses zu revolutionieren. Das übergeordnete Ziel besteht darin, automatisch Anmerkungen für Building Information Modeling (BIM)-Entwürfe zu generieren und Architekten arbeitsintensive Aufgaben zu ersparen.

Diese Arbeit schlägt eine Methode vor, um BIM-Modelle als Diagramme darzustellen und Knoten mithilfe von GNNs zu klassifizieren und so die Interkonnektivität von Elementen vorherzusagen. Zu diesem Zweck wurde in der BIM-Authoring-Software ArchiCAD eine Methode entwickelt, indem ein Plugin namens ServCAD unter Verwendung der C++ API eingeführt wurde. Die aus BIM-Modellen gesammelten Informationen werden optimiert und sind für die nächsten Schritte unerlässlich. Die extrahierten Daten werden in detaillierte Diagramme umgewandelt, die die semantischen und räumlichen Aspekte architektonischer Elemente erfassen. Diese Diagramme dienen als grundlegender Datensatz des GNN-Modells. Das Ziel besteht darin, die Wirksamkeit und Genauigkeit von GNN bei der Automatisierung des Annotationsprozesses zu bewerten, insbesondere bei der Vorhersage des Etikettentyps basierend auf der Knotenklassifizierung, gründlichen Tests und der Bewertung.

Zusammenfassend lässt sich sagen, dass die Vorhersageergebnisse für verschiedene Arten von Elementen (Wände, Zonen, Türen) über das Graph Convolutional Network (GCN) und das Graph Attention Network (GAT) bereitgestellt wurden. Beide Modelle zeigten eine außergewöhnliche Leistung bei der Klassifizierung bestimmter Etikettentypen. Bemerkenswert ist, dass das GAT Modell Genauigkeitsraten von über 90 % für die Labels '0', '2' und '3' erreichte (GCN lag ebenfalls über 80 %), wobei Label '2' eine perfekte Genauigkeit von 100 % erreichte. Allerdings gab es Unterschiede in den Genauigkeitsraten für die Labels '1' und '4'. Das GCN übertraf das GAT Modell bei der genauen Vorhersage dieser Etiketten. Später wurden die Vorhersageergebnisse untersucht und auf Beispielprojekte unterschiedlicher Komplexität abgebildet, um den automatischen Annotationsprozess mithilfe von ServCAD abzuschließen.

Stichworte: Architekturdesign, ArchiCAD, Künstliche Intelligenz, Automatisierte Annotation, BIM, C++ API, Graphische neuronale Netzwerke, Maschinelles Lernen, PyTorch, Raumbeziehungen

List of Contents

List of Abbreviations

2D	Two dimensional
3D	Three dimensional
ADP	Automatic Dimensioning Placement
AEC	Architectural Engineering and Construction
AI	Artificial Intelligence
API	Application Programming Interface
BIM	Building Information Modeling
B-Rep	Boundary Representation
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CNN	Convolutional Neural Network
CSG	Constructive Solid Geometry
DL	Deep Learning
D&T	Dimensioning and Tolerancing
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
LCS	Longest Common Subsequence
LR	Learning rate
NN	Neural Network
SFAG	Structured Face-Adjacency Graph
SOTA	State of the Art

List of Figures

List of Tables

1 Introduction

1.1 Motivation

In the AEC sector, architectural drawing serve as the fundamental means of communication, facilitating the translation of conceptual design concepts into tangible constructions. Estimates suggest that technical drawing still play a critical role in data transfer, making up 74% of the process, while three dimensional (3D) models with Model-Based Definition (MBD) are the main information source in only 26% of cases (Ruemler et al., 2017). Beyond making it easier to share and preserve design solutions, architectural drawings are very important. In addition, they help with cost calculation, procurement of raw materials, component manufacturing, quality assurance, and product assembly (Drake, 1999). Despite being a necessary component of modern architecture practices, BIM can still be improved by utilizing cutting-edge technology like AI to expand its capabilities (Zabin et al., 2022). Building and infrastructure project managers, engineers, and architects can collaborate more successfully to design, visualize, simulate, and manage projects using AI tools as part of the BIM.

Particularly in the complicated world of subterranean construction, the integration of BIM with machine learning and computer vision offers hitherto untapped opportunities to improve project visibility, reliability, and management (Huang et al., 2021). Better decision-making and automation in workflows for architectural design and construction are made possible by the incorporation of AI into BIM processes, which opens up new opportunities for creativity and efficiency.

The adoption of BIM in the AEC sector provides a rich environment for applying machine learning techniques because of the massive amount of data generated in BIM-based projects. This prospect highlights the research's emphasis on identifying common application areas, emerging trends within this context and difficulties in implementing and adopting machine learning in the AEC sector (Zabin et al., 2022). This inquiry reveals a fresh, rapidly evolving industry where machine learning is just now starting to show its potential for automating processes and extracting insights from BIM data. Furthermore, BIM has been widely adopted in many different regions due in part to government rules that favor its use (Smith, 2014). The abundance of data in the BIM shared information model ushers in a new digital approach by encouraging

collaboration in tracking and managing an asset's whole lifecycle at any point. In addition to representation and scheduling, this method has numerous applications in planning, estimating, simulation, and project management (Ciribini et al., 2016; Solihin et al., 2016). Because it follows a structured scheme, this abundance of data functions as a centralized, machine-interpretable repository (Zabin et al., 2022).

Moreover, the domains of AI and machine learning have generated unprecedented prospects for the AEC industry. Thanks to AI, people with varying backgrounds may now develop and create solutions that have the power to drastically alter established behaviors. The democratization of technology has enabled stakeholders in the AEC sector to create and implement cost-effective, time and resource efficient solutions. AI has the capacity to transform the architectural drawing and construction sector significantly. This includes automating annotation processes to predictive modeling and optimization algorithms.

The automatic annotation of existing methods such as rule-based approaches is often error-prone. It requires manual correction which is slow, and costly and results in slowing down the project. To this end, machine learning and data-driven methods can be employed to speed up the process. Since BIM models are highly linked to graph structures, Graph Neural Networks (GNNs) were mostly investigated to support this process.

A typical floor plan with basic architectural elements such as walls, doors, and zones along with some annotation elements like dimension, door label, and zone marker is shown in Figure 1.1. Even though this sample project seems simple, it is essential to record all the relationships between the components and build a graph based on these relationships in order to provide data for the machine learning portion of the thesis. Accurately representing these relationships becomes more and more necessary as architectural drawings become more complex and nuanced.

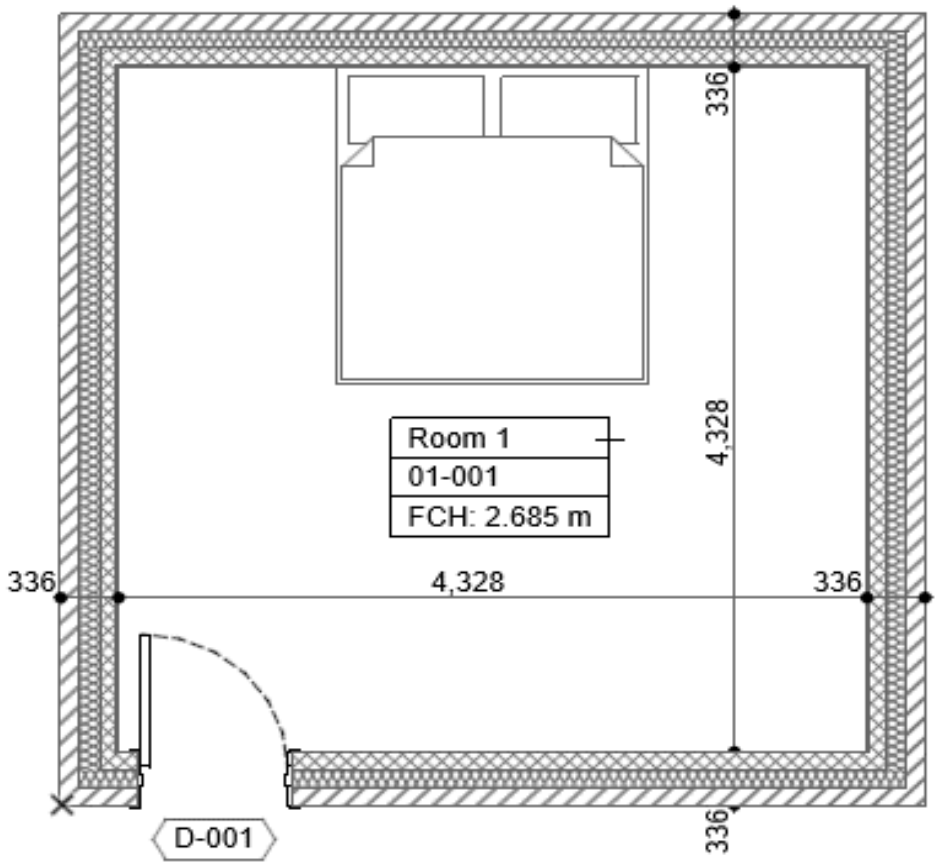


Figure 1.1: Example of a simple room modelled in BIM authoring software, ArchiCAD

ServCAD is being developed as part of this master's thesis supporting automatic annotation processes in architectural drawings. Our two goals are to: (1) automatically produce annotations for the architectural elements shown in Figure 1.1; and (2) develop a scalable and adaptable framework that can be used or improved for more complex architectural designs. The objective is to reduce the workload associated with manual annotation and optimize the annotation workflow, especially for projects involving complex and large-scale architectural designs.

On the other hand, Figure 1.2's floor plan illustrates the difficulties that arise with more complex architectural drawings as mentioned previously. This leads to the investigation of GNNs for predicting the proper label types for elements as shown in Figure 1.1. GNNs can efficiently analyze and interpret detailed architectural drawings by capturing spatial and semantic relationships which makes it possible for architects to use this information to annotate the model. However, ruled-based automatic annotation in the ArchiCAD cannot take this dynamic information into account. Consequently, using predicted label types from GNN, application programming interfaces (APIs) can be used to create automatic annotation, which will speed up the annotation process and increase efficiency in the AEC industry.

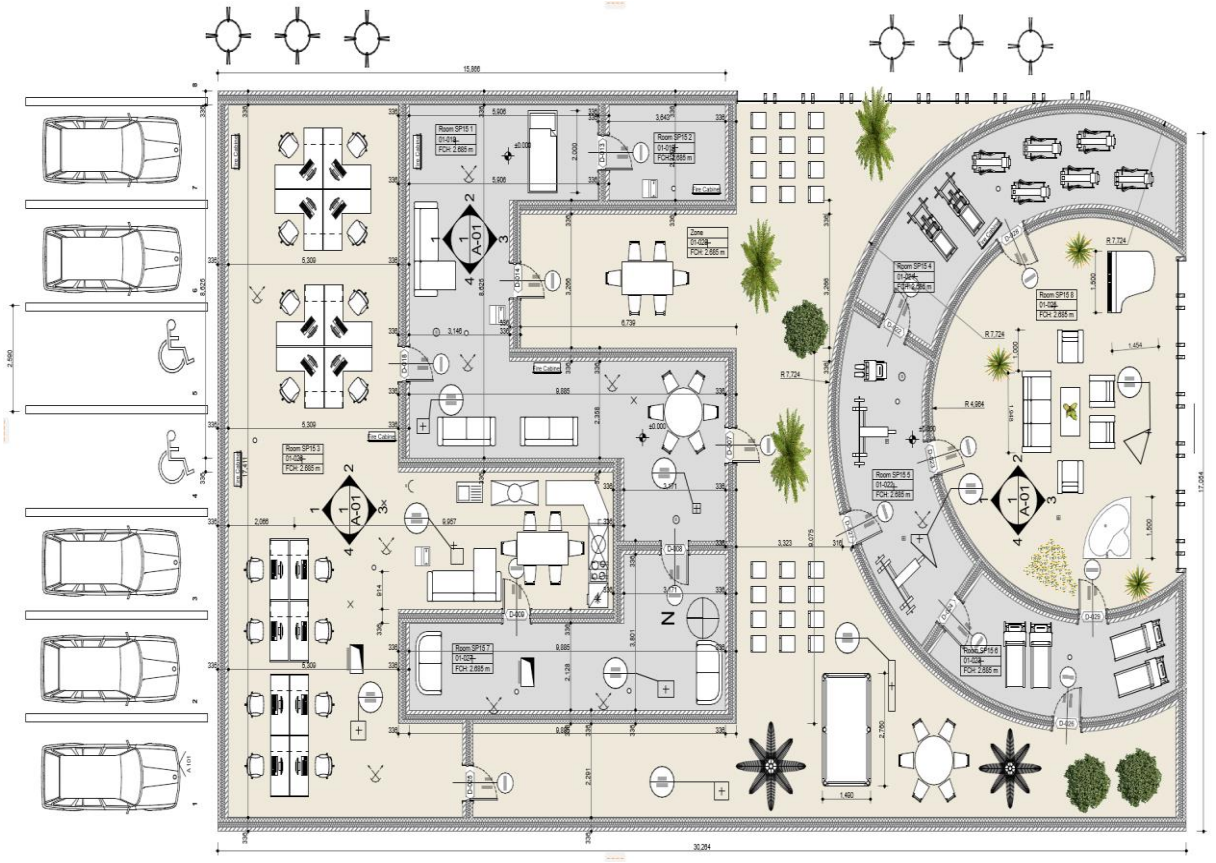


Figure 1.2: Floor plan of an example drawing, reflecting certain degree of complexity.

This introduction lays out the background of the study, outlining the difficulties in interpreting architectural drawings and the reasons for investigating GNNs for automated annotation. The approach, findings, and recommendations from this research project will be covered in detail in the upcoming chapters. This thesis aims to evolve annotation workflows in the AEC industry and open the door to more productive and successful architectural design procedures.

1.2 Research objectives

The following questions are the focus of the research:

- Can we use the BIM authoring tool (ArchiCAD C++ API) to retrieve building information, dimensional properties and annotation objects?
- How can the label classification procedure be automated during the data generation phase?
- How can we derive the connection between annotations and building elements?
- How can we use machine learning (specifically GNN) to predict annotation elements?

- How can automated annotation creation be facilitated by integrating GNN predictions back into the architectural environment (specifically ArchiCAD)?

1.3 Reading guide

The rest of the thesis follows the below chapter structure:

- Chapter 2 examines various technologies and the current advancements in automation and GNN research. It also explores theoretical frameworks for automated annotation and dimensions in the AEC sector.
- Chapter 3 explores the latest methods and technology advancements in automated BIM model annotation, utilizing geometric characteristics and machine learning algorithm and APIs.
- Chapter 4 describes the workflow created for this study, which includes using ServCAD to create graphs from BIM models, visualizing the data that is produced, and using these graphs to train the GNN architecture. It also explains how to use ArchiCAD APIs to map GNN predictions back to the BIM models.
- In Chapter 5, the efficacy of GNN models on the task of node label prediction on the custom graph dataset is evaluated by examining different model architecture and hyperparameters.
- In Chapter 6, the findings are discussed in relation to the research topics, with a focus on the study's contributions to the BIM community, the limitations of the current approach, and future areas for investigation and optimization.
- Chapter 7 presents a summary of the study's key conclusions and an appraisal of the practicality and effectiveness of employing GNNs for automated BIM model annotation. It also suggests future research directions.

2 Preliminaries and theoretical background

The section examines the evolution of automated dimensioning and annotation in CAD and BIM systems, highlighting the shift from early feature-based methods to the integration of knowledge-based systems. It draws attention to the developments in manufacturing information integration and dimensioning process automation. The review's key finding is the glaring absence of studies on the use of machine learning, particularly GNN, for enhancing automation in annotation processes. Even with these developments, there is still a significant research gap on the restricted investigation of using AI to improve BIM model annotation. This review sets the stage for future investigations aimed at leveraging AI to address the complexities of design and manufacturing integration, following a structured outline that guides through these thematic areas.

2.1 Feature based representation in CAD

CAD systems must incorporate dimensioning and tolerancing (D&T) (A. Requicha, 1977b) data to advance computer-integrated manufacturing (CIM). Current CAD models are mainly concerned with object nominal sizes and do not provide strong support for D&T and other manufacturing data. Roy and Liu (1988) proposed a feature-based representation scheme that utilizes a hybrid constructive solid geometry (CSG) and boundary representation (B-Rep) data structure to bridge this gap. The objective of this approach is to improve manufacturing orientation in CAD models by associating attributes such as tolerance with object features (Roy & Liu, 1988). Moreover, Requicha and Chan (1986) created an experimental system inside a solid modeler based on CSG, concentrating on a tolerancing theory that defines tolerances as characteristics of object features (A. Requicha, 1977a), like surfaces and edges, by using the 'variational class' concept (A. Requicha & Chan, 1986).

The framework (Roy & Liu, 1988) creates a strong foundation for advanced CAD by utilizing a complex combination of B-Rep and CSG, combined with a structured face-adjacency graph (SFAG). This hybrid method is designed to give D&T data, which are essential for processes in precision manufacturing, a more intricate representation. Through a structural connection between B-Rep, which provides detailed boundary definitions, and CSG, which facilitates hierarchical modeling of object features (geometric and non-geometric), the framework guarantees an organized and smooth flow

of information between various stages of product design and manufacturing. After the introduction of the SFAG, a complete and dynamic mapping of geometric features is made possible, which improves the model even more by retaining significant topological relationships at different hierarchical levels (low and high levels). Also, there are many different tolerances that can be attached to the SFAG since it is an explicit representation of fundamental entities (Roy & Liu, 1988).

Additionally, automated D&T was also discussed by Hillyard and Braid (1978) in their research. Although automatic dimensioning algorithms try to mimic the intentions of a designer, manual dimensioning is still better because it can capture detailed design considerations (Hillyard & Braid, 1978). These pioneering efforts have established the foundation for modern automatic dimensioning and annotating techniques, highlighting the significance of including manufacturing issues from the outset of the design process.

2.2 Automated dimensioning

2.2.1 Knowledge-Based approaches

Further exploration into automatic dimensioning systems revealed the complexities of integrating knowledge-based methodologies within CAD modeling processes. Diverse rules (for each feature, surface and form) that regulate dimension creation and organization were highlighted by a system which is written by Prolog[1] to automatically dimension 3D CAD models based on manufacturing parameters. This approach demonstrated how rules-based systems can simplify the CAD design process by cutting down on redundancy and improving the ability to choose and arrange dimensions. The exploration of knowledge-based systems marked a critical step towards intelligent, automated CAD technologies (Bond & Ahmed, 1989).

Moreover, the methodical implementation of guidelines controlling the generation of 3D dimensions for different geometric objects, including rectangular prisms and right circular cylinders, highlights the system's flexibility and adaptability in managing a range of design situations (Bond & Ahmed, 1989). By instantiating general 3D dimensions onto specific elements and strategically allocating surface dimensions, the

[1] Prolog is a strong and adaptable programming language that is used also for creating logic-based AI applications.

system ensures precise and contextually relevant dimensioning across different views. This comprehensive approach not only enhances the clarity and comprehensibility of dimensioned drawings but also streamlines the drafting process, thereby contributing to increased efficiency and productivity in Computer-Aided Manufacturing (CAM) workflows.

2.2.2 Scheme-Based approaches

Yuen et al. (1988) provided a comprehensive framework for the automatic dimensioning of objects described by solid modeling approaches, emphasizing the seamless translation of geometrical descriptions into architectural drawings dimensions. Their work, which was applied to the University of Rochester's PADL-2[1] modeler (A. Requicha, 1977b), shows the possibility for automation in creating sufficient dimensions straight from the B-Rep of solid models. This approach aims to minimize human error and ensure the correctness and sufficiency of dimensions for part definition by emphasizing the value of metrical geometry and its application in the automatic dimensioning of solid-modeled objects (Yuen et al., 1988).

The paper also presents a novel automatic dimensioning scheme that extracts metric information expressed as both angular and linear dimensions using B-Rep (Hillyard & Braid, 1978). This strategy has greatly enhanced the automation of engineering documentation and offers a theoretical foundation that could increase the accuracy and productivity of producing architectural drawings from solid models. A study conducted not only examines the workable application of this scheme on the PADL-2 system but also provides opportunities for future research in automatic tolerance representation. This implies that the suggested methods could have more uses in CAD (Yuen et al., 1988).

2.2.3 Automation for dimensioning and tolerancing in architectural drawings

Foundational reviews and evolution:

Roy et al. (1991) provide a thorough explanation of the representation, manipulation, and analysis of D&T in CAD and CAM systems. The paper organizes the literature into key categories, including how D&T are represented, synthesized, and analyzed, how

[1] PADL-2 which is an adaptable and expandable geometric solid modeling system.

tolerances (conventional and geometrical) are managed, and how this impacts CAM operations. This critical examination clarifies the merging of solid models with variational geometry to enhance CAD/CAM coherence. The goal is to automate the D&T process in order to minimize human error and improve the accuracy and productivity of design documentation and production processes. In keeping with the larger goal of improving the engineering design and manufacturing environment, this method marks a significant shift toward reduced manual intervention in drawing-based processes (Roy et al., 1991).

Building on this foundation, Yu et al. (1994) continue their methodological investigation of automatic dimensioning and tolerancing, emphasizing the importance of being able to use these critical design and production features within CAD systems. They examine the syntactic rules provided by engineering standards (Standard, 1982), assess the theoretical underpinnings of D&T, and argue that standards should advance in tandem with advances in technology. Their extensive analysis encompasses a broad spectrum of theories and models proposed over time, with particular emphasis on the differentiation between variational geometry and class (Elgabry, 1986), the creation of geometric constraints to faithfully capture dimensional and geometrical tolerances, and the direct and indirect object parameterization techniques (Yu et al., 1994).

Intelligent systems for dimensioning:

CSG and B-Rep data structures are key components in Chen et al. (2001) intelligent CAD system development for automatic dimensioning. These basic building blocks enable the system to create a 3D model of a machine part, which is necessary for accurate dimensioning and feature extraction (Yuen et al., 1988 ; Roy & Liu, 1988). The history of Boolean operations on primitives is captured by the CSG tree structure, whereas the part's boundary information such as vertices, edges, and faces as well as their connections is recorded by the B-Rep data structure. This combination facilitates the accurate determination of the coordinates of each vertex in the model, laying the foundation for intelligent dimensioning that reduces duplication and complies with drawing standards (Chen et al., 2001).

The creation of a 3D solid model using both CSG tree structure and B-Rep data structures is the first step in the methodical workflow for the intelligent dimensioning system. Three categories are used to group the model faces: planes, cylindrical surfaces, and other more intricate surfaces. The first two groups are treated with intelligent

dimensioning techniques, while the third group is treated interactively because of its complexity. The steps in the process are to identify dimensioning features for element pairs in each coordinate direction, analyze dimension redundancy, and choose the best dimensioning scheme. Then, using an expert system, dimensions are assigned and positioned in appropriate views to guarantee accuracy and the best possible placement for manufacturing needs (Chen et al., 2001).

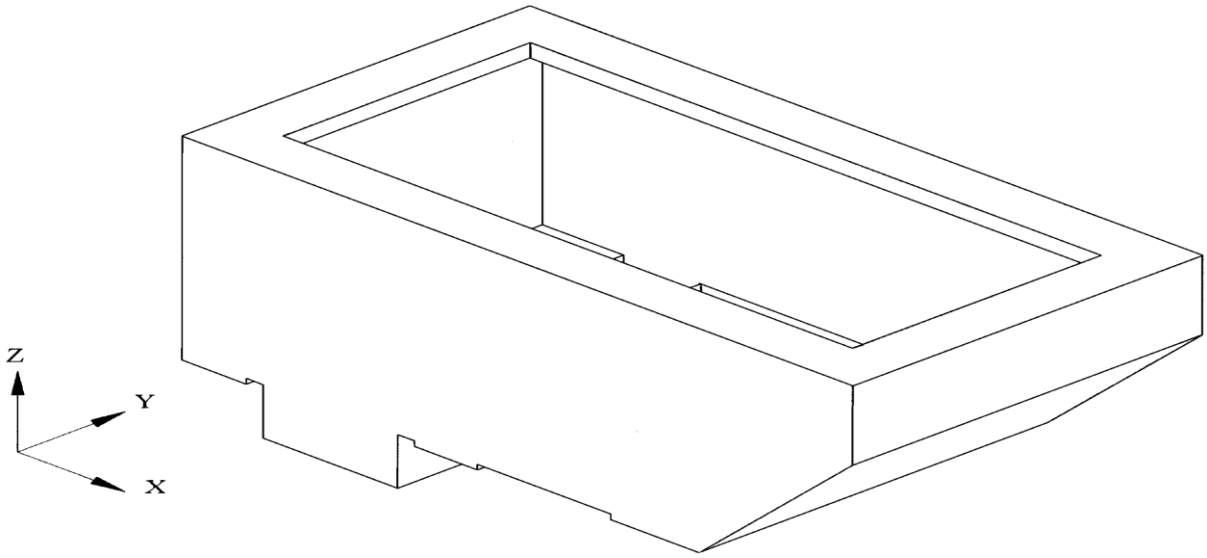


Figure 2.1: Box type element (Chen et al., 2001).

The process is visually represented in Figure 2.1 and Figure 2.2 (Chen et al., 2001), which show the system's ability to place dimensions intelligently, thereby optimizing the manufacturing and inspection processes.

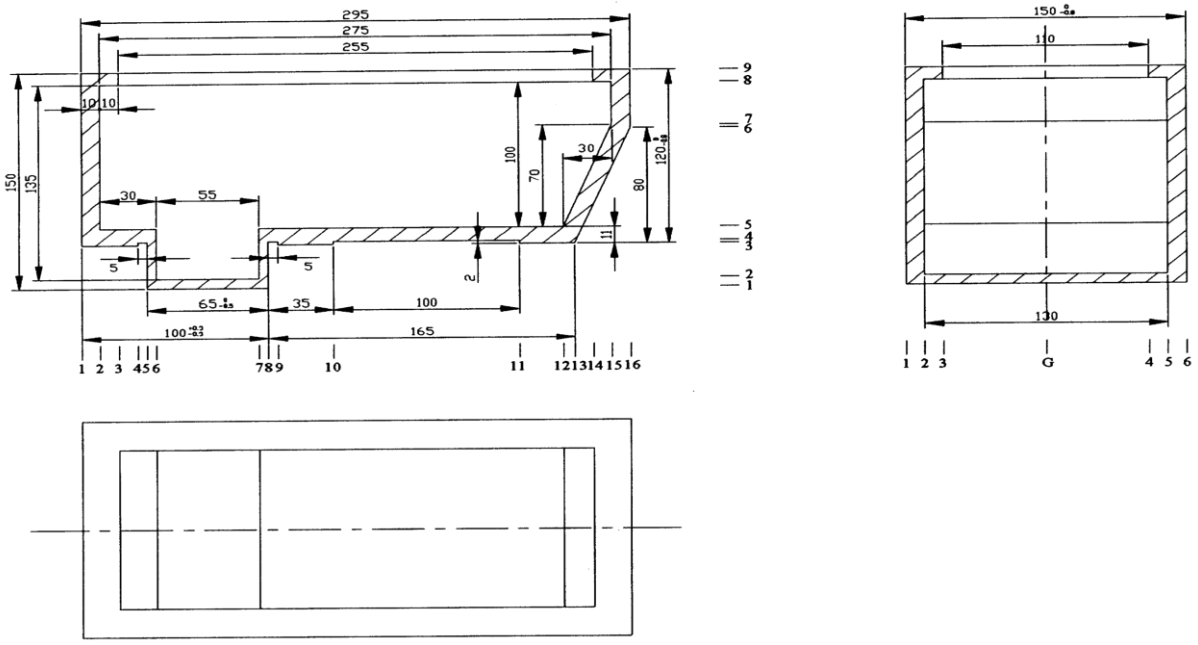


Figure 2.2: Automatic intelligent dimensioning by software prototype (Chen et al., 2001).

The research efforts of Chen et al. (2002) showed a logical progression towards the refinement and enhancement of CAD drawing automatic dimensioning. Extending the capabilities of intelligent CAD systems, they focused on the precise automatic creation of location dimensions for cylindrical surfaces, building on their foundational work from 2001 (Chen et al., 2002).

Figures 2.3 (Chen et al., 2002), which show automatic dimensioning of a multi-spindle headstock. These figures highlight the effectiveness of their approach in improving the accuracy and efficiency of CAD systems by demonstrating the system's ability for automatic dimensioning for cylindrical surfaces within complex mechanical parts. Since the intelligent system for them is outside the scope of this thesis, a reader suggested reading research from 2001 and 2002 to obtain a better understanding of it.

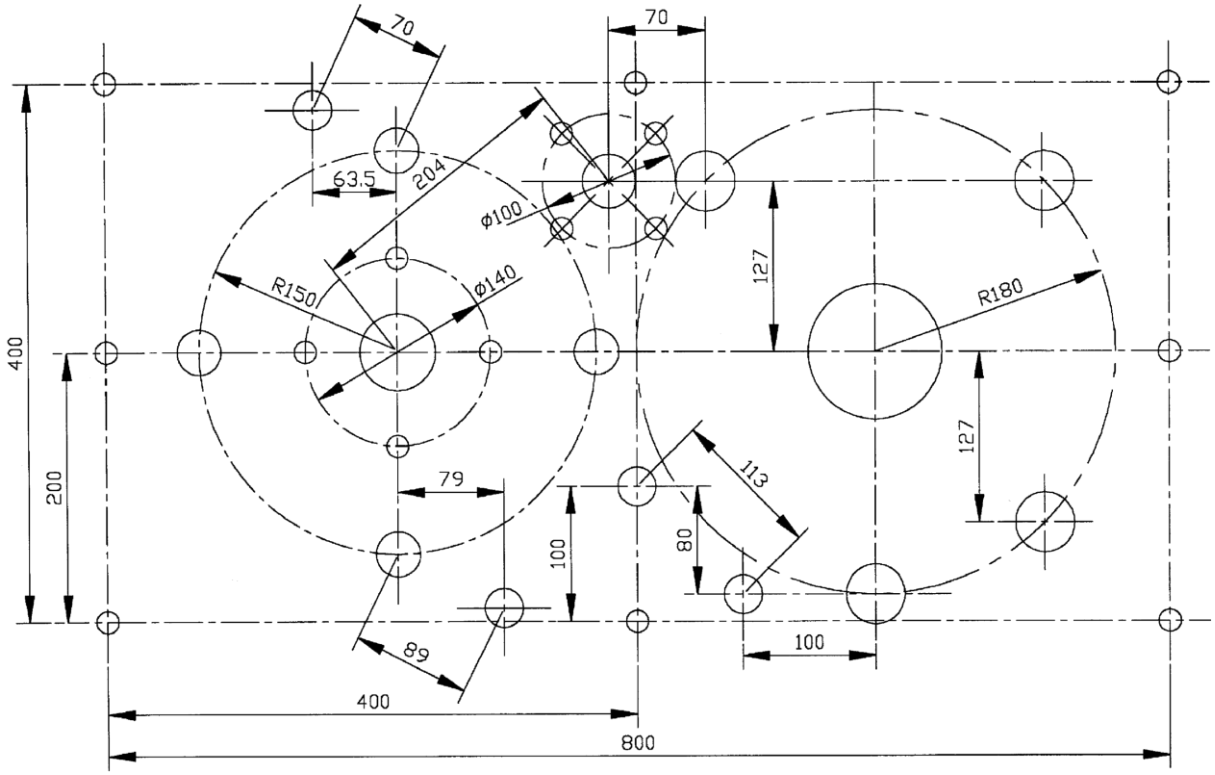


Figure 2.3: Utilizing a software prototype to intelligently locate and measure the holes in the multi-spindle headstock (Chen et al., 2002).

Automatic datum dimensioning:

Li et al. (2006) presented a new automatic datum dimensioning technique for plastic injection mould design. This technique uses a dynamic programming technique to optimize the placement of dimension tags. It addresses the challenge of avoiding dimension tag overlap and guaranteeing closeness to the features that need to be dimensioned. Additionally, it improves design productivity by facilitating seamless integration

with commercial CAD systems and enabling user-defined placement criteria. This approach is a big step in automating the mold design documentation process and demonstrates the possibilities of dynamic programming in CAD programs (Li et al., 2006).

The methodology is based on a number of important elements: (i) a preparatory phase in which key variables like dogleg angle, margin offset, and dimension tag size are set in order to make the optimization process easier; (ii) an optimization phase in which a dynamic programming technique is used to optimize dimension tag placement based on a number of factors, such as minimizing deviation from default locations and optimizing the usage of default forms (Li et al., 2006). Because they rate different characteristics of dimension placement, such as limiting divergence from default configurations and eliminating overlap between dimension tags, cost functions are vector-valued functions that are crucial to this process (Li et al., 2006). This methodical technique demonstrates the potential of dynamic programming to improve CAD integration for mold design by allowing for the quick and efficient placement of datum dimensions.

$$c_1(t_{i,j}, t_{i-1,k}) = O_A(t_{i,j}, t_{i-1,k}) + \sum_m O_R^m(t_{i,j})$$

The study found that the cost function which is depicted above ensures that the dimension tags are near the features that are being dimensioned, minimizes overlaps, and optimizes dimension tag placement in automatic datum dimensioning. The term OA is set to zero in the case where there is no overlap between the current dimension block and the prior one, indicating an ideal scenario devoid of overlaps. It assigns a relatively large value if any overlap occurs, penalizing arrangements that lead to such overlaps. The cost function called O_R^m is applied to penalize overlaps between dimension blocks di at the state $t_{i,j}$, and forbidden region R_m configurations. For a more comprehensive description of cost function, please refer to (Li et al., 2006).

Drafting automation by schemes definitions:

A different approach to enhancing the technical drawings process' effectiveness was presented by Raffaeli and Germani (2016), who created an automatic dimensioning methodology based on product family (gear motors) features. The process is divided into two primary stages: the online phase, when these schemes are automatically applied to generate dimensioned drafts, and the offline phase, where rules for drawing execution are encoded in a knowledge base created by Drafting Schemes. This innovative approach represents a substantial breakthrough in automating the production of architectural drawings (Roberto Raffaeli & Germani, 2016).

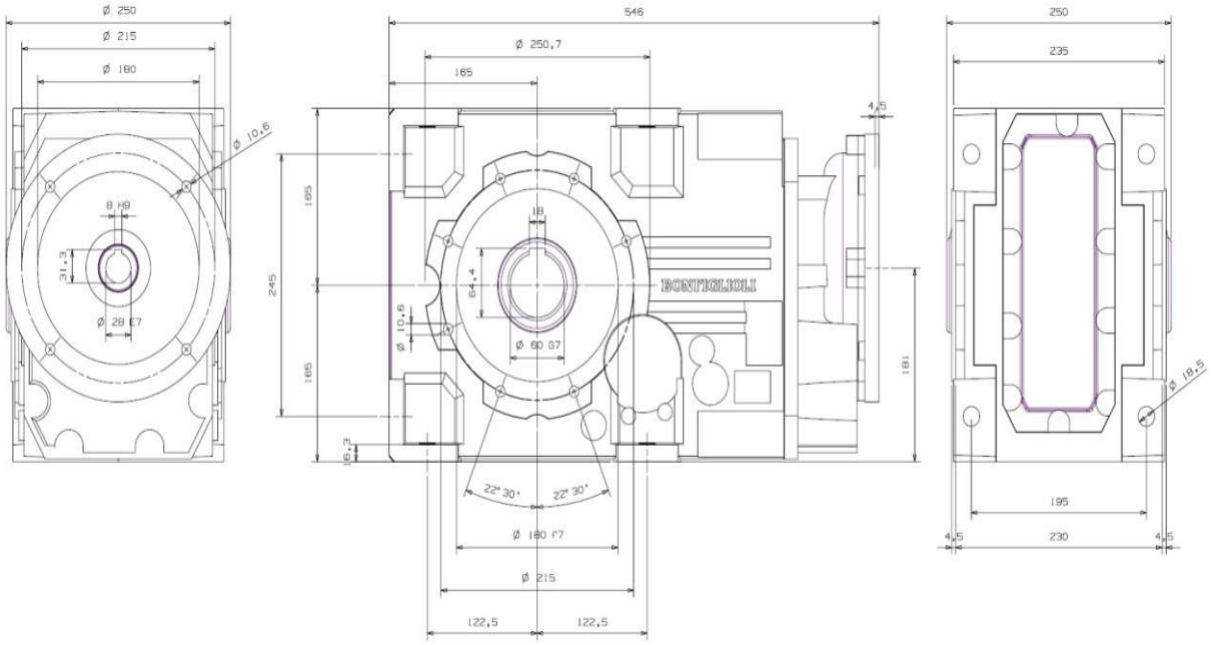


Figure 2.4: A dimensioned drawing example taken from the 'A60' gear motor 3D model (Roberto Raffaeli & Germani, 2016).

Their tests with the 'bDrafter'[1] program show that it is possible to generate a complete draft (see Figure 2.4) in an astounding average time of about 32 seconds, starting from geometry loading and feature recognition and ending with the completion of a 2D CAD model. Their study offers more insights for a comprehensive look of their findings, which go beyond the scope of this thesis (Roberto Raffaeli & Germani, 2016).

Automatic dimensioning by Automatic Dimensioning Placement (ADP) algorithm:

For automatic dimensioning in technical drawings, a solution to the ADP problem is presented by Kakoulis and Siozos (2017). Their two-step approach, which can be used to any 2D design, guarantees that dimensions are put efficiently and properly by drawing standards. They draw attention to how important automatic dimensioning placement is in reducing human mistake, speeding up the design process and preventing of financial loss (Bond & Ahmed, 1989), providing a practical solution to a CAD system's persistent (G & G, 2017).

Table 2.1 (G & G, 2017) displays a summary of their algorithm. According to researcher, future work involves applying dimensions on sections, details, and chamfers

[1] 'bDrafter' tool, a Windows-based program built on the Siemens NX 9.0 CAD system and the.NET framework.

to increase the functionality of the current software prototype (G & G, 2017). Another difficult issue would be figuring out whether symmetries are present in the input drawing and then adjusting the dimension positions to take those into consideration.

Table 2.1: Summary of ADP algorithm and definition (G & G, 2017)

Step	Action
Assumptions	Dimensioning involves line segments, circles, and arcs within a closed polygon boundary.
Definition	The center of gravity P, defined by coordinates xP and yP, averages the midpoints of all line segments in G. This aids in determining dimension placement by ensuring balance and feature proximity.
Input	Technical drawing G
Output	A dimension assignment for every graphical element in G.
1	Identify G's bounding rectangle R.
2	Compute G's center of gravity P.
3	Project line segments to R based on their relation to P.
4	Organize projected segments along R's sides.
5	Eliminate repeated dimensions.
6	Append dimensions for circles and arcs.

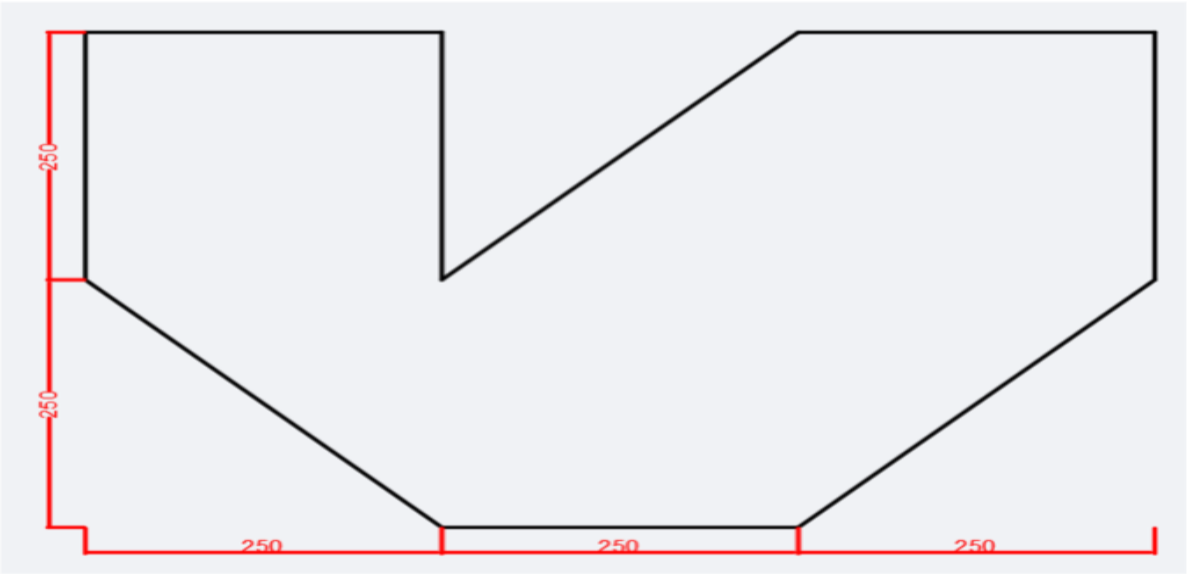


Figure 2.5: An ADP Algorithm-generated drawing with dimensions (G & G, 2017).

Figure 2.5 provides an example of an ADP algorithm on a 2D drawing (G & G, 2017). The reader is advised to consult the paper directly for a thorough understanding of their methodology and research instruments.

2.3 Advanced techniques in CAD

2.3.1 Interactive dimensioning

Kelly et al. (2015) provides an innovative framework for the interactive dimensioning of parametric models. At the core of their methodology is a real-time system that dynamically adjusts dimension lines based on the user's point of view, ensuring that dimensions remain relevant and visually consistent throughout interactive navigation. Their basic methodology involves processing every model parameter in order to derive the final dimension lines. Every parameter may result in multiple base lines and candidate lines before selecting the final line to be presented. The user-specified alignment preferences and slip behaviors work together to facilitate this process, which directs the placement of dimension lines to maximize readability and spatial coherence (Kelly et al., 2015).

Table 2.2: The summary of results contained several statistics for each model, including the total number of base lines, the number of lines in a cluster, and the measured frames per second (Kelly et al., 2015).

Name	Fig.	poly count	params	base lines (in clusters)	num clusters	pre-process
Single Tree	18	5610	6	15 (0)	0	<1 ms
Helix House	19	1495	13	210 (80)	22	<1ms
Lever	20	1178	7	45 (36)	10	<1ms
Boat	21	58214	23	37 (36)	24	2ms
Omni Tree	22	29767	18	75 (0)	0	<1 ms
Parthenon	23	132263	10	266 (240)	32	2ms
Candler	24	45366	6	2260 (2260)	47	87

If the scope of a parameter is hidden from view, their algorithm fails to return a dimension line. If not, it retrieves the base lines and uses the parameter's slip behavior to assign them to the appropriate planes. Next, it searches for preexisting guides on these planes and selects candidate lines on the basis of the results. If no candidates are found, the procedure computes them via the silhouette approach.

Table 2.2 (Kelly et al., 2015) is a comprehensive collection of performance measures for several models it is a subset of the complete table that is provided in the paper. The Helix House is shown in Figure 2.6 (Kelly et al., 2015), which illustrates the effectiveness of the methodology through the employment of both static and interactive approaches. The table displays important statistics such as the number of polygons, base lines, clusters, parameter values, and pre-processing times. For a more detailed understanding of the specific findings, including an analysis of the CPU and GPU performance of each model, it is recommended that readers read the whole study (Kelly et al., 2015).

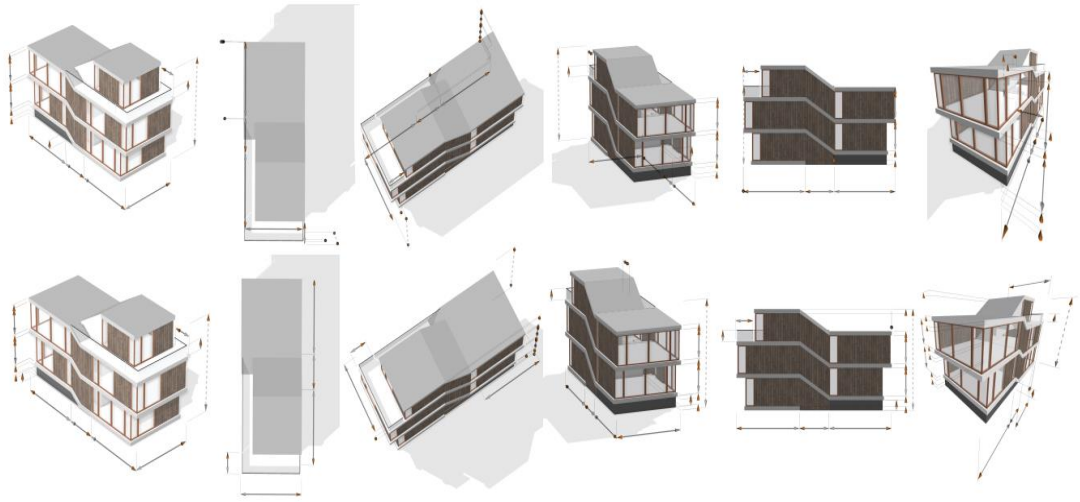


Figure 2.6: Helix House. Top: world space positions of static dimension lines. Bottom: In various fields of view and camera positions (Kelly et al., 2015).

2.3.2 Dimensioning methods for 3D modelling

The lack of a dimensioning technique specifically designed for 3D modeling in CAD systems which generally rely on conventional 2D sketching and 3D features like extrusion and revolution is addressed by Castro-Cañas and Pavón-Domínguez (2023). Their work offers a new dimensioning technique that makes use of the feature manager tree's sequential 3D modeling steps to guarantee a part's full geometric and dimensional definition. This method classifies surface and solid features in CAD software and

suggests common names and symbols for functional features, making cross-platform comprehension easier. More intuitive than traditional orthogonal views and cuts-based methods, the method emphasizes the value of 3D views and a logical step-guide, including sketching and intuitive symbols for operation orders (Castro-Cañas et al., 2023).

Readers are advised to refer to the whole work for a comprehensive understanding of their novel 3D modeling dimensioning technique, which falls beyond the specific focus of this thesis (Castro-Cañas et al., 2023). Their strategy presents a noteworthy advancement in CAD system dimensioning techniques, offering useful insights that could significantly influence the field of BIM model annotation and other related areas.

2.4 Graph Neural Networks (GNN) in CAD

The development of GNNs traces back to the initial explorations conducted in the late 1990s and early 2000s, where neural network (NN) methodologies were employed to analyze data structured in the form of graph (Zhou et al., 2020).

The earliest utilization of NNs on directed acyclic graphs set the foundation for the first investigations into GNNs (Sperduti & Starita, 1997). A significant study by Scarselli et al. (2009) established the foundation for the GNN model by presenting the core concept of using NNs to exploit graph relational structures (Scarselli et al., 2009). This groundbreaking study made many innovations possible, most notably the development of graph convolutional networks (GCNs) and graph attention networks (GATs) (Dai et al., 2018). According to Y. Li et al. (2016), these enhancements significantly increased GNNs' ability to recognize local and global patterns inside graphs (Y. Li et al., 2016).

By significantly accelerating advancements in a number of fields, including pattern recognition and data mining, NNs have revolutionized machine learning tasks such as object detection (Redmon et al., 2016; Ren et al., 2017), machine translation (Luong et al., 2015), and speech recognition (Hinton et al., 2012). Previously, the primary method for extracting relevant information from these occupations was through meticulous feature engineering (Wu et al., 2021).

A comprehensive investigation into the use of GNNs for various computational tasks is given in the section Wu et al. (2021) examined on GNNs. The explanation centers on the adaptability of GNN designs to different learning contexts, which are described as follows (Wu et al., 2021).

Supervised learning: Using data with predetermined labels, this approach trains models for accurate prediction analysis (Wu et al., 2021) .

Semi-Supervised Learning: By combining smaller amounts of labeled data with greater amounts of unlabeled data, this method improves model learning capacities. (Wu et al., 2021). Networks are able to create a reliable model that accurately assigns a class to previously unlabeled nodes (Kipf & Welling, 2017).

Unsupervised learning: It looks for patterns or structures in unlabeled data to deliver insights without predetermined labeling (Wu et al., 2021).

Additionally, the discussion touches on GNNs' capacity to yield outcomes at different levels (Wu et al., 2021).

At the Node Level: It is crucial for grouping specific nodes.

At the Edge Level: It is critical to predict the connections between nodes.

At the Graph Level: Significant for the general categorization of graphs.

In this study, the focus is placed on Graph Convolutional Networks (GCNs) and Graph Attention Networks (GATs).

2.4.1 Graph convolutional network (GCNs)

Within the field of deep learning (DL), the effort to augment GCNs' capacity to process non-Euclidean data represents a substantial development (G. Li et al., 2019). Historically, the vanishing gradient problem has restricted GCNs' depth to a limited number of layers, which has hampered their capacity to depict complex data. Inspired by the successes in deep Convolutional Neural Networks (CNN), a significant breakthrough is the development of methods combining residual/dense connections and dilated convolutions (G. Li et al., 2019). This innovative method makes it possible to train GCNs with up to 56 layers and produces GCN models that greatly outperform the conventional depth (Kipf & Welling, 2017). Such deep GCNs have demonstrated impressive performance advantages in tasks such as point cloud semantic segmentation, indicating a major breakthrough over previous architectures (Y. Wang et al., 2019).

One noteworthy development in the research is the concept of dynamic edges (Simonovsky & Komodakis, 2017), which significantly enhances the capabilities of GCNs (Y. Wang et al., 2019). This approach allows the graph's edges to be updated or recalculated at each network tier in response to the nodes' changing properties. Learning

outcomes are improved by using dynamic edge graphs since they enable the network to better adapt to the data (Yang et al., 2018). Because of this continuous recalibration of connections between nodes, the GCN is able to identify and adapt to complex patterns in the data more successfully than static designs (G. Li et al., 2019).

Figure 2.7 illustrates different GNN models built with graph convolutional layers (Wu et al., 2021). A readout layer combines the node representations into a single graph representation. An end-to-end framework for graph classification is obtained by applying a multilayer perceptron and a Softmax layer to various graph representations, as Figure 2.7 illustrates (Wu et al., 2021).

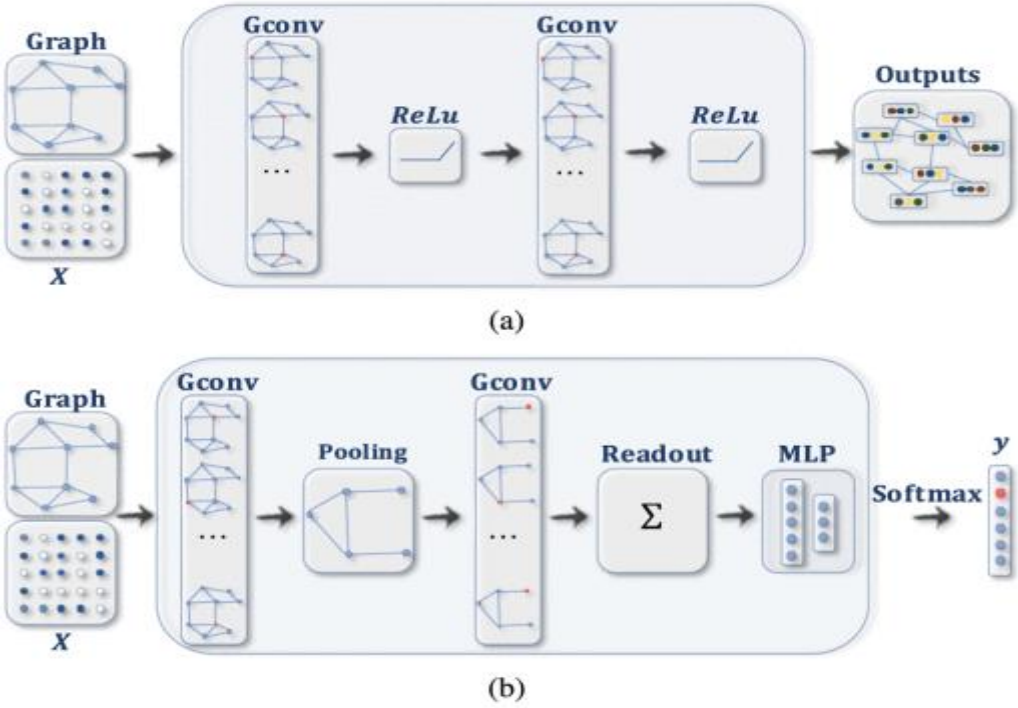


Figure 2.7: various GNN models generated with convolutional graph layers.
(a) ConvGNN using several graph convolutional layers. (b) ConvGNN for graph classification with pooling and readout layers (Wu et al., 2021).

2.4.2 Graph attention network (GATs)

CNNs have demonstrated remarkable performance in tackling tasks like image classification (He et al., 2016), semantic segmentation (Jegou et al., 2017) and machine translation (Gehring et al., 2017) because of underlying data that is arranged in a grid-like fashion. Because these systems may apply localized filters with tunable parameters at each input point, they are particularly useful for various applications (Veličković et al., 2018).

GATs represents a significant development in the field of GNNs. They address the shortcomings of previous models by incorporating masked self-attentional layers, which allow nodes to assess the relevance of the attributes of their neighbors without requiring costly matrix operations or prior knowledge of the network structure (Veličković et al., 2018). With this discovery, graph-structured data may now be represented more nuancedly, opening the door to using GATs for both transductive and inductive learning challenges. As demonstrated by Veličković et al. (2018), GATs attain or match state of the art (SOTA) results across numerous benchmarks, including citation network datasets and a protein-protein interaction dataset. A major development in graph data modeling is the capacity of GATs to attend over neighborhoods dynamically, offering a framework that is effective, easily customizable, and capable of handling a variety of graph-based activities (Veličković et al., 2018).

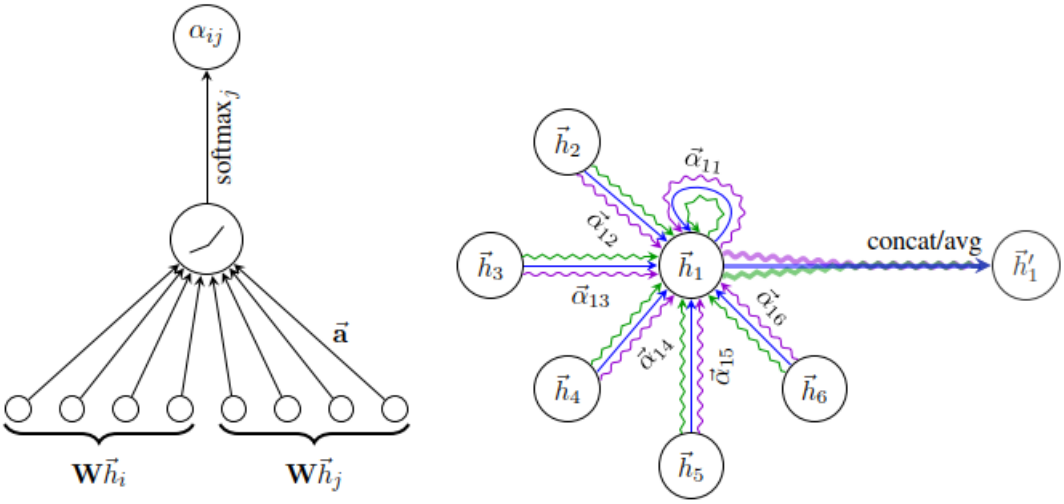


Figure 2.8: Left: A weight vector is used to parametrize the attention mechanism that is in use.Right: An example of multi-head attention (Veličković et al., 2018).

Figure 2.8 shows the graph attention mechanism of the model. Weighted node characteristics, represented by the weight vector and LeakyReLU[1] activation, are combined to determine attention coefficients. The multi-head attention approach visualization for a node and its neighbors on the right provides an example of how several attention processes are integrated, either by concatenation or averaging, to produce enhanced node attributes (Veličković et al., 2018).

[1] Leaky ReLU activation function is a well-liked solution to the shortcomings of the standard ReLU function in NNs by adding a tiny negative slope for negative function inputs.

3 Latest research and related work

3.1 API, geometric similarity and machine learning applications

Automatic drafting by API:

Buric et al. (2021) focused their inquiry on the automation of drafting in CAD systems on the development of iDrafter[1], a SOTA tool designed to automate the production of architectural drawings for mechanical parts. By utilizing the CAD system's API, the iDrafter, as shown in Figure 3.1, streamlines the process of drawing from 3D models, including dimensioning and the generation of drawing documents, views, and annotations. The study emphasizes the tool's efficiency in producing 2D architectural drawings from 3D B-Reps, addressing the laborious and prone to error nature of human drafting (Chen et al., 2001). A range of CAD models were used to evaluate the computational efficiency and resilience of the iDrafter. Automated feature recognition (Chen et al., 2001) and expanding the platform's application to additional drawing kinds are the goals of further updates (Buric et al., 2021).

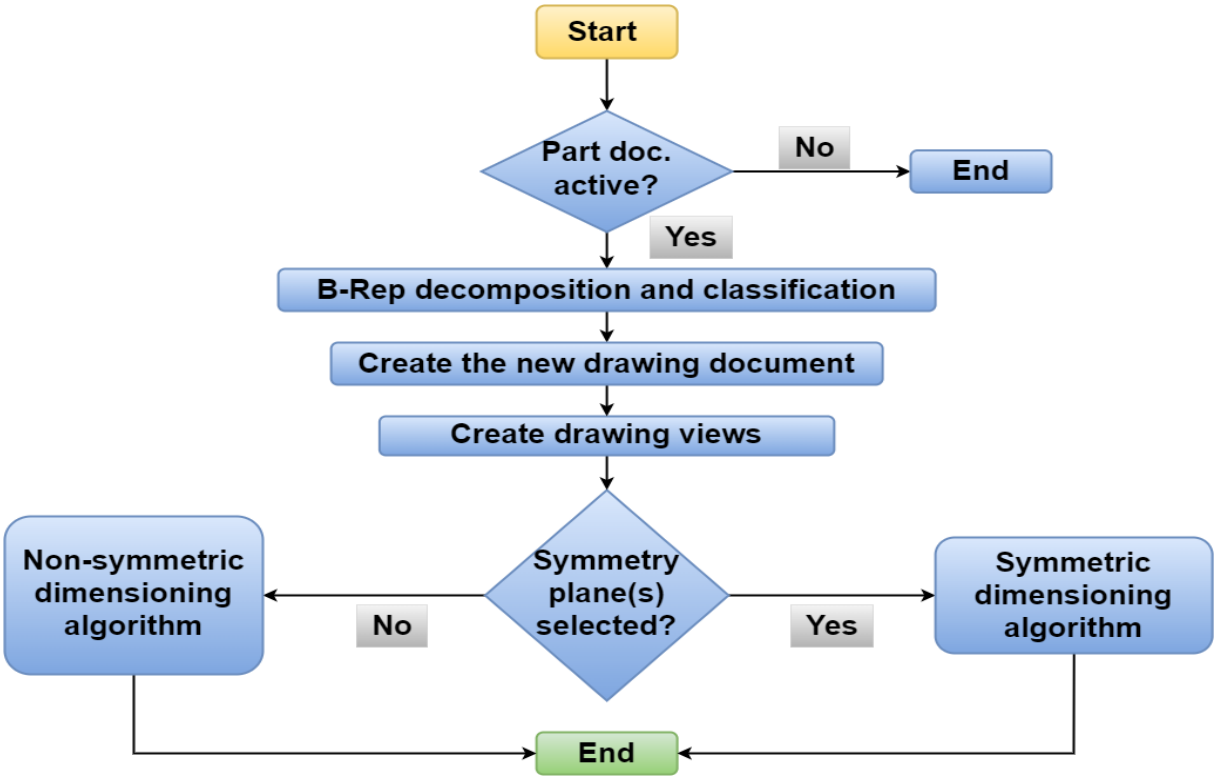


Figure 3.1: iDrafter algorithm (Buric et al., 2021).

[1] iDrafter is an automatic dimensioning tool which enables the automatic creation of 2D architectural drawings from 3D model's B-Rep.

Determining whether the current document is a part document is the first step in the iDrafter algorithm. Next, it examines and groups topological entities in the B-Rep. After that, it makes a new drawing document and generates three typical drawing views using the first-angle projection technique. The final procedures that methodically raise the precision and effectiveness of the drafting process are the D&T of these perspectives (Buric et al., 2021).

With the current version of iDrafter, only the three most popular views (front, top, and side) are created, which may not adequately depict the geometric complexity of the part being built. Additionally, the dimensioning algorithm's reliance on geometric simplifications and the removal of additional annotations restrict the tool's suitability for a broader variety of design scenarios (Buric et al., 2021).

3D Model Labeling with Geometric Similarity:

Wang, Xu, Wang, Fang, and Sun (2022) developed a novel automatic labeling method for 3D models, focusing on the use of geometric feature similarity. Their technique integrates the Longest Common Subsequence (LCS) algorithm with Freeman chain code to match and identify geometric features to find the appropriate label objects for 3D models. This approach is unique in that it can effectively and precisely automate the labeling process. Efficiency was boosted by 40% and labeling speed and accuracy were greatly improved as compared to older approaches. This invention addresses the challenges of automatic labeling by leveraging the geometric qualities of models, and it marks a significant breakthrough in the integration of image recognition and machine learning in the fields of intelligent design and digital manufacturing (Wang et al., 2022).

Figure 3.2 depicts the automatic labeling of 3D models, using geometric feature matching, clustering methods, and annotation structure optimization. This flow chart shows the sequential phases from the initial import of model design data, through the Freeman chain code computation and LCS geometry matching, to the final annotation verification and completeness check. This visualization, which illustrates how each stage leads to the precise and effective automatic labeling of 3D models based on geometric similarity, provides a concise summary of the process (Wang et al., 2022).

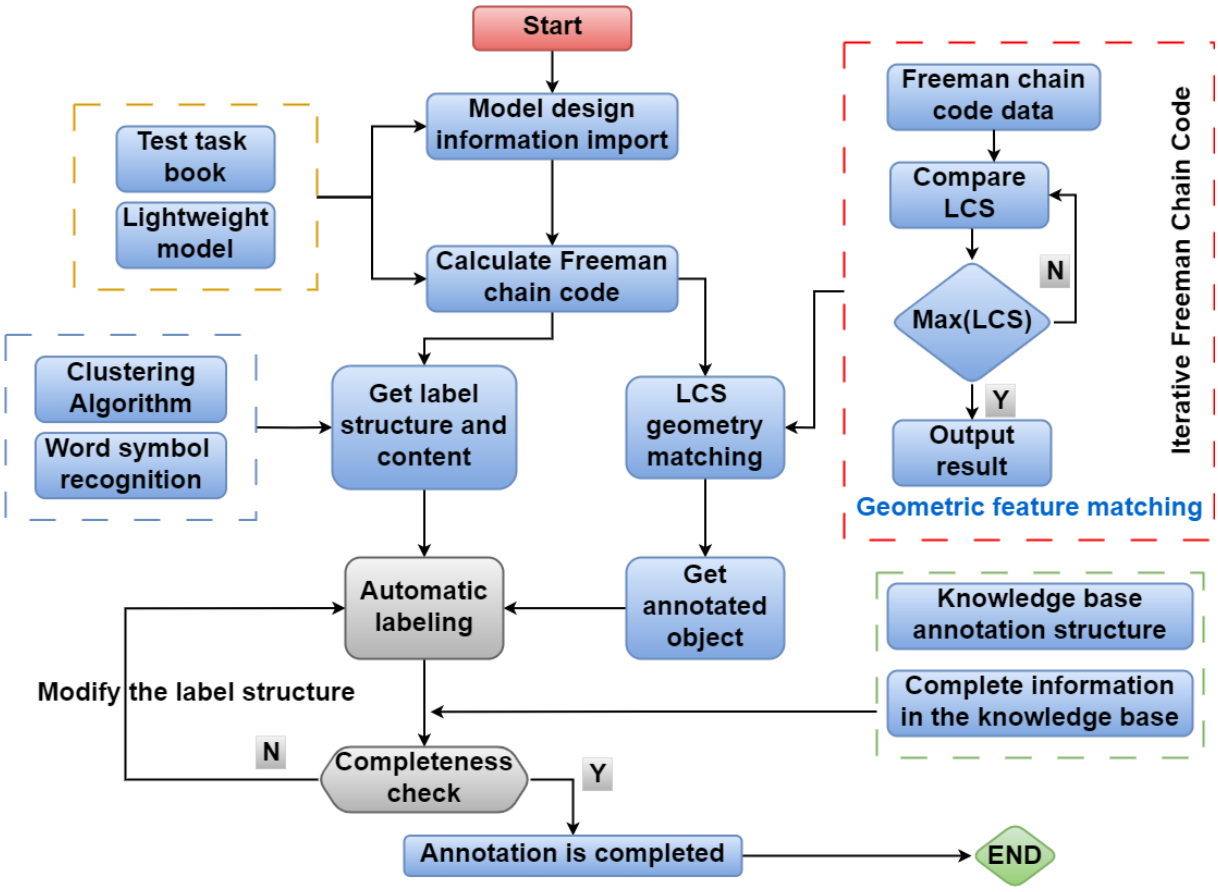


Figure 3.2: Annotation process (Wang et al., 2022).

Automated drawings by machine learning algorithm:

The advancements in AI and machine learning technologies have greatly helped the field of CAD. Villena et al. (2022) offer a novel method for automating the generation of CAD drawings using machine learning methods. The study employs two NN frameworks and a rule-based approach to automatically assign dimensions in drawings using supervised learning. While the authors admit that their approach can decrease manual labor in repetitive CAD activities, its capacity to handle complexity is limited, and rule-based automation is dependent on specialized CAD software (Villena Toro & Tarkian, 2022).

Step I of the procedure is to construct a labeled dataset for machine learning, as shown in Figure 3.3 (Villena Toro & Tarkian, 2022). This first stage is crucial to train the computer vision algorithm to recognize and classify different geometric aspects within CAD drawings. By automating the labeling process, this step aims to decrease the amount of manual intervention while increasing the precision and effectiveness of the dataset preparation phase for machine learning applications in CAD systems. A VBA script is used to create the data set automatically, producing both the image and its label. In

Step II, an advanced computer vision model called the YOLOv5[1] algorithm is put into practice. This model has been taught to recognize and classify shapes and dimensions that are present in engineering drawings. This crucial stage uses machine learning to automate the detection and localization of significant features in CAD drawings (Villena Toro & Tarkian, 2022).

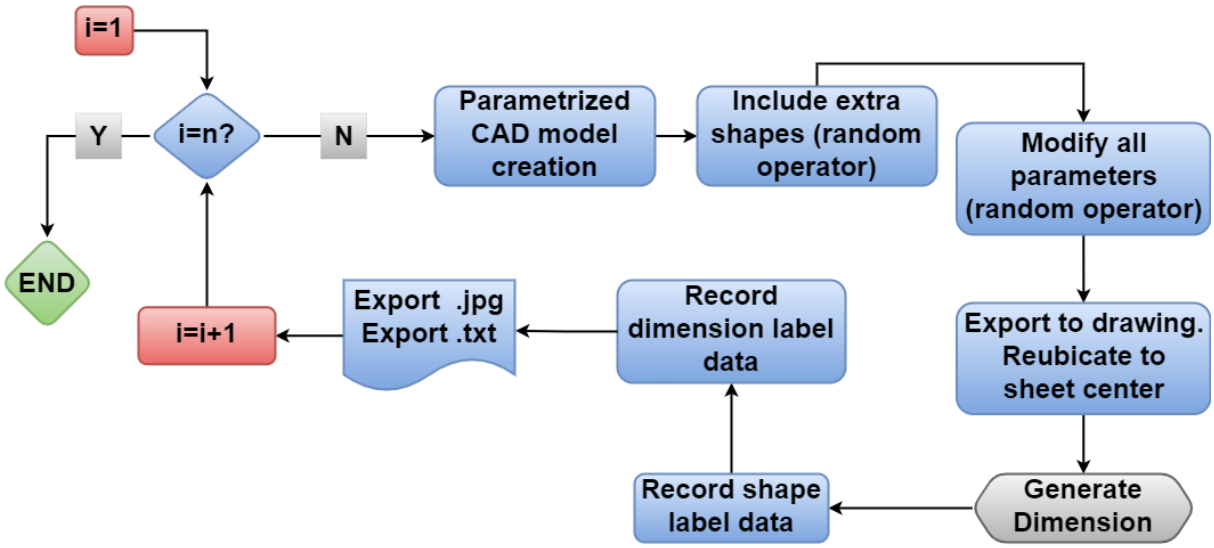


Figure 3.3: Step I, data set generation (Toro et al., 2022).

In Step III, as shown in Figure 3.4 (Villena Toro & Tarkian, 2022), a different NN is used to forecast the location and quantity of dimensions needed for each drawing, which progresses the automation process. The data is kept in matrices with the information about the bounding boxes after being divided into classes (Villena Toro & Tarkian, 2022).

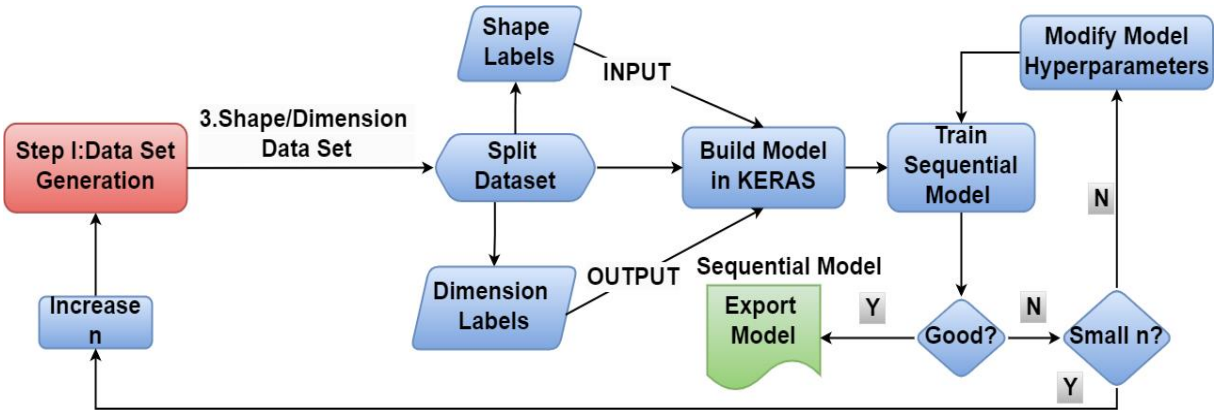


Figure 3.4: Step III, model setup (Villena Toro & Tarkian, 2022).

[1] YOLOv5, or 'You Only Look Once' version 5, is a cutting-edge object detection model. This DL algorithm uses a single NN architecture to effectively identify and categorize objects in photos or videos with high accuracy and speed.

The machine learning model's outputs are translated back into the CAD program in Step IV, which completes the automated process. The flow chart of Step IV is provided in Figure 3.5 (Villena Toro & Tarkian, 2022). In the final stage, the anticipated dimensions are instantiated straight into the engineering drawings using a rule-based script.

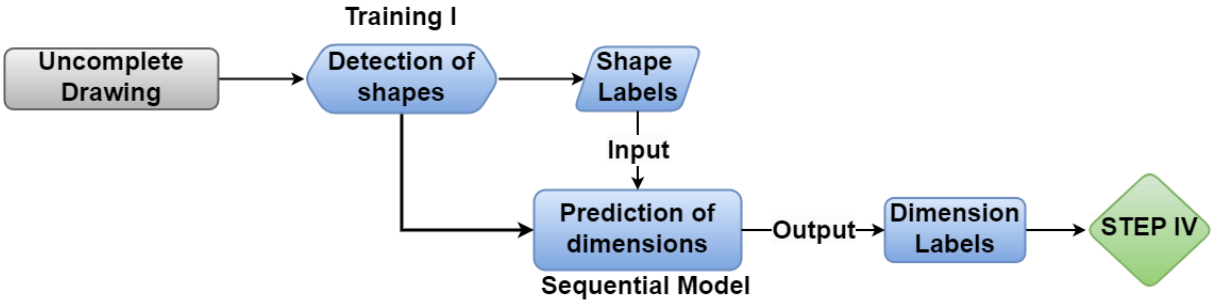


Figure 3.5: Finalizing the drawing (Villena Toro & Tarkian, 2022).

The method described in this study integrates design automation and machine learning into a framework to produce automatic and customized 2D drawings (Villena Toro & Tarkian, 2022). The four-step procedure is provided for a single case study. Future breakthroughs will broaden its application to increasingly complicated items, highlighting the ongoing need for improvements in machine learning integration into CAD platforms (Villena Toro & Tarkian, 2022). For an extensive knowledge of the methodology, readers are advised to read the entire work as it provides a deeper understanding of its development and application.

3.2 Research gaps

This study deviates from the traditional dependence on algorithmic and manual approaches by investigating the new application of machine learning and AI, particularly focusing on SOTA CAD technologies and GNNs for the automatic annotation of the architectural models. Previous research in this area has mostly relied on different methodologies and algorithms, often lacking the sophisticated features offered by machine learning and AI. This is particularly true when managing complex shapes and semantic details present in technical drawings.

In summary, there has been a noticeable void in the automatic annotation of architectural drawings in earlier studies as summarized below.

- Inadequate exploration of sophisticated machine learning and AI techniques, particularly Graph Neural Networks (GNNs), for parsing and automating annotations in architectural drafting.

- Lack of automatic dimensioning and annotation APIs to integrate machine learning results with architectural design software.
- Lack of emphasis on addressing the complex shapes and semantic details present in architectural drawings through advanced machine learning and AI methodologies. Previous researchers focused particularly on mechanical components, simple 2D and 3D shapes for automatic dimensioning and labelling.
- Prior research primarily examined CSG and B-Rep data structures for architectural drafting, along with technologies such as geometric similarity analysis and feature recognition to improve annotation.

By introducing a GNNs approach to predict the label types for architectural drawings, this thesis aims to close this gap. Furthermore, a plugin that seamlessly integrates the machine learning results with APIs is being developed to enable communication with architectural environment for automatic annotation. In the next chapters, more information on this novel strategy will be provided.

4 Methodology

Our method is developing a GNN that predicts the interconnectivity of relationships of elements in BIM authoring software. We are extracting data from architectural drawings and preprocessing them properly formulating graphs so to provide data for the training process. Finally, in the testing stage, the developed models receive architectural drawings without annotations as input. Based on the predicted label types, it subsequently produces the measurement type and the annotation itself.

Figure 4.1 illustrates the improved automatic annotation process for CAD models with ServCAD implementation in the architectural environment. The procedure commences with an architectural drawing, which serves as the foundation for data generation. The crucial data items are subsequently recorded in the file named 'ElementInfo.txt'. This textual data serves as the initial stage in establishing the foundation for structured data analysis.

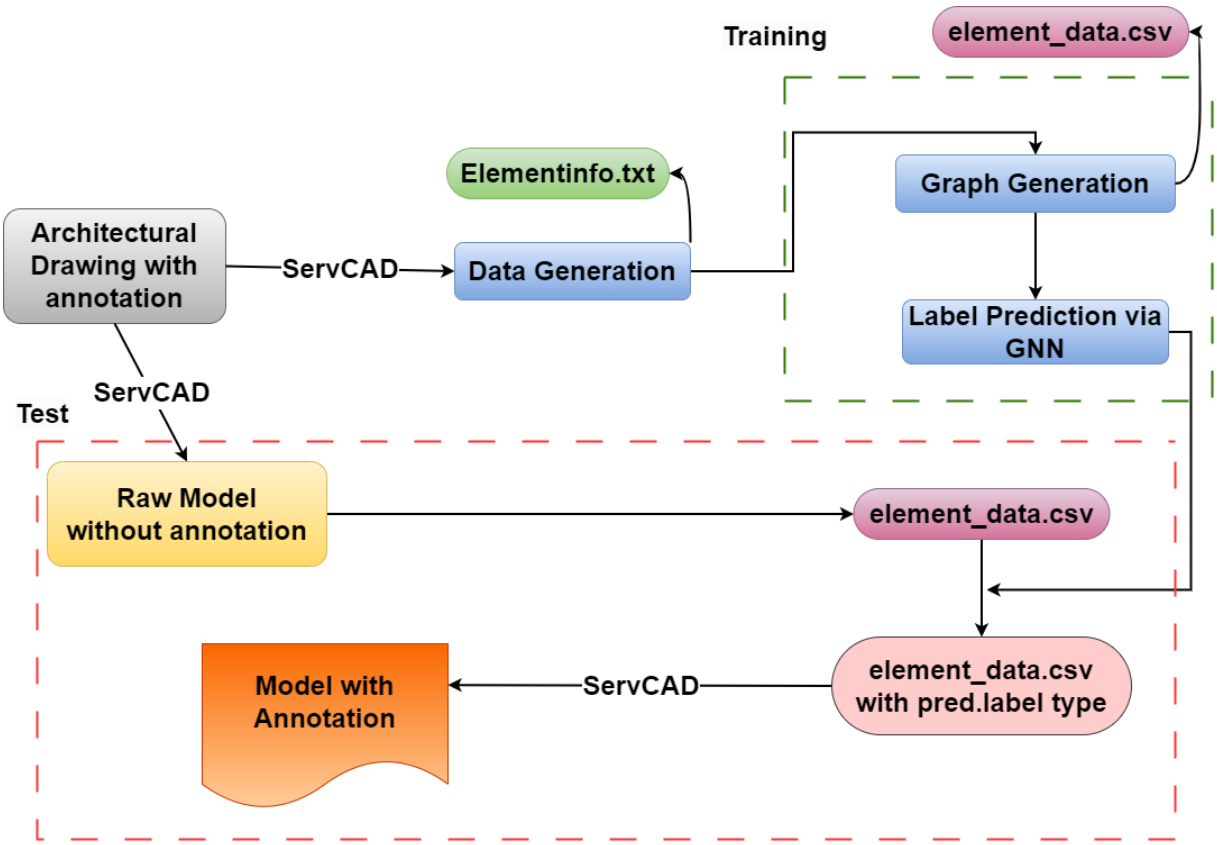


Figure 4.1: Automatic creation of annotation via GNN, proposed pipeline.

Once the data is generated, the graph construction stage utilizes the information in 'ElementInfo.txt' to construct a graph structure that acts as the basis for training the model for node categorization. Also, the complete version of the 'ElementInfo.txt' file was provided in Appendix B for Figure 1.1 to give a better understanding of the context of the file because it was foundation for all other flowing stages. All the other projects generate a similar file in the same structure with a greater length depending on the complexity of the architectural drawings.

The input file 'element_data.csv' is essential for ServCAD since it provides the necessary data for annotating the model after label prediction using GNN stage. The unprocessed architectural drawing undergoes a transformation into a model with annotations at the automatic annotation stage, which is the ultimate phase of this procedure. The content of the 'element_data.csv' can be observed in Table 4.1. It is the snipped version of the complete file which has more columns showing the all-node features that will be used for generating 'graphml' files and also for the training stage. The ‘element_data.csv' is particularly important for finalizing the proposed method being an input file for the automatic annotation part and having predicted label types as a node feature.

Table 4.1: The content of the 'element_data.csv'

node_id	element_type	guid	length	width	height	embedded_door_guid	info_string	label_type
10	Wall	3A166D5F-444B-4FFE-9C42-9E4433C7ED62	5	0.34	3	3C1ADF00-99DB-48EB-B5FC-A075685A71B1	WALL-001	1
11	Wall	26D78958-F733-42A0-9437-6A2E3AFDDDBB	5	0.34	3		WALL-002	1
12	Wall	A3831D8D-2072-464A-907B-4FF183A3BEBD	5	0.34	3		WALL-003	1
13	Wall	05526A12-85D5-41ED-9A07-FD2FA2B2489E	5	0.34	3		WALL-004	1
15	Zone	E21B4B3D-D3B6-4189-9DB9-					SPCE-001	4
16	Door	3C1ADF00-99DB-48EB-B5FC-A075685A71B1		0.9	2.1		D-001	2

In the upcoming chapters, each stage of the pipeline, starting from data production and ending with automatic annotation as depicted in Figure 4.1, will be comprehensively analyzed. A comprehensive explanation of the inputs and outputs generated at each level of the workflow will be given. A systematic analysis will be provided, illustrating the process of transforming and refining the data, commencing with the first extraction of fundamental information and culminating in the implementation of annotations.

4.1 Data extraction and preprocessing

4.1.1 Data generation

To extract all the data needed for the components in the CAD environment, ServCAD is employed throughout the data production phase. After starting the initialization process, which entails opening the output file where the collected data is supposed to be stored. Following processing, the building elements' dimensional properties are recorded and reported. These elements are labeled when the data regarding the basic elements is stored. Lastly, any further information deemed necessary is supplied, leading to the creation of the 'ElementInfo.txt' file. Figure 4.2 provides a summary of the entire process.

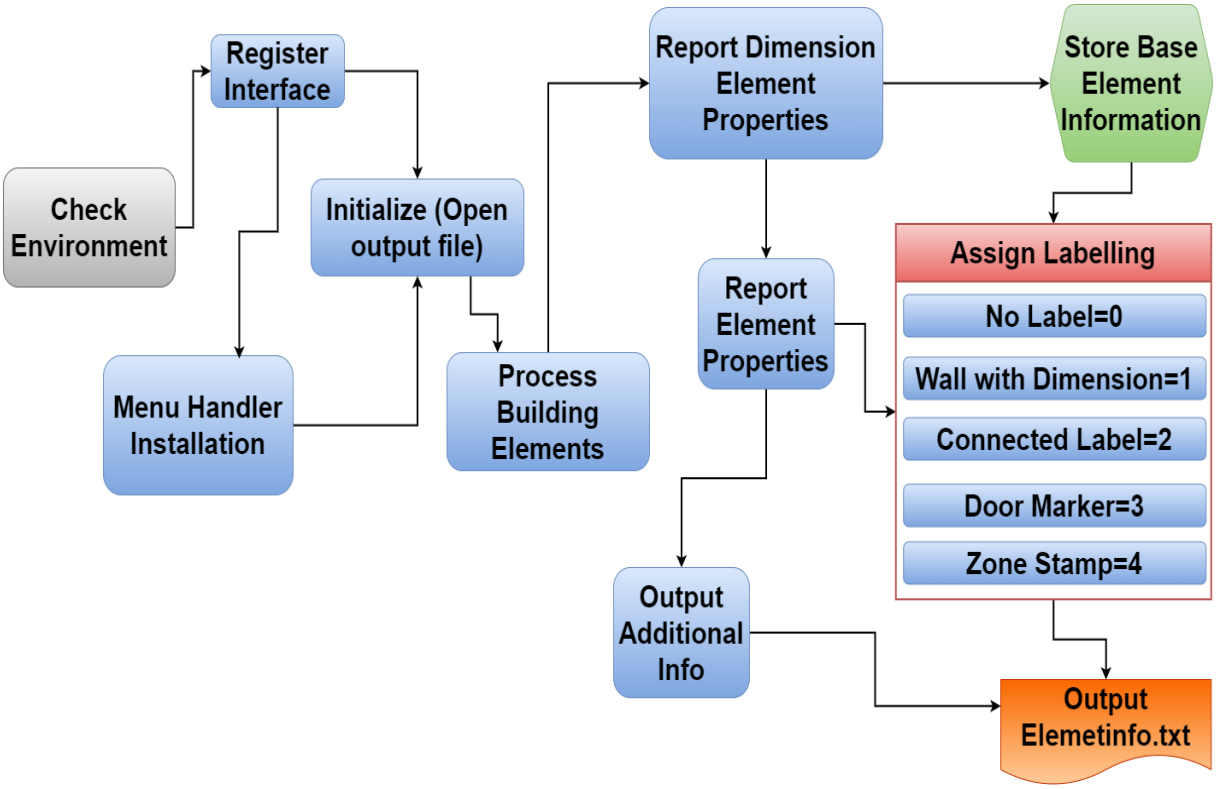


Figure 4.2 : ServCAD data generation algorithm.

Assigning the appropriate label to each building element was a crucial step in our process. When identifying walls, we first referred to the base element information stored during dimension property reporting. If any wall had attached dimensional elements, it was labeled as type 1. To identify connected labels, we utilized the ACAPI_Grouping_GetConnectedElements() function, which allowed us to check for any connected labels to an element. If found, the label type was assigned as 2. In contrast, labeling doors and zone markers was a simpler process. For doors, we extracted specific node features, enabling us to assign them as label type 3. Similarly, zone markers were identified and labeled as type 4, leveraging pre-existing node features.

Table 4.2: Extracted information from AD for further processing.

Parameter	Description
Element Type	Type of element (e.g., Wall, Slab, Door, Zone)
GUID	Global Unique Identifier for the element
Associated Element GUID	Identifier for the related element
Embedded Door GUID	Identifier for an embedded door
Zone Stamp GUID	Identifier for the zone stamp
Label GUID	Identifier for the door label
Marker GUID	Identifier for the door marker
Length	Length of the elements for wall,door and dimension nodes
Width	Width of the elements for wall, door
Height	Height of the elements for wall, door
Text	Text associated with the dimension node
Bounding Box	Coordinates defining the element's spatial extent
Room Name	Name of the room (for zones)
Room Number	Number of the room (for zones)
Room Height	Height of the room (for zones)
Info String	Additional information about the element (name etc.)
Position	The position of the element for zone marker, dimension text and dimension nodes
Label Type	Classification label assigned to the element

Table 4.2, a comprehensive depiction of the data included in the 'ElementInfo.txt' file, summarizes the information retrieved from the architectural drawings. This table contains a list of all the attributes that have been acquired. It covers all the information needed for the annotation and analysis of the CAD elements (wall, slab, zone and door), including positional data, descriptive labels, unique IDs, and geometrical dimensions.

4.2 Data representation

4.2.1 Graph generation

The 'ElementInfo.txt' file is the initial phase of the graph generation process, as seen in Figure 4.3. The system performs a loop that takes each file as input and outputs nodes and edges data, which are the fundamental building components of the graph. This data is then collected and eventually converted into data frames in order to efficiently organize the information. After conversion, the element and accompanying annotation data are refined in a filtration step. The end product of this step is a collection of CSV files, which collectively comprises of the structured graph data ready for further processing and analysis.

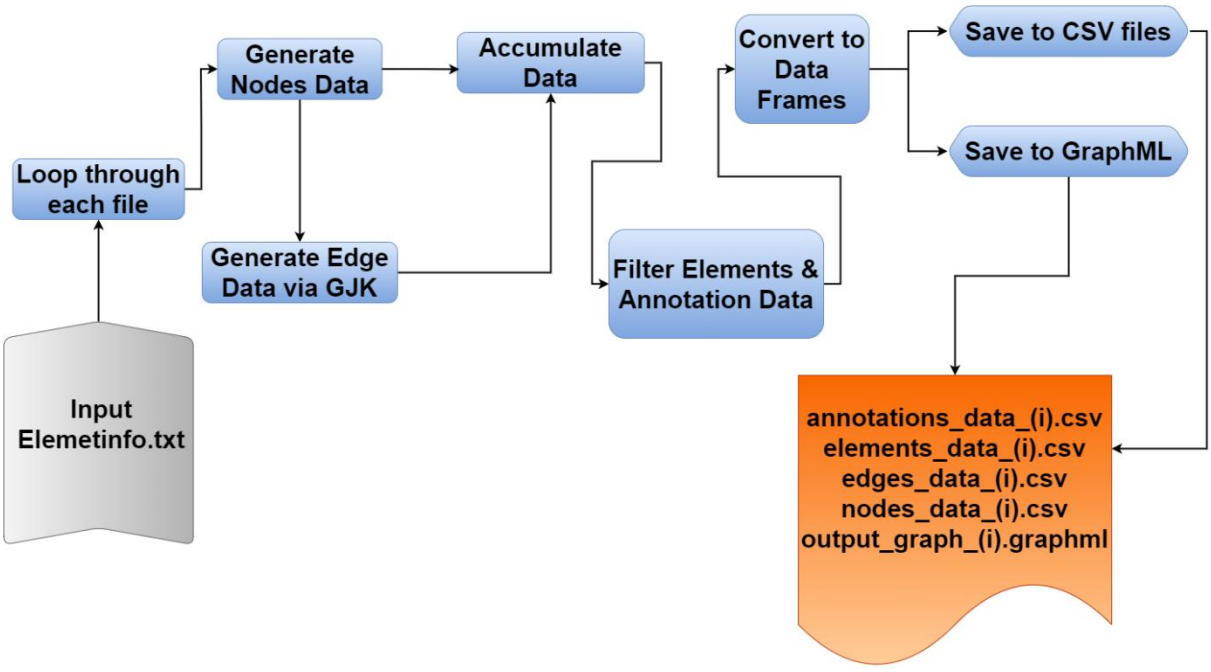


Figure 4.3: Graph generation with python.

4.2.2 Edge data creation with Gilbert–Johnson–Keerthi algorithm

The edge data in the graph construction process was produced using the Gilbert–Johnson–Keerthi (GJK) distance algorithm, as shown in Figure 4.3. The distance between elements computed by this complex method had a major role in determining the construction of edges in the graph structure. By using a tolerance level on distance (in our case is 0.1), it became possible to precisely map the connections required for graph-based representations, enabling us to precisely identify relationship between adjacent element in the architectural drawings.

4.2.3 Visualization in neo4j

After edge, node, and graph files were created, visualization was crucial to verifying the quality and integrity of the data. This verification stage made use of Neo4j[1] to display of nodes and connections. The vast scale and abundance of nodes and edges in the data made it impossible to visualize every created graph file. Graph data Figure 1.1 was visualized in Figure 4.4. The nodes, edges, and connections between them as determined by the GJK algorithm discussed in the preceding chapter are displayed in this image.

Appendix A.1 offers a more detailed representation of Figure 1.1. It offers a thorough understanding of all potential relationships between the building elements, including objects with annotations. Even though Figure 1.1 served as a starting point, the objective was to evaluate the applicability of graph generation and visualize more complex architectural drawings. Since more advanced architectural drawings are being used for training, this part was essential. Appendix A.1 included visualizations of the more intricate drawings as well. This enhanced representation draws attention to the true intricacy of the architectural drawings by eloquently depicting the intricate web of relationships and the geographical context of each component in the model. This complex picture was not the visualizations used to train the data for the subsequent steps. Graphs collected from different projects (see Figure 4.5) were assumed to have the same element type as in Figure 1.1. Only the wall, door, and zone elements were used

[1] Neo4j is incredibly scalable and flexible graph database management system for storing, querying, and assessing connected data.

for training. Slab elements were not included in the automatic annotation process and were eliminated for lack of informative value.

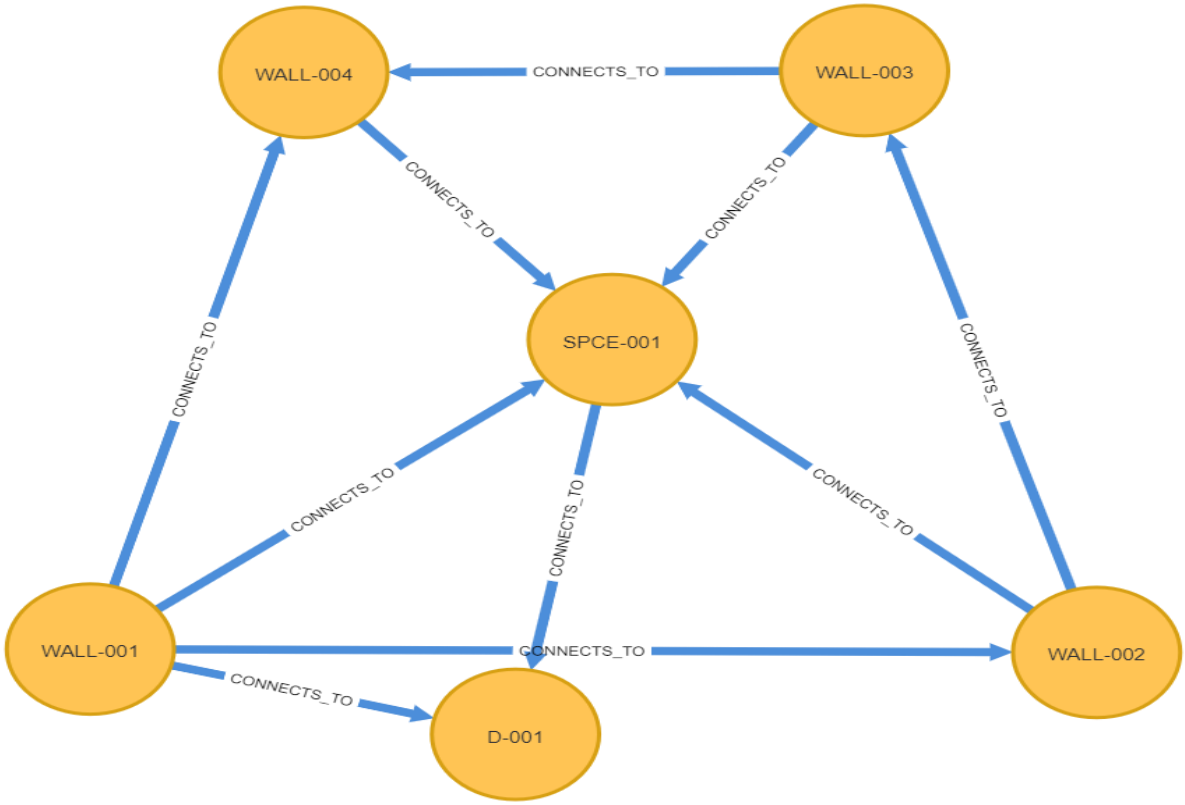


Figure 4.4: Visualization of sample project in Neo4j.

Additionally, in Appendix A.2 graph representation of some example projects was also provided with annotation nodes. Although these graphs were not used as a part of the training process, they are provided to give an idea about implementation of future works such as automatic arrangement of the annotation elements.

4.3 Dataset preperation

The node features utilized for our model training included a combination of numerical attributes and one-hot encoded categorical variables. Numerical attributes such as position coordinates (pos_x, pos_y), bounding box dimensions (bb_xmin, bb_ymin, bb_xmax, bb_ymax, bb_zmin, bb_zmax), and dimensional properties (width, height, length) were considered. Additionally, categorical features such as 'element_type', 'room_name', and 'room_number' were one-hot encoded to capture the categorical nature of these attributes. Table 4.2 illustrates the input features for our classification task and the red highlighted cell shows the one-hot encoded features. However, certain node features such as 'guid_id' and 'info_string' were disregarded in the training

process due to their non-categorical nature. Moreover, considering the classification task at hand, the output dimension of our model was determined to be 5 (see Figure 4.2), corresponding to the five different label types assigned to building elements within the architectural drawings.

After the creation of pytorch geometric, the graph's split logic was applied to the dataset as in Figure 4.5. The dataset is composed of 64 graphs total, of which 49 were produced from 7 ArchiCAD[1] sample projects and 15 were self-drawn to enhance the model performances. All the generated graphs have the same element features similar to the graph provided in Figure 4.4 without presenting the annotation nodes. The graphs include these elements used for the training, validation, and testing stages.

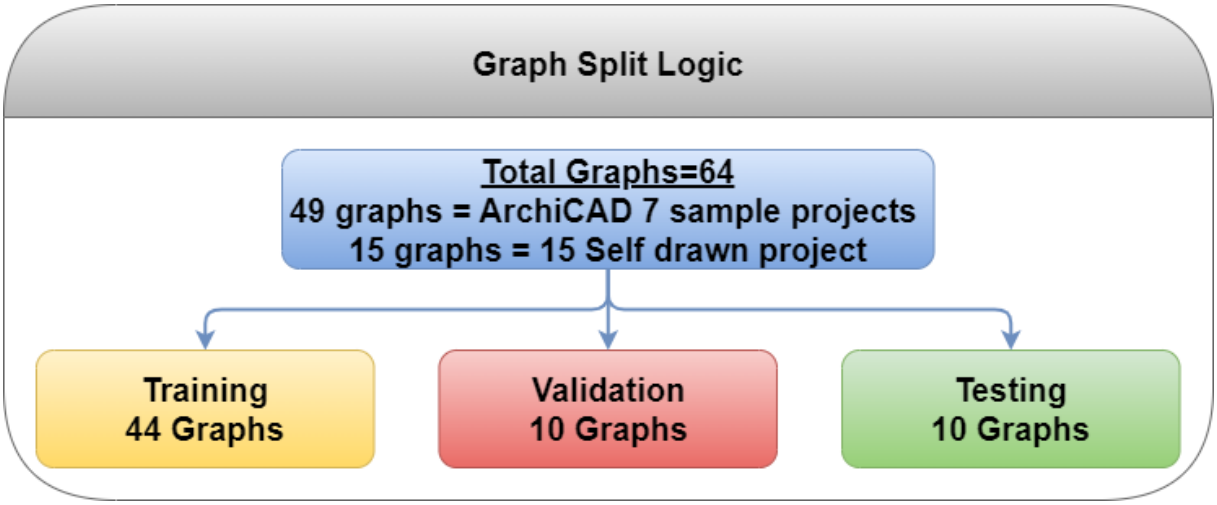


Figure 4.5: Graph split logic for training, validation and testing.

4.4 GNN Models

4.4.1 Model definition and assumptions

A thorough explanation of the architecture and definition of the GNN model utilized in this thesis was depicted in Figure 4.6. Initializing seeds is the first step in the workflow to guarantee reproducibility. The graph data is then processed by reading GraphML files. To make the node compatible with the NN, node properties are analyzed and relevant categorization nodes are one-hot encoded. One-hot encoding is carefully used before the PyTorch Geometric framework is introduced since it eliminates nodes that are unclassifiable. When building and integrating the GNN model, a customized

[1] Archicad sample projects: https://community.graphisoft.com/t5/Getting-started/Archicad-Sample-Projects/ta-p/304186

loss function called Cross-EntropyLoss with class weights is coupled with the Adam optimizer to handle class imbalance. A learning rate (LR) scheduler called 'ReduceL-ROnPlateau' is used to dynamically adjust the LR in response to validation lThioss.

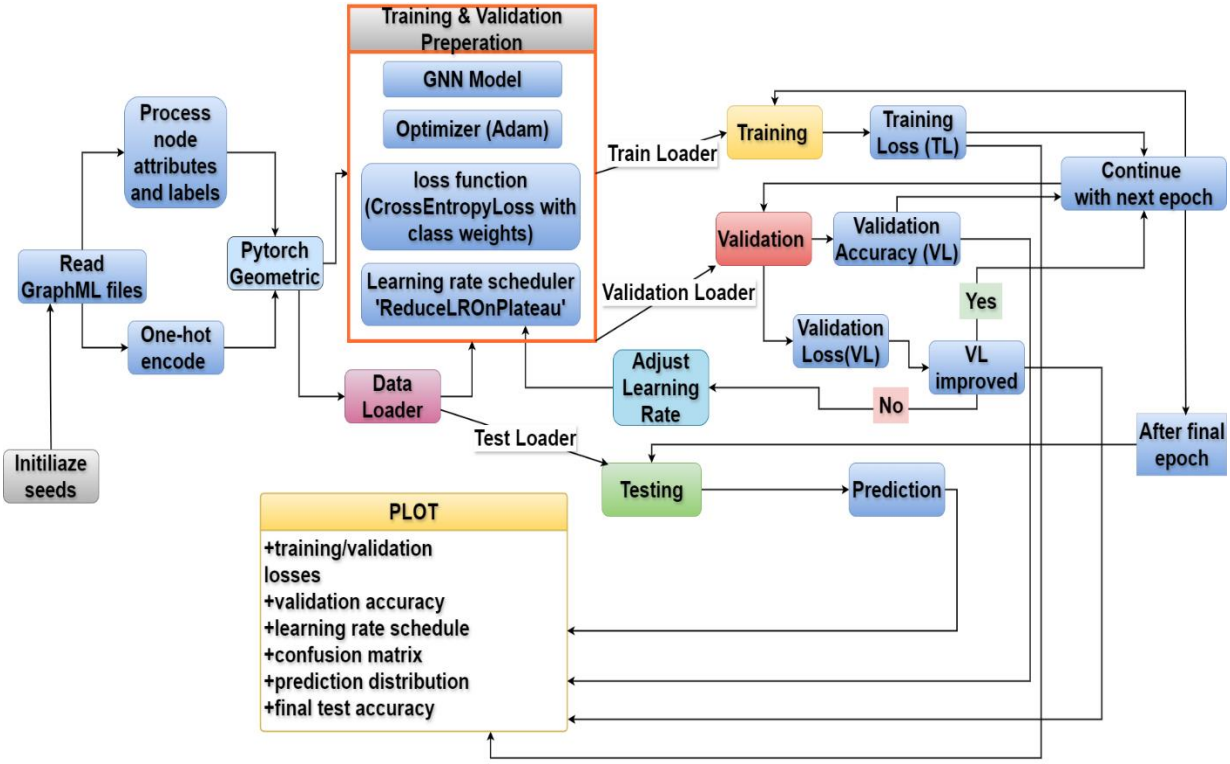


Figure 4.6: GNN training logic with pytorch

The Data Loader effectively shuffles and batches the data during the rigorous training phase, allowing the model to learn throughout each epoch. The model continues into the following epoch if the validation loss decreases; if not, a deliberate adjustment is made to the LR. Throughout this process, continuous monitoring is implemented, yielding a variety of metrics and graphics like loss charts, accuracy graphs, and confusion matrices. This all-encompassing method ensures that the model performs reliably both during training and testing, and that it is correct and generalizable in the latter stages.

Following training, the model goes through testing, when it predicts outcomes by using the patterns it discovered during training. This makes it possible to assess if the model can generalize to new, unseen data. The methodical documenting of process outcomes, including the model's accuracy and the training loss progression, will be presented in Chapter 5.

4.4.2 GCN Architecture

In order to process the graph data efficiently, the two-layer structure that defines the architecture of the GCN used in this thesis is designed to have the input dimensionality tailored to the unique properties of the nodes. The network has 256 neurons in its sizable hidden layer. As seen in Table 4.3, it concludes with an output layer meant for five distinct categories (see Figure 4.2). The Rectified Linear Unit (ReLU)[1] activation function is employed in between the convolutional layers to introduce non-linearity into the learning process. To avoid overfitting, half of the neuron activations during each training cycle are arbitrarily removed at a dropout rate of 0.5. Finally, as shown in Table 4.3, PyTorch's default settings manage weight initialization and provide a solid basis for the network's weight parameters.

Table 4.3: GCN architecture

Component	Description
Model Architecture	GCN with 2 layers
Input Dimension	Based in unique node features
Hidden Dimension	256 (Number of neurons in the first hidden layer)
Output Dimension	5 (unique labels for nodes)
Activation Function	ReLU used between the conv. layers
Dropout Rate	0.5
Weight Initialization	Pytorch default

4.4.3 GAT Architecture

In contrast to the previously mentioned GCN, the GAT model incorporates an attention mechanism in its two-layered structure, as illustrated in Table 4.4. By employing this approach, the model can focus its attention on the specific regions of the input graph that are most pertinent to the purpose of the thesis. In order to enhance the model's ability to recognize various types of relationships within the input data, the GAT

[1] ReLU is a popular activation function for NNs that creates non-linearity by directly outputting the input when it is positive and zero otherwise.

incorporates attention heads. In the first layer, specifically, 22 attention heads are employed. The lack of attention mechanism in the GCN model is a significant differentiating factor.

In order to ensure a reliable as well as effective comparison between the output of these two models, all other model parameters, such as input and output dimensions, hidden layer size, activation function, dropout rate, and weight initialization, have been kept consistent.

Table 4.4: GAT architecture.

Component	Description
Model Architecture	GAT with 2 layers
Input Dimension	Based in unique node features
Hidden Dimension	256 (Number of neurons in the first hidden layer)
Output Dimension	5 (unique labels for nodes)
Heads	24 attention heads in the first layer
Activation Function	ReLU used between the conv. layers
Dropout Rate	0.5
Weight Initialization	Pytorch default

4.5 Automatic Annotation with ServCAD

The final phase of the procedure, seen in Figure 4.7, utilizes the label predictions generated during the testing phase. These labels have been incorporated into the 'element_data.csv' file produced during the graph generation stage and serve as the foundation for further processing. The content of the element_data.csv' was already mentioned in Table 4.1 which has the all node features necessary to automatically create annotation such as their positions, dimensional properties (width, height, length), and most importantly predicted label types. Based on the predicted label type, related APIs from the ServCAD plugin are invoked to create automatic annotation. To suitably label walls, doors, and zones, for instance, several functions like CreateDimensionFor-Walls(), CreateLabel-ForDoors(), and CreateZone() are called.

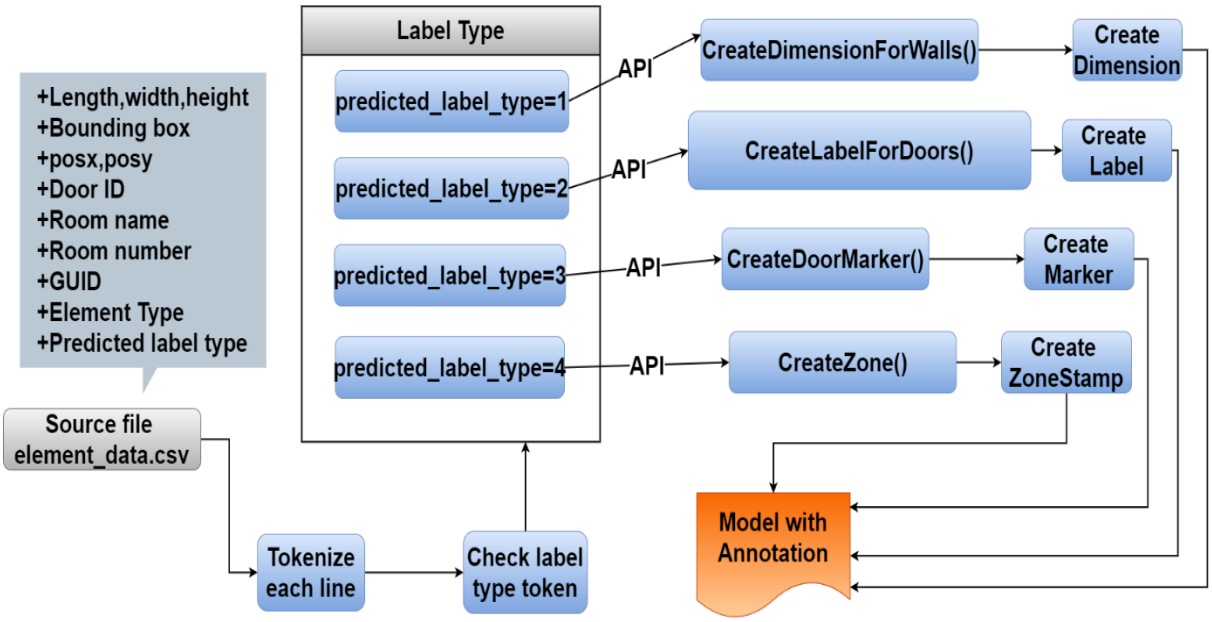


Figure 4.7: ServCAD automatic creation of annotation algorithm.

5 Results and Analysis

5.1 API functions

An essential element in automating the process of annotating architectural models was the incorporation of ServCAD with API functions[1], a list of which may be found in Table 5.1. Through a series of specific API calls, the system processed elements inside the architectural program, registered custom menus, and installed menu handlers to generate and manipulate data quickly.

Table 5.1: C++ API Functions

API Function	Explanation
ACAPI_MenuItem_RegisterMenu	Registers custom menus in ArchiCAD. Used to register custom menus for the Add-On.
ACAPI_MenuItem_InstallMenuHandler	Installs menu handler callbacks. Used to associate menu items with corresponding handler functions.
ACAPI_Element_GetElemList	Used to get lists of dimension elements, walls, slabs, zones, doors, etc., for further processing.
ACAPI_Element_Get	Used to get detailed information about a specific element, such as its properties and attributes.
ACAPI_Element_CalcBounds	Calculates the bounding box of an element. Used to determine the spatial extent of elements for reporting and analysis.
ACAPI_Element_Delete	Deletes elements specified by their GUIDs. Used to delete dimensions, labels, zones, and other annotations.
ACAPI_Element_GetMemo	Retrieves additional information (memo) associated with an element.
ACAPI_DisposeElemMemoHdls	Disposes of memory handles allocated for element memos. Used to clean up memory after processing element memos.
ACAPI_Element_GetElementInfoString	Retrieves the compound info string associated with an element.
ACAPI_Grouping_GetConnectedElements	Retrieves elements connected to a given element.
ACAPI_CallUndoableCommand	Executes an undoable command. Used to wrap the execution of command functions to ensure proper undo/redo functionality.
ACAPI_WriteReport	Writes a report string to the ArchiCAD Report window.
ACAPI_GetOwnResModule	Retrieves the resource module handle of the Add-On. Used to access resources (e.g., strings, icons) stored in the Add-On's resource file.
ACAPI_Element_GetDefaults	Retrieves default properties for a given element type.
ACAPI_Element_GetDefaultsExt	Retrieves default properties for a given element type with extended options.
ACAPI_Element_Create	Creates a new element with the provided properties. Used to create dimensions, labels, zones, and other architectural elements.
ACAPI_Element_CreateExt	Creates a new element with extended options. Used to create detail elements and markers.
ACAPI_CreateAttributeIndex	Creates a new attribute index for element properties. Used to set the category index for zones.
BMAllocateHandle	Allocates a new handle in the memory manager.
APIGuidFromString	Converts a string representation of a GUID to a GUID object.
GS::snuprintf	A safe version of sprintf that works with GS::UniString. Used to format room names and numbers into UniString objects for zone creation.
GS::Array	A dynamic array container provided by the ArchiCAD API.
GS::UniString	Used to handle Unicode text, such as room names and numbers for zone creation.

[1] API functions: https://graphisoft.github.io/archicad-api-devkit/functions.html

The C++ API had functions that allowed one to connect similar items by aligning them with anticipated labels to automate annotations, construct new elements with specific attributes, and retrieve element properties.

5.2 User interface (UI) in the ArchiCAD

Data extraction and annotation are made easy by ServCAD through a menu located inside the ArchiCAD interface. Features with names like 'Automatic Annotation, ' 'Delete ADZL', and 'Extract BE' are easily accessible. Although the 'Automatic Arrangement' feature is included in the menu, it is not examined in this thesis. Its inclusion is intended to be a work in progress; while its functionality is not yet complete, this suggests that ServCAD's potential for improvement and expansion may be realized in future projects. Figure 5.1 shows the location of the ServCAD in the BIM authoring tools, ArchiCAD 27.

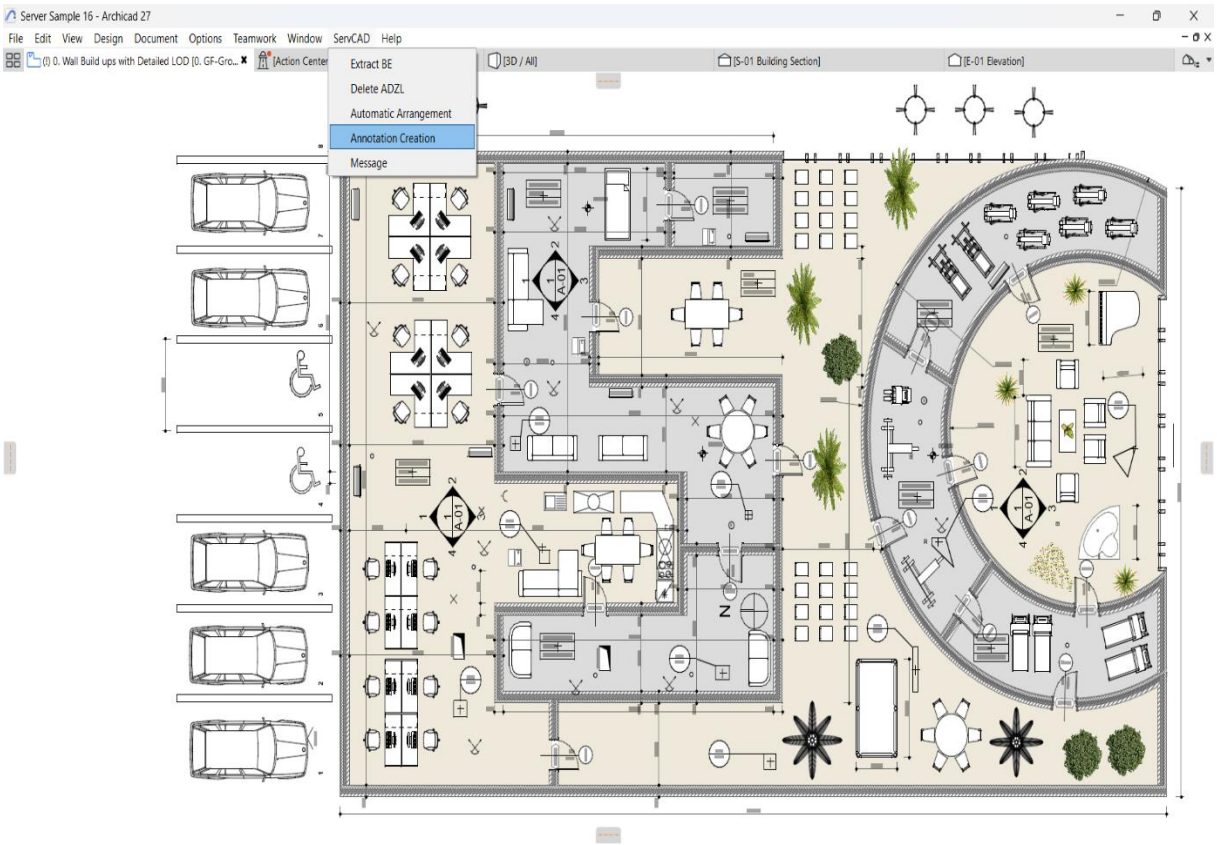


Figure 5.1: ServCAD UI in the Archicad 27

5.3 Hardware and software

To complete my thesis, a wide variety of hardware and software were used; Figure 5.2 provides specifics. A desktop computer and a laptop were employed to accommodate

the different stages and requirements of the investigation. The laptop was more portable and convenient with its Intel(R) Core(TM) i7-8550U CPU and 16GB RAM, but the desktop with its NVIDIA Quadro P2000 graphics card and 32GB RAM was a better option due to its higher processing capability. This allowed for a thorough testing of several GNN model architectures and hyperparameters.

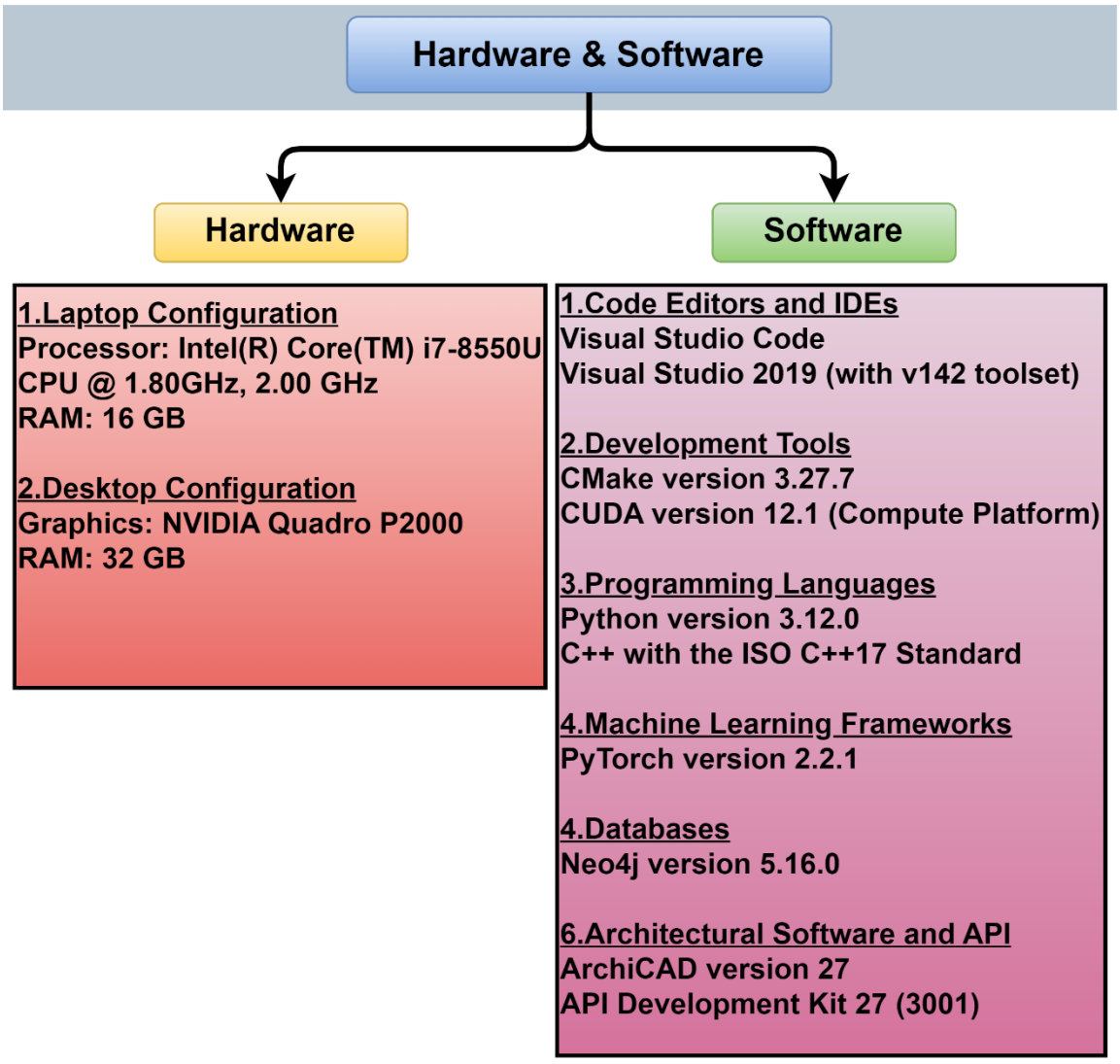


Figure 5.2: Hardware and software.

It showed that the computational difference between the CPU and GPU setups had no discernible impact on the outcomes for the same hyperparameters. For the purposes of this thesis, the results of the CPU-based setup were therefore employed. Because of the GPU's greater processing capability for more complicated architectural drawing with more nodes and label types, the choice may shift in the future in favor of employing it.

5.4 Training the GNN Model

The GNN model was developed using the methods and presumptions listed in Table 5.2. To help with the optimization process, the Adam optimizer[1] which is renowned for its adaptable LR characteristics was employed. It started out with a value of 0.01. Class weights determined from the frequency of each class in the training dataset were added to the CrossEntropyLoss[2] function, which was applied equitably to the model's loss calculations to resolve any potential class imbalance. To further prevent overfitting, L2 regularization which has a lambda value of 0.001 was employed. A batch size of 16 was used to process the data. The model training schedule, with a provisional limit of 750 epochs, included an early stopping mechanism based on validation performance to reduce overtraining and promote model generalization.

Table 5.2: GNN model definition and proposed techniques

Component	Description
Optimizer	Adam
Learning rate (Initial)	0.01
Loss function	CrossEntropyLoss
Class weights	based on the frequency of each class in the training data
L2 regularization	with a lambda value of 0.001
Batch size	16
Epochs	750 (Maximum,with early stopping if condtions are met.)

To optimize training efficiency, the LR scheduler parameters for the GNN model were configured as shown in Table 5.3. The 'min' mode was selected, indicating that the LR would be lowered if a validation loss did not improve. The LR would be reduced to a tenth of its present value whenever the requirement was met because the LR reduction factor was set at 0.1. The model was given 30 epochs of patience, which meant that it may train for a predefined period of time before changing the LR in the event that the

[1] The Adam optimizer, also known as 'Adaptive Moment Estimation', is an iterative optimization algorithm that reduces the loss function when NN are being trained.
[2] Cross-entropy loss is employed when training a supervised learning algorithm for classification tasks. It is applied when changing the model weights in a training. Minimizing the loss is the goal; a smaller loss indicates a better model.

validation loss did not improve. Lastly, the verbose setting was used to offer clarity and insights into the training dynamics. This made sure that updates on LR modifications and training status were reported.

Table 5.3: LR schedular

Parameter	Description
Mode	min (Reduce LR when a metric has stopped improving.)
Factor	0.1 (Factor by which the LR will be reduced.)
Patience	30 (Number of epochs with no improvements after which LR will be reduced)
Verbose	True

The sections that follow discuss how the GCN and GAT models trained in comparison. Comprehensive analyses comprising confusion matrices, accuracy assessments, and loss curves are used to explain the models' results.

5.4.1 Training the GCN Model

Training and validation curves:

Figure 5.3 displays the training and validation loss metrics for the GCN model over several epochs. At the beginning, there is a noticeable decrease in training and validation loss, which suggests quick learning. As the epochs proceed, the losses level off, indicating the model's convergence and stabilization. The closeness of the training and validation loss lines suggests that the model has strong generalization and is not overfitting to the training set.

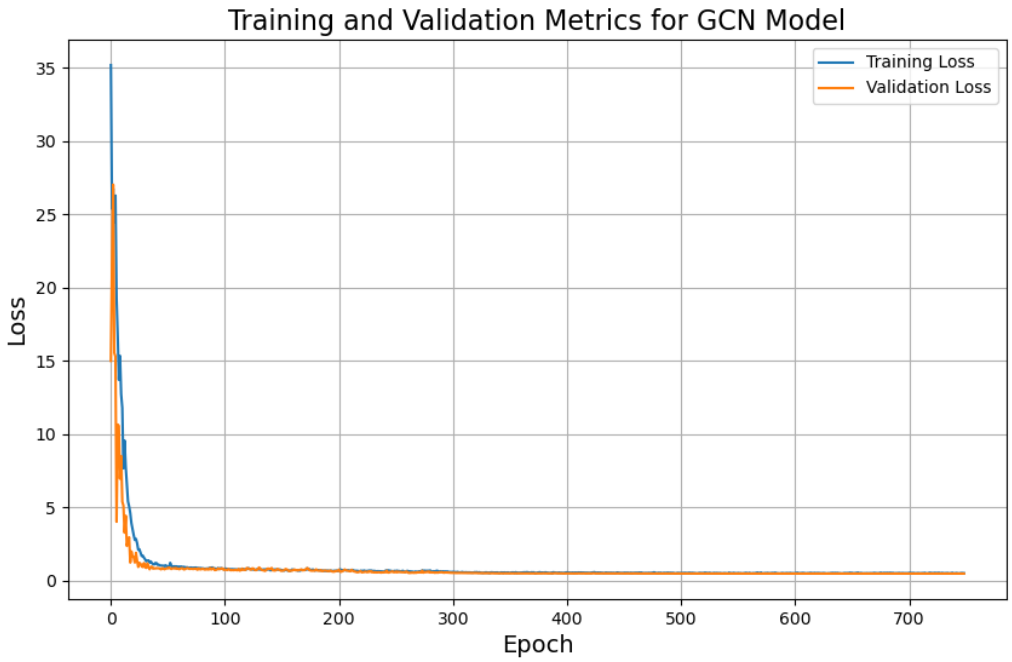


Figure 5.3: Training and validation loss over epochs for GCN.

The validation accuracy plot of the GCN model (refer to Figure 5.4) exhibits a positive trend over the course of the training epochs. After a period of rapid advancement, the model's precision stabilizes at an elevated level, suggesting that it can generalize from the training set to the validation set with effectiveness. The elevated precision plateau is an indication of the model's constant performance over time and shows that it has been successful in capturing the essential features needed for dependable prediction.

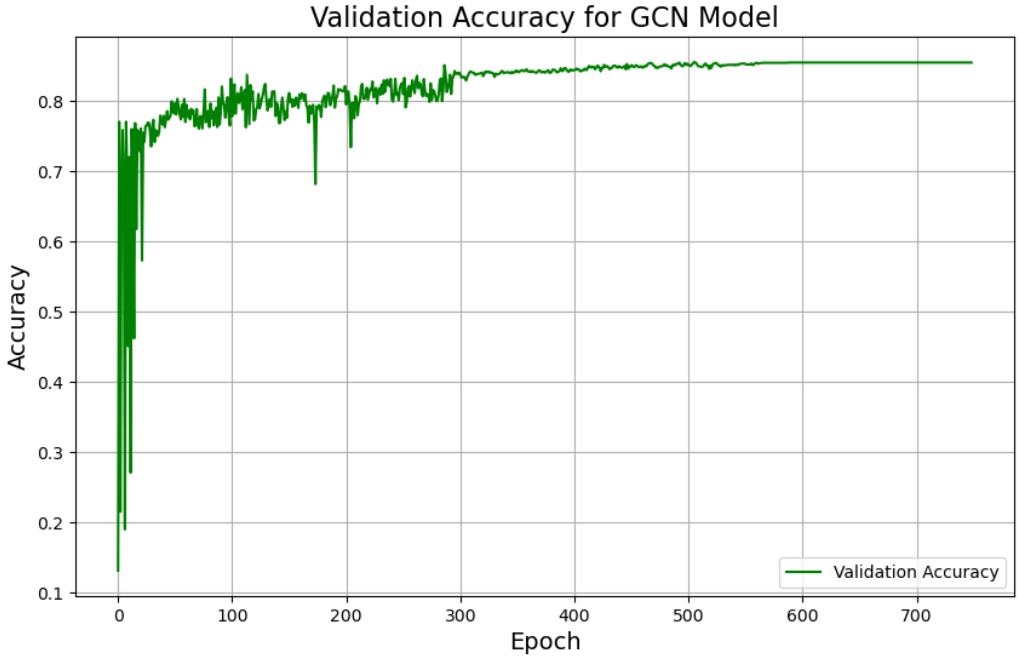


Figure 5.4: Validation accuracy over epochs for GCN

Class accuracy and confusion matrix:

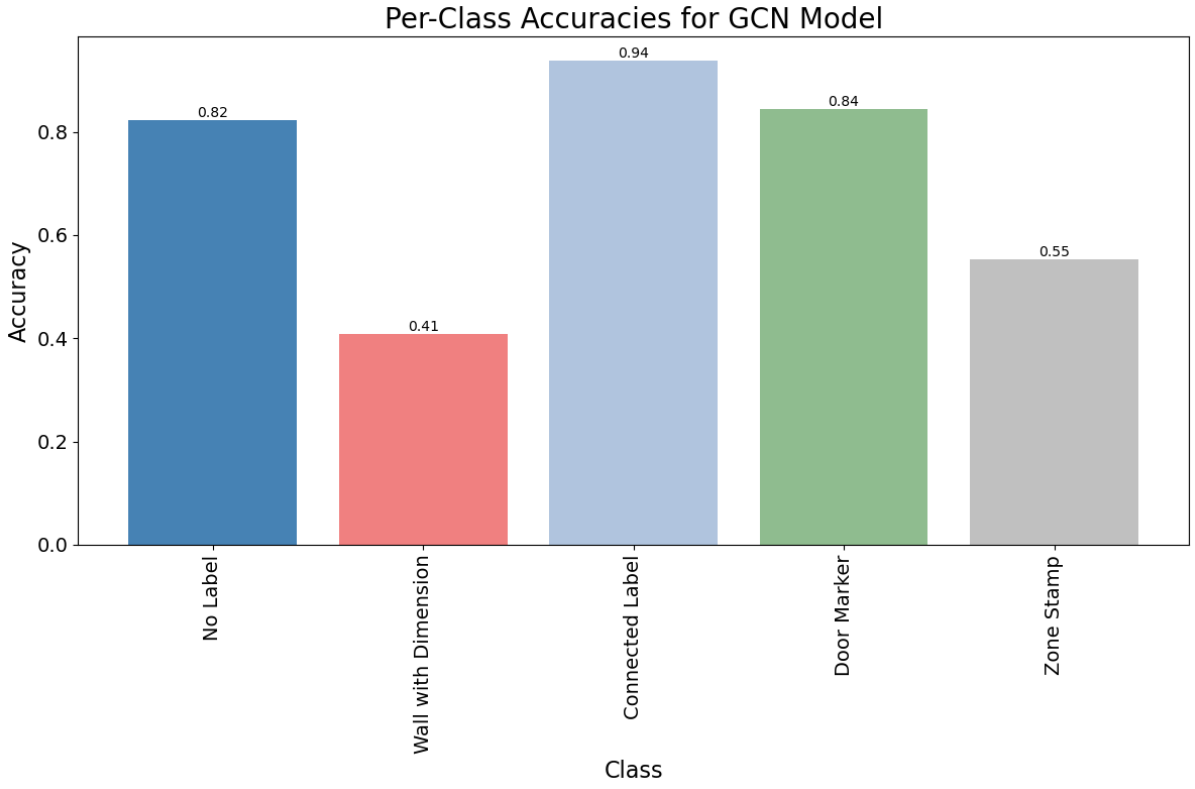


Figure 5.5: Label classification accuracy for GCN model

Figure 5.5 shows the per-class precision histogram of the GCN model, which shows how different categories function. 'No Label', 'Connected Label' and 'Door Marker' classes exhibit exceptional accuracy, while 'Wall with Dimension' and 'Zone Stamp' classes exhibit lower accuracy. This distribution of accuracies is most likely caused by the class imbalance in the training data, where the model is more frequently exposed to some classes and hence becomes more proficient in recognizing them.

Moreover, the number of precise estimates for the GCN model is displayed by the numbers in the diagonal cells of the confusion matrix (see Figure 5.6). These cells demonstrate the model's capability to accurately identify these classifications, with high numbers for 'No Label', 'Connected Label' and 'Door Marker' and reasonable counts for classes like 'Wall with Dimension' and 'Zone Stamp ' as it was also illustrated in class accuracy histogram. On the other hand, the diagonal illustrations for 'Zone Stamp' and 'Connected Label' are comparatively lower, suggesting that these areas require further model improvement.

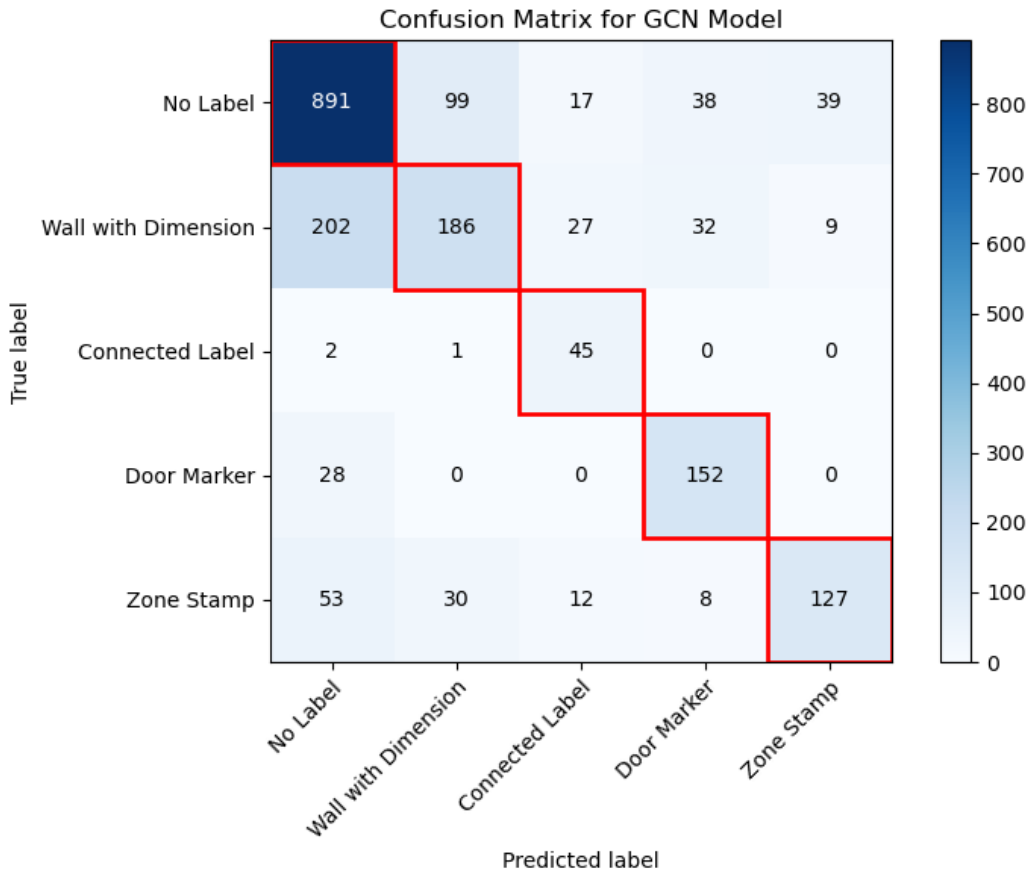


Figure 5.6: Confusion matrix for GCN model.

5.4.2 Training the GAT Model

Training and validation curves:

Like the GCN model, the GAT model exhibits an initial dramatic decrease in training and validation losses as depicted in the Figure 5.7, followed by a plateau that signifies the model has achieved a stable point of learning. The constant validation loss indicates that the GAT model, similar to the GCN model, effectively learns from the training data without overfitting. This is evident from the tight connection between the loss metrics of the two models.

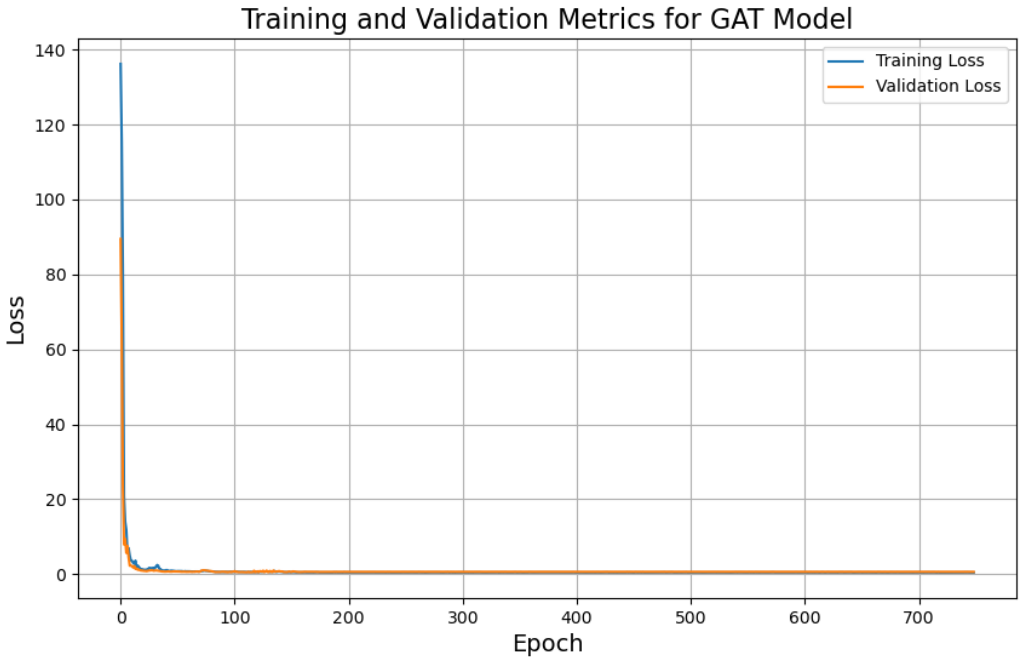


Figure 5.7: Training and validation loss over epochs for GAT.

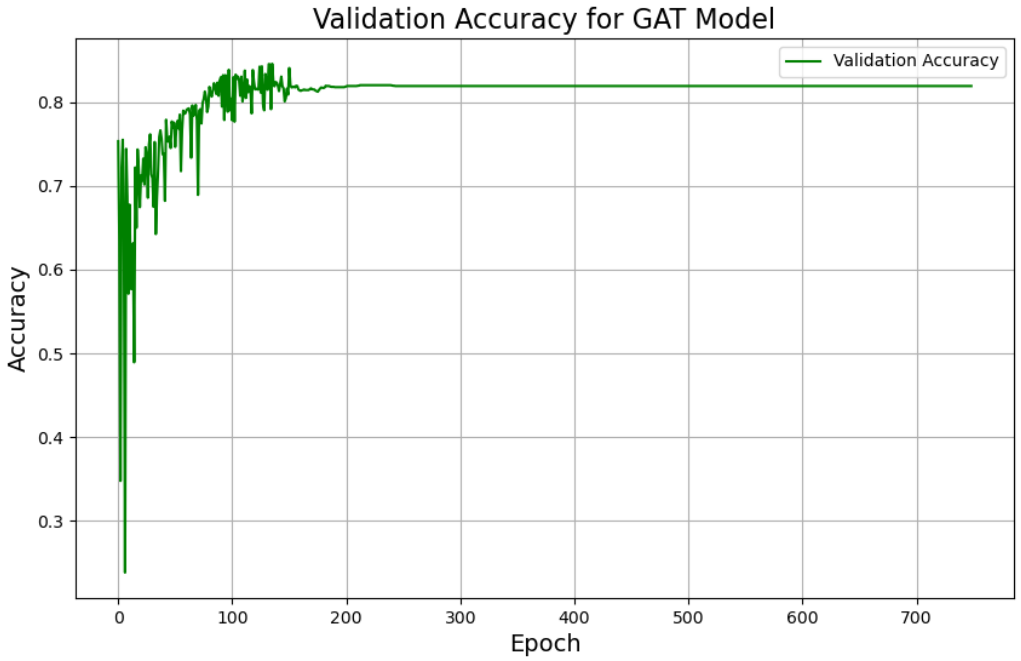


Figure 5.8: Validation accuracy over epochs for GAT

With respect to validation accuracy as provided in Figure 5.8, the GAT model maintains a consistently high level of accuracy throughout the training process, which is consistent with the excellent performance reported in the GCN model. Overall, both models achieve similar high validation accuracies, with the GCN model exhibiting somewhat more stable learning dynamics.

Class accuracy and confusion matrix:

Regarding the GAT model, the per-class accuracies indicate that certain classes, such as 'No Label', 'Connected Label' and 'Door Marker' have high levels of accuracy. In fact, 'Connected Label' achieves perfect classification (100 %), as shown in Figure 5.9. This suggests that the attention processes, which focus on important elements of the graph framework, can assist the GAT model in effectively differentiating between these categories. Nevertheless, several classes like 'Wall with Dimension' and 'Zone Stamp' exhibit noticeably lower accuracy rates, which may be attributed to various factors, including class imbalances.

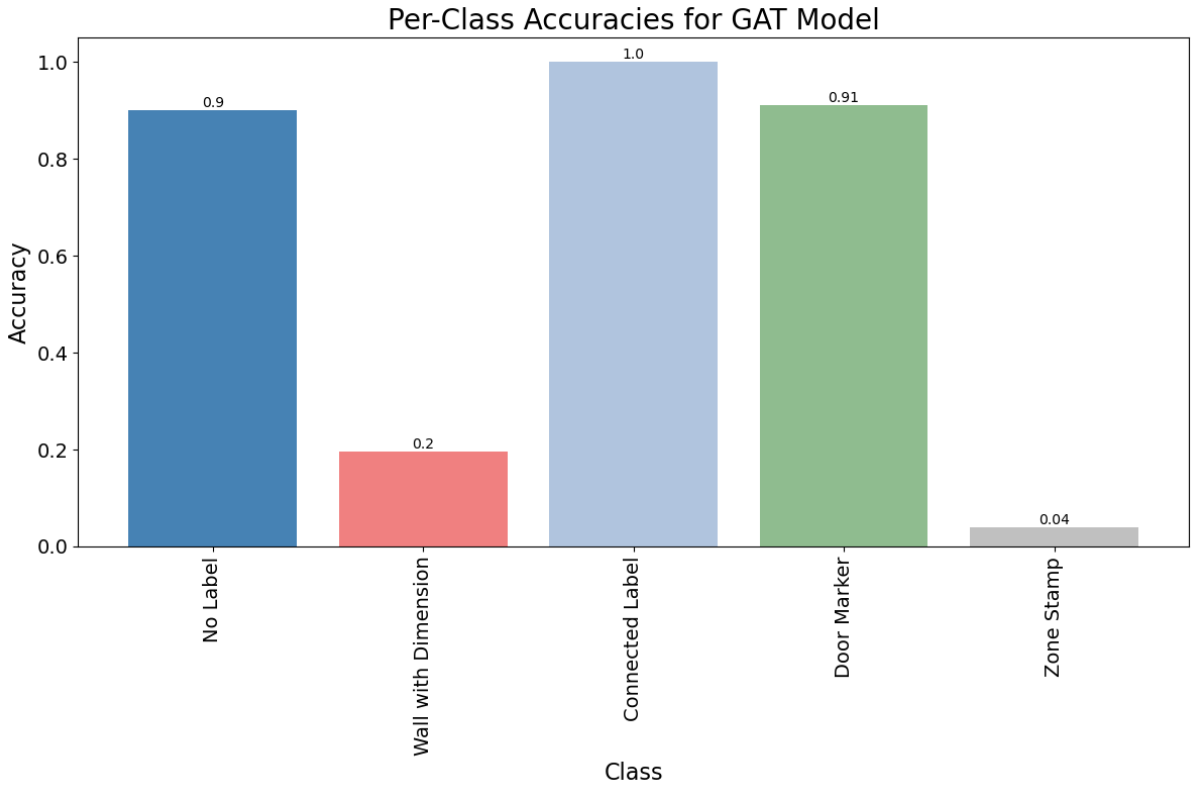


Figure 5.9: Label classification accuracy for GAT model.

The confusion matrix in Figure 5.10 of the GAT model demonstrates a substantial number of accurate predictions, suggesting that the majority of classes have been classified with precision. Predictions of the class distribution indicate a significant improvement over the GCN model specifically for label types 0,2 and 3. However, there is some degree of uncertainty, particularly with the differentiation between the 'No Label' category and other labels, and to a lesser degree regarding the predictions related to 'Zone Stamp'. These incidents of misclassification indicate potential avenues for further

development of the model, such as enhanced training procedures, data rebalancing, or architectural enhancements.

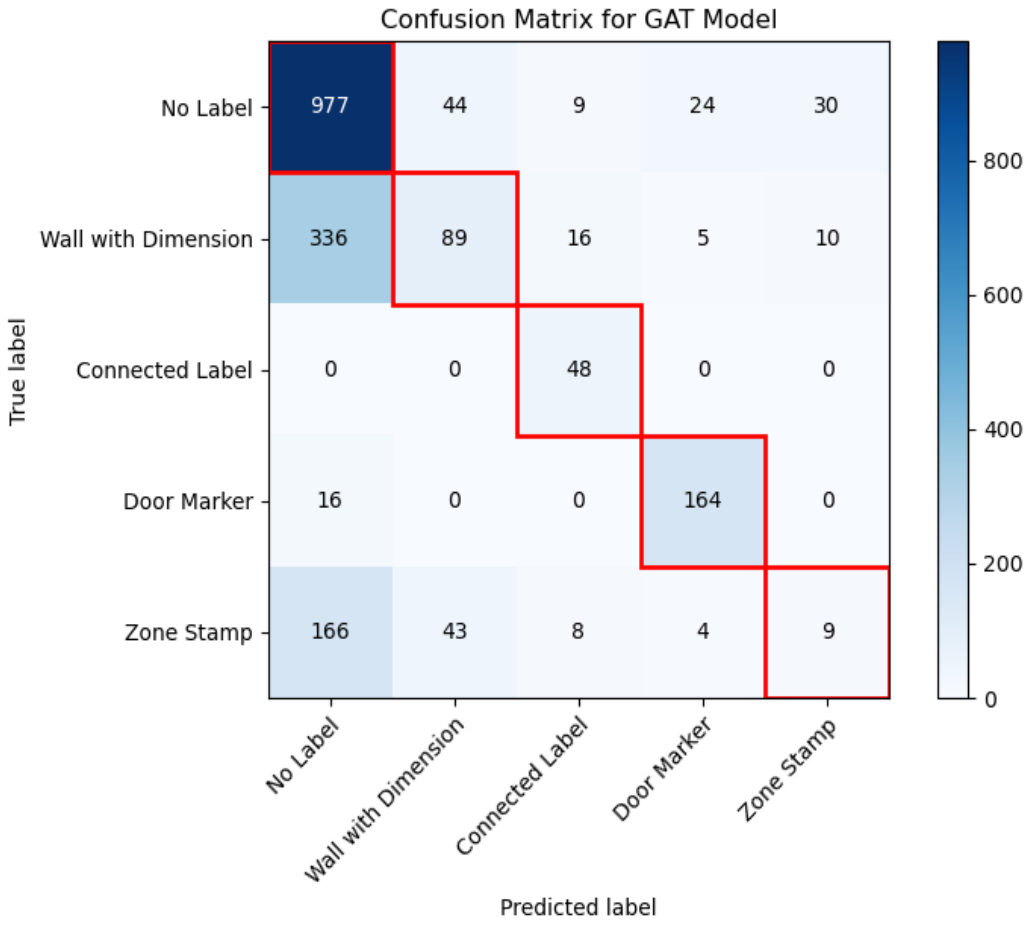


Figure 5.10: Confusion matrix for GAT model

Additionally, Table 5.4 shows the classification accuracy of various attention heads, ranging from 4 to 24, within our model for different class types. Notably, attention head 22 was found to be the most effective among them all, consistently delivering better performance across several classes. The highest accuracies were achieved for 'Connected Label (2) ' at 100%, and it was among the top performers for 'Door Marker (3)' and 'No Label (0) '. Given these results, attention head 22 was selected for detailed reporting.

Table 5.4: Class accuracy for different attention heads for GAT model

Attention Heads / Class Types	4	8	10	14	18	20	22	24
No Label (0)	93%	91%	93%	93%	88%	92%	90%	93%
Wall with Dimension (1)	10%	11%	9%	8%	26%	17%	20%	14%
Connected Label (2)	98%	96%	81%	88%	92%	88%	100%	96%
Door Marker (3)	2%	6%	9%	4%	91%	88%	91%	91%
Zone Stamp (4)	1%	0%	1%	0%	1%	2%	4%	3%

A detailed discussion of the potential causes of the variations in training results between the models will be presented in Chapter 6, which will shed light on how much better GNN predictions could be made.

5.5 Automatic creation of annotation

The results of the automated annotation procedure for the example projects are presented in this section. The prediction results for the test graphs will demonstrate how the annotating capabilities of the model are applied in practical situations, giving a clear image of how the model's predictions are translated into annotations in the architectural drawings.

Three architectural drawings are shown in this Figure 5.11, Figure 5.12, Figure 5.13, Figure 5.14 to illustrate the annotation capabilities of our ServCAD. The ground truth drawing, which offers comprehensive annotations and dimensions used as the reference standard and is shown in the bottom right panel. The raw model, which is provided in the top portion of the figures without any annotations, is created using this ground truth. This raw model is subjected to the automatic annotation procedure that is carried out using ServCAD. The bottom left panel displays the automated annotation results, which demonstrate how the GNN predictions were mapped back to the architectural environment.

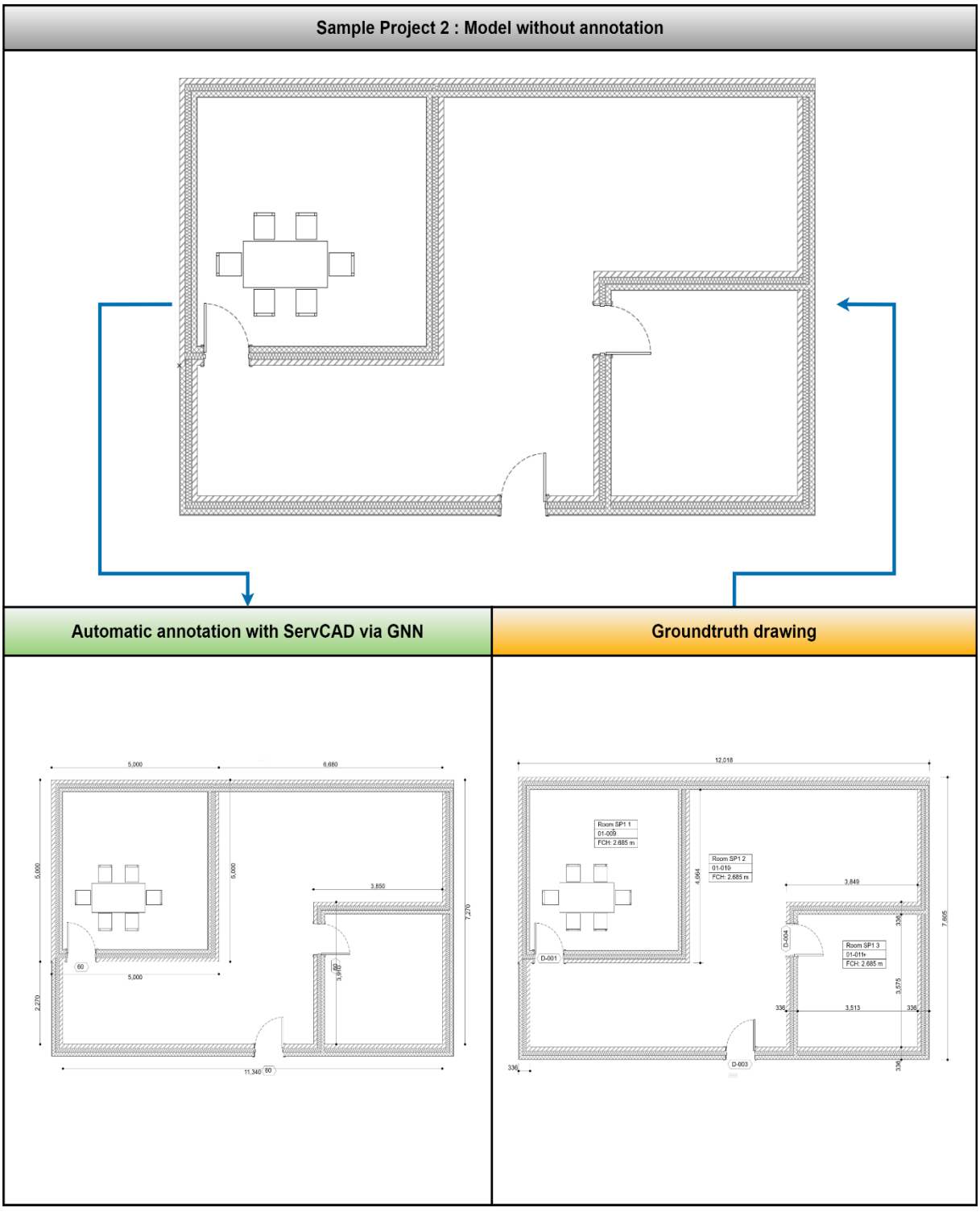


Figure 5.11: Result of the automatic annotation for Sample Project 2 with ServCAD

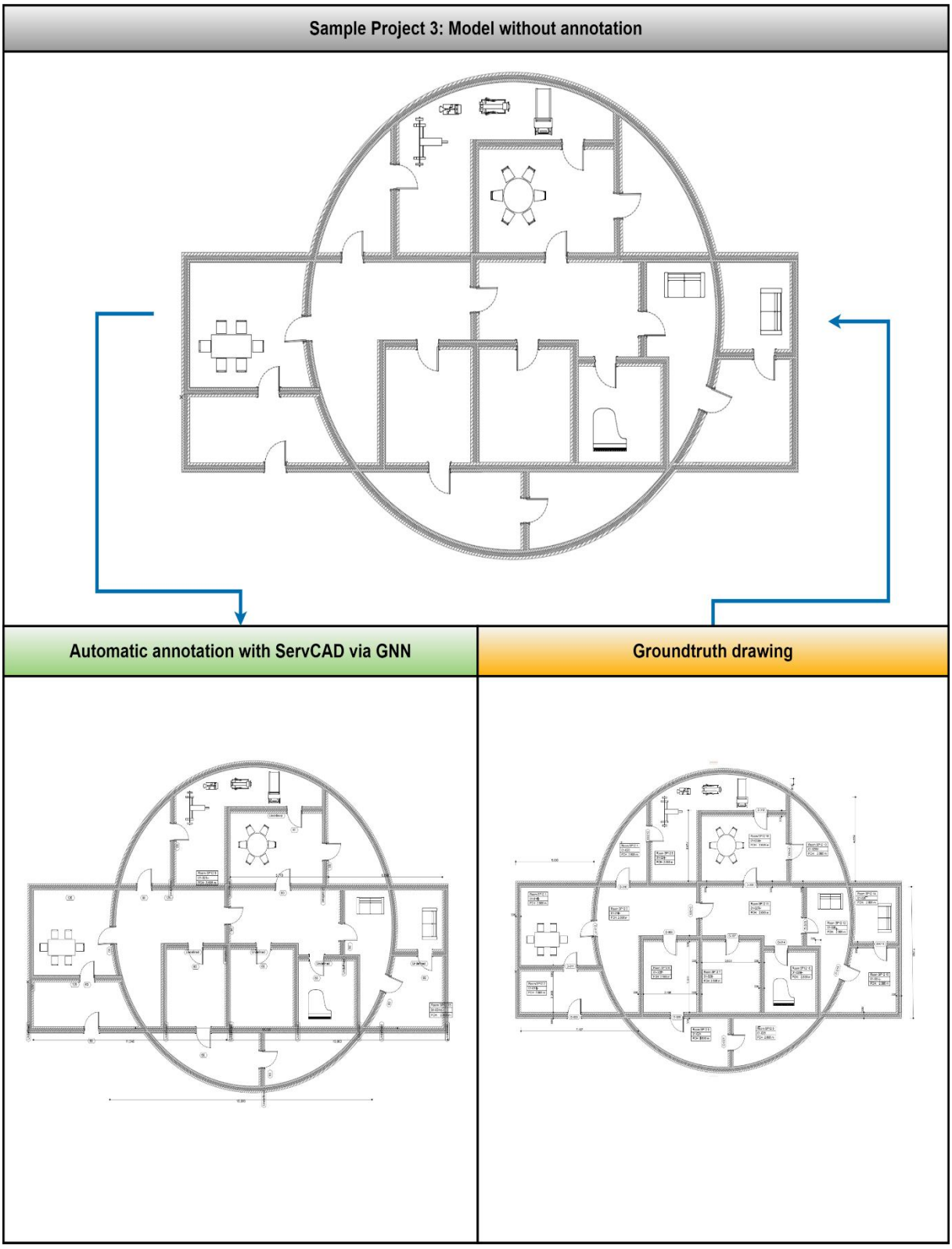


Figure 5.12: Result of the automatic annotation for Sample Project 3 with ServCAD

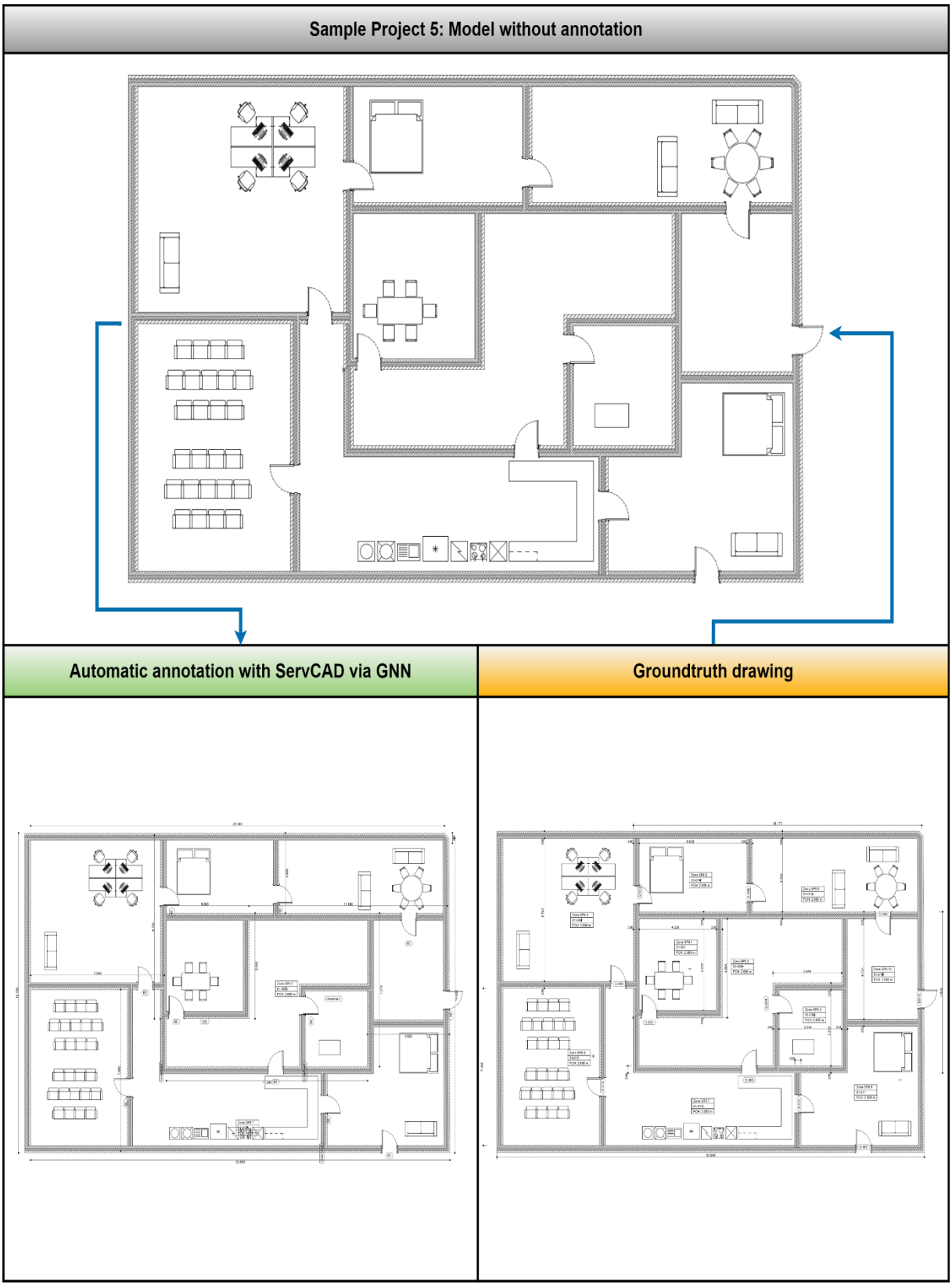


Figure 5.13: Result of the automatic annotation for Sample Project 5 with ServCAD

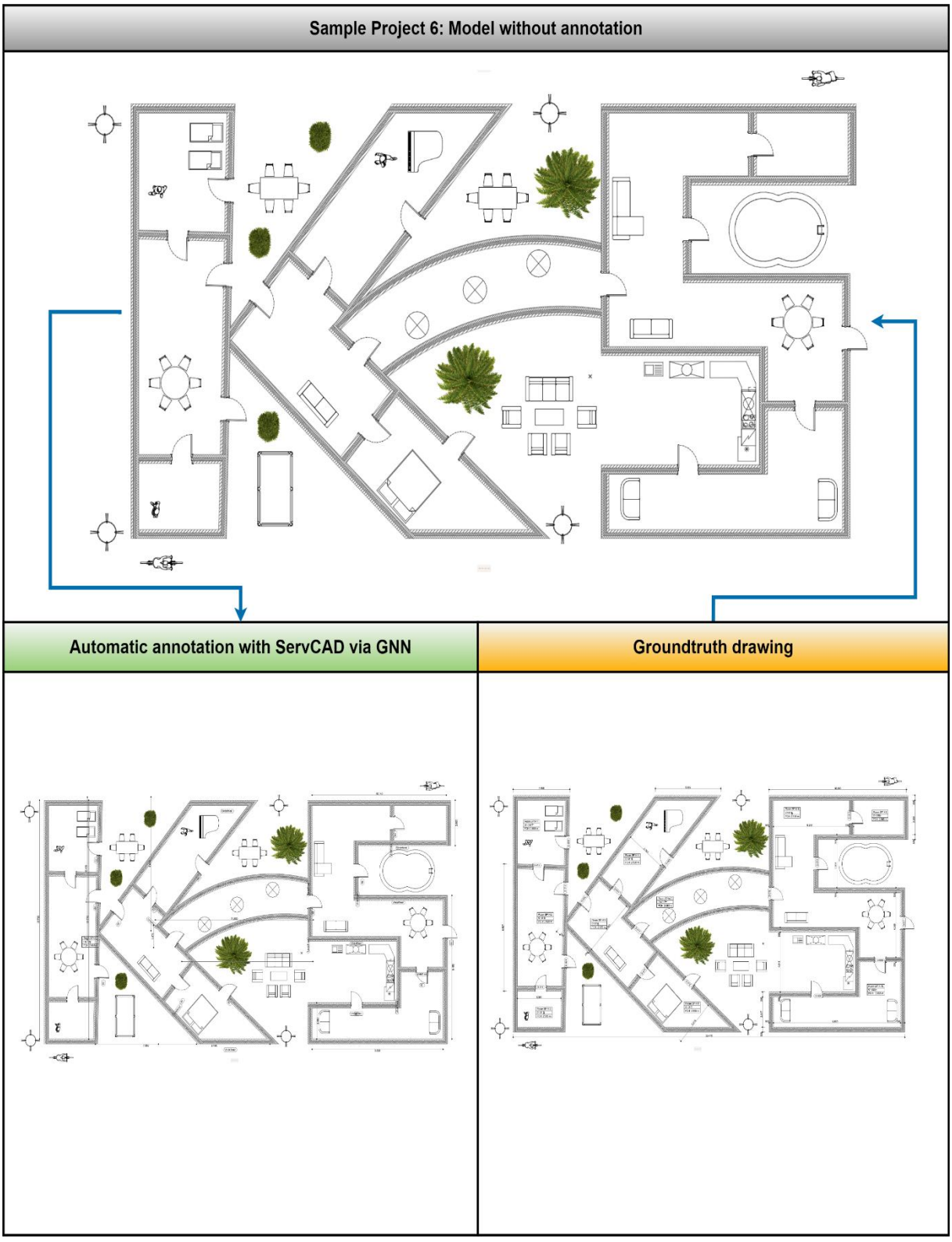


Figure 5.14: Result of the automatic annotation for Sample Project 6 with ServCAD

6 Discussion and Limitations

6.1 Discussion

This study investigates the relatively uncharted topic of AI in architectural drafting while also adding to the existing understanding in response to the research questions presented in section 1.2. Architectural drawings are intricate and multi-layered, requiring a level of understanding and interpretation that is challenging for existing methods to give without machine learning and AI underpinnings.

This research offers a significant step towards enhancing automatic annotation processes by merging GNNs, powered by machine learning and AI, to make sketching more visually appealing and context aware. By focusing on automatic annotation using GNNs, this study contributes to resolving current technical limitations and imagines a future where AI technologies will significantly accelerate the architectural design process from inception to completion. Since machine learning is inherently scalable, these approaches can be expanded to more complicated designs in the future. This adaptability demonstrates the broader revolutionary potential of the study and establishes the foundation for future advancements in architectural design and annotation at various levels of complexity.

The data generating process we employ confirms the efficacy of the ServCAD system in efficiently extracting data from the architectural environment. Data extraction was successful not just for our initial sample project (Figure 1.1), but also for an additional 22 projects (7 from official ArchiCAD, 15 projects self-drawn) of varying complexity. This accomplishment showcases ServCAD's proficiency in navigating architectural data and accurately capturing crucial element relationships and information. To guarantee the data acquired is relevant and coherent, the procedure adhered strictly to established assumptions, focusing on significant architectural components such as walls, slabs, zones, and doors, while omitting others.

This methodical approach to data collection highlights ServCAD's precision and agility in obtaining pertinent information from complex architectural projects. The accuracy of the example projects was clarified by visualizing the nodes and edges with neo4j. Appendix A.1 and Appendix A.2 showed the visualization of the certain graphs. As we already mentioned, the Appendix A.1 was our foundation for the machine learning part.

Appendix A.2 also includes the annotation nodes to show the true picture of example projects to inspire the future works. The distinctive data extraction needs of our GNN models were successfully met, establishing a solid foundation for training and testing the models using real architectural data. The efficacy of ServCAD in this scenario not only showcases the potential of integrating SOTA machine learning techniques into architectural drawings, but also underscores the need of targeted data extraction methods in facilitating substantial AI applications inside BIM workflows.

The investigation of GNNs, with a specific emphasis on GCNs and GATs, for the purpose of predicting label categories in technical drawings has produced valuable and enlightening results. Both models underwent a thorough evaluation under similar conditions, including the same hyperparameters and graphs for training, validation, and testing. This was done to ensure a fair comparison between the two models. The results suggest that the GAT model demonstrates higher performance in reliably predicting certain label types (see Figure 5.9). GAT model achieved the highest accuracies for 'Connected Label (2)' with 100% and for 'No Label (0)' and 'Door Marker (3) ', it has the 90 % and 91% accuracy respectively. This is explained by the GAT model's capacity to dynamically determine each node's relative importance within the graph, providing a more complex interpretation and representation of the architectural drawings. On the other hand, GCN shows better performance to predict more label types (see Figure 5.5) with reasonable amount.

However, the low performance of the GAT model on certain labels such as 'Wall with Dimension' and 'Zone Stamp' can be largely because of class imbalance and the nature of the neighborhood structure of the graph data. The effectiveness of GAT models is deeply influenced by the neighborhood structure, as these models use the relationships and interactions between nodes to make predictions. If these features are not effective in differentiating the classes, this could lead to lower performance of the model. Given the limitation of needing to fair compare results between the GAT and GCN models without extensive hyperparameter modifications, it would be beneficial to explore GAT specific settings that could potentially address these issues.

6.2 Limitations

Data Generation

Our reliance on the ArchiCAD C++ API was a major limitation during the data generation phase, despite all of its features. Its capacity to record all the features needed for

our models is restricted. The range of architectural elements we could include was restricted by the function ACAPI_Grouping_GetConnectedElements(), which only worked with specific element types: API_WindowID, API_DoorID, API_SkylightID, API_LabelID, and API_OpeningID. Limiting the extraction of bounding boxes for dimension elements and dimension text which are represented by a single point, also reduced the amount of detail in our dataset. Notwithstanding these limitations, the C++ API proved to be really beneficial in procuring particularized facts for our inquiry. The Python API became less appropriate, particularly when it comes to capturing complicated element relationships, because it is more current and has more constraints than the C++ API.

Moreover, our research was based on an analysis of a model project that included basic architectural elements such the zones, slabs, doors, and walls depicted Figure 1.1. The vast number of labels and annotations found in more complex architectural drawings is not included in this as a part of the thesis. As a result, the present API version has limitations regarding the automatic labeling in more complicated designs. However, our created algorithms are scalable and could be applied in the future to more complex architectural drawings, similar to ServCAD. Since our main interests are significant building elements and annotations, this adaptability is very crucial.

Model Architecture and Training

Data from 22 projects were added to our dataset for model training, some of which featured highly complex drawings with a range of floor plans and architectural elements. The goal of this project was to improve the dataset's comprehensiveness, with a focus on a variety of wall kinds, including interior partitions and outside, fire-rated, and both. Dimensions were given the type 1 name, indicating their relationship to dimensions and their essential function in dimensioning, because they are mostly associated with outside walls in the architectural context. However, a significant portion of the information had a label class '0', suggesting that certain walls had no dimensions, which created a clear imbalance in the label classification.

For final results, CPU based system was used (see Figure 5.2). In particular, this hardware choices helped to efficiently handle GCNs and GATs. It is important to acknowledge that, even with the careful comparison of the GCN and GAT models under identical hyperparameter and training conditions, the research into potential model performance might have been constrained by the focus on these two models alone, as

well as the specific hyperparameter configuration. Regarding accuracy in prediction and model efficiency, alternative GNN architectures or even a different hyperparameter configuration might produce better results.

Additionally, as the confusion matrixs in Figure 5.6 and Figure 5.10 illustrates, the training results exhibit variations in prediction accuracy between various label classes. This volatility is mostly caused by the imbalanced nature of our dataset, especially when it comes to dimension connections with various wall kinds. By introducing the class weights, the models demonstrated a unique capacity to offset these imbalances and offer a more accurate label prediction across several architectural components. However, particularly in GAT model, some label classes ('1' and '4') accuracies are low. This could be because the limitation that we had for model architecture, special attention on the GAT model with increasing layer size and other hyperparameters (such as batch size, attention heads), may result in better performances.

This study shows the importance of label distribution imbalances, and it also suggests that future research should examine a wider variety of GNN models and hyperparameter configurations in order to achieve better results. Moreover, larger-scale trials could be made possible by advancements in computational hardware and optimization algorithms, which could lead to the discovery of more effective techniques for the automatic annotation of intricate architectural models.

Integration into Architectural Environment

There were several challenges in reintegrating the predictive capabilities of the models into the architectural environment, chief among them being limitations in the API used to replicate dimensions and annotations in architectural software. We were able to effectively build wall dimensions and zone and door markers using the API, however we encountered problems when attempting to further modify and enhance the look of these components. For example, when we attempted to enhance the appearance of dimensioned objects, texts, and labels, the capability of the API did not satisfy our expectations.

These shortcomings highlight the need for a more robust and flexible API that can manage the intricate adjustments necessary for improved architectural drawings. The ability to automatically produce dimensions for significant architectural elements, such as walls and markers for zones and doors, was an amazing accomplishment in spite of these difficulties. The limits of the existing API, however, emphasize how important

it will be to update the program in the future to have a genuinely integrated system that can handle the demands of intricate architectural projects. These gaps could be filled by providing additional options for modifying and enhancing the appearance of dimensioned items and annotations.

As architectural drawings and documentation move more and more toward digital and automated procedures, APIs will need to change to accommodate more intricate changes. These advancements would enhance the overall caliber and use of digital architectural drawings in addition to simplifying the integration of AI-generated comments into architectural workflows. Expectations for more powerful versions of APIs reflect a broader industry trend toward improving the flexibility and interoperability of digital design tools, which is necessary for the effective use of AI technology in the architectural domain.

7 Conclusion

7.1 Summary

This thesis extensively examined several important questions regarding the application of advanced machine learning techniques, especially GNNs, to improve architectural design processes. By addressing the questions, the current study has significantly improved the field and created methods that have the potential of fostering the adoption of increasingly sophisticated and automated architectural drafting procedures. The primary algorithm of the study, ServCAD, demonstrated its effectiveness in both the data production stage and the annotation creation process based on predicted label types, which made it beneficial for automating and enhancing architectural design operations. The answers to the research questions from section 1.2 are provided below,

1. Information about building and annotation properties, such as dimensional properties and annotation objects like dimensions and labels, were successfully extracted using specialized C++ API functions. Table 5.1 provides a detailed overview of the API function that supports the functionality of the ServCAD algorithm and outlines the various functions and capabilities of this API. This table contains a thorough documentation of the API functions used in the study.
2. During the data generation phase, a large amount of C++ API utilized for the successful automation of the label categorization process as it also illustrated in the Figure 4.2. By using the identified C++ API, methods were developed that automatically associate labels with matching building elements, ensuring a seamless transmission of relevant data for GNN training.
3. The C++ API was helpful to get interconnectivity between architectural elements and annotations. This information improved the dataset for GNN training by enabling accurate label and dimensional property mapping. As it already provided in the Table 5.1, some functions were useful such as ACAPI_Grouping_GetConnectedElements() to get relationship been element and connected label.
4. The study explored how to configure a GNN to accurately predict label types by optimizing the network architecture, fine-tuning hyperparameters, and ensuring the quality and relevance of the training data that is extracted from the architectural environment.

5. Annotation creation was made automated by reintegrating GNN predictions into the architectural environment. Figure 5.11, Figure 5.12, Figure 5.13 and Figure 5.14 shows how the ServCAD system can dynamically generate concise and contextually relevant annotations within architectural drawings, exemplifying how well it can automate the development of annotations.

This thesis provides valuable knowledge and practical approaches to enhance the automation and intelligence of the architectural drafting and design processes, effectively addressing the difficulties it raised. The development and implementation of the ServCAD algorithm, reinforced by the tacit use of a C++ API, has demonstrated the potential of GNNs for automatic annotation of architectural drawings. This achievement is a critical step toward the digital transformation of architectural practices, as well as a means of generating chances for further research and development in the field of architecture. In summary, this study represents a major advancement in the digital transformation of architectural drawing and documentation workflows by highlighting the potential of GNNs to improve the automation and intelligence of BIM procedures.

7.2 Contributions

The creation of the ServCAD plugin for ArchiCAD is a notable addition to the CAD community as well as an important part of the thesis. It becomes clear that ServCAD is a strong and adaptable tool that can extract a wide range of data from architectural drawings, even those with complex and intricate structures. ServCAD demonstrated its efficacy in precisely capturing the dimensional and relational features of architectural elements through rigorous development and testing.

Furthermore, this thesis explored the use of GNNs to predict label types in architectural drawings, which are a component of intricate architectural compositions, which is a novel application. Moreover, the ServCAD algorithm has shown effectiveness in producing raw models from annotated architectural drawings, indicating that its potential goes beyond data extraction. This feature creates opportunities for improved architectural analysis in addition to making it easier to test innovative annotation strategies. The successful generation of dimensions, zone stamps, and door labels effectively shows how ServCAD is implemented efficiently to close the gap between design and documentation.

7.3 Future Works

In terms of further analyzing and utilizing the architectural drawings, ServCAD's future development holds great potential. One important thing that needs to be done is to make ServCAD more flexible so that it can handle the complexity of contemporary architectural drawings. With continued technological progress and API development, ServCAD can be enhanced to capture complex relational structures and extract more data from architectural drawings. ServCAD can grow to analyze a greater variety of architectural elements by going beyond the analysis of standard elements like walls, slabs, zones, and doors. This will increase the tool's usefulness and suitability in a variety of architectural contexts.

In addition, future versions of ServCAD could focus on including sophisticated functionalities meant to automate several aspects of architectural analysis and visualization. Apart from its current features of data generation, raw model creation, and automatic annotation, ServCAD can be improved to expedite the arrangement process for improved annotation visualization. In addition to increasing productivity, ServCAD's automation of these processes gives architects and designers a better understanding of architectural compositions, which aids in well-informed decision-making and design improvement. These developments ultimately establish ServCAD as a complete and essential tool in the toolbox of architectural professionals, leading to creativity and quality in architectural analysis and design.

8 References

Bond, A. H., & Ahmed, S. Z. (1989). Knowledge-based automatic dimensioning. *Artificial Intelligence in Engineering*. https://doi.org/10.1016/0954-1810(89)90023-X

Buric, M., Brcic, M., & Skec, S. (2021). Towards Automated Drafting in CAD Systems. *2021 4th International Conference on Electronics and Electrical Engineering Technology*, 233–238. https://doi.org/10.1145/3508297.3508335

Castro-Cañas, L., Pavón-Domínguez, P., Castro-Cañas, L., & Pavón-Domínguez, P. (2023). *Dimensioning Method for 3D Modelling*. https://doi.org/10.1007/978-3-031-20325-1_52

Chen, K.-Z., Feng, X.-A., & Lu, Q.-S. (2001). Intelligent dimensioning for mechanical parts based on feature extraction. *Computer-Aided Design*, *33*(13), 949–965. https://doi.org/10.1016/S0010-4485(00)00132-9

Chen, K.-Z., Feng, X.-A., & Lu, Q.-S. (2002). Intelligent location-dimensioning of cylindrical surfaces in mechanical parts. *Computer-Aided Design*, *34*(3), 185–194. https://doi.org/10.1016/S0010-4485(01)00076-8

Ciribini, A. L. C., Mastrolembo Ventura, S., & Paneroni, M. (2016). Implementation of an interoperable process to optimise design and construction phases of a residential building: A BIM Pilot Project. *Automation in Construction*, *71*. https://doi.org/10.1016/j.autcon.2016.03.005

Dai, H., Kozareva, Z. L., Dai, B., Smola, A. J., & Song, L. (2018). Learning steady-states of iterative algorithms over graphs. *35th International Conference on Machine Learning, ICML 2018*, *3*.

Drake, P. J. (1999). *Dimensioning and tolerancing handbook*. McGraw-Hill New York.

Elgabry, A. K. (1986). A framework for a solid-based tolerance analysis. *ASME Proceedings for Computers in Engineering Conference*, *2*, 79–84.

G, A., & G, K. (2017). On the Automatic Dimensioning of Technical Drawings. *Sixth International Conference on Advances in Mechanical and Robotics Engineering - AMRE 2017*, 30–34. https://doi.org/10.15224/978-1-63248-140-5-41

Gehring, J., Auli, M., Grangier, D., & Dauphin, Y. N. (2017). A convolutional encoder model for neural machine translation. *ACL 2017 - 55th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)*, *1*. https://doi.org/10.18653/v1/P17-1012

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*. https://doi.org/10.1109/CVPR.2016.90

Hillyard, R. C., & Braid, I. C. (1978). Analysis of dimensions and tolerances in computer-aided mechanical design. *Computer-Aided Design*, *10*(3). https://doi.org/10.1016/0010-4485(78)90140-9

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., & Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition. *Ieee Signal Processing Magazine*, *November*. https://doi.org/10.1109/MSP.2012.2205597

Huang, M. Q., Ninić, J., & Zhang, Q. B. (2021). BIM, machine learning and computer vision techniques in underground construction: Current status and future perspectives. *Tunnelling and Underground Space Technology*, *108*. https://doi.org/10.1016/j.tust.2020.103677

Jegou, S., Drozdzal, M., Vazquez, D., Romero, A., & Bengio, Y. (2017). The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, *2017-July*. https://doi.org/10.1109/CVPRW.2017.156

Kelly, T., Wonka, P., & Mueller, P. (2015). Interactive Dimensioning of Parametric Models. *Computer Graphics Forum*, *34*(2), 117–129. https://doi.org/10.1111/cgf.12546

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.

Li, C. L., Lee, Y. H., & Yu, K. M. (2006). Automatic datum dimensioning for plastic injection mould design and manufacturing. *The International Journal of Advanced Manufacturing Technology*. https://doi.org/10.1007/S00170-004-2374-2

Li, G., Muller, M., Thabet, A., & Ghanem, B. (2019). DeepGCNs: Can GCNs go as deep as CNNs? *Proceedings of the IEEE International Conference on Computer Vision*, *2019-October*. https://doi.org/10.1109/ICCV.2019.00936

Li, Y., Zemel, R., Brockschmidt, M., & Tarlow, D. (2016). Gated graph sequence neural networks. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*.

Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*. https://doi.org/10.18653/v1/d15-1166

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*. https://doi.org/10.1109/CVPR.2016.91

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6). https://doi.org/10.1109/TPAMI.2016.2577031

Requicha, A. (1977a). *Mathematical models of rigid solid objects*.

Requicha, A. (1977b). *Part and assembly description languages: I--dimensioning and tolerancing*.

Requicha, A. A. G., & Chan, S. C. (1986). Representation of Geometric Features, Tolerances, and Attributes in Solid Modelers Based on

Constructive Geometry. *IEEE Journal on Robotics and Automation*, *2*(3). https://doi.org/10.1109/JRA.1986.1087053

Roberto Raffaeli, P. C., & Germani, M. (2016). Automation of drafting execution by schemes definitions and feature recognition. *Computer-Aided Design and Applications*, *13*(4), 459–470. https://doi.org/10.1080/16864360.2015.1131538

Roy, U., & Liu, C. R. (1988). Feature-based representational scheme of a solid modeler for providing dimensioning and tolerancing information. *Robotics and Computer-Integrated Manufacturing*. https://doi.org/10.1016/0736-5845(88)90004-X

Roy, U., Liu, C. R., & Woo, T. C. (1991). Review of dimensioning and tolerancing: representation and processing. *Computer-Aided Design*. https://doi.org/10.1016/0010-4485(91)90045-X

Ruemler, S. P., Zimmerman, K. E., Hartman, N. W., Hedberg, T., & Barnard Feeny, A. (2017). Promoting model-based definition to establish a complete product definition. *Journal of Manufacturing Science and Engineering, Transactions of the ASME*, *139*(5). https://doi.org/10.1115/1.4034625

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, *20*(1). https://doi.org/10.1109/TNN.2008.2005605

Simonovsky, M., & Komodakis, N. (2017). Dynamic edge-conditioned filters in convolutional neural networks on graphs. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017-January*. https://doi.org/10.1109/CVPR.2017.11

Solihin, W., Eastman, C., & Lee, Y. C. (2016). A framework for fully integrated building information models in a federated environment. *Advanced Engineering Informatics*, *30*(2). https://doi.org/10.1016/j.aei.2016.02.007

Sperduti, A., & Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, *8*(3). https://doi.org/10.1109/72.572108

Standard, A. (1982). Y14. 5M (1982) Dimensioning and Tolerancing. *American Society of Mechanical Engineers, New York, NY*.

Veličković, P., Casanova, A., Liò, P., Cucurull, G., Romero, A., & Bengio, Y. (2018). Graph attention networks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*. https://doi.org/10.1007/978-3-031-01587-8_7

Villena Toro, J., & Tarkian, M. (2022, August 14). Automated and Customized CAD Drawings by Utilizing Machine Learning Algorithms: A Case Study. *Volume 3B: 48th Design Automation Conference (DAC)*. https://doi.org/10.1115/DETC2022-88971

Wang, S.-C., Xu, J.-Y., Wang, L.-N., Fang, X.-F., & Sun, H.-R. (2022). *Research on Automatic Marking Method of 3D Model Based on Geometric Feature Similarity*. https://doi.org/10.1007/978-981-16-7381-8_54

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., & Solomon, J. M. (2019). Dynamic graph Cnn for learning on point clouds. *ACM Transactions on Graphics*, *38*(5). https://doi.org/10.1145/3326362

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(1). https://doi.org/10.1109/TNNLS.2020.2978386

Yang, J., Lu, J., Lee, S., Batra, D., & Parikh, D. (2018). Graph R-CNN for Scene Graph Generation. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11205 LNCS*. https://doi.org/10.1007/978-3-030-01246-5_41

Yu, K. M., Tan, S., & Yuen, M. M. F. (1994). A review of automatic dimensioning and tolerancing schemes. *Engineering With Computers*. https://doi.org/10.1007/BF01200178

Yuen, M. M. F., Tan, S., & Yu, K. M. (1988). Scheme for automatic dimensioning of CSG defined parts. *Computer-Aided Design*. https://doi.org/10.1016/0010-4485(88)90023-1

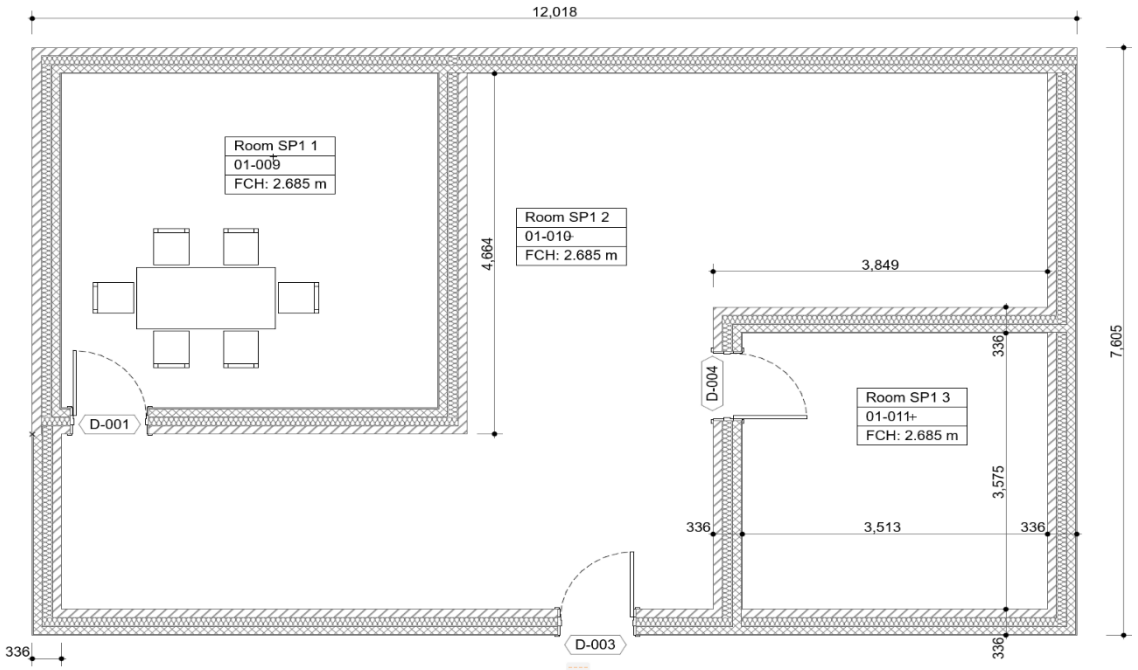
Zabin, A., González, V. A., Zou, Y., & Amor, R. (2022). Applications of machine learning to BIM: A systematic literature review. *Advanced Engineering Informatics*, *51*. https://doi.org/10.1016/j.aei.2021.101474

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

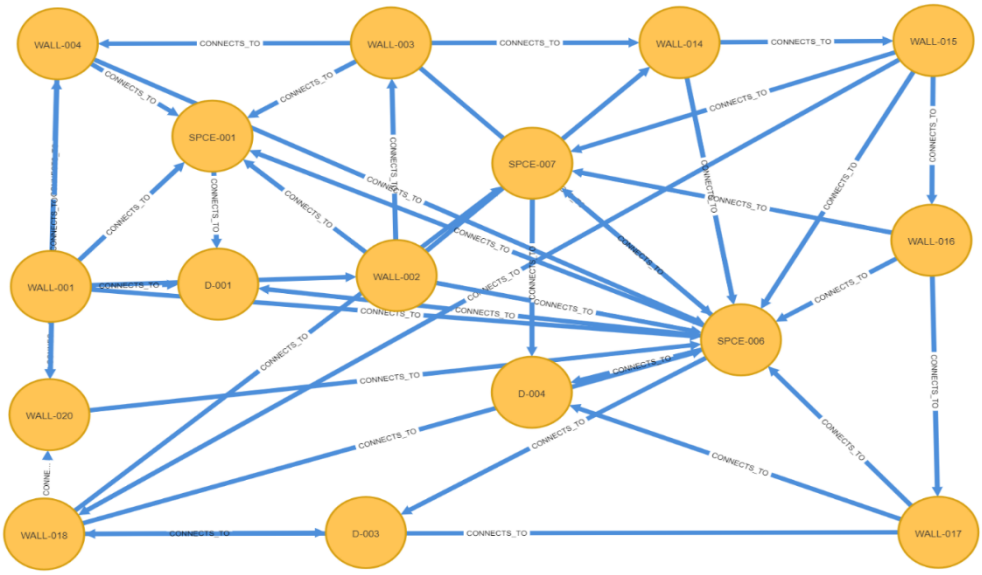
9 Appendix A

Appendix A.1: Graph visualization for sample projects used for machine learning part without annotations.

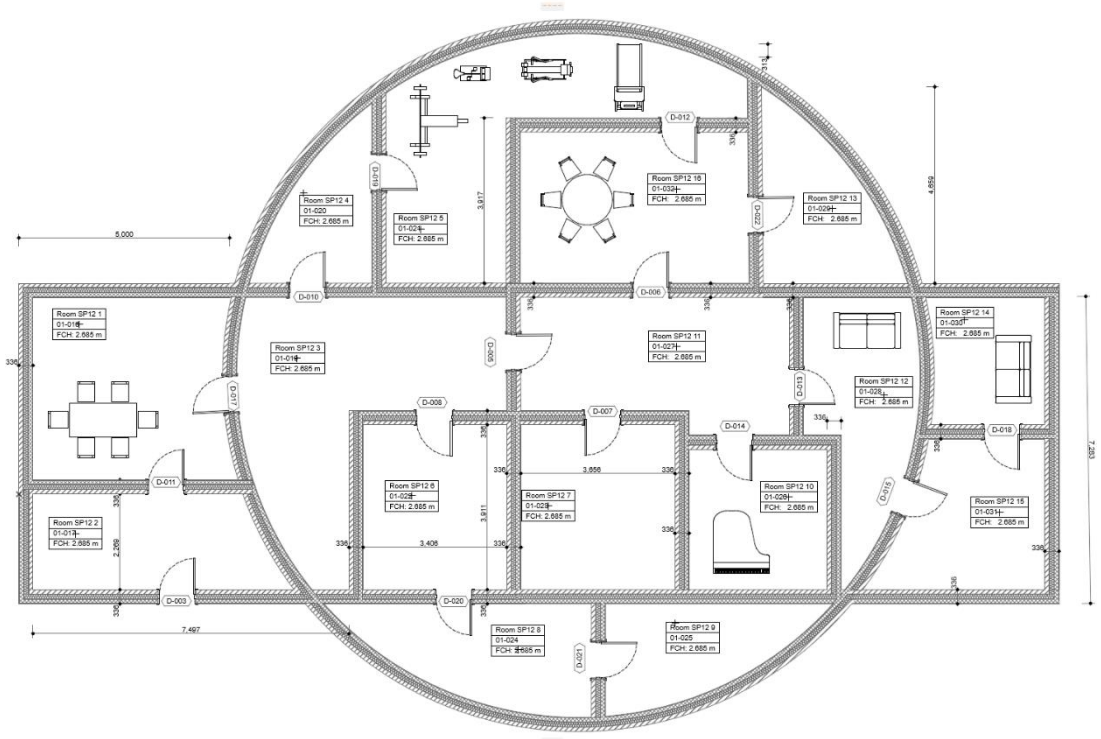
Sample Project 2: Original Drawing



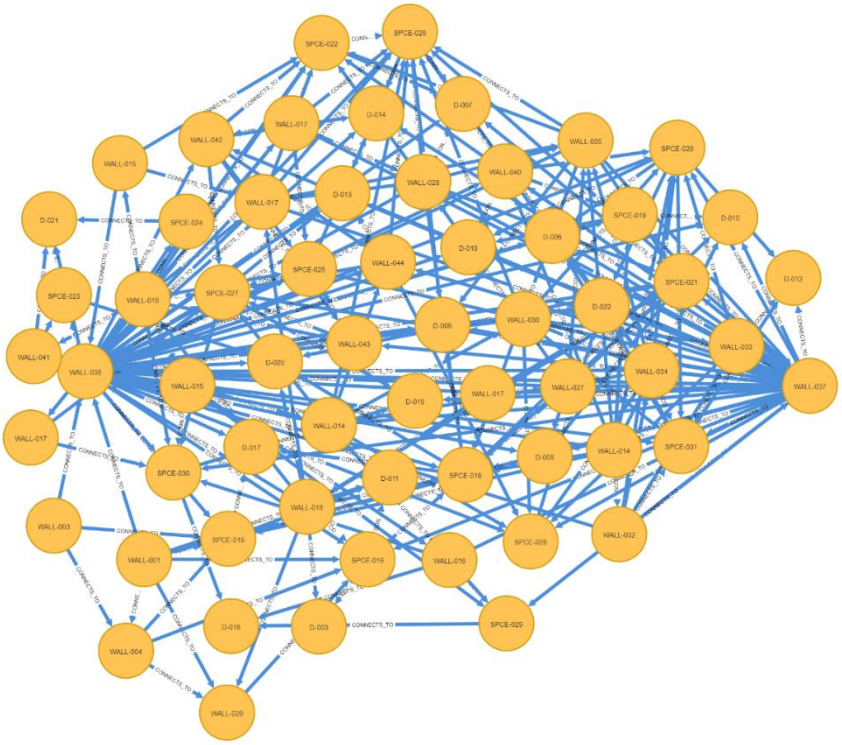
Graph generation by neo4j



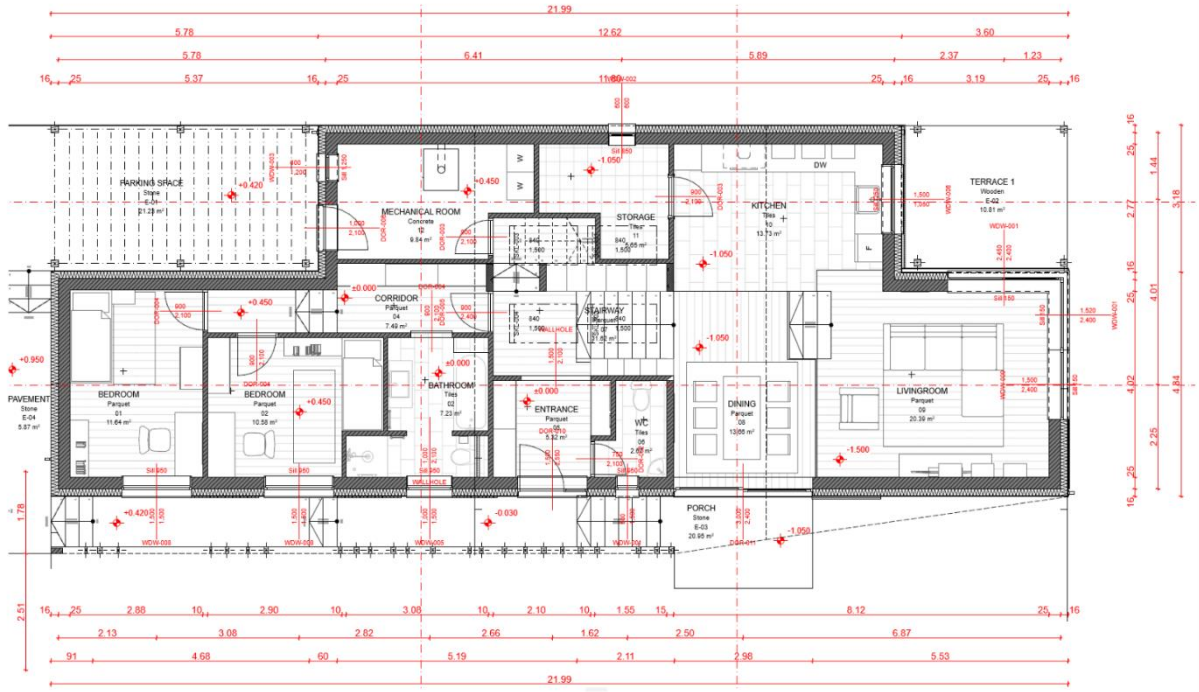
Sample Project 3: Grountruth Drawing



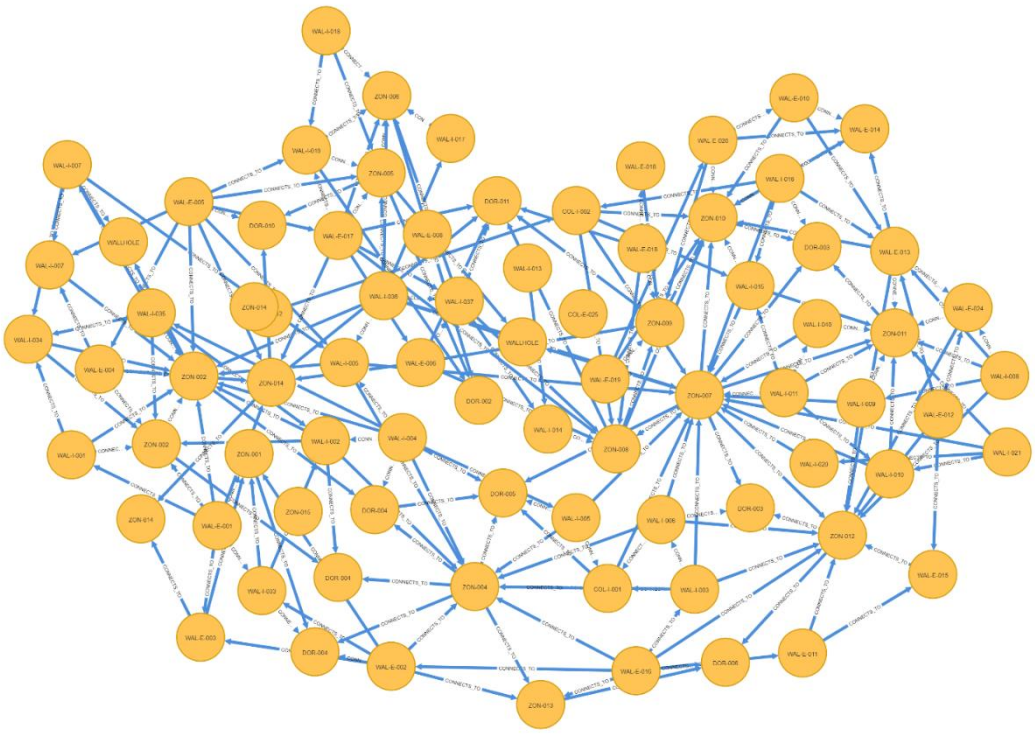
Graph generation by neo4j



Sample Project 4: Groundtruth Drawing

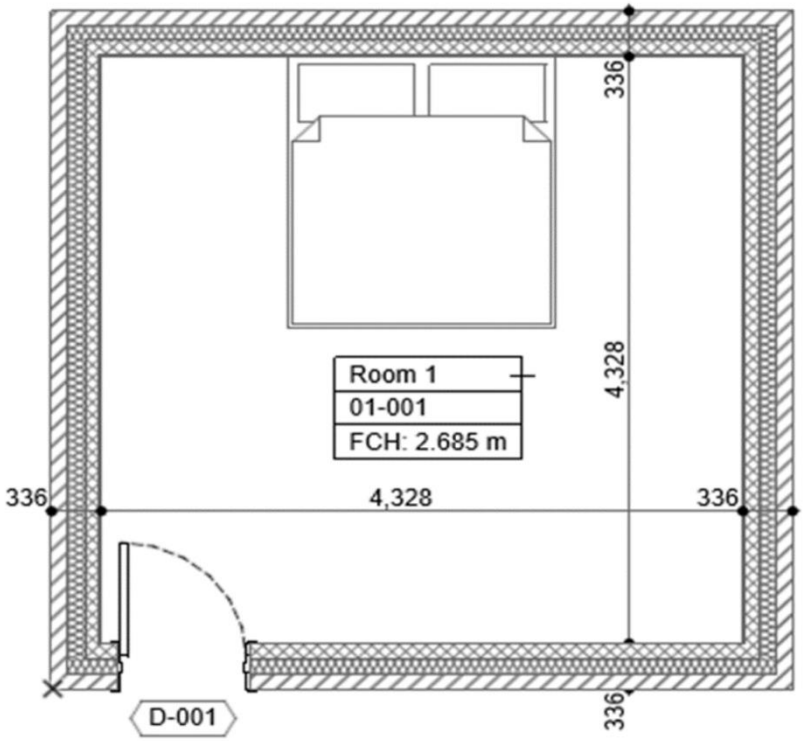


Graph generation by neo4j

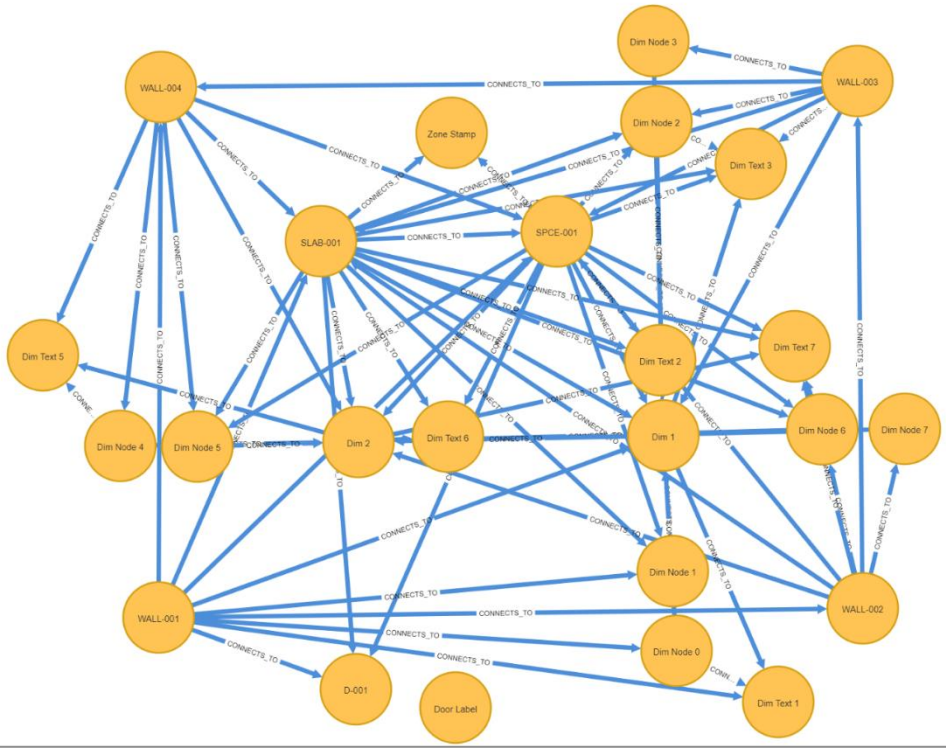


Appendix A.2: Graph visualization for sample projects including annotation nodes

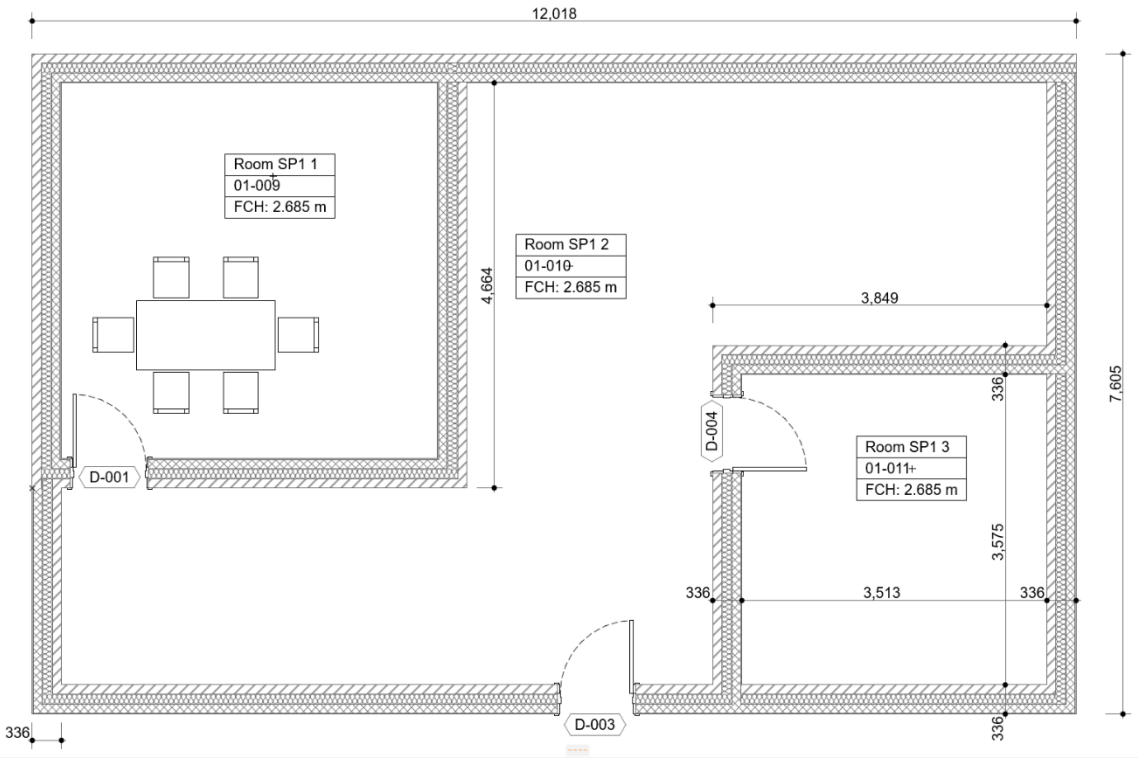
Sample Project 1: Groundtruth Drawing



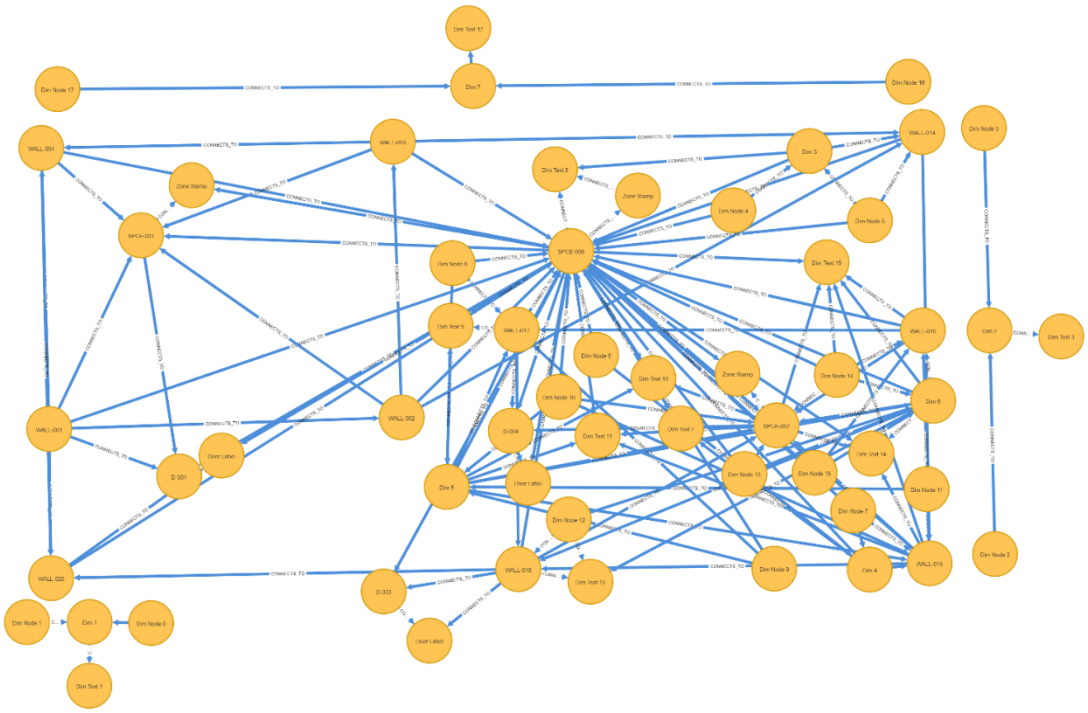
Graph generation by neo4j including annotation nodes



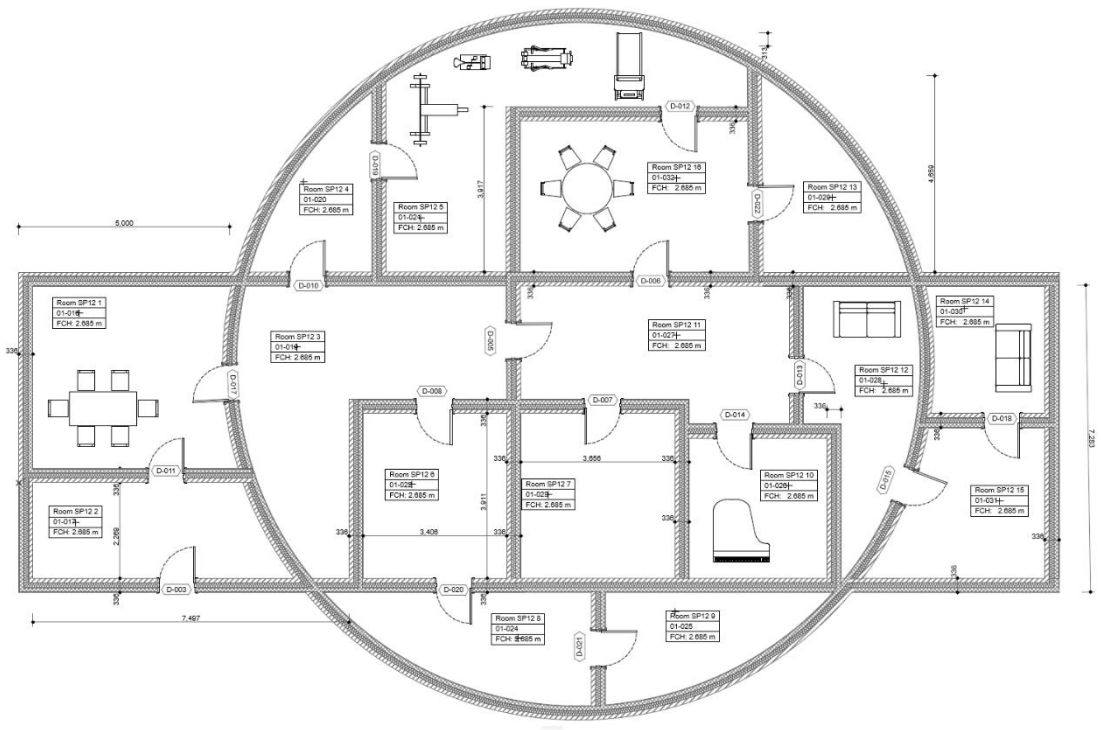
Sample Project 2: Groundtruth Drawing



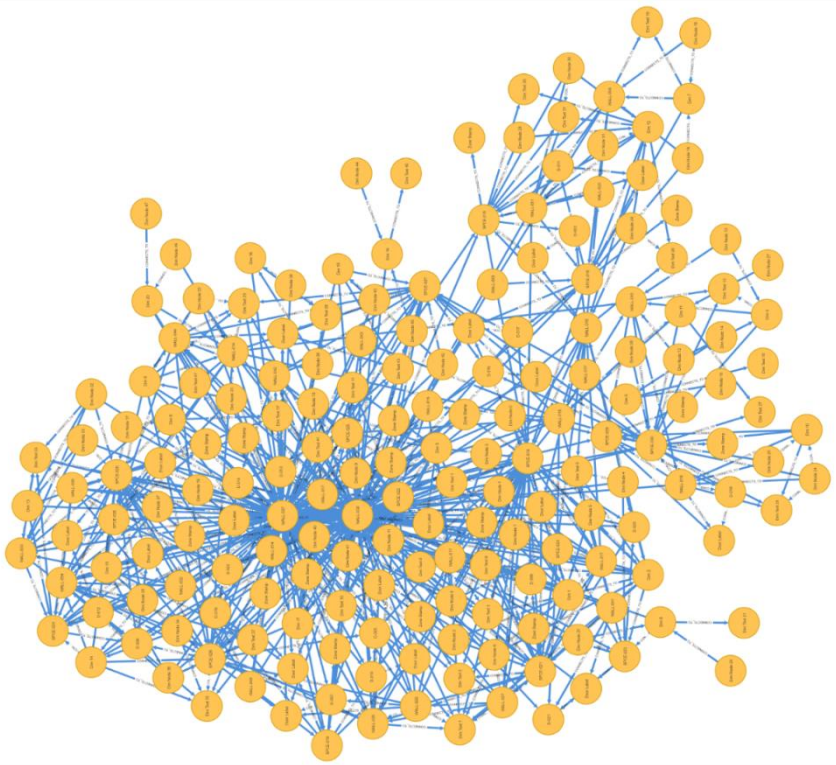
Graph generation by neo4j including annotation nodes



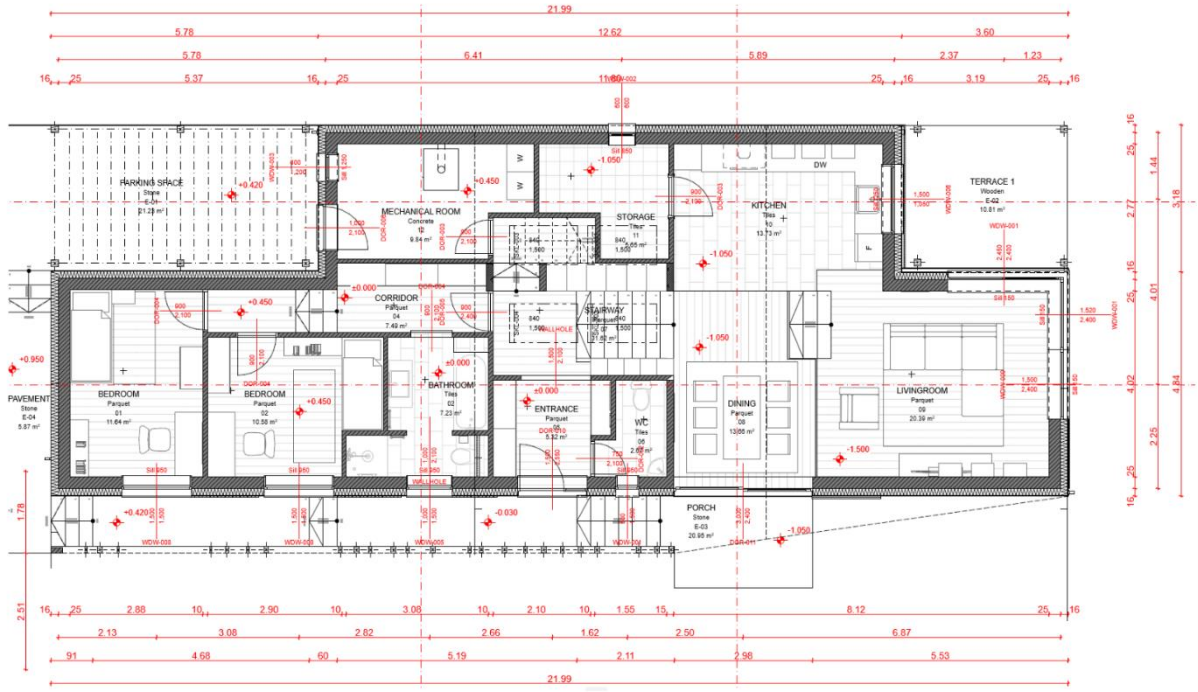
Sample Project 3: Original Drawing



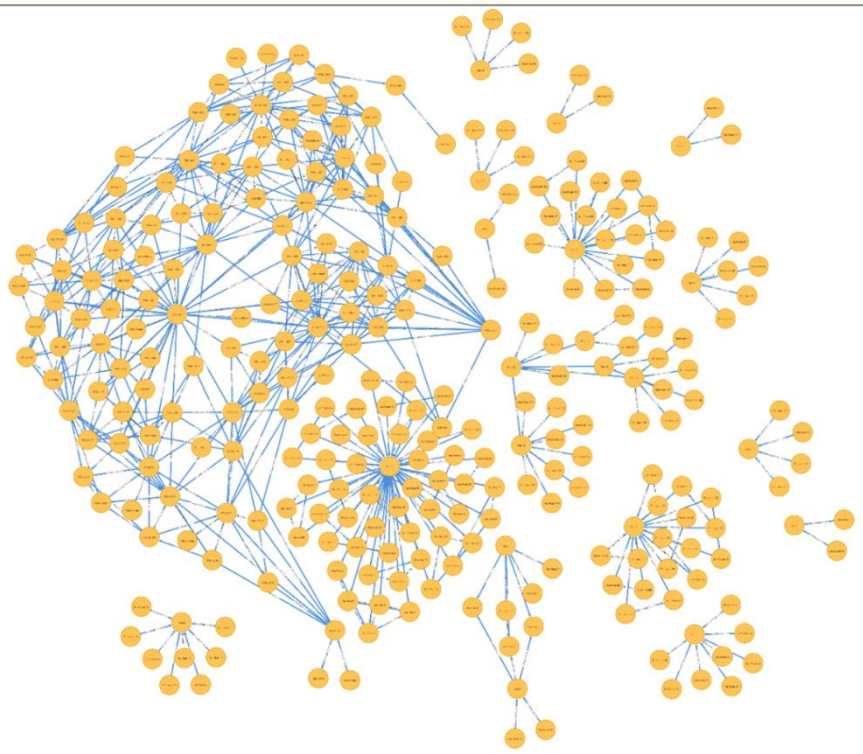
Graph generation by neo4j including annotation nodes



Sample Project 4: Groundtruth Drawing



Graph generation by neo4j including annotation nodes



10 Appendix B

The content of 'ElementInfo.txt' for Figure 1-1
Element Type: DimNode 0, GUID: 3FE05E78-CFCD-4703-9E90-E86F8ED02076, Associated Element GUID: 05526A12-85D5-41ED-9A07-FD2FA2B2489E, Text: 0, Length: 0.00, DimNode 0 Bounding Box: [(-0.00, 1.31, 0.00), (0.00, 1.32, 0.00)], Position: (0.00, 1.31), Info String: Dim Node 0 Element Type: DimNode 1, GUID: 3FE05E78-CFCD-4703-9E90-E86F8ED02076, Associated Element GUID: 05526A12-85D5-41ED-9A07-FD2FA2B2489E, Text: 336, Length: 0.34, DimNode 1 Bounding Box: [(0.33, 1.31, 0.00), (0.34, 1.32, 0.00)], Position: (0.34, 1.31), Info String: Dim Node 1 Element Type: DimNode 2, GUID: 3FE05E78-CFCD-4703-9E90-E86F8ED02076, Associated Element GUID: 26D78958-F733-42A0-9437-6A2E3AFDDDBB, Text: 4,328, Length: 4.33, DimNode 2 Bounding Box: [(4.66, 1.31, 0.00), (4.67, 1.32, 0.00)], Position: (4.66, 1.31), Info String: Dim Node 2 Element Type: DimNode 3, GUID: 3FE05E78-CFCD-4703-9E90-E86F8ED02076, Associated Element GUID: 26D78958-F733-42A0-9437-6A2E3AFDDDBB, Text: 336, Length: 0.34, DimNode 3 Bounding Box: [(4.99, 1.31, 0.00), (5.01, 1.32, 0.00)], Position: (5.00, 1.31), Info String: Dim Node 3 Element Type: Dimension, GUID: 3FE05E78-CFCD-4703-9E90-E86F8ED02076, Dimension Bounding Box: [(-0.05, 1.22, 0.00), (5.05, 1.41, 0.00)], Length: 5.00, Info String: Dim 1 Element Type: DimNode 4, GUID: 053D03C4-B54B-424F-B1F3-C157E92A5358, Associated Element GUID: 3A166D5F-444B-4FFE-9C42-9E4433C7ED62, Text: 0, Length: 0.00, DimNode 4 Bounding Box: [(3.89, -0.00, 0.00), (3.90, 0.00, 0.00)], Position: (3.90, 0.00), Info String: Dim Node 4 Element Type: DimNode 5, GUID: 053D03C4-B54B-424F-B1F3-C157E92A5358, Associated Element GUID: 3A166D5F-444B-4FFE-9C42-9E4433C7ED62, Text: 336, Length: 0.34, DimNode 5 Bounding Box: [(3.89, 0.33, 0.00), (3.90, 0.34, 0.00)], Position: (3.90, 0.34), Info String: Dim Node 5 Element Type: DimNode 6, GUID: 053D03C4-B54B-424F-B1F3-C157E92A5358, Associated Element GUID: A3831D8D-2072-464A-907B-4FF183A3BEBD, Text: 4,328, Length: 4.33, DimNode 6 Bounding Box: [(3.89, 4.66, 0.00), (3.90, 4.67, 0.00)], Position: (3.90, 4.66), Info String: Dim Node 6 Element Type: DimNode 7, GUID: 053D03C4-B54B-424F-B1F3-C157E92A5358, Associated Element GUID: A3831D8D-2072-464A-907B-4FF183A3BEBD, Text: 336, Length: 0.34, DimNode 7 Bounding Box: [(3.89, 4.99, 0.00), (3.90, 5.01, 0.00)], Position: (3.90, 5.00), Info String: Dim Node 7 Element Type: Dimension, GUID: 053D03C4-B54B-424F-B1F3-C157E92A5358, Dimension Bounding Box: [(3.80, -0.05, 0.00), (3.99, 5.05, 0.00)], Length: 5.00, Info String: Dim 2

The content of 'ElementInfo.txt' for Figure 1-1, continued

Element Type: Wall, GUID: 3A166D5F-444B-4FFE-9C42-9E4433C7ED62, Length: 5.00, Width: 0.34, Height: 3.00, Embedded Door GUID: 3C1ADF00-99DB-48EB-B5FC-A075685A71B1, Info String: WALL-001, Wall Bounding Box: [(0.00, 0.00, 0.00), (5.00, 0.34, 3.00)], Label Type: 1

Element Type: Wall, GUID: 26D78958-F733-42A0-9437-6A2E3AFDDDBB, Length: 5.00, Width: 0.34, Height: 3.00, Info String: WALL-002, Wall Bounding Box: [(4.66, 0.00, 0.00), (5.00, 5.00, 3.00)], Label Type: 1

Element Type: Wall, GUID: A3831D8D-2072-464A-907B-4FF183A3BEBD, Length: 5.00, Width: 0.34, Height: 3.00, Info String: WALL-003, Wall Bounding Box: [(0.00, 4.66, 0.00), (5.00, 5.00, 3.00)], Label Type: 1

Element Type: Wall, GUID: 05526A12-85D5-41ED-9A07-FD2FA2B2489E, Length: 5.00, Width: 0.34, Height: 3.00, Info String: WALL-004, Wall Bounding Box: [(0.00, 0.00, 0.00), (0.34, 5.00, 3.00)], Label Type: 1

Element Type: Slab, GUID: E05FCCC9-D4F7-429C-82E8-157268CB44D6, Info String: SLAB-001, Slab Bounding Box: [(0.34, 0.34, -0.05), (4.66, 4.66, 0.00)], Label Type: 0

Element Type: Zone, GUID: E21B4B3D-D3B6-4189-9DB9-FF2BC8EF4F45, Zone Stamp GUID: 9242F349-3501-4A25-BE75-4657E5AA5E3B, Position: (2.54, 2.07), Room Name: Room 1, Room Number: 01-001, Room Height: 2.69, Zone Stamp Bounding Box: [(1.91, 1.70, 0.00), (3.17, 2.44, 0.00)], Info String: SPCE-001, Zone Bounding Box: [(0.34, 0.34, 0.00), (4.66, 4.66, 2.69)], Label Type: 4

Element Type: Door, GUID: 3C1ADF00-99DB-48EB-B5FC-A075685A71B1, Width: 0.90 , Height: 2.10 , Label GUID: 392FEA53-C27B-4718-B3AC-E27D1FCA0669, Label Bounding Box: [(-0.56, -0.34, 0.00), (1.24, -0.09, 0.00)], Embedded in Wall GUID: 3A166D5F-444B-4FFE-9C42-9E4433C7ED62, Info String: D-001, Door Bounding Box: [(0.40, -0.02, 0.00), (1.38, 1.08, 2.10)], Label Type: 2

Element Type: Zone Stamp, GUID: 9242F349-3501-4A25-BE75-4657E5AA5E3B, Zone Stamp Bounding Box: [(1.91, 1.70, 0.00), (3.17, 2.44, 0.00)], Info String: Zone Stamp

Element Type: Door Label, GUID: 392FEA53-C27B-4718-B3AC-E27D1FCA0669, Label Bounding Box: [(-0.56, -0.34, 0.00), (1.24, -0.09, 0.00)], Info String: Door Label

Element Type: DimText 0, GUID: 3FE05E78-CFCD-4703-9E90-E86F8ED02076, DimText 0 Bounding Box: [(-0.05, 1.34, 0.00), (0.45, 1.84, 0.00)], Text: 0, Position: (-0.05, 1.34), Info String: Dim Text 0

Element Type: DimText 1, GUID: 3FE05E78-CFCD-4703-9E90-E86F8ED02076, DimText 1 Bounding Box: [(-0.31, 1.34, 0.00), (0.19, 1.84, 0.00)], Text: 336, Position: (-0.31, 1.34), Info String: Dim Text 1

The content of 'ElementInfo.txt' for Figure 1-1, continued

Element Type: DimText 2, GUID: 3FE05E78-CFCD-4703-9E90-E86F8ED02076, DimText 2 Bounding Box: [(2.14, 1.34, 0.00), (2.64, 1.84, 0.00)], Text: 4,328, Position: (2.14, 1.34), Info String: Dim Text 2

Element Type: DimText 3, GUID: 3FE05E78-CFCD-4703-9E90-E86F8ED02076, DimText 3 Bounding Box: [(4.35, 1.34, 0.00), (4.85, 1.84, 0.00)], Text: 336, Position: (4.35, 1.34), Info String: Dim Text 3

Element Type: DimText 4, GUID: 053D03C4-B54B-424F-B1F3-C157E92A5358, DimText 4 Bounding Box: [(3.87, -0.05, 0.00), (4.37, 0.45, 0.00)], Text: 0, Position: (3.87, -0.05), Info String: Dim Text 4

Element Type: DimText 5, GUID: 053D03C4-B54B-424F-B1F3-C157E92A5358, DimText 5 Bounding Box: [(3.87, -0.31, 0.00), (4.37, 0.19, 0.00)], Text: 336, Position: (3.87, -0.31), Info String: Dim Text 5

Element Type: DimText 6, GUID: 053D03C4-B54B-424F-B1F3-C157E92A5358, DimText 6 Bounding Box: [(3.87, 2.14, 0.00), (4.37, 2.64, 0.00)], Text: 4,328, Position: (3.87, 2.14), Info String: Dim Text 6

Element Type: DimText 7, GUID: 053D03C4-B54B-424F-B1F3-C157E92A5358, DimText 7 Bounding Box: [(3.87, 4.35, 0.00), (4.37, 4.85, 0.00)], Text: 336, Position: (3.87, 4.35), Info String: Dim Text 7

11 Declaration

I hereby affirm that I have independently written the thesis submitted by me and have not used any sources or aids other than those indicated.

München, 1. Mai 2024 Server Çeter

Vorname Nachname