Exploring Architectural Floor Plan Appropriateness in Context of Bangladesh Leveraging Graph Neural Networks in Spatial Context

by

Tanjim Noor
24341103
Mahir Tasin Islam
24341104
Tiham Shafi Islam
24341115
Mahid Atif Hosain
21101170
Md. Irtiza Anam
24341101

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

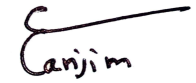
Department of Computer Science and Engineering
Brac University
September 2024

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing the degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material that has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

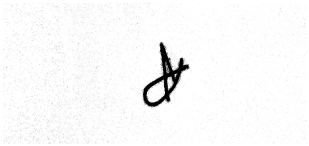
Student's Full Name & Signature:



Tanjim Noor
(24341103)

Mahir Tasin Islam
(24341104)

Tiham Shafi Islam
(24341115)



Mahid Atif Hosain
(21101170)

Md. Irtiza Anam
(24341101)

Approval

The thesis/project titled "Exploring Architectural Floor Plan Appropriateness in Context of Bangladesh Leveraging Graph Neural Networks in Spatial Context" submitted by

1. Tanjim Noor (24341103)
2. Mahir Tasin Islam (24341104)
3. Tiham Shafi Islam (24341115)
4. Mahid Atif Hosain (21101170)
5. Md. Irtiza Anam (24341101)

Of Summer, 2024 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on September 17, 2024.

Examining Committee:

Md. Tanzim Reza
(Supervisor)
Senior Lecturer
Department of Computer Science
and Engineering
BRAC University

Dr. Farig Yousuf Sadeque
(Co-Supervisor)
Associate Professor
Department of Computer Science
and Engineering
BRAC University

Dr. Md. Golam Rabiul Alam
(Thesis Coordinator)
Professor
Department of Computer Science
and Engineering
BRAC University

Sadia Hamid Kazi
(Head of Department)
Chair Person and Associate Professor
Department of Computer Science
and Engineering
BRAC University

Abstract

This paper investigates the use of Graph Neural Networks (GNNs) for classifying architectural floor plans and establishing the applicability of international floor plans with respect to Bangladeshi architectural standards. Flooring plan data is mainly derived from Chinese residential designs, which are converted into graph-based representations where rooms represent the nodes, and the connections through doors form the edges. Node features are prepared that include room area, centroid coordinates of the room, and room type, while door connections form unweighted edges. Three GNN models—GCN, GraphSAGE, and GAT are tested to evaluate their effectiveness in this binary classification task. GraphSAGE yielded the best performance among all the three GNN models tested, showing 87.09% test accuracy and an AUC-ROC score of 0.9512, with good generalization on unseen data. This work illustrates how GNNs can capture spatial relations from architectural data to enable scalable solutions for cross-cultural design evaluation and urban planning. It contributes to the increasingly important intersection of AI and Architecture by going beyond image-based traditional approaches and introducing a framework that automatically assesses the appropriateness of architectural designs concerning different cultural contexts.
Keywords: Architectural floor plans, Cross-cultural suitability, Graph Neural Networks, Spatial relationships, Binary classification, Bangladesh, Urban Planning, GraphSAGE, Design evaluation, Automated assessment, Model performance.

Acknowledgement

Firstly, all praise to the Great Allah for whom our thesis has been completed without any major interruption.
Secondly, we would like to thank our supervisor Md. Tanzim Reza, and co-supervisor Farig Yousuf Sadeque, who gave us the incentive to start our thesis on this topic.
Also, we would like to express our gratitude towards Arch. Abu Sayed Samiul Islam, Arch. Tahsina Islam, Samiha Rahman Khaled, and other annotators for their contributions. Finally, our parents without whose constant support we would not have been able to finish this report.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background on Floor Plan Classification

Architectural design is profoundly shaped by a variety of cultural, geographical, and climatic influences, resulting in notable discrepancies in floor plans across various regions. Such variations arise from distinct local preferences, lifestyle needs, and environmental circumstances, which dictate the organization of spaces within a structure. Nevertheless, the majority of current architectural datasets are tailored to specific regions, thereby constraining the applicability of these designs in diverse geographic or cultural settings. For example, a floor plan designed to meet the compact and efficient housing requirements of urban China may not be as appropriate for the more expansive and climatically varied environment of Bangladesh. This constraint underscores the necessity for more adaptable design evaluation frameworks that can effectively assess architectural appropriateness across different geographical contexts. As urbanization and global architectural exchange accelerate, developing automated tools to assess and adapt foreign designs to local contexts is essential for creating functional, culturally appropriate buildings.

Over the last years, advances in AI and ML have enabled new ways of automating architectural design and evaluation processes. In particular, GNNs have recently shown great potential due to their powerful ability to model complex relationships inherent in graph-structured data, such as spatial configurations of floor plans. Unlike traditional image-based architectural models, GNNs are strong at learning from structured data where rooms can be represented as nodes and their connections as edges. That makes them suitable for tasks like the floor plan analysis, when spatial relationships between rooms are critical to define the overall functionality and flow of a building. This work is going to explore the ability of Graph Neural Network architectures, which include Graph Convolutional Networks (GCN) [6], GraphSAGE [5], and Graph Attention Networks (GAT) [8], to generalize spatial attributes of floor plans and apply them in unfamiliar cultural settings.

Our research focuses on evaluating the adaptability of residential floor plans, which are originally from China, for application in Bangladesh. The core objective is to apply GNN-based classification methods to analyze spatial data from Chinese floor plans and determine whether these designs align with the functional and cultural needs of Bangladesh. Given the rapid urbanization in both countries, developing

an efficient and scalable method to assess and adapt foreign architectural designs is crucial for future urban planning and housing development. Through this approach, the present study not only tackles the particular issue of modifying Chinese architectural designs for the context of Bangladesh but also adds value to the wider domain of evaluating cross-cultural architectural design.

In the process, this research contributes to advancements in AI-driven architecture by extending the application of GNNs beyond conventional domains like social networks or molecular biology, where GNNs have traditionally excelled. Here, GNNs are applied to architectural floor plans to show their versatility in learning from spatial relationships and functional dependencies between rooms. GNNs are particularly good for this task since they capture both local and global patterns in graph-structured data. Their capability makes them identify subtler design elements that serve as critical determinants for the functional suitability of a floor plan in context.

The study then builds upon a GNN's capability of binary classification between suitable and unsuitable floor plans in Bangladeshi architecture. Based on the graph-based representation of such floor plans, the research utilized models like GCN, GraphSAGE, and GAT, which uniquely offered advantages in learning complex spatial features. As typical examples, GCNs are effective for handling large-size graphs while modeling local patterns [6]; GraphSAGE is advantageous in generalizing to unseen nodes by aggregation of information from neighborhoods [5]. GATs introduce an attention mechanism that lets the model weigh node importance, which could be particularly helpful when trying to highlight key rooms or spatial features whose relevance to the classification task may vary [8]. By integrating the GNN architectures, the research not only addresses the immediate challenge of adapting Chinese floor plans for use in Bangladesh but also sets the stage for the broader applications of GNNs in architectural design. The aim is to develop an automated, scalable solution with which to assess and adapt architectural designs across cultures and geography. The paper illustrates how GNNs can be useful in the advancement of AI in architecture, helping to better the design process to be more cross-culturally adaptive and thereby informing and improving urban planning practices more effectively across the world.

1.2 Problem Statement

Architectural floor plans vary significantly across regions due to differences in cultural preferences, lifestyle, and climatic conditions. However, most available datasets focus on floor plans from specific regions, limiting the applicability of those designs in different geographic or cultural contexts. In this study, we aim to evaluate the appropriateness of residential floor plans from China for use in Bangladesh. Using a Graph Neural Network (GNN) framework, we analyze spatial data from these plans to determine whether they align with the architectural needs of Bangladesh. By leveraging GNN architectures GCN, GraphSAGE, and GAT in a binary classification task, this research explores whether GNNs can effectively generalize spatial features in architectural designs and identify patterns relevant to local contexts.

1.3 Research Contribution

This research aims to apply GNNs to test the appropriateness of Chinese floor plans for Bangladesh by analyzing spatial data with GNNs. The goal is to develop an efficient automated method of architectural design evaluation across cultural and geographical contexts by applying GNN to assess spatial data.

- **Cross-cultural relevance**: The research aims to adapt foreign architectural designs to local contexts by using GNNs to automatically assess the appropriateness of floor plans for Bangladesh, thus offering scalable solutions for urban planning.
- **Spatial data analysis**: The study focuses on the coordinates of the rooms and the spatial relationships defining the floor plans, highlighting how GNNs learn complex structural patterns for an exact way to assess the functional suitability of an architectural design.
- **Advancement in Architectural AI**: The research strives to contribute to the ever-evolving AI in architecture by showing the capabilities of GNNs for extending beyond the typical image-based models through spatial context, which could eventually improve design evaluation processes around the world.

Chapter 2

Related Work

2.1 Literature review

Graph Neural Network [1] is a model that extends the current neural network methods of effectively dealing with data structured in graph form: acyclic, cyclic, directed, and undirected. Based on the initialization and estimation of its parameters using a supervised learning algorithm, the model is formulated as a graph-to-point mapping. Experiments also show that the identified information diffusion mechanism in the GNN model can generate effective and generalizable features since it keeps consistent data structures and preserves relationships. The GNN process can perform better without preprocessing than other techniques like random walk models or recursive neural networks, which are limited in their ability to learn specific types of graph structures. Applications include computer vision, molecular chemistry and biology, pattern recognition, and data mining. The authors suggest investigating avenues such as processing dynamic graphs, and relational learning over data ontologies for future investigation. The model highlights GNN models as a robust backbone in graph-based data processing taking the best of both neural networks (for representation) and Graph theory(dependent elements).

Kipf and Welling (2017) present a method for supervised learning on graph-based data by employing Graph Convolutional Networks (GCNs) [6]. These networks implement a first-order estimation of graph convolutions to handle large graphs efficiently by scaling linearly with the number of edges while capturing local graph patterns and node characteristics in hidden layer representations effectively. Through testing, on citation networks and a dataset of knowledge graphs, the GCNs show performance compared to methods. The method avoids using graph-based regularization by incorporating the adjacency matrix of the graph into the model. This enables the distribution of gradients, from loss and representation learning, for both labeled and unlabeled nodes. The authors introduce a layer-based propagation rule for GCN, inspired by a first-order spectral graph convolution approximation, which ensures efficient and scalable semi-supervised classification. The experimental results show how accurate and computationally efficient the model is. Through a comparative analysis of various propagation models and an evaluation of the training time per span, they conclude that their model has advantages for extensive use. Additionally, Kipf and Welling go over the drawbacks of their GCN technique as well

as potential advancements in the future, such as memory needs and managing directed edges or edge properties. Using the trade-off between accuracy and efficiency, their work defines GCN as an important tool for graph-based data classification.

The introduction of graph attention networks (GATs) [8] by Velikovic et al. (2018) identified a breakthrough using masked self-aware layers, allowing nodes to join features of neighbors with different weights, increasing expressiveness and efficiency without expensive matrix operations. This greatly increases the expressiveness and efficiency without the expensive matrix operations involved in diffusion and convolution. State-of-the-art GATs show state-of-the-art performance on benchmarks such as Cora, Citeseer, Pubmed, and protein-protein interaction data, which testifies to their at-most efficiency for dealing with graph-structured data. GATs address several key issues in the previous approaches of spectral-based approaches: The models do not depend on the whole graph structure from the beginning and can assign different importance to different nodes within a neighborhood. These models have successfully achieved or matched state-of-the-art performance across well-established node classification benchmarks, both transductive and inductive, especially with unseen graphs used for testing. The other potential future enhancements may involve the handling of larger batch sizes, model interpretability, the extension of the method to handle graph classification, and the inclusion of edge features to treat a wider range of problems. From early GNNs to sophisticated models such as GATs, these demonstrate very significant advances in this domain, in terms of computational challenge and enhancement of models' learning capabilities from complex graph structures.

Rossi et al. (2020) introduce Temporal Graph Networks (TGNs) [16], a novel paradigm for deep learning on dynamic graphs represented as sequences of time-stamped events. TGNs combine memory modules with graph-based operators, yielding a much more computationally efficient solution that outperforms existing methods. The general framework can seamlessly determine several prior models as specific instances for the learning of dynamic graphs. It contains memory components for storing the history of nodes, message functions to model interactions between nodes, and embedded modules for producing up-to-date node embeddings. Thus, it alleviates the problem of memory staleness. Moreover, TGNs enable efficient parallel processing and are easily applicable for both transductive and inductive tasks. Extensive experiments are carried out to establish the effectiveness of the framework that achieves the state-of-the-art performances on several benchmark datasets. More importantly, the ablation studies did provide detailed information that underlines how important the memory module and graph-based embedding are in maintaining accuracy while being computationally efficient. Future work on TGNs will relate to the exploration of more advanced configurations, enhancing model interpretability, and applications in different domains, and possibly leveraging edge features and graph classification tasks.

House-GAN's [15] method for creating house layouts makes use of a relational generative adversarial network that is graph-constrained. Given as input, a bubble diagram—representing architectural constraints such as room types and their spatial adjacency—produced a set of axis-aligned bounding boxes of rooms. It uses convolutional message-passing neural networks to handle the graph-encoded constraints within relational networks of both the generator and discriminator. Evaluations based on over 117,000 real images of floor plans verify that the proposed

House-GAN significantly outperforms the existing approaches in generating realistic, diverse, and compatible house layouts. These results show the capability of House-GAN in automating house designs and hence assisting an architect to get an efficient, aesthetic-looking design within budget and time. The authors intend to make their code and dataset accessible to everyone to encourage more study and advancement in this field.

FloorplanGAN [17] proposes a new generative adversarial framework committed to the generation of architectural floorplans, solving the peculiar difficulties that arise because of their vector graphic and raster image dual nature. However, the traditional deep learning models choke on these characteristics, FloorplanGAN combines a vector-based generation process with room area constraints, complemented by raster-based visual discrimination using convolutional layers. A differentiable renderer effectively bridges the gap from a vector generator to a raster discriminator. The model employs a self-attention mechanism to capture interrelationships among rooms, enhancing the coherence and realism of generated layouts. The experimental results are verified by various objective metrics and user studies that prove the effectiveness and feasibility of FloorplanGAN and further demonstrate that it performs significantly better than other state-of-the-art approaches for the generation of accurate and editable floorplans, able to be easily integrated into existing CAAD workflows. This approach automates and accelerates the usually iterative architectural design process; hence, it is a promising solution to accomplish the challenging task of residential floorplan generation.

The development of automatically generating floorplans, integrated with deep learning techniques, has significantly evolved from traditional manual methods. Probably the most frequent method is that of Wu et al. (2019), which proposes a data-driven approach toward the automatic generation of residential floorplans given the boundaries of the floor [11]. In general, the contribution of Wu et al. (2019) focuses on the two-stage process of the technique in question, similar to how humans usually create a floorplan. It predicts room locations first, starting with the living room, which is the central element in most modern residential designs, and refines the structure by placing the walls to make the layout coherent and practical. It is based on the Wu et al. dataset of 2019, developed from the RPLAN dataset that includes over 80,000 floorplans from real residential buildings and thus provides sufficient data for learning architectural patterns. The rich, dense annotations within the dataset of labeled rooms and walls thus enable neural networks to predict configurations of rooms consonant with typical human designs. First, the iterative prediction model places the living room, which is the most important room for enhancing the plausibility of the generated layouts and logically situating and connecting the other rooms. This data-driven approach enjoys much more flexibility compared to earlier manual or rule-based methods. It freely allows the generation of realistic floorplans directly from boundary inputs, without the need to specify detailed constraints by designers. Results from user studies prove that the system is indeed capable of generating efficient floorplans, the quality of which is comparable to professionally designed ones. The approach of Wu et al. does constitute a significant landmark in automated architectural design, as this indeed cuts time and expertise for the generation of floorplans considerably.

A new concept in architectural layout synthesis [4] has been presented in a paper

published by Elsevier B.V. in the journal "Automation in Construction" in 2016. The basic thrust of this research has been on manipulating highly irregular shapes without a particular structure in the plane, enabling designers to impose patterns from any arbitrary image onto the layout. Layout automation historically has shown great promise in supporting design activities; however, its practical adoption remains limited. The primary reasons are that most of the automation approaches are confined to right-angle polygons, and there is no mechanism to input high-level preferences by architects themselves. This research fills this gap by feeding the automated tool with an image of arbitrary patterns, thus giving more room for flexibility and creativity in the design process. The methodology used herein is strong: It extracts irregular regions from images through the use of statistical region merging and then uses sub-graph matching and simulated annealing in constructing topologically feasible layouts. It also introduces methods for measuring similarities between desired templates and irregular rooms to make sure the final designs meet the initial requirements. The results derived are promising. When users input images of regular patterns, resultant layouts prove to be predictably regular. Complex images produce highly irregular layouts, which demonstrates the flexibility and adaptability of the proposed system. The generated layouts are very irregular yet true to the patterns of the input image; hence, the designer could visually assess and iterate on their design. Considering the limitations, first and foremost is the fact that the layout automation program only focused on room dimensions and adjacencies. It did not deal with realistic specifications a building should have, such as building materials, natural lighting, heating, and energy conservation. This may be one of the probable ways for research and development shortly. The paper presents a new synthesis of an architectural layout, which has a lot of flexibility and creativity during the process. The methodology is solid and results are promising, but certain aspects need to be improved, particularly with more realistic building specifications.

Contour detection is a very crucial stage in the development of semantic segmentation and image classification [7]; many challenges are still associated with it, though, especially when the contours are incomplete or unclosed. Gong et al. (2018) review existing approaches to contour detection that fall into four categories: pixel-based, edge-based, region-based, and DCNN-based methods. While traditional approaches have evolved significantly, it was the advent of DCNNs that fully revolutionized contour detection by being able to exploit their outstanding performance for image recognition tasks. The DCNN-based approaches include holistically nested edge detection and fully convolutional networks, which introduced deep learning into contour detection to enhance precision and efficiency. Considering the above advantages, effective integration of higher-level features and enhancement of the robustness and generalization of these models still need to be overcome. In this direction, future research will focus on feature combination optimization, using prior shape information, and attempting to develop weakly-supervised learning techniques that can mitigate some of the current methodologies.

Park et al. (2023) proposed an AI-driven methodology [25] that may help the early design phase in architecture through their study entitled "Floor plan recommendation system using graph neural network with spatial relationship dataset," published in the Journal of Building Engineering. In this paper, by using graph neural networks combined with a spatial relationship dataset, this system tries to

recommend appropriate floor plans based on the needs of the client. Traditional floor-plan selection is based on manual case studies and subjective filtering through books, and real estate websites, which is extremely time-consuming and imprecise. This research uses the model SimGNN to quantify graph similarity for efficiently and effectively recommending floor plans with specific spatial configuration requirements. The research process creates a large dataset of house floor plans, analyzes spatial relationships, and trains the GNN models for the prediction of similarities among graphs. It follows from these results that the proposed system will give recommendations of high accuracy; therefore, it will result in considerable improvement in efficiency during the pre-design phase. The result of this is that the derivation of a suitable floor plan is much faster. Most importantly, the system guarantees that such a design objectively meets the client's specifications by using a data-driven approach. The authors go on to point out that their system can be used with other building types; this suggests that it could be popular and continue to be extended in further research into architectural design automation.

The paper "Graph Structure Extraction from Floor Plan Images and Its Application to Similar Property Retrieval" [20] by Yamada et al. (2021) presented a novel approach to handle such challenges provided by diversified and non-standardized drawing styles in traditional floor plan images that made computational analysis quite hard. The authors make use of a two-tiered methodology: deep learning for semantic segmentation, which identifies and annotates various regions in an image of a floor plan down to a pixel level of detail, while a rule-based transformation would convert such segmented images to their respective graph representations where the rooms become nodes and their connections are modeled as edges. This would bring forth structured yet mathematically tractable graphs that, at best, capture the spatial relationships in floor plans and go as high as 92 percent accuracy. The graph-based representation will surely enhance the comparison, evaluation, and retrieval beyond the limitation with which traditional image-based searches present floor plans. The proposed system significantly extends the functionalities of property retrieval by allowing for detailed searches with specifications in terms of spatial arrangement and relationship, other than only by room type and quantity. This will ensure that the search results are not affected by the change in style of the images, hence providing a better and more reliable retrieval experience. Besides, this study is significant because it entails the wide applicability of this method when the correctness of a similar property retrieval system is assured, thereby accurately identifying similar-configured properties using the graph structures once again. The authors now present an end-to-end method that succeeds in outperforming the best state-of-the-art methods based on deep neural networks for floor plan analysis, establishing a new state-of-the-art in the automation and retrieval of floor plans for the real estate industry.

The results are incomparable using the T2D model on the generated floor plan by T2D [23]. Among them, the T2D without using any boundary information, namely, T2D, gained a very high micro IoU of 54.34 and a macro IoU of 53.30, much higher than baseline models. This was achieved because the Seq2Seq model effectively controlled the generation of the target box sequence, guided by the salient information extracted from the given linguistic instructions. The traditional text-conditional image generation methodologies cannot suffice for this context, as their relevant design

is directed toward generating artistic images concerning high-level visual concepts inferred from short texts, which cannot satisfy the requirements laid out by multiple instructions with diverse constraints, a necessity in some design tasks. Also, while trained only on artificial instructions and then tested by ones written by humans, there happens a big fall in its performance, hence proving that yes, the gap in language distribution between artificial and human instructions does exist. This gap was reduced to a considerable extent when a preliminary warming-up of the model was done using artificial instructions before fine-tuning it with human instructions, which achieved an amazing increase of more than 10 IoU scores. That suggests that the artificial and human instructions were each other's positive factors, be the language gap as it may, in training. Moreover, the floor plan boundary uniformly came out as a sequence of boxes, relentless in the improvement of the performance of Seq2Seq within all conditions of training. This further assured the mere feasibility of the particular strategy as a valid method of effectively integrating the constraints of the floor plan. It also pointed out that quantitative results concern IoU score, indicating indirect checking of generated floor plans on their alignment with respective language instructions by considering overlap amongst generated layouts and their ground-truth counterparts. Moreover, referring to the difficulty of the task, one had to consider that an IoU score cannot be taken as a hint at poor generation, since one instruction in language can correspond to multiple correct floor plans. Quantitative results need further refinement to interpret them well enough to have a proper understanding of performance in this respect. Human judgments give more direct insight into how well the generated floor plans were aligned with the language instructions. A subset of this test set, T2D, was already evaluated by humans; it consisted of 100 randomly selected instructions from different annotators. In this study, five volunteers with NLP experience were required to assess to which degree the instructions given in the source language, according to four concrete alignment criteria, correspond to target floor plans. Volunteers scored each criterion on a scale between 1 and 5 and also provided a global alignment of the generated floor plan concerning whether it followed all specifications described in the instruction. This same subjective assessment had been done for both T2D-generated and ground-truth floor plan designs. Results from human evaluation indeed showed that 85 percent of the ground truth satisfied all specifications for each of the partial alignment criteria. That would be a good indication that this dataset contained great human instructions that have communicated their designs into the ground truth. On the contrary, T2D did not record scores below 3.5, thus stipulating that the model can predict at least 50 percent of the rooms concerning their positions, sizes, and relationships. It still showed divergence from the ground-truth design relating to room location and relationship pointing to aspects that would need further improvement. To give a better understanding of human performance on the T2D task, volunteers were asked to design the floor plans for 100 samples taken from the same subset used for human evaluation. The IoU scores which resulted showed that human subjects do tend to outperform the model of T2D. Even the floor plans generated by humans-naturally much closer to following the input instructions-received the highest IoU score of about 63 percent, compared with the ground truth floor plans. This observation brought out the implicit diversity in design whereby one set of language instructions could map to many plausible interpretations of the floor plan.

Graph Neural Networks [21] have emerged very fast as powerful ways to learn from

graph-structured data, finding applications in domains ranging from chemistry and biology to social networks and recommendations. The power of GNNs comes with their efficacy in modeling both local neighborhood information along global graph structure, enabling tasks as far-reaching as node classification to graph-level prediction. In recent years, various works have focused on improving the expressivity, scalability, and generalization of GNN architectures. One of the problems that has often made it difficult to develop GNNs in a way that allows their comparison among different models fairly and homogeneously is the lack of standardized benchmarks. Traditional datasets, such as Cora, Citeseer, and TU datasets, are usually too simple to reflect the complexity that new models developed are meant for; therefore, new benchmarking frameworks need to be more rigorous. With this in mind, Dwivedi et al. 2022 presented one of the most complete GNN benchmarking frameworks that "is modular, user-friendly on top of PyTorch and DGL". It consists of a set of medium-sized real and synthetic datasets representative of various graph-related tasks, such as node classification, edge classification, and graph regression. Comparisons among models are fair and reproducible owing to the use of fixed parameter budgets. The introduction of diverse datasets such as ZINC and AQSOL in the benchmark widened the expressiveness of GNN research, and models can be put to tasks that are relevant to chemistry, such as molecular property prediction. Moreover, this has provided intuition about how GNNs are to be improved, specifically regarding aggregation functions, pooling mechanisms, and positional encodings. For example, Laplacian eigenvectors have been proposed as a type of positional encoding that significantly boosts the performance of message-passing GNNs for synthetic datasets. The development of standardized benchmarks has overall been crucial to the development of GNN research, as these have largely facilitated a basis in which new architectures can easily be explored, and the performance scalability and robustness of these models tuned.

Automated floor plan generation has recently received extensive attention in the form of deep learning and AI-enabled generative models. Various methods have been devised for this, focusing on creating realistic layouts from minimum user input. Graph2Plan [14] is one of the major contributions in this category. It presented a new paradigm of floorplan generation with GNNs and CNNs, where the former processes layout graphs and the latter processes build boundaries. Hu et al., 2020. Most of the earlier methods were largely developed based on either procedural or optimization techniques. As an example, Arvin and House (2002) used spring systems to develop indoor layouts given certain design objectives, whereas Merrell et al. (2010) used the approach of stochastic optimization for residential layouts. However, these approaches lacked flexibility and could not adaptively handle the increasingly complex user constraints. Recent methods, for instance, that of Wu et al. 2019, employed deep learning for floorplan generation but these provided limited user control and hence restricted capability in making room layouts as desired. Graph2Plan overcomes this limitation by taking, as input, sparse constraints on room count, connectivity, and layout preferences and processing these through GNNs to retrieve relevant floorplan layouts from a large-scale dataset, namely RPLAN. By leveraging the RPLAN dataset collection of more than 80,000 annotated floorplans-Graph2Plan can produce more architecturally valid designs. This system takes its root from the previous research in scene synthesis and layout generation, where Johnson et al. 2018 proposed an image composition model conditioned on scene graphs and Wang

et al. 2019 introduced Scene Generation with Neural Networks. Unlike previous methods, Graph2Plan allows both fully automatic mass generation of floorplans and detailed, user-guided designs, targeting a wide range of applications from virtual world creation to large-scale urban planning.

GNNs have achieved outstanding success in the problem of applying large-scale graph structures in various domains over recent years. Traditional algorithms, such as DeepWalk by Perozzi et al. (2014) for node embedding, and matrix factorization-based approaches, e.g., Tang et al. (2015), consider transductive learning for the most part. These approaches mandate that during training, the entire graph should be known; therefore, they cannot manage evolving graphs and unseen nodes efficiently. While real-world networks, such as social networks and protein-protein interaction networks, dynamically change, there is a growing demand for inductive methods that could generate embeddings of unseen nodes efficiently. To avoid such a limitation, GraphSAGE (Hamilton et al., 2017) proposed an advanced inductive method [22] for learning node representation through the exploitation of node feature information, such as text attributes and structural properties. Unlike the transductive methods, GraphSAGE does not pre-train the embeddings for each node; instead, it learns an aggregation function that could create the embedding based on a node's local neighborhood. This aggregation step allows GraphSAGE to generalize across unseen nodes and new subgraphs. The flexibility makes it highly applicable to real-world tasks, such as node classification and link prediction, especially in such dynamic environments as citation networks, Reddit discussions, and biological networks. GraphSAGE architecture has been one of the key contributions of this paper to the field, introducing different types of aggregating functions such as mean, pooling, and LSTM-based aggregators, which gave state-of-the-art performance compared to previous methods on both inductive and transductive learning tasks. It also showed the efficiency and scalability of the framework on various benchmarking datasets such as citation data, and protein-protein interaction datasets, among others, showing significant improvement in F1 scores compared to prior methods. To conclude, GraphSAGE brought a step-change in both scalability and flexibility for GNNs regarding evolving graphs and inductively generating embeddings of unseen nodes, hence setting the new standard for inductive learning in graph-structured data.

GNNs have been used to serve several graph-structured data analysis tasks, including node classification, link prediction, and graph classification. Most of the classical models, such as Kipf & Welling, 2017, and DeepWalk by Perozzi et al., 2014, are designed for transductive learning, where the whole graph is present at the time of training. That is, such models are in a transductive setting, which means that they do not generalize well to unseen nodes or new graphs. Dynamic environments, then, are when the graph keeps changing in nature with continuously coming data; hence, this forms a basic challenge. In most real-world applications such as social networks, citation networks, and protein interaction networks, graphs are real-time-updated. In their influential paper [5], "Inductive Representation Learning on Large Graphs", Hamilton, Ying, and Leskovec (2017) introduced GraphSAGE (Sample and Aggregate), an inductive framework designed to overcome the limitations of transductive models. GraphSAGE learns to generate node embeddings by sampling and aggregating features from a node's local neighborhood rather than requiring individual em-

beddings for every node in the graph. GraphSAGE induces generalization to unseen nodes and even to new graphs through induction. GraphSAGE uses node features, which can be some form of textual attributes or even node degrees, to produce useful embeddings that capture both local and global graph structures. GraphSAGE first introduced three kinds of aggregation strategy: mean, pooling, and LSTM-based aggregators, which can flexibly combine neighbor information in many ways. They also proved that GraphSAGE outperformed traditional transductive methods in various benchmarking tasks that included evolving citation graphs and protein-protein interaction networks. This is because it could come up with much better embeddings for the unseen data, hence superior performance in classifying posts to subreddits. Its scalability and adaptability to different graph structures keep on being developed either for the supervised or unsupervised model. The paper "Inductive Representation Learning on Large Graphs" stirred a new revolution within the domain of GNNs by enabling the scaling of inductive learning. The efficiency of doing node embedding on unseen data by GraphSAGE began to become an important factor in dynamic environments and was a strong solution for industries or fields of research reliant heavily on large-scale graph analytics. The present paper gave way to GNNs, pointing out the induction learning needed in most higher-order graph-based tasks.

Graph neural networks have attracted widespread interest because of their ability to model graph-structured data. Most of the traditional GNNs focus on learning node embeddings through neighborhood information aggregation; two very popular such methods include the GCN method by Kipf and Welling (2017), and GraphSAGE by Hamilton et al. (2017). But usually, these models fail to capture rich information provided by edge features. Edges in graphs, especially in domains like molecular networks, represent important relationships. For instance, chemical bonds are crucial for tasks such as molecular property prediction. Trying to tackle this challenge, Yang and Li (2020) extended the proposal with the introduction of the Node and Edge Neural Network [18] (NENN), a brand-new architecture that encompasses both node and edge features. NENN leverages a dual-level hierarchical attention mechanism to enhance the learning process. The model iteratively passes through node-level and edge-level attention layers so that it can mutually learn node and edge importance. The two layers ensure node embeddings capture their connecting edges, and vice versa, to better perceive the graph structure. Extensive experiments on several benchmark datasets have demonstrated the effectiveness of NENN. Node classification: On citation networks such as Cora, Citeseer, and Pubmed, the model outperforms traditional GNN models, including GCN and GraphSAGE. NENN has been run for graph classification and regression on molecular data sets, and compared to the state-of-the-art baselines, it yields significant gains in performance for Tox21 and HIV, demonstrating its capacity for handling both node and edge feature learning efficiently. In short, NENN gave a very nice response to the challenges associated with incorporating edge features into GNNs, yielding superior performance across a wide variety of graph-related tasks.

GNNs have recently gained a lot of attention due to their capability for modeling graph-structured data and have thus harnessed applications from the preponderant number of people in tasks ranging from node classification, and link prediction, to graph classification. Most early GNN models focus on the aggregation[10] update of information from the neighbor nodes to compute node embeddings; some

of them are GCN by Kipf and Welling, 2017, and GraphSAGE by Hamilton et al., 2017. GCNs were very influential; they could aggregate node features and graph structures in a simple, scalable way. However, as GNNs were applied to ever bigger and bigger graphs, these methods finally started to have some issues, such as over smoothing, where node representations become indiscriminate across deeper layers. GraphSAGE introduced inductive learning by neighborhood sampling, enabling scalability of embedding generation in large, dynamically evolving graphs but did so at some cost in terms of theoretical expressiveness, at least considering successor models such as the GIN model of Xu et al. (2019). Despite such development, one important bottleneck still exists in the field of GNNs: a systemic lack of reproducibility and comparability of results. Most of the model investigations are performed on traditional datasets, whose experimental settings are inconsistent, including ambiguous hyperparameter tuning and methods for data splitting. In such a case, comparing different models is difficult since each evaluation may result in a performance that is either biased or excessively optimistic. Regarding this, Errica et al. (2020) indicated a systematic re-evaluation of the state-of-the-art five GNN models of DGCNN by Zhang et al. (2018), DiffPool by Ying et al. (2018), ECC by Simonovsky and Komodakis (2017), GIN by Xu et al. (2019), and GraphSAGE across nine different bench-mark chemical and social data-sets. Some key points were thereafter developed in the developed study. Notably, this involves structure-agnostic baselines that outperform many GNN models on D & D and PROTEINS datasets, respectively, representatives of chemical domains. Again, this underlines the fact that in those tasks, such as molecular property prediction, where the graph structure has a direct bearing on the chemical properties, GNN needs to make proper use of the node and edge features. The performances of GNN models like GIN were remarkable in social datasets such as IMDB-BINARY and REDDIT-5K once additional structural information like node degree was included. The work by Errica et al. brought lots of rigor into GNN evaluation through a very transparent and reproducible framework, thus allowing future researchers to compare their models without any unfairness. By actually normalizing these evaluation protocols, the authors have provided a starting point for more massive strides in the design and performance evaluation of Graph Neural Networks. In fact, only a few datasets have full coverage of all structural features, examples of which are chemical-domain datasets. Unlike previous research using general datasets from the chemical and social domains, this study deploys GNN models in an architectural floor plan domain. This is a specific challenge, besides other common applications of GNN, by representing the rooms as nodes and the relations between them as edges. This will help enhance reproducibility by developing GNN architectures with specializations that target unique graph structures of architectural layouts; hence, contributing usefully to the evolving domain of GNN research.

Cross-entropy loss functions have formed the backbone of many classification tasks [24], most notably in neural networks. They find applications, mainly paired with softmax output, in training models in quite a few domains ranging from image recognition to language processing. A very desirable property of the cross-entropy loss is that it is Bayes consistent-meaning that its minimization will also minimize asymptotically the zero-one classification loss (Zhang, 2004). This would make sense in an ideal setting: infinite hypothesis space, large dataset. However, in most realistic scenarios, when the hypothesis space needs to be limited limited neural networks

the datasets are not that big, and there haven't been many clear theoretical guarantees about the cross-entropy loss. Recent work by Mao et al. (2023) answers this by introducing H-consistency bounds serving as a non-asymptotic guarantee for cross-entropy and any other comp-sum loss functions, hence giving insight into the zero-one loss estimation error in terms of the surrogate loss estimation error within a given hypothesis set. In fact, in most practical problems, H-consistency bounds are much more informative than the standard Bayes consistency guarantees since the hypothesis set is always restricted. On the other side, Mao et al. also discuss the inimitability gaps of some popular loss functions in the comp-sum family and show that these gaps present a major part in determining whether some loss functions work or not in practice. The authors also address the question of adversarial robustness, a recent concern of modern machine learning. It is documented that neural networks might be unusually sensitive to small and sometimes imperceptible perturbations of input data and, hence, considerably degrade the performances. To this end, Mao et al. developed a new class of smooth adversarial compositional sum losses tailored for adversarial robustness. These loss functions introduce a smooth term into the combinatorial sum counterparts, guaranteeing theoretical improvement of the model in terms of robustness. Empirical evaluations on benchmark datasets such as CIFAR-10 and CIFAR-100 showed that models trained using adversarial loss functions outperformed state-of-the-art robustness techniques like TRADES from Zhang et al., 2019, both in adversarial and non-adversarial settings. In fact, from a theoretical viewpoint, this has been a great leap into generalizing the H-consistency bounds and introducing smooth adversarial loss functions into the application of machine learning in practice. These will be useful in enhancing the robustness of neural networks by providing the ability to compare and select, depending on the task, the most appropriate loss function among those presented in this paper.

Optimization has always acted as the key influencer in training a model of machine learning, let alone deep neural networks. Variants concerning Stochastic Gradient Descent [2] (SGD) have gained wide acceptance due to their efficiency in computation and handling big data. However, standard methods of SGD completely fail under sparse gradients or nonstationary objectives. To handle such issues, several adaptive learning rate techniques have been invented, which include the technique of AdaGrad by Duchi et al. in 2011, RMSProp by Tieleman & Hinton in 2012, and most recently Adam by Kingma & Ba in 2015. Adam stands for Adaptive Moment Estimation. This extends the basis from Adagrad and RMSProp toward adaptive estimates of both the first and second moments of gradients to deduce the learning rate individually for each parameter. The key merits of Adam include competence in handling sparse gradients and adapting to the objectives' non-stationarity, therefore enabling this algorithm to perform very efficiently on deep learning tasks and at scales. Secondly, its hyperparameters make intuitive sense, and for most practical applications, little adaptation is required. Its compact memory footprint and computational efficiency have led to widespread adoption for tasks ranging from basic image classification to complex natural language processing. They also propose a variant of the optimization algorithm, which they refer to as AdaMax, based on the infinity norm and well-suited to problems with unbounded updates of the parameters. AdaMax extends the Adam update rule in such a way that for some applications in large nonconvex optimization problems leads to an algorithm with

improved numerical stability. This extension keeps the merits of adaptive learning rates but allows better stability at the same time. Empirical results are presented by Kingma and Ba (2015), where, on an array of models and datasets, Adam and AdaMax outperform other stochastic optimization methods: namely, SGD with Nesterov momentum and Adagrad on logistic regression, multi-layer neural networks, and convolutional neural networks. Also, it illustrates the way Adam is superior for convergence and more reliable to converge than the other methods for both noiseless and noisy data with sparse gradients. In turn, the emergence of Adam and its variant, AdaMax, significantly beat some new paths in machine learning optimization techniques. These algorithms represent strong and effective solutions for some challenges usually coming from training great neural networks and becoming irreplaceable tools in academic research and some practical applications.

Graph neural networks have gained an unbeatable reputation in the last few years because of their strong ability to model complicated relational data, including social networks, protein interactions, and molecular structures. In principle, GNNs are neural networks to capture the dependencies or relationships between nodes in a graph using some aggregation mechanism. Early models introduced by Kipf & Welling 2017, such as Graph Convolutional Networks, extend the operation of convolutional neural networks into graph domains and set a strong solution to tasks such as node classification and link prediction. These models typically assume a static graph and generalize badly to dynamic or very large graphs arising in several real applications. Zhou et al. (2021), in their review entitled "Graph Neural Networks: A Review of Methods and Applications," suggested the development and advance that had been made in the GNN models [19] over the years across a wide variety of domains. The paper examines a large number of GNN variants, including Graph Attention Networks by Velickovic et al. (2018) and Graph Recurrent Networks in their development to attack specific challenges arising in graph-based learning. GATs use the attention mechanism to guide attention differentially to neighbors; hence, they perform well on tasks whose node relationships are unequally distributed. Similarly, the GRNs capture temporal information and could therefore be applied to tasks making use of dynamic graphs like time-evolving networks. It puts the applications of GNNs systematically into view from structured tasks. The GNNs can model the relationships that are explicitly or implicitly represented across these variable types. In consequence, they have become the crucial ingredient in most bio-informatics, knowledge graphs, and combinatorial optimization settings. Some of the design and application issues of GNNs remain open despite its large number of applications. Among them, the problems of scalability, interpretability, and integration of heterogeneity remain very important trends. Based on these problems, this paper has taken up some research directions related to constructing much more effective graph neural network models, to refine the theoretical basis to decisively raise the level of handling large-scale dynamic graphs. In a nutshell, Zhou et al. (2021) reviewed GNN models concerning some important milestones and prospects. Their effort in this area of science has demonstrated very clearly the flexibility of GNNs and active effort toward their improvement because of current model shortcomings. This positions GNNs as an essential tool for the future of machine learning based on graph-structured data.

Chapter 3

Methodology

3.1 Work Plan

The work plan begins with **Dataset Collection**, where floor plan data is gathered. Then we extract room coordinates, room area, and room connectivity from the floor plan image. This extracted information is used to make nodes and edges which represent the floor plan in **graph**. An **annotation framework** is built, defining the protocols and standards that should be maintained by the annotator in labeling floor plans in different classes. The labeled floor plan is paired against each graph accordingly. These graphs are **augmented** by rotating the original floor plan to increase the data sample size. After suitable **models** are chosen, Node features are normalized, and encoding techniques are applied. The entire dataset is loaded by converting the node features and edges into graph object. After **train-test split**, the different Model undergoes training. All the results of different models are gathered and used for **analysis**. Derive conclusion from result analysis

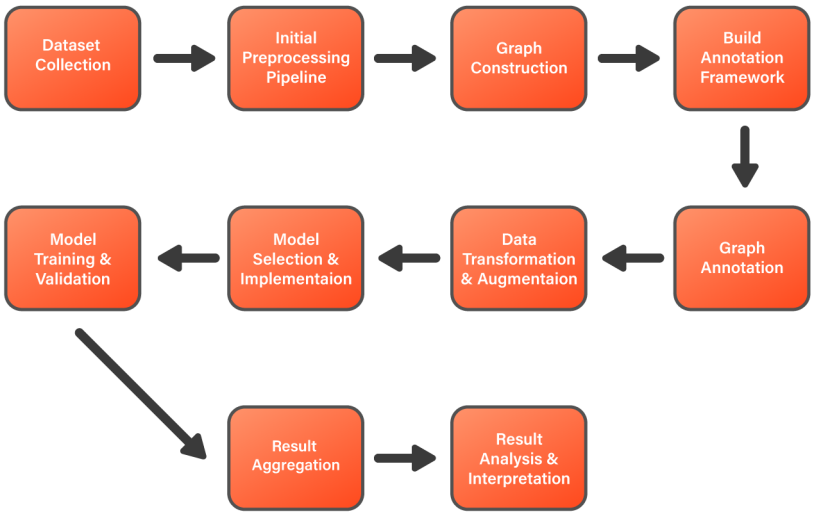


Figure 3.1: Workplan

3.2 Dataset Overview

The dataset used in this study is derived from the RPLAN[12] dataset, which comprises over 80,787 real-world floor plans in China. The dataset provides a wide variety of room types, floor layouts, and connections, crucial for the architectural design tasks addressed in this research. The data is presented as 256 x 256 pixel pre-processed images. In the data encoding schema for the floor plan model, four channels are used to represent different aspects of the floor plan.

3.3 Initial Preprocessing Pipeline

3.3.1 Initial Data Cleaning

The dataset used in this study undergoes a detailed cleaning process to ensure it is suitable for graph-based processing. The initial step involves converting the raw floor plan images into structured graph representations. This is achieved by first extracting the contours of essential architectural features using binary masks on the channels, such as walls and doors, and simplifying these shapes while retaining their fundamental geometry. The simplification reduces computational complexity, making the data more efficient for downstream tasks. Once simplified, the contours are converted into vertices, which capture the structural layout of the floor plan. These vertices are then organized into a graph format, where rooms are represented as nodes and doors as edges, modeling the spatial relationships between different components. This transformation prepares the dataset for further processing stages, ensuring that it maintains the critical architectural details necessary for the task at hand. A brief overview is shown in Figure 3.2

3.3.2 Feature Extraction: Image Encoding and Vertex Data

The feature extraction process leverages the pre-processed floor plan images provided by the RPLAN dataset, which includes a structured four-channel encoding schema that captures essential aspects of the architectural elements.

- **Channel 1** encodes the exterior walls with a pixel value of 127, designates the front door with a pixel value of 255, and marks other areas with 0.
- **Channel 2** assigns specific integers to various room types, starting from 0 for non-room areas and incrementing for different rooms, such as 1 for the master room, 2 for the kitchen, and up to 17 for the interior door.
- **Channel 3** differentiates between rooms with identical labels by using distinct integers.
- **Channel 4** distinguishes between exterior and interior areas, marking the exterior with a pixel value of 0 and the interior with 255.

This structured encoding allows for effective differentiation between critical components of the floor plans. Subsequently, the contours derived from these encoded images are transformed into vertex data. The vertices represent specific points in the spatial layout, which are then organized into a format suitable for graph construction.

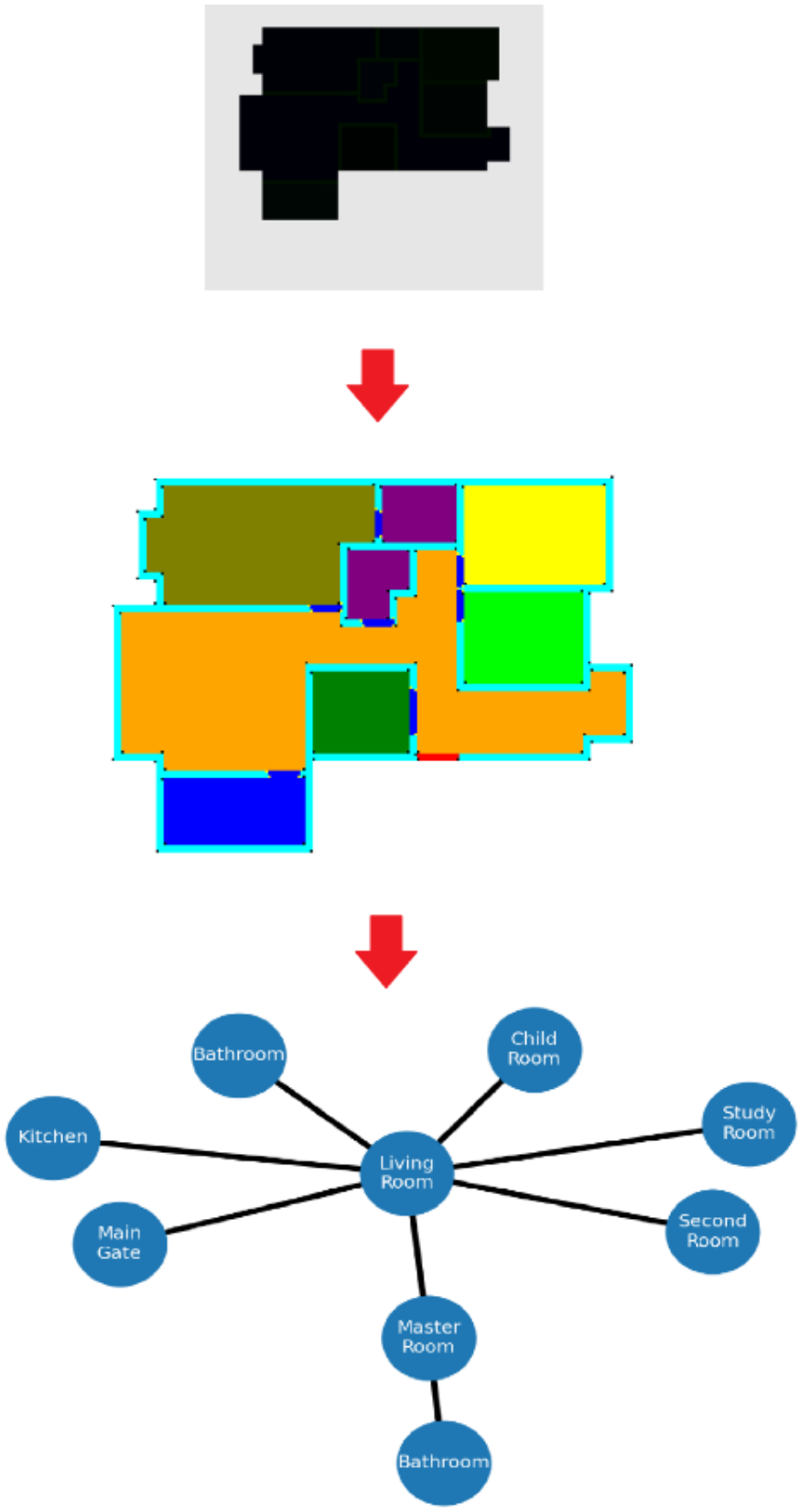


Figure 3.2: Initial Data Preprocessing Pipeline

3.3.3 Contour Extraction and Simplification

The contour extraction process begins with the generation of binary masks from the pre-processed floor plan images, which serve as the foundational input for identifying architectural features. These binary masks isolate critical elements such as walls, doors, and rooms, allowing for precise contour detection.

- **Contour Detection**: This phase utilizes the $find\_contours()$ function from the *skimage.measure* module to accurately identify the boundaries of architectural elements within the binary masks. By scanning the image at a defined intensity level, this function effectively delineates features such as walls, doors, and rooms, ensuring that the essential structural components are captured.
- **Polygon Simplification**: Following contour detection, the identified contours undergo simplification to reduce the number of vertices representing each shape while retaining critical geometric features. This process streamlines the contour representation by approximating the outline of each shape with fewer points, thereby minimizing computational complexity. By effectively reducing the complexity of the contours, this technique enhances the efficiency of subsequent data processing stages while preserving the essential characteristics of the architectural elements.

3.3.4 Graph Construction: Nodes and Edges

- **Node Creation**: The nodes are created representing the rooms where each node is attributed with features such as room vertices, the area, type of room.
- **Edge Establishment**: Spatial relationships and adjacency between architectural components are analyzed to establish edges. For instance, doors are connected to the rooms they lead into, representing direct relationships between nodes that stand for connected rooms. We used our custom special algorithm to find which doors are connecting which two rooms and as a result, made an edge between them. As an edge feature, vertices of the connecting door between the nodes or rooms are being stored.

As the main gate holds significant information, an exception was made. It is stored with a dummy node termed "Main Gate" and the main gate's vertex information is stored as the edge feature between the dummy room and the room which is connected to the main gate.

3.3.5 Artifacts

The constructed graph is then checked to be valid, as the data itself does not have any problems. The problems, namely two of them were detected. After the filtering process, the dataset was reduced to 51K functional data. The problems are as follows:

- **One door connecting multiple rooms**: In the dataset, there are multiple cases where two rooms were connected by the same interior door, which made the graph take only one of the multiple rooms that were connected. In this case, if all the nodes in our graph are not equal to the number of rooms, it is

not selected in the functional data pool. This appeared in 11521 images of the dataset. The small blue rectangles represent the internal doors in Figure 3.3.

Figure 3.3: One door Connecting multiple rooms problem

- **Multiple doors being morphed during simplification**: The contouring and simplification process which was carried out sometimes morphed multiple door coordinates together when they were on top of each other, or if they were too close. If the interior door vertex count was not 4 (as all doors are represented as rectangles in our dataset), it is not selected in the functional data pool. This appeared in 17920 images of the dataset. The small blue rectangles represent internal doors in Figure 3.4, the artifacts within the door outline the problem.

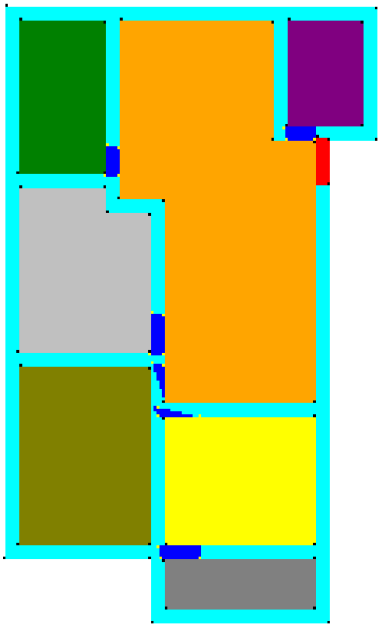


Figure 3.4: Multiple Door Morphing Problem

3.4 Initial Filtered Dataset

3.4.1 Visualization

The graphs derived from initial pre-processing are then acquired which numbers around 51K. The data is then visualized to understand the information that exists in them.

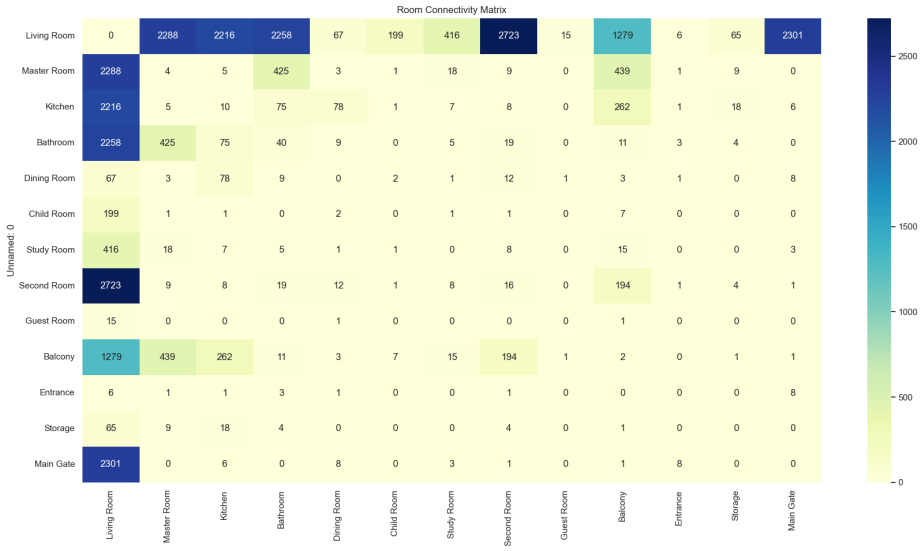


Figure 3.5: Room Adjacency Matrix

The Room Adjacency Matrix in Figure 3.5 demonstrates the network of room interactions within a house layout. Here, the Living Room is revealed as the central hub, with particularly strong connections to essential areas such as the Master Room, Kitchen, Bathroom, and Second Room. This centrality suggests that the Living Room functions as the main passage and gathering area in the house. Interestingly, certain rooms like the Guest Room, Entrance, and Child Room show minimal connectivity, which reflects their more secluded or specific roles within the home. Asymmetries in connectivity, such as the robust link from the Living Room to the Main Gate that isn't reciprocated, may indicate movement flow patterns or sensor positioning within the home

The Histograms in Figure 3.6 reveal the distribution of connectivity across different rooms. The Living Room presents a multi-modal distribution, with peaks at various ranges such as 2000-3000 and 6000-8000, hinting at different intensities of connections to other spaces. Other rooms like the Master Room, Kitchen, and Bathroom display right-skewed distributions, with their peaks concentrated at lower values, indicating that while they maintain some strong connections, most of their interactions are at lower levels. Utility areas like the Guest Room and Storage maintain narrow, low ranges of connectivity, underscoring their limited interaction with other parts of the house. Individual sampling of data also reveals that graphs can be cyclic.

The initial filtered dataset emphasizes the role of the Living Room as the dominant node in the house's connectivity network. As demonstrated across the visualiza-

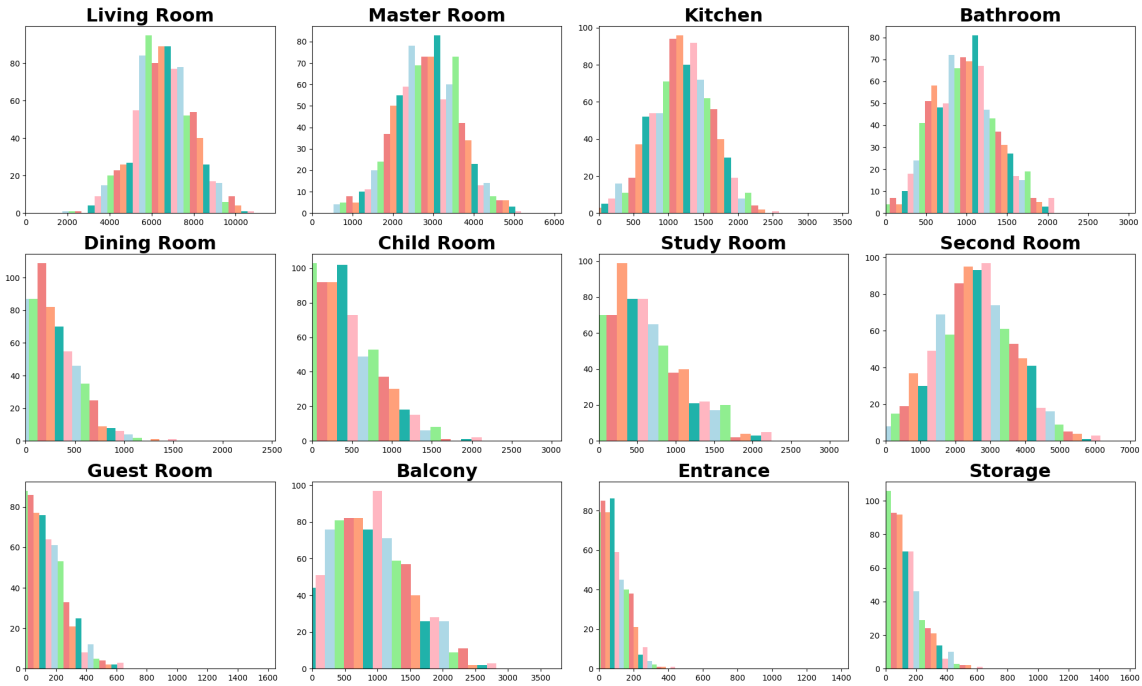


Figure 3.6: Histogram of Room Areas

tions, it functions as the main conduit for movement and interaction. A hierarchical connectivity structure emerges, where the Living Room is followed by the Master Room and Kitchen in terms of connectivity importance, then by other main rooms like the Bathroom and Second Room. Utility spaces and smaller rooms are the least connected. The visualizations also reveal occasional high outliers in connectivity between certain rooms, suggesting significant but infrequent interactions. This hierarchical structure and the presence of outliers provide deeper insights into how spaces are utilized within house layouts in China. This is important as further down the pipeline, as a random subset of the dataset is considered for annotating. Comparing both datasets provides insights into the information the model was trained and evaluated.

3.5 Annotation Process and Labeling

In the field of architectural floor plan classification, the annotation process is necessary for evaluating the spatial structure and zoning features of a quality family residence. This annotation scheme aims to classify residential floor plans according to their spatial properties, zoning configuration, interior relationships, and overall design efficiency in the context of residential floor plans in Bangladesh. Classification is based on a hierarchical system and helps to create an effective guide for the standardization of the assessments done in architectural design, providing a valid measure of its evaluation. Especially in the context of Bangladesh, urban residential architecture presents some distinct challenges and opportunities.

The ranking framework encompasses four distinct categories: Type A, Type B, Type C, and Type D, in which each category represents a different level of design quality and spatial effectiveness.

These annotations are applied to ensure whether a floor plan is ideal for family living, functionally inadequate, or not recommended for standard residential application. Moreover, the labeling process takes into consideration regional architectural conventions, spatial behaviors, and different residential requirements associated with Bangladesh, which also provides a different perspective on the judgment of floor plans. The approach to this annotation involves reviewing and categorizing features related to room location, space utilization, interrelationships between different areas, privacy concerns, and traffic flow patterns. Data labeling is hereby required as an important step in training the models to be used in the accurate classification of floor plans based on these spatial features. The section below describes how the annotation labels were contextualized for Bangladesh's residential architecture and further details the exact workflow that was followed while labeling the data to ensure consistency and reliability in the classification process.

3.5.1 Annotation Labels in Context of Bangladesh

In the context of Bangladesh, residential architecture must address several local relevant issues, including constrained spatial conditions, the need for efficient zoning layouts, and significant privacy requirements. The developed labeling system for categorizing floor plans acknowledges these factors by providing a framework that reflects the basic needs and practices of the country. The classification system is categorized into four distinct types of residential floor plans, Type A, Type B, Type C, and Type D.

- **Type A:** Type A residences are considered ideal and give us an overview of proper zoning with optimal space utilization. It provides strong internal connections between areas, with a clear division between public, semi-private, and private spaces. Proper zoning holds a large portion of a residence's design quality. The arrangements of the zoning are crucial. When it comes to zoning, privacy is important in this category, which indicates the private zones, such as bedrooms, kitchens, etc. The spatial arrangement of zoning within a residential design should be in a manner that private areas are placed far from public areas, like the living room, dining area, and primary entrance. If it is otherwise, this would be considered a poor design in the context of Bangladesh. This setup ensures that the private areas are kept away from external visitors, keeping privacy intact. Secondly, the number and arrangement of bathrooms are important. For instance, in a three-bedroom apartment, there should be at least two bathrooms, one being attached to one bedroom and another common bathroom to accommodate household needs, which is generally practiced in the context of Bangladesh. In addition, Type A houses show a coherent pattern of circulation smoothly guiding the flow between public and semi-private areas logically and in a decent sequence, which is essential for both comfort and functionality.

  Similarly, like zoning, space utilization is another critical factor. Type A residence designs have minimal to no wastage of space. Every room is thoughtfully designed to optimize spatial efficiency, and also ensure maximum utilization and the arrangement of spaces. So that there will be no unused or leftover

spaces within the layout. Space ratio is also a major part of a residence design. It ensures each room's size is proportionate to its function and overall layout. Proper space ratio enhances comfort and usability, avoiding crowding or unused space. It is an important aspect in maximizing the entire residence in terms of flow and efficiency.

In the context of Bangladesh's geography, orientation is one of the key concepts while mapping a floor plan. Due to the geographical location, north-south oriented residences are favored, because they allow natural ventilation, and also facilitate better airflow throughout the space. As Bangladesh is a hot and humid country, west-facing apartments are not preferred due to excessive heat exposure during the daytime. This orientation increases room temperatures and makes the living environment uncomfortable. On the other hand, as Bangladesh is an overpopulated nation, and buildings are next to each other, north-south-oriented apartments are quite rare to find. For natural ventilation, and better airflow, semi-opened spaces like balconies are needed. Airflow in a residence is aligned with the ventilation allocations. Ventilation is one of the major concerns in designing Type A apartments. Considering the climatic condition of Bangladesh, the inclusion of a balcony and proper orientation of rooms can ensure cross-ventilation, which is highly valued. Even though it is quite hard to meet all the ideal conditions in Type A apartments, an architect always tries to create the optimal living environment for their clients that prioritizes comfort and functionality.

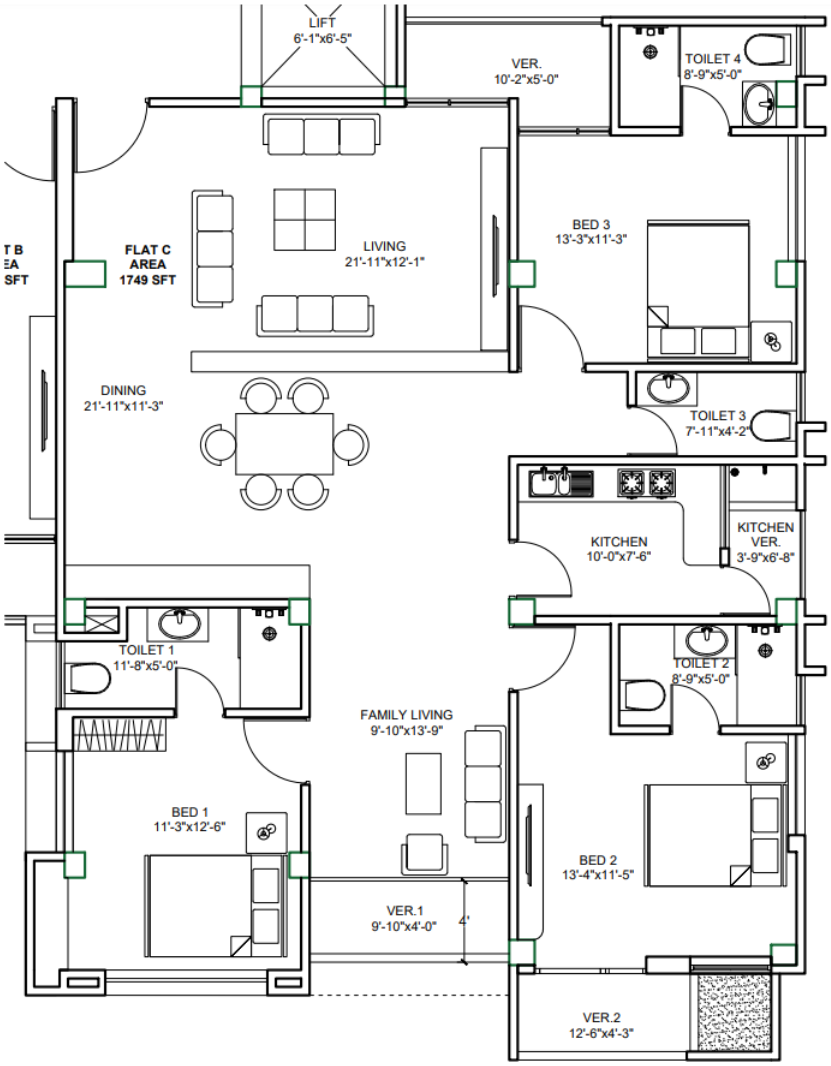


Figure 3.7: Type A apartment's floor plan design

- **Type B:** Type B residences are functional and meet all the standard requirements within the context of Bangladesh. However, it lacks the design precision, quality, and the characteristic of Type A residences. Though they still follow zoning and space ratio principles. However, the internal relationships between spaces are not as strong as they should be, which ends up resulting in less optimal space utilization. This category of residence mostly creates confusion in the separation of public and semi-private areas. For example, a kitchen may be placed too close to the main entrance, which disrupts the flow of the zoning. Similarly, while privacy is a key aspect of residential design, most of the older buildings in Bangladesh do not provide adequate separation for private zones. But still, they are livable and meet the basic needs of their inhabitants. While they do not meet the standards of Type A designs, still Type B designs are acceptable and also commonly seen in Bangladesh. These residences often fulfill basic functional requirements and needs but they still lack in space efficiency and maintenance of privacy concerns.

  In Type B residences, zoning practices are applied, but with compromises. Public, private, and semi-private zones may overlap or be poorly defined, which eventually leads to confusion in the use of spaces. Mainly, the difference within the zoning placements of semi-private and private zones. For example, there can be a dining room in front of the main entrance instead of a living room. It is not convenient, but as long as other basic standard requirements are fulfilled, it is not that big of a deal. This indicates the flow from the main entrance to public areas and eventually from semi-private to private zones is not as smooth, which can result in a less comfortable living experience, but it can be standard in the context of Bangladesh. Typically, Type B residences embody a more functional design philosophy, emphasizing practicality rather than aesthetic appeal or spatial refinement.

Figure 3.8: Type B apartment's floor plan design

- **Type C:** Type C residences are different from the pattern found in Type A and Type B residential models. A floor plan of this category may suggest zoning with flaws, not optimal space allocation, and weak interconnections between different zones. Not only that, Type C residences are marked by inferior room layouts, poor provision for privacy, and utilization of space is not proper and may create confusion. Also services may not work properly. Most of the time, these types of residential designs are more suited for non-family residential purposes, such as hostels or commercial spaces. The spatial relationship among rooms does not allow for ease of living, and key facilities such as bathrooms and kitchens may be misplaced or inadequately provided. Normally, accommodations of this type are not at all suited to permanent residential purposes by a family but would be more suited to temporary accommodation or shared living arrangements where privacy and space efficiency are less of a priority.

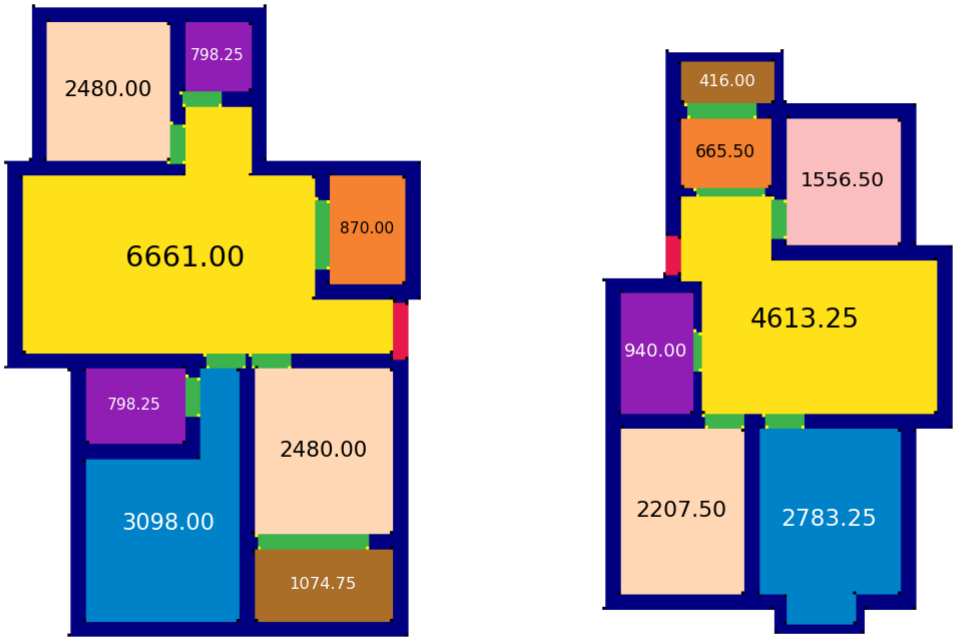


Figure 3.9: Type C apartment's floor plan design

- **Type D:** Within these floor plan classification systems, Type D serves as a benchmark for identifying designs that fail to meet basic residential standards, and also lack the fundamental characteristics of a functional floor plan. Type D is the worst-case scenario in residential design. The floor plan types classified as Type D could not reach the minimum requirements mentioned earlier and are not recommended for residential purposes. Generally, most residences with Type D show disorderly spatial arrangement, improper zoning, and ineffective utilization of space. Unfortunately, these designs may still exist due to outdated construction practices or a lack of regulation, but some of these plans are contemporary standards. The flow in these residences is usually poor, and there is no distinction between public and private areas, which then leads to a very disorganized living environment.

  The Type D residential constructions should be discarded or avoided for modern residential purposes, as they don't even satisfy the basic design requirements. The spatial flow of the spaces is fragmented, the rooms are often poorly ventilated, and privacy is near to nonexistent. Such designs may continue in areas where there is either older or less regulation, but they do not meet the needs or expectations of modern living.

Figure 3.10: Type D apartment's floor plan design

The hierarchical classification of residential floor plans mentioned earlier provides a clear concept for evaluating the spatial effectiveness, zoning, and overall design quality of family residences in the context of Bangladesh. Addressing such key architectural factors like privacy, space utilization, and functionality, helps the ranking system to guide the design process towards creating a more livable and sustainable modern living.

3.5.2 Data Labeling Workflow and Adjustments

The classification of residential floor plans, Types A, B, C, and D provided a vivid understanding of different zoning, space utilization, and strong internal relationships between the spaces that impact the livability and functionality of a residence. In this way, it has been established through this classification that optimum designs, the ones meeting the strict conditions of Type A, are very rare, especially when considering those floor plans obtained from the RPLAN dataset from China, different cultural and architectural backgrounds.

To have an in-depth understanding of these categories, we approached a professional architect from a local firm. The insights gained from these consultations helped us refine our concepts of classification and further adapt them to the specific needs of residential architecture in Bangladesh. Afterward, the architect trained a group of 13 architecture students in an iterative process to filter out floor plans that satisfied the set criteria for family residences in Bangladesh. Through multiple cycles of training and feedback, the students developed the required skills necessary for identifying appropriate designs based on zoning, room interrelationships, and space utilization. This approach will ensure that the classification system is tuned to the professional architectural experience.

However, since all the floor plans were sourced from China, it became necessary to eliminate those designs that were irrelevant to the Bangladeshi context. All of the floor plans did not reflect regional architectural preferences and were not aligned with them, for example, room positioning, circulation flow between the rooms, and ventilation, which are critical for livability in a hot, humid country like Bangladesh. To help us with this filtering process, we developed a website called "Filtering Floor Plans," which was hosted online for the architect's team to access and contribute to. To streamline the process, we developed and hosted a website called "Filtering Floor Plans," where the trained students could filter and classify the floor plans according to our ranking system. The architects used this platform to evaluate and label the floor plans based on the guidelines.

Through this website, our goal was to identify floor plans based on the classification, Type A, B, C, and D. However, after reviewing between 800 and 1,000 plans, no plans were found that fully satisfied the requirements of Type A. This led to the conclusion that our dataset was not directly applicable to real-world residences in Bangladesh without significant modification. So, the limitation caused us to create two subcategories under Type B: High B and Low B. It allowed us to represent floor plans better. The ones that were close to Type A but required minor modifications as High B and those that met basic functional requirements but fell short of optimal design as Low B.

- **High B:** The High B category includes floor plans that are close to the ideal standards of Type A, but slight modifications are needed to meet the optimum standards. Generally, such plans show appropriate zoning and effective usage of space, with clear distinctions between public, semi-private, and private areas. The room layouts and relationships are strong, but small adjustments, like repositioning the main entrance or modifying the orientation of certain rooms are needed to bring the design to Type A quality.

The High B plans represent a distinct class of architectural layouts that, while not perfect in every respect, form a very sound basis for family residence in Bangladesh. Minor adjustments to these plans allow them to meet the needs of the people's comfortable living by providing efficient use of space and maintaining coherent room sequences to strengthen the interconnections. Overall, the High B class provides a practical approach to the development of near-optimal designs with reduced redesign effort.

- **Low B:** The Low B category consists of family residences that are functional but make more significant compromises compared to High B or Type A plans, but similar to the low-end Type B category. Even though this category of architectural design follows the proper zoning principles, room relationships or spatial arrangements might not be optimal for family living. For example, a Low B would entail a rearrangement where the dining room and living room are switched, or there is only one bathroom shared between two bedrooms. These modifications make the design less suitable for larger families, but still, they are functional for smaller households and couples.

  In contrast, the Low B plans resemble the normal Type B designs, which are very common in Bangladesh, still despite the fact, they are not suitable for contemporary family living. Such a design would meet the minimum functional requirements but is poorly designed in terms of space utilization and circulation flow. Despite these shortcomings, Low B plans are still suitable for families living in the context of Bangladesh, particularly for smaller families with fewer demands on space.

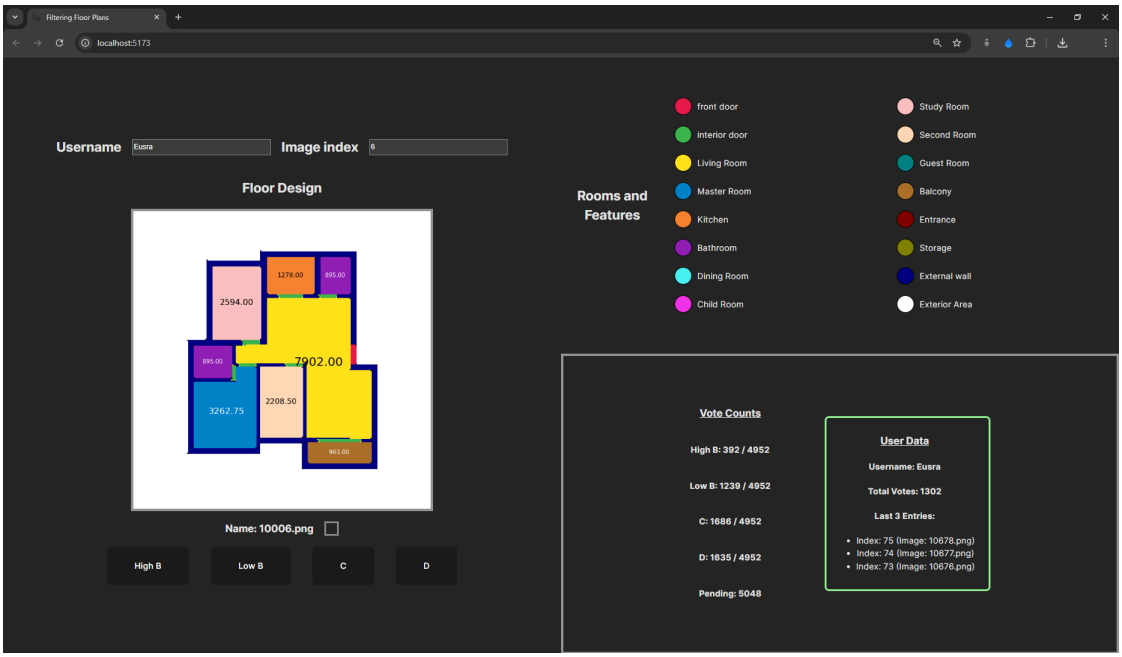


Figure 3.11: Architectural Floor Plan Filtering Website's User interface

The website has a voting mechanism that enables each participant to classify an image into one of the following classes: High B, Low B, C, or D. For each floor plan image, the most voted category is assigned. The corresponding graph of the image is included in the final dataset, if there were a minimum of 5 votes by different annotators. This democratic approach helps ensure that these classes reflect the collective agreement of users' judgment based on established criteria.

Our machine learning model works by converting the floor plan images to bubble diagrams, representing the rooms as each node and holding essential information like the centroid of each room, total area, room types like master bedroom or kitchen, etc, and interconnections between rooms. However, the model does not capture ventilation and orientation, which are the key factors in evaluating the livability of floor plans in the context of Bangladesh. Despite this limitation, the architecture students worked within the constraints of the available data to classify the floor plans.

The filtered dataset after the initial pre-processing provided plans in graph structure. To address this issue, images with color-coded labels are produced from the graph data and are integrated into the website for the annotators to annotate. After giving the inputs to the website, the final output of labeled data is stored in MongoDB for further analysis and the use of our model.

Our initial goal was to filter and annotate a subset of the original dataset, which contained 51,000 floor plans. Due to time constraints, and the intensity of the task, we were able to filter and annotate 9,028 plans. Moreover, the architecture students participating in the project also had their studies and could not commit fully to this project. Similarly, the architects assisting us had their professional responsibilities, limiting the time they could allocate to the project. But among these, 897 plans were classified as High B, 3,165 as Low B, 2,753 as Type C, and 2,213 as Type D. The subset of the dataset was chosen chronologically as the filename, for which the first 10.5K entries are annotated and the remaining part of the dataset was not labeled for reasons mentioned above. These results point to the challenge of retrieving the best floor plans from this data, even more so considering that none of the retrieved plans met the strict criteria of Type A. Although more precise results could be obtained by further refinement and subdivision of each class, and also by modifications of the data for particular uses in the context of Bangladesh.

3.6 Final Dataset Visualizations

The annotated dataset of 9k graphs holds information that is different from the superset of 51K graphs. To understand the final dataset, the information is then visualized.

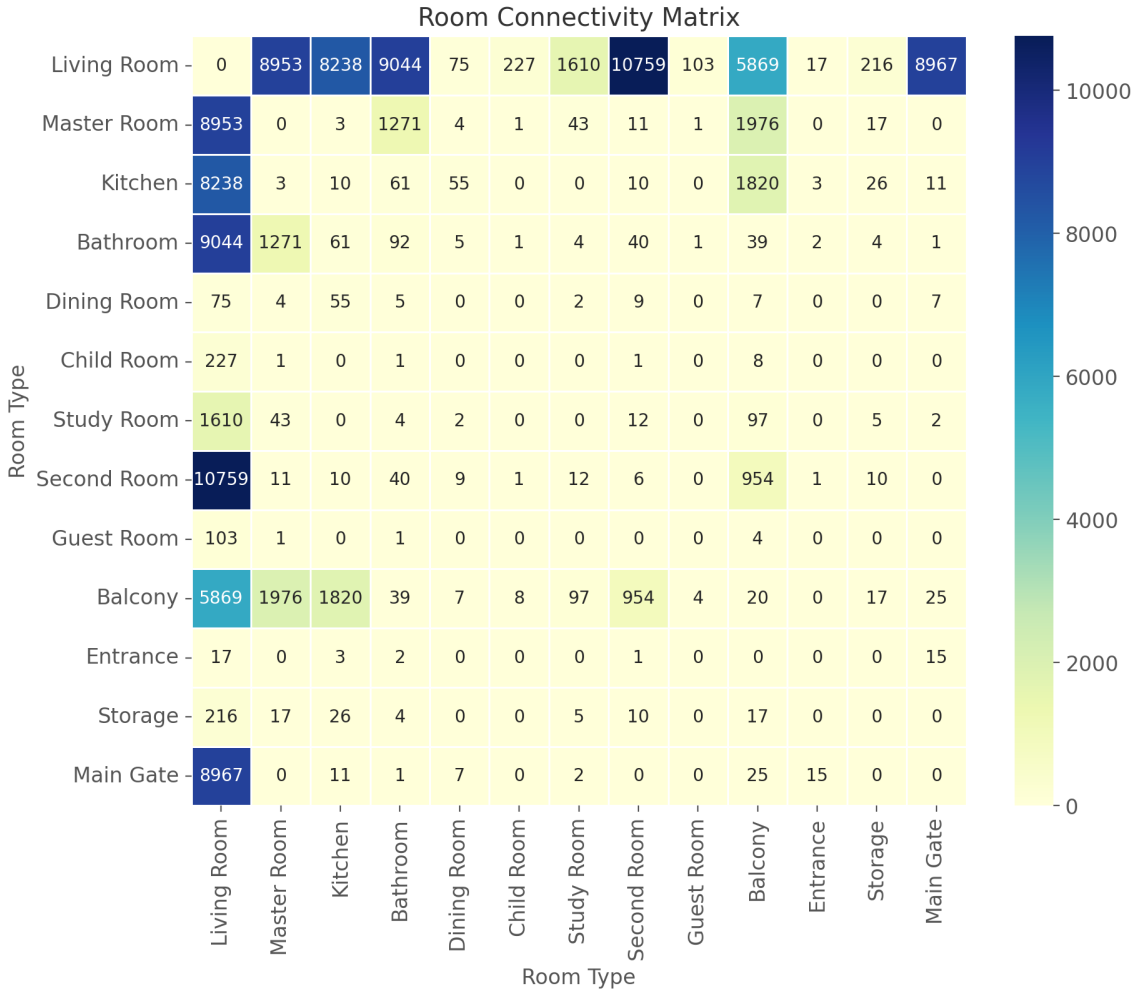


Figure 3.12: Room Connectivity Matrix

In the figure 3.12, the living room is the most connected space, linking to almost all other rooms. Second room and balcony also show high connectivity. Master room, kitchen, and bathroom have moderate connections. This suggests a central role for the living room, with second room and balcony as important secondary connective spaces.

In figure 3.13, the living room area is roughly normally distributed with a right skew. Master room is more symmetric. Kitchen is right-skewed with a clear peak. Bathroom is heavily right-skewed. Second room shows a bimodal distribution. Balcony is right-skewed with a long tail. Notably, dining, child, entrance, and storage rooms have tall bars near zero, indicating they're often absent in many house designs.

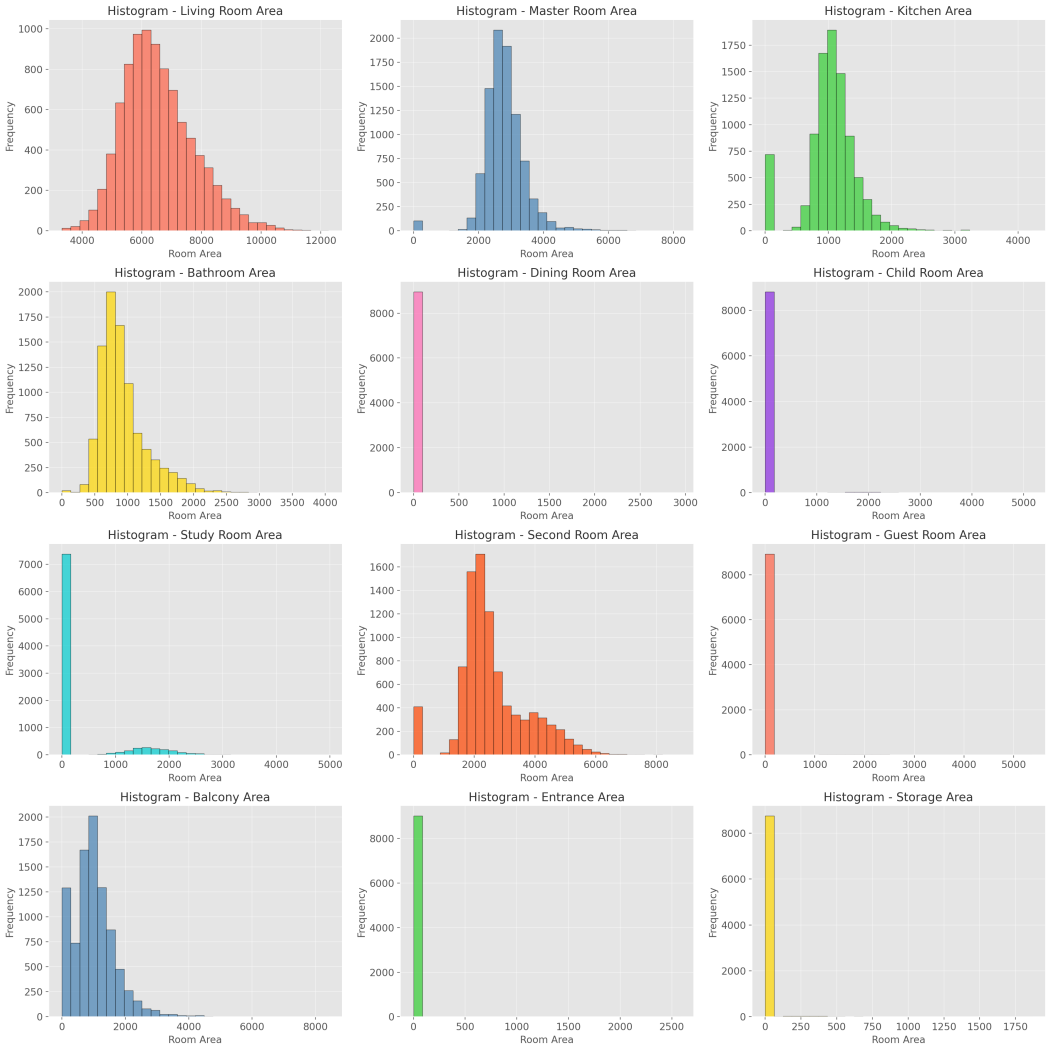


Figure 3.13: Histograms of Room Areas

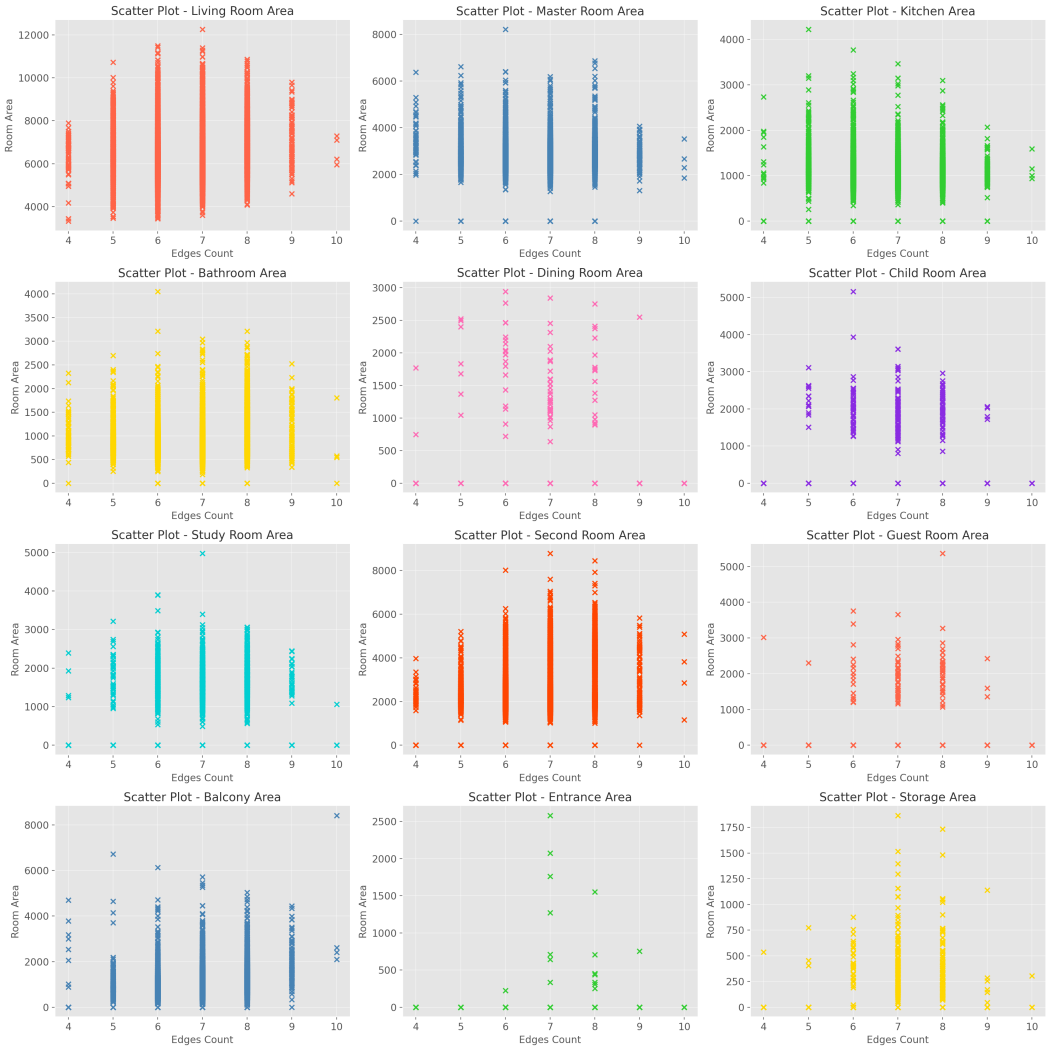


Figure 3.14: Scatter Plots of Room Areas vs. Edge Count

In figure 3.14, the living room shows a slight positive correlation between area and edge count. The second room area increases more clearly with edge count. Most other rooms show little correlation. The balcony area slightly decreases with more edges. This suggests only some rooms (mainly living and second) tend to be larger when more connected.

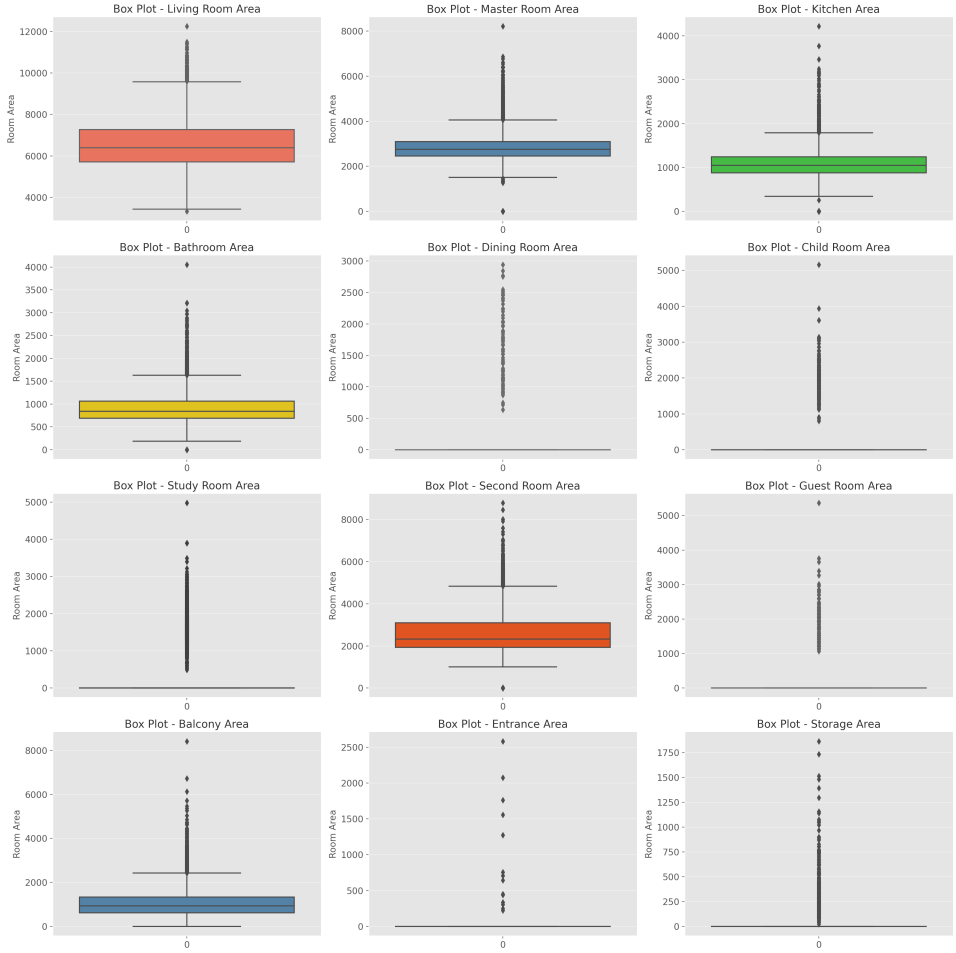


Figure 3.15: Box Plots of Room Areas

In figure 3.15, the living room has the largest median area and widest range. The master and second rooms follow in size. The kitchen, bathroom, and balcony show moderate, less variable sizes. Smaller rooms (dining, child, study) have compact distributions. Entrance and storage are consistently small. Outliers in most room types indicate some unconventional designs.

3.6.1 Final Dataset Analysis

The data reveals common patterns in house layouts. Living rooms are central, largest, and most connected. The master and second rooms are also significant. Kitchen and bathroom sizes are more standardized. Balconies are important connective spaces despite moderate sizes. Smaller rooms (dining, child, guest) show less variability and connectivity and are often absent in many designs. Room size-to-connectivity relationships vary by type. The second room's bimodal distribution

suggests two distinct design approaches. This data highlights a clear hierarchy of room importance and reveals how different spaces are prioritized in home designs, with some rooms being optional in many layouts.

The final annotated graph data, selected chronologically due to resource constraints, represents a more focused version of the initial filtered graphs, though some information from the initial dataset may have been missed. While the final dataset maintains the centrality of key rooms like the Living Room and its connections to spaces such as the Master Room and Kitchen, it filters out less frequent interactions seen in the initial graphs. This results in a more concentrated view of room connectivity, but the chronological selection process may have overlooked certain patterns present in the broader dataset, potentially limiting the diversity of interactions captured.

3.7 Dataset Transformation and Refinement

3.7.1 Annotation Transformation

The annotations that are collected are then aggregated following the majority vote. The data from the annotation site is passed in JSON format, which is converted to an HDF5 file containing the floor plan information and the label value from the JSON file. However, as binary classification is to be performed, the label values containing "A" and "B" are converted to "1" signifying yes, and label values containing "C" and "D" are converted to "0" meaning no.

The edge feature from filtered data is dropped, and all the individual vertex information for each node is converted to a single 2D array representing the center of the room. In doing so, the information about the front door will also be lost, but to counter it, the dummy node was modified with information from the door, where its centroid became the centroid of the front door.

3.7.2 Data Augmentation

As the information of the 2D array is being passed as a node feature, data augmentation was done with rotation. For example, a labeled data that has a label value of "1" will still be the same (as direction is not being considered for this data) if it is rotated by any amount of degrees. However, after doing a rotation, the node feature of the centroid will change according to the rotation, while the pattern will be the same, hence the label should still be the same. Doing such data augmentation allows us to introduce new data from the labeled data, while also allowing the model to explore new spaces for it to learn.

This still poses a challenge as within the 9028 labeled data, there are 4062 "1" labels and 4966 "1" labels. Thus, the class imbalance will worsen if the data is rotated in multiple angles uniformly. For this, graphs containing the label "0" are rotated 7 times at angles 0 (unchanged), 45, 90, 135, 180, 225, and 270 degrees, & graphs containing the label "0" are rotated 6 times at angles 0 (unchanged), 45, 90, 135, 180, 270 degrees. Doing so our total dataset size becomes 58230 labelled graphs. This can also help combat the problem's ambiguous nature by reinforcing the same patterns so that the model can learn better.

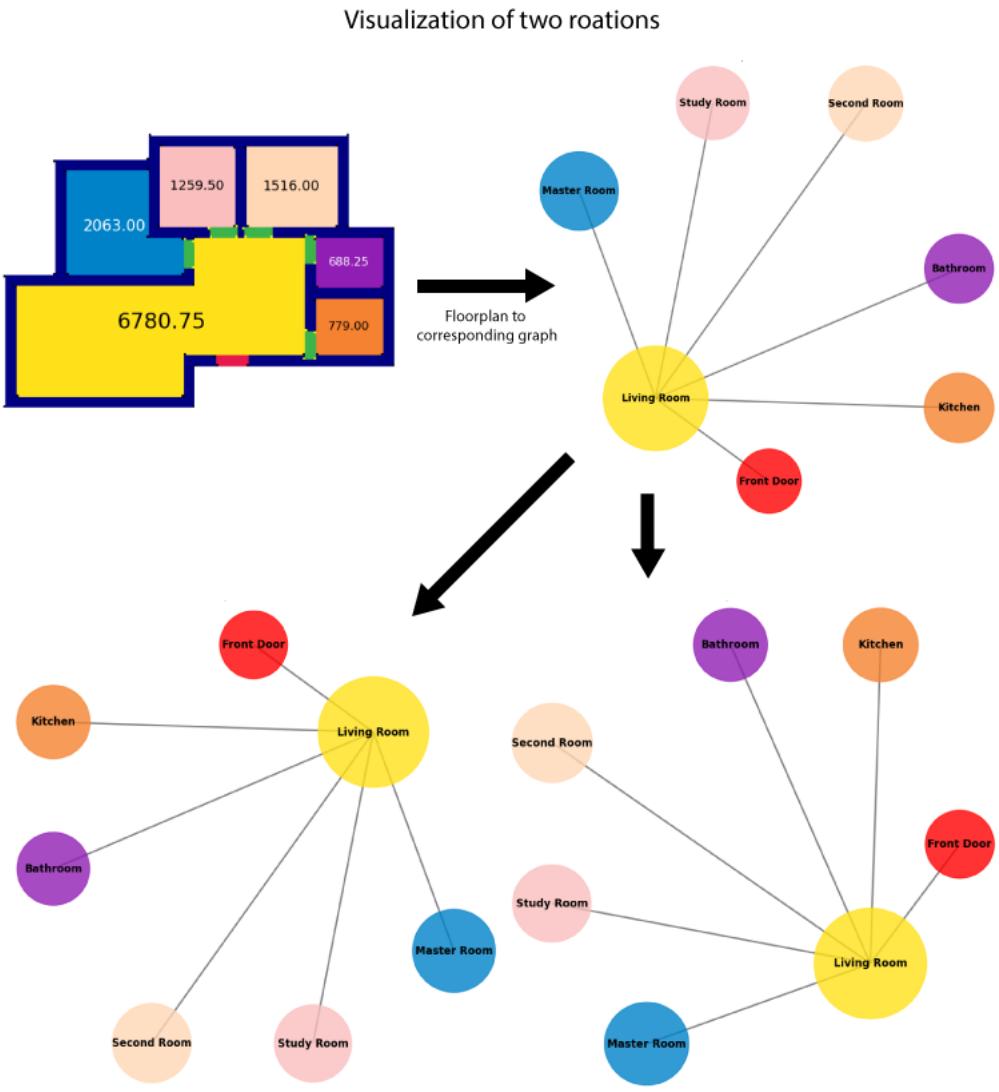


Figure 3.16: Data augmentation through rotation

3.8 Model Selection

3.8.1 Model Selection Criteria

The primary objective of this study is to perform graph-level binary classification to determine whether a given architectural floor plan aligns with Bangladeshi architectural standards. Considering the nature of the data and the task, we require models capable of capturing complex relationships within graphs and generalizing well to unseen data.

Criteria for model selection included:

- **Ability to Handle Graph-Structured Data**: Models must effectively process and learn from graph representations of floor plans.
- **Scalability**: Models should be scalable to accommodate large datasets and complex graphs.
- **Expressiveness**: Models need to capture both local and global patterns within the graphs.
- **Computational Efficiency**: Reasonable training and inference times are essential for practical applicability.

After evaluating various Graph Neural Network (GNN) architectures that meet these criteria, we selected three models known for their effectiveness in graph classification tasks.

3.8.2 Selected Models

The models chosen for this study are:

- **Graph Convolutional Networks (GCN)**
- **Graph Attention Networks (GAT)**
- **GraphSAGE**

These models were selected based on their unique approaches to aggregating and updating node information:

- **GCN**: Efficient and suitable for capturing general graph structures through spectral convolution.
- **GAT**: Incorporates attention mechanisms to weigh the importance of neighboring nodes, enhancing the model's ability to focus on relevant features.
- **GraphSAGE**: Capable of handling large graphs through inductive learning and flexible neighborhood aggregation methods.

Chapter 4

Model

4.1 Data loading and Train-test Split

4.1.1 Feature Representation in GNN Models

Effective feature representation is crucial for the performance of GNN models. In this study, each node and edge in the graph is associated with specific features that capture both geometric and semantic information.

Node Features:

- **Normalized Area**: Represents the size of the room, normalized between 0 and 1.
- **Normalized Centroid Coordinates**: (c_x, c_y), normalized to a range of [0, 1] based on the maximum coordinate value.
- **One-Hot Encoded Room Type**: A binary vector indicating the type of room among 13 possible types.
- **Positional Encodings**: Computed using sine and cosine functions to capture spatial relationships.

Edge Features:

- Edges represent connections between rooms (e.g., doors) and are unweighted in this study.

4.1.2 Data Processing and Graph Construction

Node and Edge Feature Selection

Area Normalization:

$$\text{normalized\_area} = \frac{\text{area} - \text{min\_area}}{\text{max\_area} - \text{min\_area}}$$

Centroid Normalization:

$$c_x = \frac{x_{\text{centroid}}}{\text{max\_coordinate}}, \quad c_y = \frac{y_{\text{centroid}}}{\text{max\_coordinate}}$$

One-Hot Encoding of Room Types:

Each room type is converted into a binary vector of length 13, where the index corresponding to the room type is set to 1.

Positional Encoding:

Applied using the formula mentioned in **attention**:

$$\text{PE}_{(p,2i)} = \sin\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right)$$
$$\text{PE}_{(p,2i+1)} = \cos\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right)$$

where p is the position (normalized centroid coordinate) and d_{model} is the dimensionality of the encoding (set to 64 in this study). By using a positional encoding, the model is given a structured way to understand the relative position of rooms in the layout. This way, the model can learn a meaningful graph representation of the floor plans in the dataset,

Graph Construction from Annotated Data

The process involves:

1. **Data Extraction**:
   - Reading nodes and edges from the HDF5 files.
   - Extracting attributes such as area, centroid coordinates, and room types.
2. **Feature Preparation**:
   - Normalizing area and centroid coordinates.
   - One-hot encoding room types.
   - Computing positional encodings for each node based on centroid positions.
   - Concatenating all features to form the node feature matrix X.
3. **Edge Index Construction**:
   - Edges are extracted and formatted into an edge index tensor required by PyTorch Geometric.
   - Edge indices represent connections between nodes in the graph.
4. **Label Assignment**:
   - Binary Labels in the dataset for each graph are extracted. This label used to train the binary classifer mode,
5. **Data Object Creation**:

- Using the `Data` class from PyTorch Geometric to encapsulate node features, edge indices, and labels into a single graph data object ready for model training.

4.1.3 Train-Test Split

Initially, 58230 sample of data was loaded from the hdf5 file into a data list. But before training, a stratified split is performed on the dataset to ensure an even distribution of classes across the training, validation, and test sets. The function begins by extracting the labels from the data list, and then balances the dataset by selecting an equal number of samples from each class. This balance is achieved by identifying the smaller class and restricting the larger class to the same number of samples.

The indices for both classes are shuffled to ensure randomness, and then the dataset is split into training, validation, and test sets. This stratification ensures that the class proportions remain consistent across all splits.

In the end we have 28434 samples for each of the classes which we split by into training, validation and test set in the ration of 60, 20 and 20 respectively. This ended us with 34120 training samples, 11374 samples for validation and test each.

Set	**Class 0 Samples**	**Class 1 Samples**	**Total Samples**	**Split Ratio**
Balanced	28,434	28,434	56,868	100%
Training	14,217	14,217	34,120	60%
Validation	5,687	5,687	11,374	20%
Test	5,687	5,687	11,374	20%

Table 4.1: Train-Validation-Test Split with 60-20-20 Ratio

4.2 Model Overview

In this section, we provide the theoretical background, historical development, and mathematical foundations of the three Graph Neural Network (GNN) models applied in this study: Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), and GraphSAGE.

Graph Convolutional Networks (GCN)

Historical Development

GCNs were introduced by Kipf and Welling in 2016 [6], pioneering the extension of deep learning techniques to graph-structured data. They formulated a localized first-order approximation of spectral graph convolutions, enabling efficient and scalable learning on graphs.

Mathematical Foundations

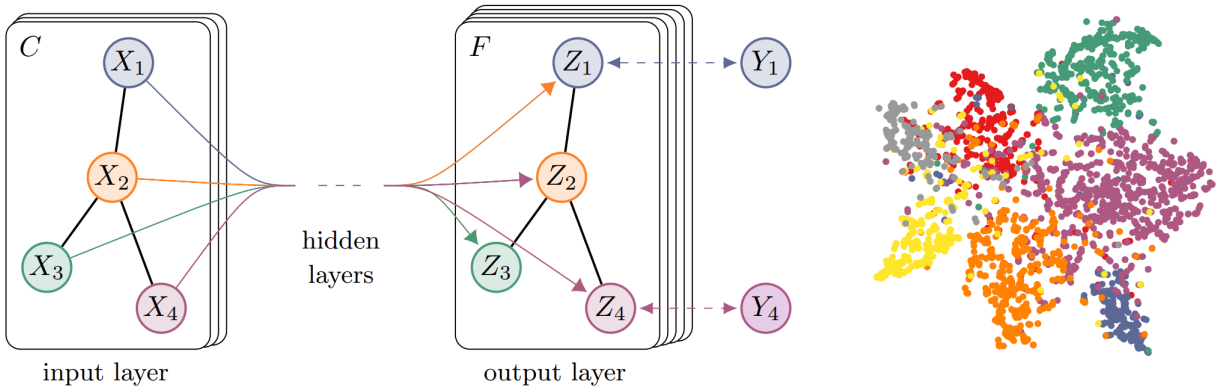
GCNs are rooted in spectral graph theory, where the convolution operation is defined in terms of the eigenfunctions of the graph Laplacian. The layer-wise propagation rule for a GCN is:

$$H^{(l+1)} = \sigma\left(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}H^{(l)}W^{(l)}\right)$$

where:

- $H^{(l)}$ is the matrix of node features at layer l,
- $\hat{A} = A + I$ is the adjacency matrix with added self-loops,
- $\hat{D}$ is the degree matrix of $\hat{A}$,
- $W^{(l)}$ is the trainable weight matrix,
- σ is an activation function (e.g., ReLU).

This formulation allows the model to aggregate information from a node's immediate neighbors, effectively capturing local graph structures.



(a) Graph Convolutional Network (b) Hidden layer activations

Figure 4.1: GCN Model Illustrated in [6]

Graph Attention Networks (GAT)

Historical Development

GATs were proposed by Veličković et al. in 2017 [8], introducing attention mechanisms to GNNs. GATs allow nodes to weigh the importance of their neighbors during feature aggregation, addressing limitations of earlier models that treated all neighbors equally.

Mathematical Foundations

GATs compute attention coefficients to weigh the influence of neighboring nodes:

1. **Compute Unnormalized Attention Coefficients**:

$$e_{ij} = \text{LeakyReLU}\left(\vec{a}^{\top}[Wh_i \,||\, Wh_j]\right)$$

2. **Normalize Attention Coefficients**:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

3. **Compute Node Representations**:

$$h_i' = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} W h_j\right)$$

where:

- h_i and h_j are the input features of nodes i and j,
- W is the weight matrix,
- $\vec{a}$ is the attention vector,
- $||$ denotes concatenation,
- σ is an activation function.

This mechanism allows the model to focus on the most relevant parts of the graph.

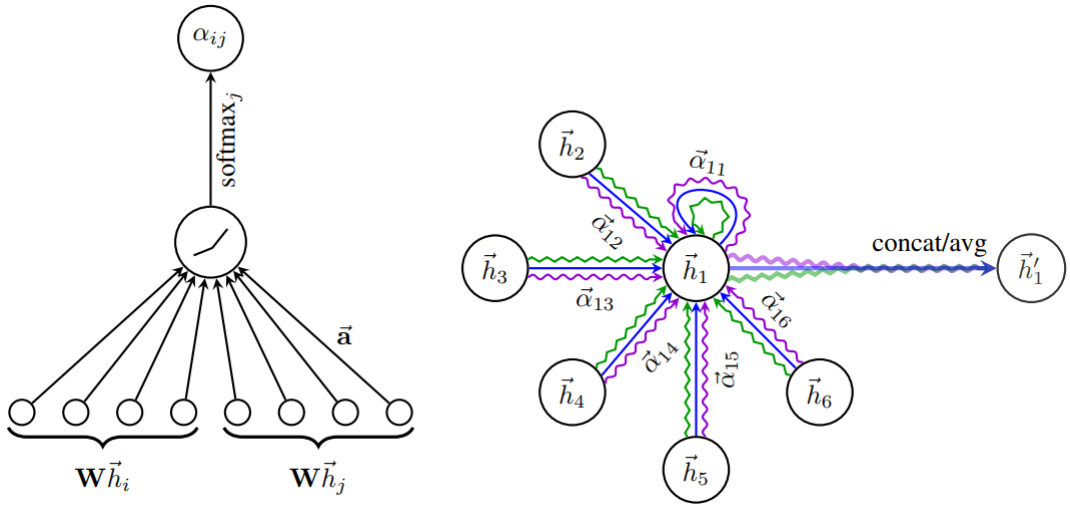


Figure 4.2: Attention mechanism illustrated in [8]

GraphSAGE

Historical Development

GraphSAGE was developed by Hamilton et al. in 2017 [5] to improve scalability and inductive learning capabilities in GNNs. It introduced a framework for generating node embeddings by sampling and aggregating features from a node's local neighborhood, allowing the model to generalize to unseen nodes or graphs.

Mathematical Foundations
GraphSAGE updates node embeddings by aggregating neighbor information:

1. **Aggregate Neighbor Representations**:

$$h_{\mathcal{N}(i)}^{(k)} = \text{AGGREGATE}^{(k)}\left(\{h_j^{(k-1)}, \forall j \in \mathcal{N}(i)\}\right)$$

2. **Compute Node Representations**:

$$h_i^{(k)} = \sigma\left(W^{(k)} \cdot \text{CONCAT}\left(h_i^{(k-1)}, h_{\mathcal{N}(i)}^{(k)}\right)\right)$$

where:

- $h_i^{(k)}$ is the embedding of node i at layer k,
- $\text{AGGREGATE}^{(k)}$ is a permutation-invariant aggregator function (e.g., mean, max),
- $W^{(k)}$ is the trainable weight matrix,
- σ is an activation function.

GraphSAGE allows for inductive learning by learning how to aggregate features from a node's local neighborhood.

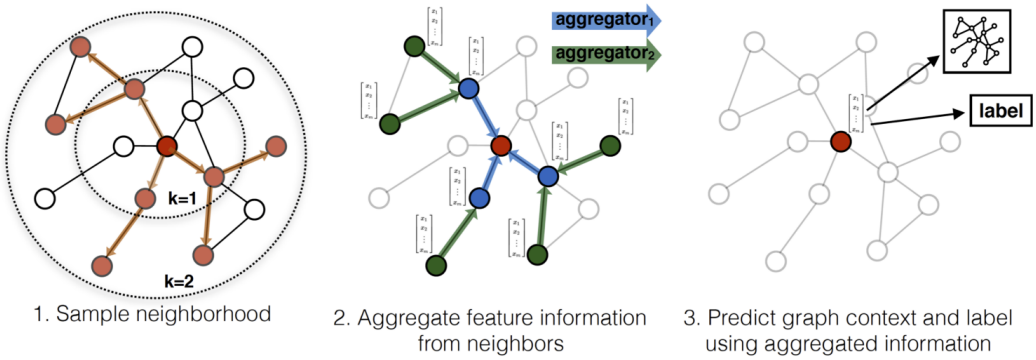


Figure 4.3: Visual illustration of the GraphSAGE and aggregate approach [5]

4.3 Model Architecture

4.3.1 Abstract Model Architecture

The custom models in this study follow a common abstract architecture designed for classifying architectural floor plans. This architecture comprises the following key components:

- **Graph Convolutional Layers:** All models use graph convolutional layers (e.g., GCNConv, GATConv, SAGEConv) to aggregate and transform node features, enabling the model to learn spatial and topological relationships between rooms in the floor plans.

- **Global Pooling:** After the graph convolutional layers, a global pooling mechanism (mean, max, or a combination of both) is applied to condense node-level embeddings into a single graph-level representation. This step is crucial for producing a fixed-size vector that summarizes the entire floor plan.
- **Fully Connected Network (FCN):** The pooled graph representation is passed through a fully connected network, consisting of one or more linear layers. These layers further refine the graph-level features and produce the final classification output.
- **Binary Classification Output:** The final output layer performs binary classification, determining whether a given floor plan conforms to the specified architectural standards.
- **Loss function and Optimizer:** The binary cross-entropy [24] loss function (via nn.CrossEntropyLoss) is used to optimize the model. The model parameters are updated using the Adam optimizer [3]. Additionally, L2 weight decay is applied to regularize the model.

4.3.2 Model-Specific Differences

Although the abstract architecture remains consistent across all models, each model incorporates specific adjustments:

GCN Architecture

The *GCN* model uses two GCNConv layers to propagate node information, with global mean pooling applied afterward. The fully connected network consists of two linear layers that process the node embeddings and output the classification result.

GAT Architecture

The *GAT* model introduces graph attention mechanisms through its GATConv layers, with each attention head focusing on different regions of the graph. It applies both global max and mean pooling, which are concatenated before passing through the fully connected network.

GraphSAGE Architecture

The *GraphSAGE* model uses SAGEConv layers with a "mean" aggregation function to gather information from neighboring nodes. Afterward, global mean pooling is applied, followed by fully connected layers to produce the final classification output.

4.3.3 Hyperparameter Summary

The hyperparameter tuning is done using Optuna library [9] which uses a form of Bayesian optimization called TPE (Tree-structured Parzen Estimator) for hyperparameter tuning. The best hyperparameters used for each model are summarized in Table 4.2.

Table 4.2: Hyperparameters for Each Model

Hyperparameter	**GCN**	**GAT**	**GraphSAGE**
Learning Rate	0.001	0.00007	0.0005
Batch Size	32	32	16
Optimizer	Adam optimizer without weight decay	Adam optimizer with weight decay of 4.5e-5	Adam optimizer with weight decay of 1e-6
Number of Layers	2 (GCNConv and FCN)	2	2
Hidden Dimension	256	512	256
Number of Heads	N/A	1	N/A
Dropout Rate	Not applied	0.271	0.06

Table 4.3: Comparison of Custom GNN Model Architectures

Feature	**GCN**	**GraphSAGE**	**GAN**
Number of Convolution Layers	2	2	2
Type of Convolution Layers	`GCNConv`	`SAGEConv`	`GATConv`
Aggregation Function	Spectral Convolution	Mean	Attention
Node Feature Transformation	$80 \rightarrow 256$, then 256	$80 \rightarrow 256$, then 256	$80 \rightarrow 512$
Consistency in Embedding Size	Maintained at 256	Maintained at 256	Maintained at 512
Attention Mechanism	N/A	N/A	Yes (single head)
Pooling Mechanism	Global Mean Pooling	Global Mean Pooling	Global Max and Mean Pooling
Fully Connected Network (FCN)	Two Linear Layers: $64 \rightarrow 64$, $64 \rightarrow 2$	Two Linear Layers: $256 \rightarrow 256$, $256 \rightarrow 2$	Two Linear Layers: $512 \rightarrow 512$, $512 \rightarrow 2$
Dropout Rate	N/A	0.06	0.271
Output Classes	Binary Classification (2 classes)	Binary Classification (2 classes)	Binary Classification (2 classes)
Learning Rate	0.001	0.0005	0.00007
Batch Size	32	16	32
Optimizer	Adam (no weight decay)	Adam (weight decay=1e-6)	Adam (weight decay=4.5e-5)

Chapter 5

Results and Discussion

5.1 Result Analysis

The models are evaluated on different statistical analysis and the final results are analyzed and compared between the models.

5.1.1 GCN

The performance of the GCN model is analyzed based on statistical analysis:

- **Training and Validation Accuracy Over Epochs**
  The accuracy curves over 300 epochs demonstrate steady improvement in both training and validation accuracy. Initially, the model shows a rapid increase in accuracy, which begins to plateau after around 150 epochs. The **final validation accuracy is approximately 85%** which indicates good generalization capability with minimal overfitting. The closeness between the training and validation curves suggests that the model has learned the patterns in the data well and generalizes effectively to unseen samples.

- **Training and Validation Loss Over Epochs**
  The loss curves complement the accuracy curves by showing the decline in both training and validation loss over epochs. The sharp drop in the early epochs indicates that the model learned quickly which further decreases with more training. The final validation loss stabilizes around **0.30**, indicating effective learning without significant divergence between the training and validation curves, further confirming that the model does not suffer from overfitting.

- **Confusion Matrix**
  The GCN model correctly classified **4945** instances of Class 0 and **4949** instances of Class 1. The false positives and false negatives were relatively low, with **742** misclassified instances for Class 0 and **738** for Class 1. This score suggests that the model has high true positives and low false positives in both classes.

- **Precision-Recall Curve**
  The precision-recall curve provides insight into the model's ability to maintain high precision and recall across different thresholds. The GCN model's **AUC**

score of 0.954 indicates strong performance in distinguishing between the two classes. The curve maintains a high level of precision across a broad range of recall values. This proves that the model effectively handles class imbalance and lower probability of false positives.

- **Receiver Operating Characteristic (ROC) Curve**
  The ROC curve is another measure of classification performance. The **AUC score of 0.9507** suggests excellent discrimination capability. Again, the curve is well above the diagonal line further supports that the model is making accurate predictions across both classes.

- **t-SNE Visualization of Graph Embeddings**
  The t-SNE plot illustrates the clustering of graph embeddings learned by the model. Here, each point representing a sample from either Class 0 (blue) or Class 1 (orange). The separation between the two clusters is visible, suggesting that the model successfully learned to represent the classes differently. While there is some overlap, the overall clustering pattern indicates that the embeddings contain meaningful features that allow the model to differentiate between the two classes. Even if there is some overlap, the overall clustering pattern indicates that the embeddings contain meaningful features that allow the model to differentiate between the two classes.

Summary:
The GCN model demonstrates strong performance on the binary classification task. The confusion matrix highlights its balanced classification with low false positives and low false negatives, while both the precision-recall and ROC curves shows lower probability of making mistakes. The training and validation metrics suggest effective learning and generalization without overfitting. The t-SNE visualization further supports that the learned embeddings provide useful representations for classification. Overall, the GCN model is well-suited for the graph-based binary classification task.

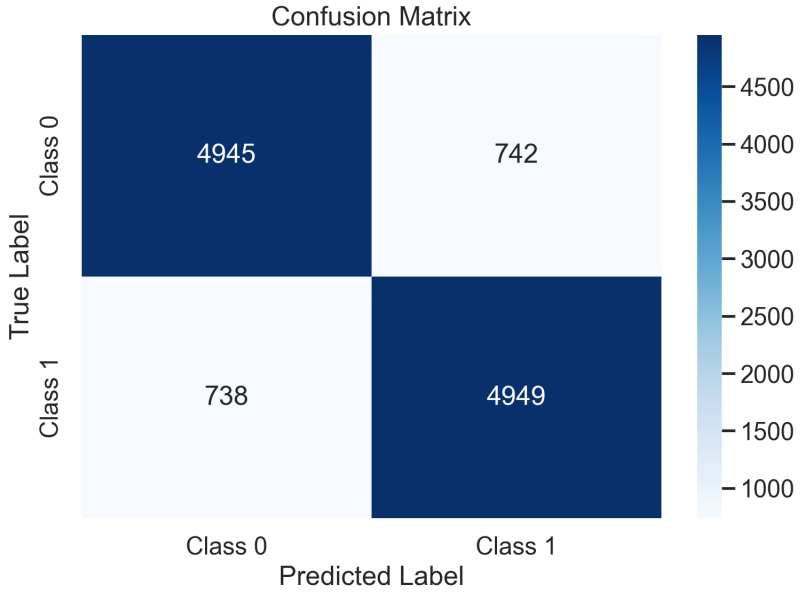
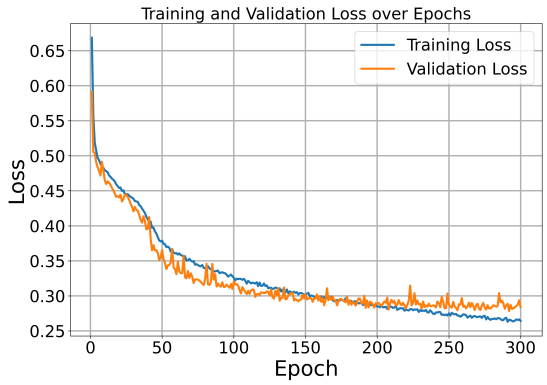
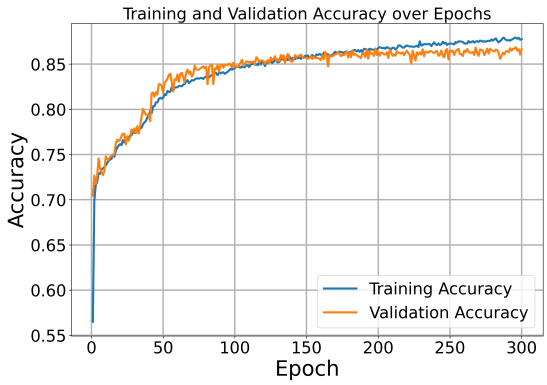


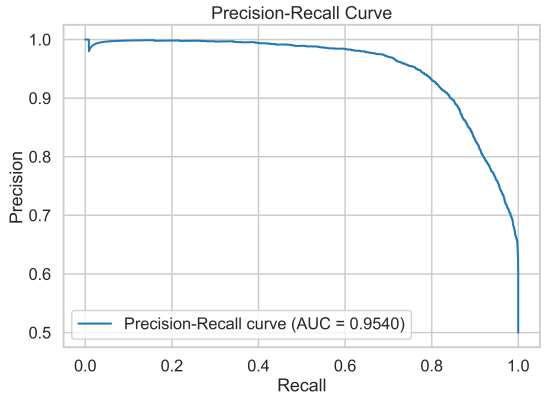
Figure 5.1: Confusion Matrix for GCN



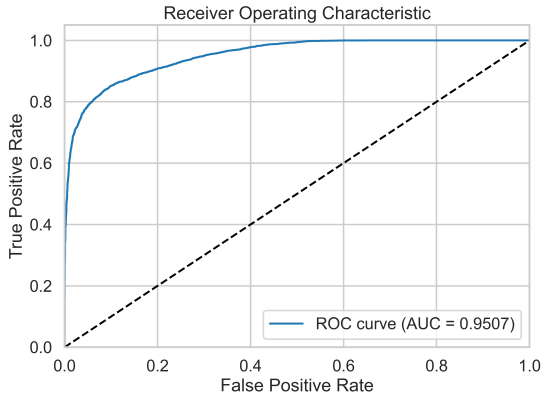
(a) Training and Validation Loss over Epochs



(b) Training and Validation Accuracy over Epochs



(c) Precision recall curve for GCN



(d) Receiver Operating Characteristic

Figure 5.2: Training and Validation Metrics over Epochs for GCN

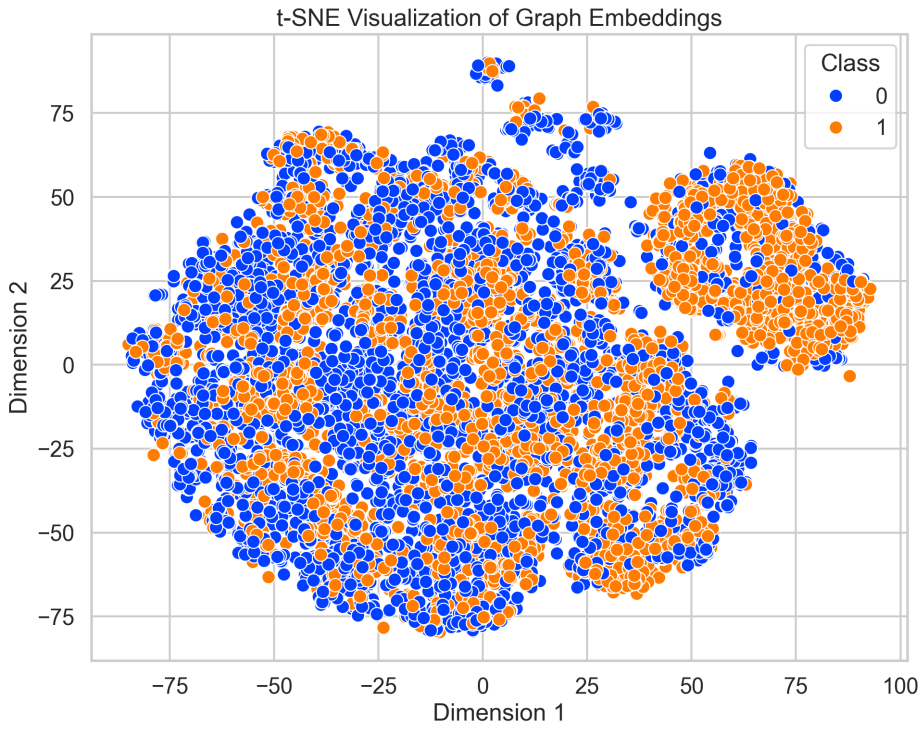


Figure 5.3: t-SNE Visualization for GCN

5.1.2 GraphSAGE

The performance of the GCN model is analyzed based on statistical analysis:

- **Training and Validation Accuracy Over Epochs**
  The accuracy curves over 300 epochs show a steady increase in both training and validation accuracy. Similar to the GCN model, the accuracy plateaus after around 150 epochs, with the final validation accuracy reaching approximately **85%**. The training and validation accuracy are close, which suggests that the model generalizes well without overfitting.

- **Training and Validation Loss Over Epochs**
  The training and validation loss curves show consistent improvement, with both losses decreasing over time. The final validation loss stabilizes around **0.30**, indicating that the model converged effectively. The loss curves for both training and validation data closely follow each other, confirming the model's ability to learn without overfitting.

- **Confusion Matrix**
  The confusion matrix shows that the GraphSAGE model correctly classified **4973** instances of Class 0 and **4933** instances of Class 1. The false positives and false negatives were slightly higher compared to the GCN model, with **714** misclassified instances for Class 0 and **754** for Class 1. However, the model still demonstrates a balanced classification across both classes.

- **Precision-Recall Curve**
  The precision-recall curve for the GraphSAGE model shows an **AUC score of 0.9543** which is very similar to GCN model. This indicates that the model

retains high precision across a broad range of recall values.

- **Receiver Operating Characteristic (ROC) Curve**
  The ROC curve shows the **AUC score of 0.9512** which reflects strong discriminative power. Again, this score is similar to GCN which is also evident in the curve.

- **t-SNE Visualization of Graph Embeddings**
  The t-SNE plot shows the clustering of graph embeddings generated by the model. Although some overlap exists, there is a clear separation between the two classes (Class 0 in blue and Class 1 in orange). The overall clustering pattern shows that the model has learned meaningful representations of the data, which contribute to its classification performance.

Summary:
Just like GCN, GraphSAGE model also performs well on the binary classification task. It shows a balanced classification in the confusion matrix and strong performance metrics in both the precision-recall and ROC curves. The evidence of good generalization ability is shown in accuracy and loss curves. The t-SNE visualization further supports the model's capacity to differentiate between the two classes through meaningful graph embeddings. Overall, GraphSAGE shows comparable performance to the GCN model and proves to be a suitable model for the graph-based binary classification task.

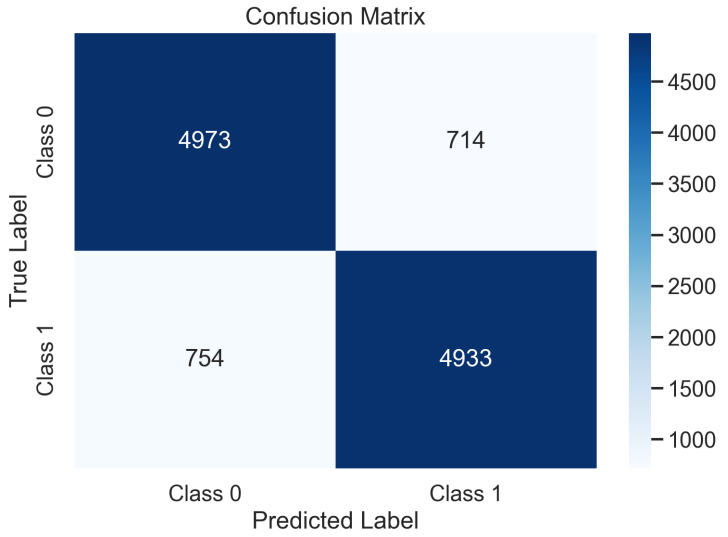
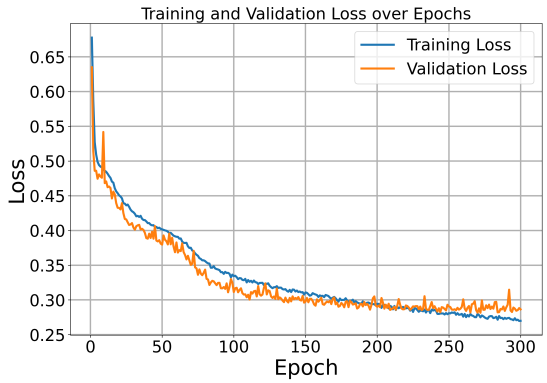
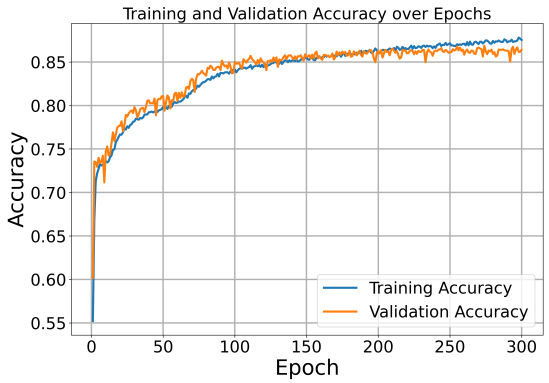


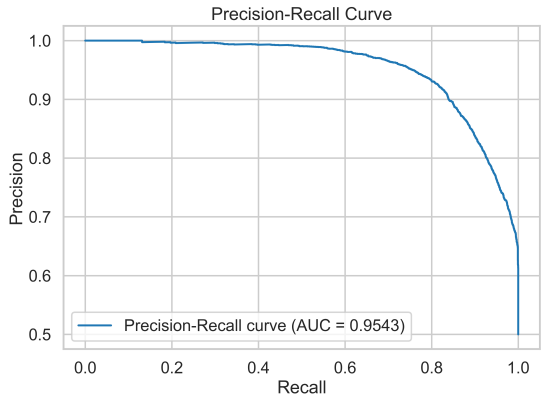
Figure 5.4: Confusion Matrix for GraphSAGE



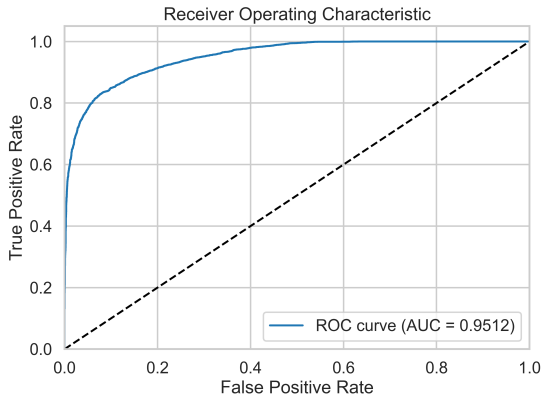
(a) Training and Validation Loss over Epochs



(b) Training and Validation Accuracy over Epochs



(c) Precision-Recall Curve for GraphSAGE



(d) AUC-ROC Curve for GraphSAGE

Figure 5.5: Training and Validation Metrics over Epochs for GraphSAGE

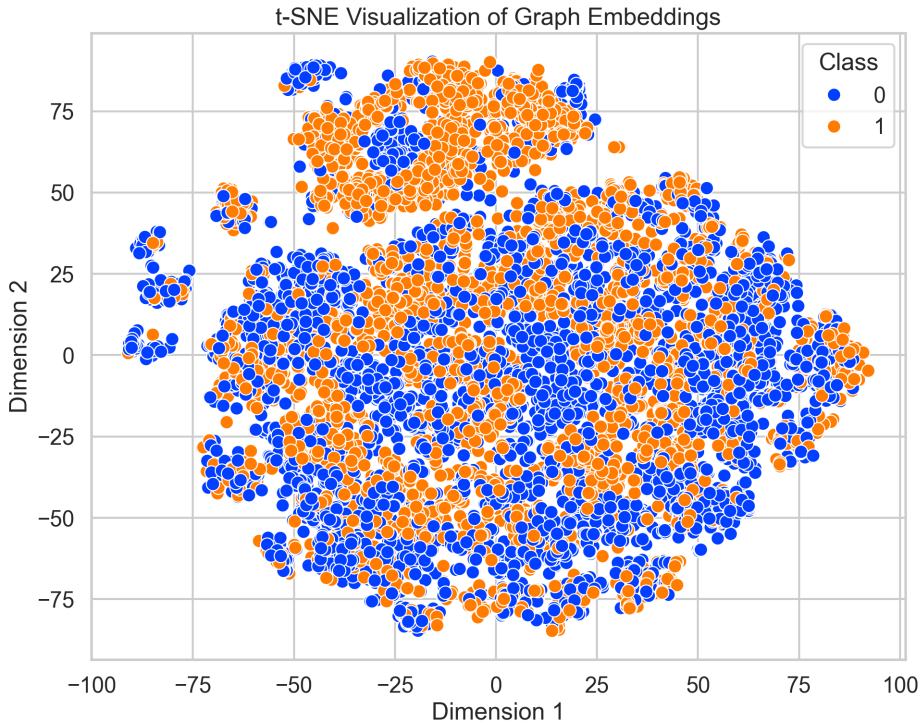


Figure 5.6: t-SNE Visualization for GraphSAGE

5.1.3 GAT

The performance of the GAT model for the graph binary classification task is summarized through six key visualizations, as explained below:

- **Confusion Matrix**
  The confusion matrix indicates that the GAT model correctly classified **4086** instances of Class 0 and **2243** instances of Class 1. However, the model also misclassified **1945** instances of Class 1 as Class 0 which indicates a noticeable imbalance in performance. It demonstrates where Class 1 is harder for the model to predict accurately compared to Class 0.

- **Precision-Recall Curve**
  The precision-recall curve shows that the GAT model has an **AUC score of 0.84**. While the precision remains relatively high at lower recall values, the precision significantly decreases as recall approaches 1, indicating that the model struggles with maintaining precision as more positive samples are considered.

- **Receiver Operating Characteristic (ROC) Curve**
  The ROC curve shows that the model achieves an **AUC score of 0.86**, suggesting moderate discriminative power. The curve does not reach as close to the top-left corner as the previous models (GCN or GraphSAGE), which reflects the model's lower overall ability to differentiate between the two classes.

- **Training and Validation Accuracy Over Epochs**
  The training accuracy steadily increases over time, reaching around **0.74**, while the validation accuracy fluctuates more significantly, indicating some instability during training. The variation in the validation accuracy suggests potential overfitting or difficulty in generalizing to the validation data.

- **Training and Validation Loss Over Epochs**
  The loss curves reveal a decrease in both training and validation loss, with the training loss stabilizing after several epochs. However, the validation loss exhibits severe fluctuations, This shows that the model may not generalize as well to unseen data.

- **t-SNE Visualization of Graph Embeddings**
  The t-SNE visualization of the embeddings shows that while the GAT model has learned some level of distinction between the two classes (with some distinct clusters). But there is significant overlap between the two classes. This overlap further confirms the model's difficulties in fully separating the two classes, particularly Class 1.

Summary:
The GAT model shows moderate performance in the binary classification task. The confusion matrix reveals a challenge in identifying Class 1 instances, with higher false negatives compared to other models. The AUC scores in both the precision-recall and ROC curves reflect this moderate performance. The validation accuracy over the epoch is also unstable. The t-SNE visualization suggests the model struggles to distinguishing between the two classes. Overall, while the GAT model performs noticably worse than GraphSAGE and GCN.

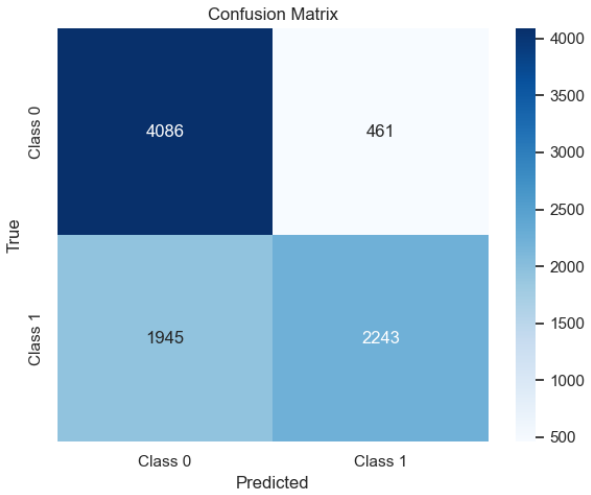
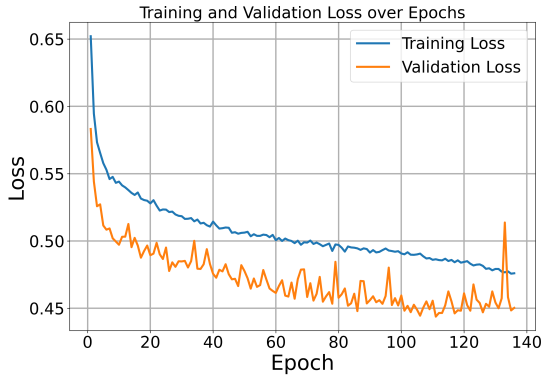
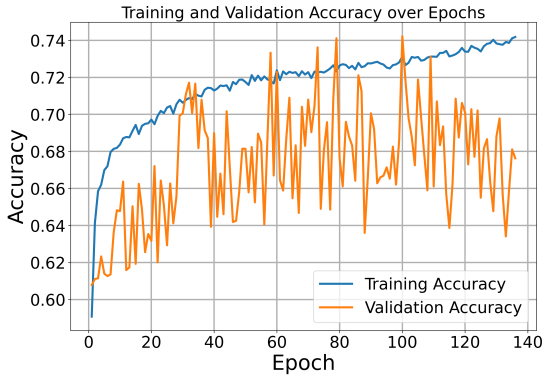


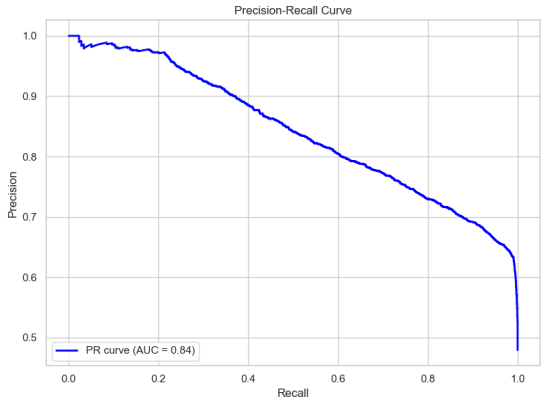
Figure 5.7: Confusion Matrix for GAT



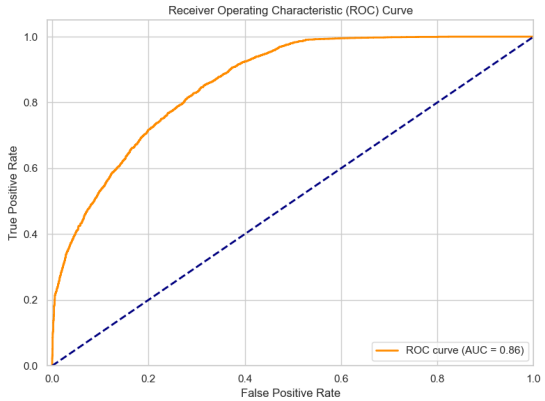
(a) Training and Validation Loss over Epochs



(b) Training and Validation Accuracy over Epochs



(c) Precision-Recall Curve for GAT



(d) AUC-ROC Curve for GAT

Figure 5.8: Training and Validation Metrics over Epochs for GAT

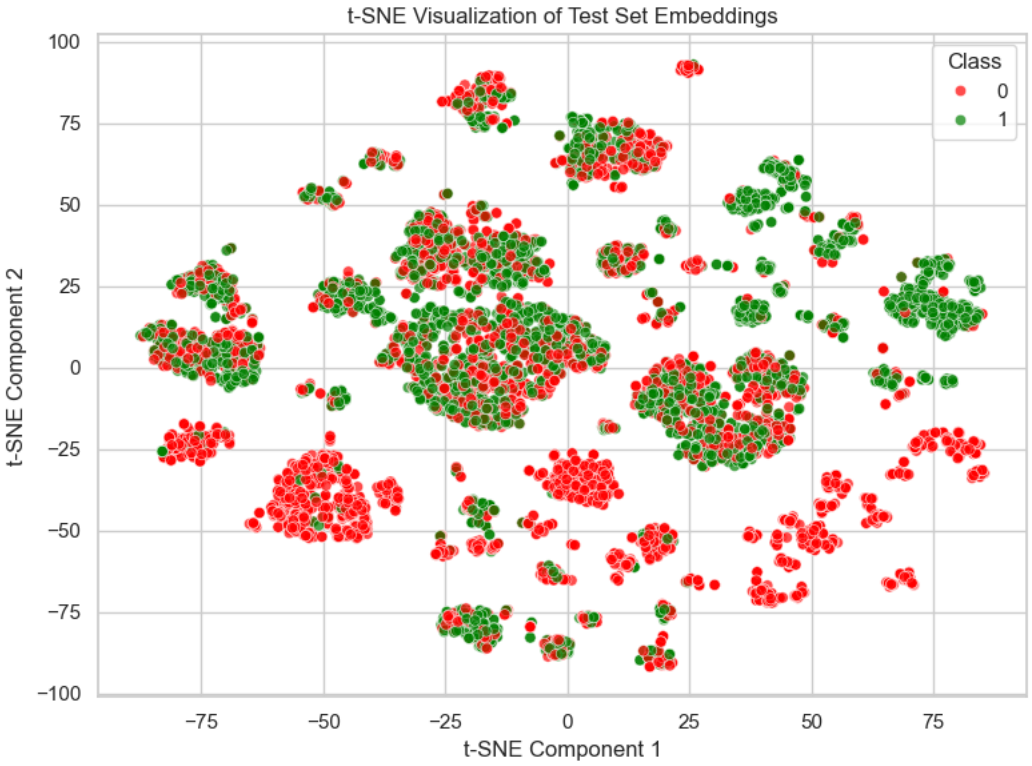


Figure 5.9: t-SNE Visualization for GAT

5.1.4 Model Comparison: Loss and Accuracy

- **Loss Comparison:**
  GCN and GraphSAGE: Both models exhibit nearly identical loss patterns in both training and validation. They begin with higher losses (0.60-0.65) and steadily decrease over the course of 300 epochs. By the end of the training, both models achieve training losses around 0.25 and validation losses around 0.30-0.35. This indicates efficient learning with good generalization, as the models maintain a smooth and stable decline in both training and validation loss, without significant fluctuations or divergence.

  GAT: In contrast, the GAT model struggles with higher initial losses and slower convergence. It starts at a higher training loss (0.65) and plateaus around 0.50, showing it doesn't minimize the loss as effectively as GCN and GraphSAGE. The validation loss exhibits a high degree of variance, oscillating between 0.80 and 0.90, particularly after 100 epochs. This instability suggests GAT is prone to overfitting or underfitting, unable to generalize well to the validation set.

- **Accuracy Comparison:**
  GCN and GraphSAGE: These models achieve similarly high levels of accuracy in both training and validation. The training accuracy climbs steadily and stabilizes at around 0.85 for both models, indicating robust learning from the training data. Their validation accuracy curves also stabilize close to 0.85, demonstrating that these models generalize well to unseen data without signs

of overfitting. The consistency between training and validation accuracy confirms their balanced performance.

GAT: The GAT model exhibits significantly lower training and validation accuracy compared to GCN and GraphSAGE. While its training accuracy plateaus at around 0.70, the validation accuracy fluctuates erratically between 0.40 and 0.70, reflecting its struggle to maintain consistent performance. The instability in validation accuracy further highlights GAT's difficulty in generalizing to new data, making it less reliable compared to GCN and GraphSAGE for this task.

Summary:
GCN and GraphSAGE perform similarly and show strong overall performance. Their loss and accuracy curves indicate efficient learning, good generalization, and stable convergence without overfitting. In contrast, GAT lags behind both models, with slower convergence, higher loss, and unstable validation accuracy. These issues suggest that GAT is less suited for this specific binary classification task without further tuning.

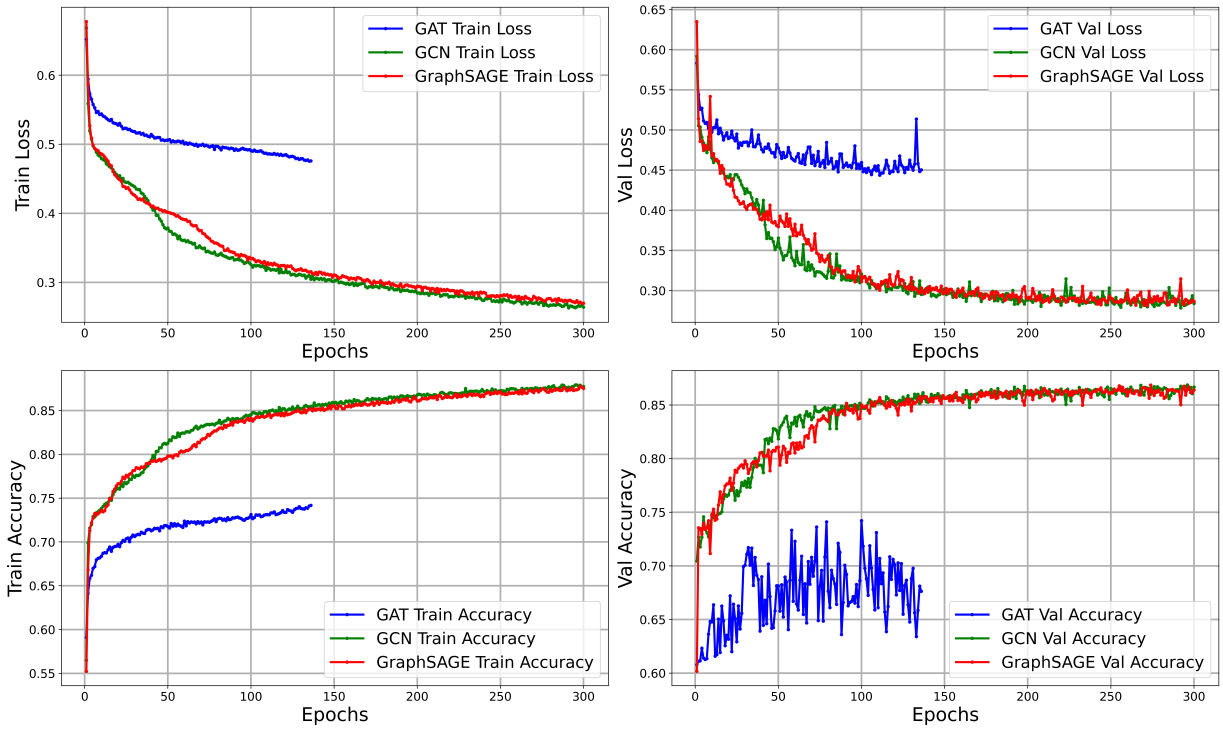


Figure 5.10: Loss and Accuracy between models over the epochs

Test Results

The models are evaluated across various metrics necessary to determine their competence in capturing the patterns in architectural floor plans and distinguishing between them. Here, **Accuracy** measures the overall correctness of the model. Although it can be misleading in imbalanced datasets, we trained, validated and tested our with equal number of class present in the dataset. **Precision** indicates how many of the predicted positive cases are actually true positives, it is useful to understand which model has the lowest false positive. **Recall** (or sensitivity) measures the ability to identify all true positives, which in our case is useful to determine which model can the extract the highest number of applicable floor plan from the total dataset. **F1 Score** shows the balance between precision and recall. Lastly, **AUC-ROC** evaluates a model's ability to distinguish between classes. This is very important for our task which is binary classification.

Table 5.1: Test Results for GAT, GCN, and GraphSAGE Models

Metric	**GAT**	**GCN**	**GraphSAGE**
Loss	0.4462	0.2806	**0.2787**
Accuracy	0.7246	0.8699	**0.8709**
Precision	0.8295	0.8696	**0.8736**
Recall	0.5356	**0.8702**	0.8674
F1 Score	0.6509	0.8699	**0.8705**
AUC-ROC	0.8609	0.9507	**0.9512**

The table clearly shows GCN and GraphSAGE outperforms GAT across all the metrics.GraphSAGE shows marginally better performance than GCN in most areas, particularly in precision and AUC-ROC. Although GAT has decent accuracy and precision, it falls short in all other metrics. GAT struggles with recall, leading to a lower F1 score. On the other hand, GCN and GraphSAGE showcases excellent accuracy, precision, and recall. Both of them have very good AUC-ROC score which indicates they are able to distinguish between the classes properly. GraphSAGE is the best model based on the results. It has better scores in all metrics except recall.

The results suggest that GCN and GraphSAGE models are the most suitable for tasks requiring balanced accuracy, precision, and recall. These models are better suited for architectural design validation, where minimizing both false positives and false negatives is important. In contrast, GAT struggles with recall is unsuitable for most cases compared to GraphSAGE. Overall, GraphSAGE is the clear outlier in terms of its performance, but GCN is not too far behind.

5.2 Limitations and Challenges

5.2.1 Ambiguities in Architectural Concepts

As architecture is a true field of creativity and aesthetics it does not hold any kind of perfections. The conceptualization of residential floor plans is ambiguous because of a set of socio-economic, cultural, and geographical influences. It varies from man to man according to their social aspects, economic standards, cultural background, geographical challenges, etc. These elements contribute to a multifaceted understanding of what constitutes an ideal living environment, underscoring the complexity and subjectivity of architectural design. The work of the architect does not involve mere implementation of the design but also various needs and expectations of the client, while also considering local conditions and material availability.

Architecture is a living atmosphere where one can experience their living style according to their taste. Individuals from different walks of life generally exhibit distinct preferences regarding their residence structures. For instance, a highly paid doctor or businessperson would appreciate modern and spacious architectural spaces that reflect luxury. On the other hand, an artist or photographer would prefer a more eclectic and aesthetic style that allows for creativity and flexibility. This reflects not only personal choice but also the socio-economic factors that can shape one's lifestyle and aspirations. That is why personal taste and cultural tendencies influence architectural design. Therefore, architects engage in a thorough understanding of their client's needs, ensuring that their designs resonate with the client's values and financial capabilities.

Geographical and climatic conditions add another complexity to the architectural design process, which makes it more ambiguous. For example, in Bangladesh, an architect would have to consider certain environmental hazards, such as heavy rainfall, high humidity, and extreme temperature variations throughout the year. These factors dictate the choice of construction materials, structural designs, and spatial arrangements within residential plans. An architect should continue being aware of such local features to ensure that the design is workable and viable. Design therefore needs to balance aesthetic appeal with functional requirements to achieve both the client's desires and the needs of the environment.

Moreover, the changing nature of architectural trends and standards adds another layer of ambiguity to residential design. Generation wise it varies. As societal norms evolve, the expectations concerning the living environment also change. For instance, the increase in remote work has led to a higher demand nowadays. For that, the home office spaces are changing the priorities in residential design. Architects must remain adaptable, embracing new concepts and methodologies that align with contemporary lifestyles.

The architectural concept is inherently ambiguous by nature and shaped by diverse personal preferences, socio-economic factors, and environmental challenges. While efforts are made to maintain standards and present various design options to meet individual abilities and choices, it is the architect's job to determine the best solutions within the client's overall financial range. There are endless possibilities while designing a residential floor plan, but establishing fundamental principles and services can make decision-making easier. This ambiguity in architecture presents an

opportunity for continuous exploration and innovation, which allows architects to create spaces that truly reflect the complexity of human experience.

5.2.2 Challenges in Labeling Floor Plans with Local Annotations

As previously mentioned, the categorization of residential floor plans is High B, Low B, Type C, and Type D. It poses fundamental challenges, especially when considering the subjective aspects of architectural design shaped by socio-economic, cultural, and geographical influences. When architectural students participated in the annotation process, they encountered significant difficulties while filtering out the floor plans based on these classifications. The main challenge was that diverse interpretations arose from distinct individual viewpoints. Each classification embodies a unique understanding of optimal residential design, which is often shaped by the diverse needs and expectations of clients.

- **Subjectivity in Classification:** The classification system for floor plans necessitates a level of subjectivity, as individual evaluators may perceive the suitability of a design differently. Whereas one evaluator might classify a floor plan to be Low B concerning its functional attributes and principles of zoning, another evaluator might classify it as Type C on account of deficiencies perceived either in spatial relationships or room layout. This discrepancy in viewpoints may lead to plans being considered suitable for various classifications, thereby causing ambiguity regarding their applicability to prospective occupants.
- **Contextual Influences on Perceptions:** Floor plan classifications are deeply rooted in the context of local socio-economic conditions. For an example, what constitutes an ideal residence for a wealthy individual may vastly differ from the standards of a low-income family. A High B classification, considered suitable for a reasonably comfortable family lifestyle, might prove beyond the able means of economically deprived families, who would wish to consider Type C or Type D plans as more in line with their economic constraints. As a result, the same floor plan may be viewed as a viable residence for one group while considered inadequate by another.
- **The Role of Client Expectations:** Architects also have to balance their client's expectations and needs, which complicates the labeling process. Lifestyle, cultural background, and individual needs all make a difference in what clients perceive to be an adequate residential environment. What might be considered an inadequate design in one case could easily be labeled as ideal by another client, further adding to the pervasive ambiguity of categorization.
- **Implications for the Filtering Process:** The challenges inherent in labeling floor plans necessitate a flexible approach within the classification system. The architectural students who were part of the filtering process found it difficult to justify their classification from other perspectives. The iterative nature of the filtering process emphasizes that mutual discussions among the evaluators are required so that a consensus can be reached keeping in mind that architectural design is a variant of different interpretations and preferences.

The complexity of labeling residential floor plans with local annotations underscores the multifaceted nature of architectural design. Factors such as subjectivity, contextual elements, variability in application, and client anticipations all play a role in the difficulties encountered in forming conclusive classifications. This uncertainty illustrates the imaginative and dynamic quality of architecture, necessitating a sophisticated comprehension of both design principles and the varied social contexts within which these plans are situated.

5.2.3 Model Limitations

The task of Graph binary classification depends on the representational learning capability of GNN. GNN models use message-passing layers to aggregate and transform node features by sharing information between the neighbors. These node-level embeddings are condensed by pooling operation, making a graph-level representation. This process has many varieties and factors involved, which makes it complex. In this paper, three different types of models are used for graph binary classification. However, the core architectures in the models are fairly simple. The architectural floor plan information is converted into a graph because the connectivity information between rooms can be easily represented on the edges. However, the node features and edge features determine how well the floor plan information can be learned by the GNN models. In this paper, only three node features(area, centroid, room type) and no edge features are present in the graph. This is done due to the higher model complexity of handling edge features and node features, which are dynamic. For example, room coordinates with dynamic vertex count can hold spatial information better than a centroid, also edge features with door vertex coordinates can give the model crucial door location information. However this would increase model complexity, and implementing this requires more research. Moreover, different pooling and aggregate functions could be used which are good at handling spatial data (eg. diffpool). So, in this paper, the models are limited by not being handled to dynamic node features and edge features and not using other advanced pooling, and aggregation operations. This results in limited graph-level representation has room to improve.

Chapter 6

Applications and Future Directions

6.1 Enhance Dataset in the Context of Bangladesh

The attempt at adapting foreign architectural data for local applications poses its unique challenges, especially when faced with datasets that are not inherently designed for the target environment, Bangladesh. In our work, we sought to use a dataset of 51,000 floor plans from China to evaluate and filter them for our application in the context of Bangladesh, where the hot and humid climate requires certain attributes for residential design. Although our dataset is quite broad in terms and valuable for the architecture field. However, it lacks major factors like orientation and ventilation, these are the factors that are essential for assessing livability in Bangladesh. To bridge this gap, we have made several enhancements to the dataset while understanding certain limitations that influence model performance.

Our machine learning model plays a crucial role in the data processing pipeline, which converts initial images of floor plans into bubble diagrams showing rooms as independent nodes carrying critical information on the centroid position of each room, total area, and types of rooms-labeled, including master bedrooms, kitchens, and bathrooms, etc. Additionally, it displays the interrelationships among the rooms in the bubble diagrams and also allows us to evaluate the internal space ratio and spatial zoning. However, even if the model is effective in capturing floor plan data's structural elements, it does not account for ventilation and orientation, which are necessary for designing functional residences in Bangladesh's aspect.

6.1.1 Enhance Dataset with Orientation Considerations

Orientation is a crucial part of the residence design. In the context of Bangladesh, as it is a hot, humid climate country, orientation plays a huge role. Proper orientation can enhance the conditions of a building in terms of natural lighting, ventilation, thermal comfort, and airflow, which significantly helps us to reduce dependence on artificial cooling. Generally, in Bangladesh, residences with west-facing facades are avoided, because they are exposed to the intense afternoon sun, which can make the indoor environment uncomfortable to live in because of the unbearable temperature. Because of that, we typically preferred north and south-facing orientations, as they allow for better natural ventilation and smooth airflow. Still, always it is

not possible to prefer a north-south facing facade apartment because of the location and building's front directions.

In the context of our dataset, the lack of information about orientation is a major limitation. To address this issue in the future, it would be beneficial to enhance the data by considering the orientation of the dataset. It can be either by adding metadata to the floor plans or by developing a model capable of acknowledging the orientation based on external factors like balcony positions, or the location of main entrances. While window placement is an important factor too in residential design, the dataset we used in this study, does not contain any information about window orientation. However, enhancing the dataset by incorporating window orientation and training the model accordingly so that it could provide valuable insights and improve the accuracy of spatial assessments in the context of Bangladesh. This will enable our model to grade floor plans more accurately, especially those falling into the High B and Low B classifications. In particular, many of those currently graded as High B could receive a Type A designation if orientation is optimized. Similarly, some of the Low B plans will be further downgraded because of their west-facing facade orientations, which would increase the thermal load on the building and consequently affect its overall livability.

6.1.2 Incorporate Ventilation Factors

As we know, Bangladesh is a hot and humid country, proper ventilation is essential in residential design. Decent airflow can reduce indoor temperatures and can improve air quality. Which eventually makes the environment more livable without heavily relying on artificial cooling systems. However, the dataset used for the study does not contain any explicit information about windows. But windows are the key element to determine to what extent a floor plan can help with natural ventilation and airflow. Cross-ventilation is also necessary for residential plans. The lack of information limits and constrains the precise evaluation of the residential floor plan's livability.

The existing machine learning model is mainly centered on transforming floor plans into bubble diagrams, which capture fundamental spatial information like dimensions of rooms, their classifications, and their interconnections. Even though this approach is effective in evaluating zone relationships, but it fails to incorporate ventilation considerations, such as the positioning of windows, vital for analyzing airflow between different rooms. Knowing this limitation, students of architecture had to make subjective assessments on ventilation, based on room size, room configuration, and the presence of a balcony.

For the future directions, incorporating windows into the dataset would significantly enhance the accuracy of our classifications. This crucial attribute would be able to let the model consider key aspects like cross-ventilation and natural airflow, which become essential in the perspective of Bangladesh's climatic condition. Accommodating window data would enable better classification of the floor plans, and therefore, modifications could be suggested that would significantly enhance livability, particularly for High B plans that could potentially be upgraded near Type A. By addressing this gap in the dataset, we could make more informed and meaningful contributions to residential design practices in Bangladesh.

6.2 Application of GNNs in Spatial Context

Graph Neural Networks have emerged as powerful tools for machine learning with data represented in graph structures, thereby enabling the modeling of complex relationships and dependencies manifested in those graph topologies. In particular, GNNs are very helpful in spatial scenarios since they can model and explore spatial relationships that integrate node attributes with graph structures. That ability makes GNNs very successful in tasks related to urban planning, geospatial analysis, and architectural design-encompassing tasks where spatial relationship understanding is fundamental.

6.3 Future Improvements in GNN Architecture

In our current research, we annotated 9028 floor plans with 4 labels which were converted to 2 labels but we plan to increase the data set by increasing the amount of annotated data and incorporating multiclass classification.

In the present paper, the use of the NENN [18] model enables the effective incorporation of edge features, such as the data on door locations that were excluded in our earlier GraphSAGE model. While GraphSAGE is primarily focused on node features and the pooling of nearby neighborhood information, NENN offers a hierarchically deployed dual-level attention mechanism that enables the integration of node as well as edge features. This becomes specifically helpful in the present context since the relations through doors connecting different rooms are edge features that carry crucial spatial information capable of influencing the structure of the entire graph. The alternating layers in NENN allow the model to learn knowledge not only about the importance of neighboring nodes but also about that of the neighboring edges, hence improving the embeddings of both nodes representing rooms and edges signifying doors. Such dual-level attention makes the model fully exploit the rich information provided by the edge features and improve the classification accuracy of floor plans relevant to Bangladeshi architecture.

We can try to enhance the performance of our GraphSAGE model by considering more advanced pooling techniques such as DIFFPOOL [13], replacing the current global mean aggregator. DIFFPOOL instantiates a differentiable, end-to-end pooling model that allows the model to learn hierarchical representations by clustering the nodes and, hence, producing coarser graph representations in each layer. Incorporating DIFFPOOL will allow us to further coarsen the graph through multiple layers, enabling the model to capture both local node-level information—in this case, individual room features—and higher-level structural patterns that include clusters of connected rooms and their spatial relations. We consider this to be a very suitable hierarchical approach to graph classification tasks, where higher-level understanding of general architectural structure is crucial to make distinctions between floor plans. This replaces using global mean pooling with DIFFPOOL and would improve our ability to make the most of edge features, such as door locations, but also train effectively on the spatial associations among rooms. Such a method is likely to significantly boost the model's skill at classifying complex architectural floor plans, and in particular in activities targeted at recognizing patterns unique to Bangladeshi architecture.

Chapter 7

Conclusion

Graph Neural Networks (GNNs) have demonstrated significant promise for tasks involving classification on graphs, including the examination of architectural floor plans. In this research, we employed GCN, GraphSAGE, and GAT models to categorize floor plans according to their suitability for Bangladesh, finding that GCN and GraphSAGE surpassed GAT in terms of accuracy, precision, and generalization. However, the most important issues we encountered were because of vagueness in classifying architectural features. We tried to assure coherence by allowing a system of votes between different annotators to take over. As a second point, there are only three node featuresroom area, centroid, and room typeand no edge features which can be considered in our models, providing more spatial details. Despite these challenges, GCN and GraphSAGE outperformed most of the compared algorithms, proving to be suitable for this binary classification task. Future work may focus on incorporating dynamic node and edge features and advanced pooling techniques in order to improve model complexity and representational learning. Overall, this research contributes to the field by applying GNNs to architectural design evaluation and highlighting areas for further improvement.

Bibliography

[1] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009. DOI: 10.1109/TNN.2008.2005605.

[2] P. K. Diederik, "Adam: A method for stochastic optimization," *(No Title)*, 2014.

[3] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[4] H. Hua, "Irregular architectural layout synthesis with graphical inputs," *Automation in Construction*, vol. 72, pp. 388–396, 2016, ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2016.09.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092658051630231X.

[5] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[6] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, 2017. arXiv: 1609.02907 [cs.LG].

[7] X.-Y. Gong, H. Su, D. Xu, Z.-T. Zhang, F. Shen, and H.-B. Yang, "An overview of contour detection approaches," *International Journal of Automation and Computing*, vol. 15, no. 6, pp. 656–672, 2018, ISSN: 1751-8520. DOI: 10.1007/s11633-018-1117-z. [Online]. Available: https://doi.org/10.1007/s11633-018-1117-z.

[8] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph attention networks*, 2018. arXiv: 1710.10903 [stat.ML].

[9] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery data mining*, 2019, pp. 2623–2631.

[10] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," *arXiv preprint arXiv:1912.09893*, 2019.

[11] W. Wu, X.-M. Fu, R. Tang, Y. Wang, Y.-H. Qi, and L. Liu, "Data-driven interior plan generation for residential buildings," *ACM Trans. Graph.*, vol. 38, no. 6, Nov. 2019, ISSN: 0730-0301. DOI: 10.1145/3355089.3356556. [Online]. Available: https://doi.org/10.1145/3355089.3356556.

[12] W. Wu, X.-M. Fu, R. Tang, Y. Wang, Y.-H. Qi, and L. Liu, "Data-driven interior plan generation for residential buildings," *ACM Transactions on Graphics (SIGGRAPH Asia)*, vol. 38, no. 6, 2019.

[13] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, *Hierarchical graph representation learning with differentiable pooling*, 2019. arXiv: 1806.08804 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1806.08804.

[14] R. Hu, Z. Huang, Y. Tang, O. van Kaick, H. Zhang, and H. Huang, "Graph2plan: Learning floorplan generation from layout graphs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 1–11.

[15] N. Nauata, K.-H. Chang, C.-Y. Cheng, G. Mori, and Y. Furukawa, "House-gan: Relational generative adversarial networks for graph-constrained house layout generation," in *European Conference on Computer Vision (ECCV)*, Springer, 2020, pp. 162–177.

[16] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, *Temporal graph networks for deep learning on dynamic graphs*, 2020. arXiv: 2006.10637 [cs.LG].

[17] P. Wu, Z. Tang, S. Jiang, *et al.*, "Floorplangan: Vector residential floorplan adversarial generation," *Automation in Construction*, vol. 120, p. 103 366, 2020.

[18] Y. Yang and D. Li, "Nenn: Incorporate node and edge features in graph neural networks," in *Asian conference on machine learning*, PMLR, 2020, pp. 593–608.

[19] J. Zhou, G. Cui, S. Hu, *et al.*, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[20] M. Yamada, X. Wang, and T. Yamasaki, "Graph structure extraction from floor plan images and its application to similar property retrieval," in *2021 IEEE International Conference on Consumer Electronics (ICCE)*, 2021, pp. 1–5. DOI: 10.1109/ICCE50685.2021.9427580.

[21] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, *Benchmarking graph neural networks*, 2022. arXiv: 2003.00982 [cs.LG].

[22] M. Yoon, Y. Wu, J. Palowitch, B. Perozzi, and R. Salakhutdinov, "Graph generative model for benchmarking graph neural networks," *arXiv preprint arXiv:2207.04396*, 2022.

[23] S. Leng, Y. Zhou, M. H. Dupty, W. S. Lee, S. Joyce, and W. Lu, "Tell2Design: A dataset for language-guided floor plan generation," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 14 680–14 697. DOI: 10.18653/v1/2023.acl-long.820. [Online]. Available: https://aclanthology.org/2023.acl-long.820.

[24] A. Mao, M. Mohri, and Y. Zhong, "Cross-entropy loss functions: Theoretical analysis and applications," in *International conference on Machine learning*, PMLR, 2023, pp. 23 803–23 828.

[25] H. Park, H. Suh, J. Kim, and S. Choo, "Floor plan recommendation system using graph neural network with spatial relationship dataset," *Journal of Building Engineering*, vol. 71, p. 106 378, 2023, ISSN: 2352-7102. DOI: https://doi.org/10.1016/j.jobe.2023.106378. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352710223005570.

Appendix
Code Snippets for Data Split and Model

Graph Convolutional Network

The code use for defining GCN Model:

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import GCNConv, global_mean_pool

class GNNGraphClassifier(nn.Module):
    def __init__(self, num_node_features, hidden_dim, num_classes):
        super(GNNGraphClassifier, self).__init__()
        self.conv1 = GCNConv(num_node_features, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, hidden_dim)
        self.fc1 = nn.Linear(hidden_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, num_classes)

    def forward(self, x, edge_index, batch):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        x = F.relu(x)

        x = global_mean_pool(x, batch)

        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        return x
```

Figure 7.1: Code Defining GCN Model

GraphSAGE

The code use for defining GraphSAGE Model:

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import SAGEConv, global_mean_pool

class GNNGraphClassifier(nn.Module):
    def __init__(self, num_node_features, hidden_dim, num_classes, num_layers=2, dropout_rate=0.0):
        super(GNNGraphClassifier, self).__init__()
        self.num_layers = num_layers
        self.convs = nn.ModuleList()
        self.convs.append(SAGEConv(num_node_features, hidden_dim))
        for _ in range(num_layers - 1):
            self.convs.append(SAGEConv(hidden_dim, hidden_dim))
        self.fc1 = nn.Linear(hidden_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, num_classes)
        self.dropout = nn.Dropout(dropout_rate)

    def forward(self, x, edge_index, batch):
        for conv in self.convs:
            x = conv(x, edge_index)
            x = F.relu(x)
            x = self.dropout(x)

        x = global_mean_pool(x, batch)

        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

Figure 7.2: Code Defining GraphSAGE Model

Graph Attention Network

The code use for defining GAT Model:

```python
from torch_geometric.nn import GATConv, global_max_pool, global_mean_pool

class GATNet(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim,
                 num_heads, dropout_rate, num_layers):
        super(GATNet, self).__init__()
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.output_dim = output_dim
        self.num_layers = num_layers
        self.num_heads = num_heads

        self.convs = torch.nn.ModuleList()
        self.lins = torch.nn.ModuleList()

        for i in range(num_layers):
            conv = GATConv(
                input_dim if i == 0 else hidden_dim * num_heads,
                hidden_dim,
                heads=num_heads,
                concat=True,
                dropout=dropout_rate
            )
            lin = torch.nn.Linear(hidden_dim * num_heads, hidden_dim * num_heads)
            self.convs.append(conv)
            self.lins.append(lin)

        self.fc1 = torch.nn.Linear(hidden_dim * num_heads * 2 * num_layers, hidden_dim)
        self.fc2 = torch.nn.Linear(hidden_dim, output_dim)

        self.dropout = torch.nn.Dropout(dropout_rate)

    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index, data.batch

        pooled_outputs = []
        for conv, lin in zip(self.convs, self.lins):
            x = conv(x, edge_index)
            x = F.elu(x)
            x = lin(x)
            x = self.dropout(x)
            x1 = torch.cat([global_max_pool(x, batch), global_mean_pool(x, batch)], dim=1)
            pooled_outputs.append(x1)

        x = torch.cat(pooled_outputs, dim=1)

        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout(x)
       x = self.fc2(x)

        return F.log_softmax(x, dim=-1)
```

Figure 7.3: Code Defining GAT Model