

ArchitectureGAN

Amit Divekar¹, Shrey Srivastava¹, Ved Kulkarni¹

¹ Department of Computer Science, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai Campus, Chennai, India

Email: amitvishvas.divekar2019@vitstudent.ac.in, shrey.srivastava2019a@vitstudent.ac.in,
ved.kulkarni2019@vitstudent.ac.in

Mob: +91-92841-99286, +91-98811-50018, +91-95615-40271

Abstract – This paper proposes a model to automatically generate floor models, given only its boundary. The model takes a two-phase approach, first determining the locations of various rooms and then drawing the wall (keeping the outer boundary in mind). The living room is the central piece of a house. Most rooms are adjacent to it and it is often centrally located. Thus, the model first positions the living room, and the other rooms are positioned adjacent to it after that. The use of GANs allows us to imitate human designers and produce, what is in all sense, a possible-in-real-world house layout. RPLAN is a dataset consisting of 80K floor plans of real houses. 10K layouts from this dataset were used to train our model. After training, our dataset was able to generate realistic floor plans in just about 4 seconds. This is way less time than what a human would need to draw a similar plan.

Keywords – GANs, House-GAN, Floor Plan

1 INTRODUCTION

A floor plan is a fundamental element of a building’s architecture. A floor plan is basically a blueprint of walls, which in turn define the rooms. So, a floor plan defines the size, shape and location of all rooms along with doorways and other miscellaneous features. Currently, interior designing is practiced exclusively by humans. However, like a lot of other professions, there is a shortage of architects and interior designers, almost everywhere in the world. Also, hiring an architect is expensive. Also, the process of coming up with a good plan is a time-consuming process and not perfect. It involves a lot of trial and error and guesswork.

Thus, there is a need to automate the process of generating floor plans to help make it more accessible and possibly, better. Similar processes are already used in popular games. Minecraft generates huge worlds from a seed value. Endless runner games like Subway Surfer and Temple Run also generate a part of their world.

Some techniques have been proposed to generate floor plans. [2,3,4] However, these need inputs for room sizes, positions and adjacencies. This defeats the whole purpose of automatic plan generation. Normal people don’t know all these factors, and so only trained professionals with adequate knowledge of interior design can use such programs.

Thus, we aimed to develop a learning-based process that will need nothing other than the outline of the floor. Everything else, the number of rooms, their placement in the space as well as their size and shape will be determined by the program. To do this, we trained our model on a dataset of real floor

plans. We have used a part of the RPLAN dataset that was developed by Wenming Wu et al. [1] The dataset 80K plans. Due to time constraint, and more importantly, the constraint of processing power, we only 10K plans from it.

The main objective of this model is to learn from the dataset of plans, designed by a professional and thus, generate such plans that they will be indistinguishable from that designed by a professional. Hence, a Generative Adversarial Network is the perfect machine learning framework for this task. The generator will try to generate plans that pass the discriminator's evaluation. Eventually, the generator will be able to produce plans comparable to that drawn by a trained interior designer in a fraction of the time that they would take.

We design a two-phase approach to learn the floor plan based on the observation that professionals design floor plans in two stages: [12]

- Determining room connections and positions.
- Computing room sizes and wall positions.

To predict the room positions, an iterative learning method [13,14] is used with one unique modification - a living room first approach to improve the plausibility of our generated floor plans. Then, given the calculated positions of the rooms, an encoder-decoder network is trained to predict the wall positions. The result is an image of the floor plan with pixelated walls. With a vectorisation program that uses customized rules, we transform it into a vector representation.

After training, our program is able to generate floor plans comparable to what a human would draw, in just around 4 seconds which is way less than a human would take.

2 LITERATURE SURVEY

2.1 BUILDING-GAN: GRAPH-CONDITIONED ARCHITECTURAL VOLUMETRIC DESIGN GENERATION, CHANG KAI-HUNG ET AL. [6]

In the research work by Chang kai-hung et al, they have used Graph Conditioned Architectural Design for building GAN. For any professional building design, a primary and critical step is its volumetric design. In this paper they have used an input program graph to generate a volumetric design that has a 3D representation. It is referred to as a 3D voxel graph. The designs generated by this novel way are evaluated on the basis of three criteria. These are: quality, diversity and connectivity accuracy. This was assessed by architects who were made to generate an actual building design based on the volumetric design developed in this present study.

2.1.1 Advantages of this Work

The authors have put forth several advantages of this method. These are as follows:

- The work provides a novel pipeline termed as Building-GAN to improve efficacy on the professional front.
- The designing method used is very useful in the architectural field and the construction industry.
- The design provides a 3D view of the building based on the input graph which is easier for interpretation as compared to the traditional methods that used 2D approach.
- By interacting with the building-GAN the architects and designers can produce several valuable designs.
- The building designs created by this method are aesthetically appealing and realistic.

2.1.2 Limitations of this Work

The designs generated using this model of building-GAN showed some flaws such as missing nodes, missing edges and disconnected rooms. This may be because the discriminator lacks the information about the specific program nodes which the voxel nodes points to.

2.1.3 How we plan to tackle these limitations

Instead of taking the entire layout as an entity and establishing boundaries as the primary task, we apply a two-stage approach and departmentalise the work to achieve better accuracy and have more defined details. This is one of the main issues that is tackled elegantly in our implementation of house GAN.

2.2 APARTMENT FLOOR PLANS, HAO ZHANG ET AL. [7]

In this study, Hao Zheng et al. in the year 2020, make the use of GAN in architectural design. While drawing architectural plans it is very necessary for the designers to define minute details. Hence such images always have information to support the designs. The researchers have used GAN to learn and generate image data in creating apartment floor plans when boundaries were given. Through the machine learning of images that show details of the plan drawings the learning program can build a model to learn the connections between two given images and then the evaluation program can generate architectural drawings according to the boundaries entered by the designers. They have developed a Chinese and a Japanese model and compared both.

2.2.1 Advantages of this Work

- The design obtained by using the Japanese model shows the accurate positioning of every room inside the apartment.
- The Chinese model gives a precise prediction in layout design and drawing pattern.
- Only based on the boundary designs the GAN could produce accurate designs of the rooms.

2.2.2 Limitations of this Work

The Japanese model could not provide the details of internal furnishings.

2.2.3 How we plan to tackle these limitations

We have incorporated a visualization model. It is based on the framework of Visual Studio 2013. It does not have the decor but the floors are furnished enough to make the difference in the rooms apparent.

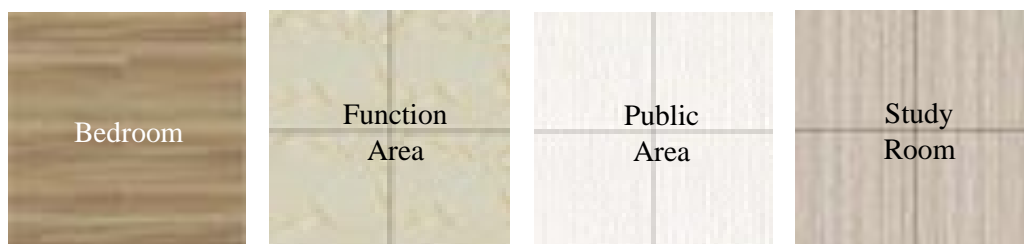


Figure 1

2.3 COLOUR-PATTERNS TO ARCHITECTURE CONVERSION THROUGH CONDITIONAL GENERATIVE ADVERSARIAL NETWORKS, DIEGO NAVARRO MATEU ET AL. [8]

This study is based on testing the feasibility of using conditional Generative Adversarial Networks (cGANs) as a tool for coding architecture into colour pattern-based images and translating them into 2D architectural representations. A series of scaled tests are performed to check the feasibility of the

hypothesis. A second test assesses the flexibility of the trained neural networks against cases outside the database.

2.3.1 Advantages of this Work

- It helps in simplifying and compressing reality into easy and efficient information that can be further used as a powerful tool to generate complex architectural design.
- The technique of cGAN utilizes omni light and occlusion shadows in getting the final output, thereby making the representation a pseudo 3D one.

2.3.2 Limitations of this Work

Since the technique is based on colour- patterns, there are chances of changes in opposition to do nothing as each colour will have a different frequency.

2.3.3 How we plan to tackle these limitations

We have eradicated the problem by keeping the visualization separate from the model itself. After the generation of walls and location of rooms, we run the visualization module on Visual Studio. This keeps the colour part away from the GAN and the layouts generated are with black background and green walls. Then when the visualization module is run, we get the floors and the walls as they were tagged by the GAN algorithm.

2.4 HOUSE-GAN: RELATIONAL GENERATIVE ADVERSARIAL NETWORKS FOR GRAPH-CONSTRAINED HOUSE LAYOUT GENERATION, NELSON NAUATA ET AL. [9]

In this research work by Nelson N et al, they have used a graph constrained generative adversarial network, whose generator and discriminator are built upon relational architecture. The paper has proposed architecture for a new house layout generation problem, whose task is to take the number and types of rooms with their spatial adjacency as an architectural constraint and produce a set of axis-aligned bounding boxes of rooms. We measure the quality of generated house layouts with the three metrics: the realism, the diversity, and the compatibility with the input graph constraint. They have evaluated the model both quantitatively and qualitatively.

2.4.1 Advantages of this Work

- This particular method has performed well in compatibility matrices.
- The results obtained are realistic.
- The system has the best diversity.

2.4.2 Limitations of the Work

Improper room size or shapes for a given room type, misalignment of rooms and inaccessible rooms.

2.4.3 How we plan to tackle these limitations

We have conditionals for this. We have set certain standards for the model and that helps us side-line the inputs that might cause issues like they did in the paper by Nelson. These conditions are:

- The total area of the floor plan is larger than 60 square meters and less than 120 square meters.
- The number of rooms in the floor plan is larger than 3 and less than 9.
- The floor plan has a living room.
- The proportion of the area of the living room to the total area of the floor plan is larger than 0.25 and less than 0.55.
- The average area of each room is larger than 10 square meters and less than 20 square meters.

2.5 END-TO-END GENERATIVE FLOOR-PLAN AND LAYOUT WITH ATTRIBUTES AND RELATION GRAPH, XIN HAN D ET AL. [10]

In this paper, the authors have proposed an end-to-end model for producing furniture layout for interior scene synthesis from a random vector. This proposed model is aimed to support professional interior designers to produce interior decoration solutions more quickly. The proposed model combines a conditional floor-plan module of the room, a conditional graphical floor-plan module of the room, and a conditional layout module. They have conducted the testing of their model on the real-world interiors' dataset, evaluated their work with the existing methods.

2.5.1 Advantages of this Work

The current model outperforms all the existing state of art interiors dataset.

2.5.2 Limitations of this Work

The generation of vectors is based only on a random variable.

2.5.3 How we plan to tackle these limitations

Instead of using a random variable for vectorization we use Jointly Gaussian Random Variables that are normalised and have been fine-tuned by latest works in the field of Regression Theory.

2.6 ROOF-GAN: LEARNING TO GENERATE ROOF GEOMETRY AND RELATIONS FOR RESIDENTIAL HOUSES, YIMING QIAN ET AL. [11]

In this work the authors have proposed a novel method Roof-GAN, that generates structured geometry of residential roof structures as a set of roof primitives and their relationships. Here, the generator produces a structured roof model as a graph, which consists of primitive geometry as raster images at each node, encoding facet segmentation and angles; inter-primitive collinear/coplanar relationships at each edge; and primitive geometry in a vector format at each node, generated by a novel differentiable vectorizer while enforcing the relationships. The discriminator is trained to assess the end-to-end primitive geometry. Qualitative and quantitative evaluations are done.

2.6.1 Advantages of this Work

- It maintains the same topology as the ground truth in comparison to the house-GAN.
- There is consistency in the visually similar roof structures.
- It has a remarkable compositional capability.

2.6.2 Limitations of this Work

- Theoretically, this model of GAN is as sound as they come. But lacks in operation.
- Lack of deep testing and modular testing of the models.

2.6.3 How we plan to tackle these limitations

With the code we have a testing module which is called upon the first iteration before the testing begins and runs more than 50 tests with all kinds of combinations possible. It gives a proper perspective into the integration of the model that is lacking in Roof-GAN. As the capabilities of a model are limited by the software and hardware at hand, these tests help to understand whether the model is getting trained evenly as we do not need either the discriminator or the generator to dominate over the other. These tests keep everything in perspective and give us an idea if we need to tweak the models as GAN is very susceptible to changes on either of the models mentioned above.

3 PROPOSED WORK

3.1 GENERATIVE ADVERSARIAL NETWORKS

GAN is a machine learning framework that consists of not one, but two neural networks. [15] One is called a generator, and the other, a discriminator. The generator is just told what it needs to generate, but not the ground truth (actual data), even for training. It has to solely depend on the loss score generated by the discriminator. The job of the discriminator is to compare the actual data and the generated data, and tell if they are the same or not. It gives the loss depending on how dissimilar it thinks the data is. Initially, the generator will generate very poor data, and the discriminator will easily be able to tell that its not the real data. However, with more epochs, it will generate better data. Eventually, it will be able to fool the discriminator. This will force the discriminator to train more (increase learning rate), till it is again able to distinguish between real and generated data. This cycle will continue. Ideally, with a perfect model and a lot of training, the generator will get so good that its generated data is completely indistinguishable from actual data. This, is of course, not yet possible in real-world scenarios. Still, we are able to obtain really good generators that work really well for practical scenarios.

3.2 RPLAN DATASET

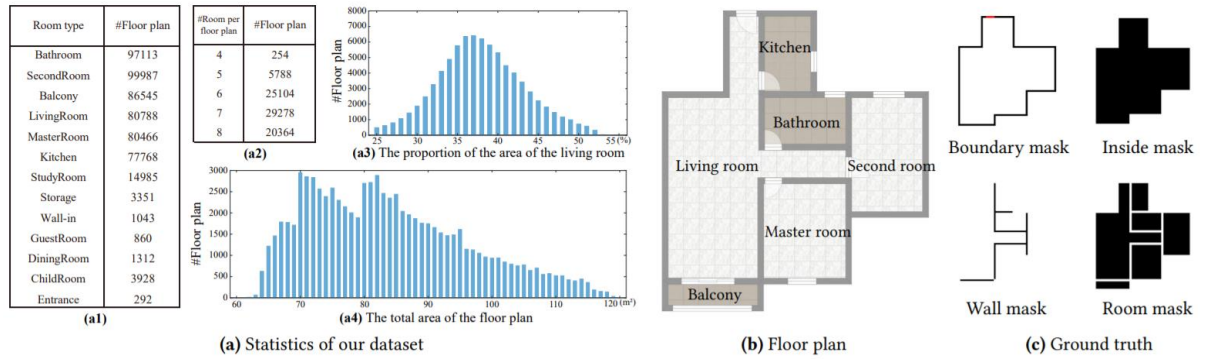


Figure 2

3.2.1 Data collection

The data has been collected more than 120K floor plans from real-world residential buildings in the estate market in Asia. All floor plans in the dataset have no copyright issue, this makes it easy to freely use the dataset. Each floor plan has a vector-graphics representation within a squared region of $18\text{m} \times 18\text{m}$, including the geometric and semantic information as shown in Fig. 2 (b). For the sake of applying learning scheme, we convert each floor plan into a 256×256 image.

3.2.2 Filtering

Real-world residential buildings often have some small areas for the flue, elevator and equipment platform. These are not our target since these areas are too small and may be randomly set. To avoid the interference of these factors and enhance the reliability of the dataset, some non-standard data has been filtered out for training and testing. The dataset, after filtering, has 13 kinds of rooms. Further, only the floor plans that satisfy all the following requirements are kept:

- The total area of the floor plan is larger than 60 m^2 and less than 120 m^2 .
- The number of rooms in the floor plan is larger than 3 and less than 9.
- The floor plan has a living room.

3.3 TWO PHASE APPROACH

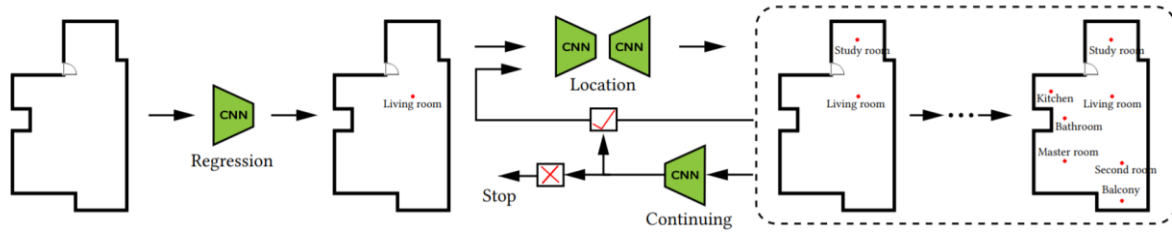


Figure 3

3.3.1 Locating Rooms

3.3.1.1 Living room First Strategy

Key observations: The living room is the centre of the modern home and thus, has are two key attributes, it is usually located in the middle of the floor plan and is adjacent to most other rooms. Based on these observations, we develop a living room first strategy, which predicts the location of the living room first (Fig. 3). Once its position is determined, the connections can be decided by detecting the adjacencies between it and other rooms. A separated prediction model for the living room helps to improve the predictive accuracy and the overall rationality of the floor plan, as shown in Fig. 4.

Our iterative strategy: We propose the following iterative strategy (Fig. 3):

- Computing the location of the living room.
- Deciding what type of room to add and where.
- Deciding whether to add another room. If yes, go to step 2; otherwise stop the algorithm.

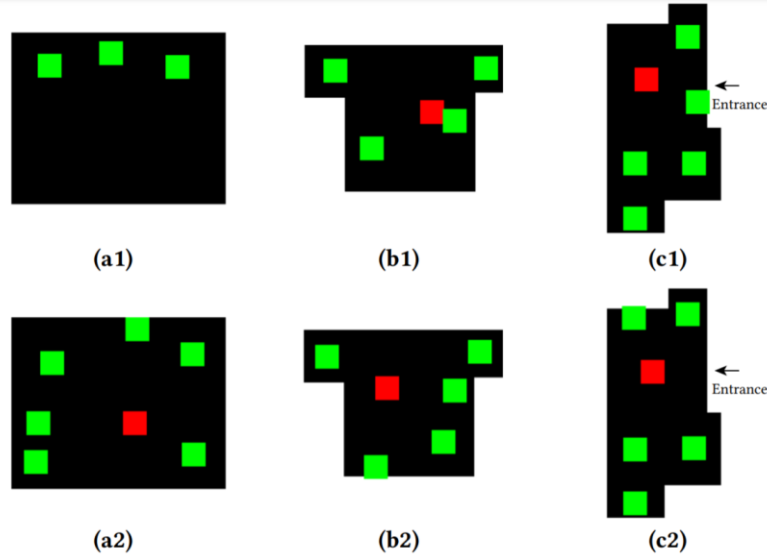


Figure 4

3.3.1.2 Living room regression

We determine the location of the living room through a regression network.

Training dataset: We build a training dataset for the regression network with a multi-channel image as the input and the location of the living room as the regression target. Since the shape of the living room is a polygon, we represent the location of the living room as the centroid of the polygon. The multi-channel input includes the following information at each pixel, which defaults to 0:

- Inside mask: taking a value of 1 for the interior.
- Boundary mask: taking a value of 1 for the exterior walls and 0.5 for the front door.
- Entrance mask: taking a value of 1 for the front door.

Network architecture: We use a modified Resnet-34 [16] to extract the spatial features from the multi-channel input. The Resnet-34 architecture is modified to use 256×256 multi-channel images as inputs. We drop the last average pooling layer and fully connected (FC) layer, and append two convolution layers. We then use batch normalization (BN) and leaky rectified linear unit (leaky ReLU) between two convolution layers. We add an average pooling layer at the end of the network to obtain two-dimensional coordinates. We train the regression network using the robust smooth L1 loss [17], which is less sensitive than the L2 loss.

3.3.1.3 Room type and location

Given the previously generated room types and locations, we use an encoder-decoder network to determine a new room type and location.

Training dataset: To make the dataset amenable to train our prediction network, we simplify the representation of each room in the floor plan (Fig. 5). We use a small square (18×18) centred on the centroid of a room to represent this room. We then build a training dataset for the prediction network by randomly removing rooms from the floor plan of the dataset. Since our prediction network is based on the living room regression, we ensure that all samples in the training dataset contain a living room. The input for our location network is also a multi-channel image. Except for the channels in living room regression, we add additional room masks (i.e., the 18×18 squares at the room centroids) for each individual existing room in the floor plan. Finally, we add another channel that contains all existing room masks to allow the prediction network to have a global grasp of the overall existing room distribution. The target of our prediction network is a labelled image that is the same size as the input image. Aside from room types, we add three more labels: EXISTING (i.e., a pixel belonging to existing rooms), NOTHING (i.e., a pixel belonging to the interior but not occupied by any rooms), and OUTSIDE (i.e., a pixel belonging to the exterior).

Network architecture: For the network architecture, we borrow the atrous spatial pyramid pooling (ASPP) from [18]. We modify the ASPP architecture based on Resnet-34, and drop their last upsampling layer. Instead, we append an atrous convolution layer and a deconvolution layer at the end of the network. The deconvolution layer is used to upsample to the target size. We find this modification improves the accuracy. Since our location network performs a pixel-classification task, we use averaged pixel-wise cross entropy loss.

Sampling: Our prediction map is generated with noise, which is very common for large generative models. To reduce the impact of the noise, we adopt a more direct sampling method to obtain the type and location of the new room at the same time, as shown in Fig. 6. Our sampling process contains the following steps:

- For each pixel p with a room label in the prediction map, we compute the number of pixels (denoted as N_p), which have the same label as p and are in the 18×18 neighbourhood of p .
- Choose the location of the pixel that has the greatest N_p as the centre of the newly added room.
- The type of the new room is the same as the label of the chosen pixel and the new room is represented as the corresponding 18×18 square.

If more than one candidate pixel has the greatest N_p , we pick one randomly when sampling.

Discussions: We use an 18×18 square to improve the stability of the learning process. Since most pixels are with non-room labels, learning an 18×18 square is much easier than learning a single point with X-Y coordinates. It also weakens the category imbalances. Since we filter out the floor plans in

the training dataset with very small rooms, the 18×18 square will not because those squares of different rooms overlap with each other.

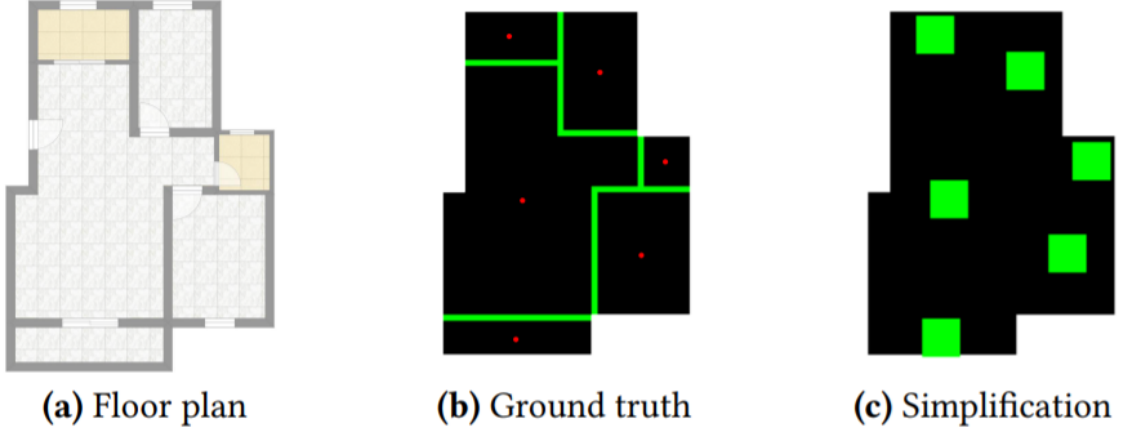


Figure 5

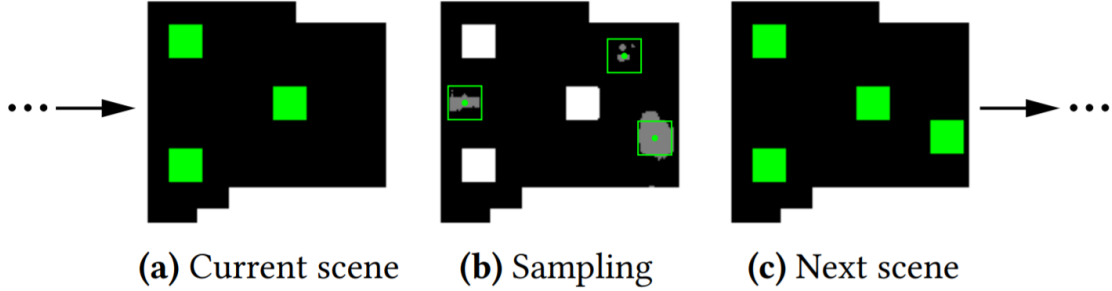


Figure 6

3.3.1.4 Continuing Network

Given a set of generated room types and locations, we develop a continuing network to determine whether to terminate the algorithm.

To build the training dataset for the continuing network, we first adopt representation simplification for each floor plan in our dataset. Then we randomly choose half of the floor plans in our dataset as negative samples with label TRUE (i.e., continue) and the rest as positive samples with the label FALSE (i.e., terminate). For each room in one negative sample, we discard it with a 50% chance. The input for the continuing network is the same as the location network, except that we add a count vector, whose dimension is the number of room types, for the existing rooms to the input. We adopt a similar network architecture to [14], but we modify it with Resnet-34. For the last three fully-connected layers, we only use leaky ReLU activation between these layers. The continuing network performs a binary classification task, so we use the standard binary cross entropy loss.

3.3.2 Locating Walls

Prediction-based locating strategy. The next step is to locate walls to allocate space for each room.

Previous constraint-based methods can be directly applied by formulating room locations into constraints. However, to generate a plausible floor plan, additional constraints, such as geometric constraints and topology constraints, should be provided. On the one hand, these inputs complicate the design process for users, since constraint design requires increased consideration. On the other hand, a system that includes too many constraints may have no solutions due to the contradictions between constraints. Therefore, we propose a prediction-based locating strategy, as shown in Fig. 7.

Specifically, we first use an encoder-decoder network to predict walls in discrete pixels given the

boundary and room locations. Then, a post-processing step is applied to convert the predicted walls into a vector representation.

3.3.2.1 Encoder-Decoder prediction

After locating rooms, we should now have a series of room types and locations. The next step is to build walls based on this information. Given the boundary and predicted rooms, we use an encoder-decoder network to predict the locations of walls.

To build a training dataset for our wall locating network, as we did before, we first adopt a representation simplification for each floor plan in our dataset. The input for the network is the same as the room locating network. Our training target is a labelled image that is the same size as the input image. We have three kinds of labels for each pixel: WALL (i.e., a pixel belonging to walls), NOTHING (i.e., a pixel belonging to the interior but not part of any walls) and OUTSIDE (i.e., a pixel belonging to the exterior). We treat room doors as parts of walls, though we can predict locations for room doors at the same time. We will pursue this further in future work. We use the same network architecture as in the room locating network due to the similar pixel-classification tasks. We then use averaged pixel-wise cross entropy loss to train our network.

3.3.2.2 Vectorization

Our network generates a wall map with discrete pixels representing walls. Although we obtain the approximate outlines of walls from the wall map, there are still a few issues remaining. We use a post-processing step to convert the wall map into a vector representation. We implement the vectorization using four steps, as shown in Fig. 8.

- Decompose and fit the noise predicted walls into rectangular parts. For a noise wall map (Fig. 8 (a)), we first perform the morphological closing operation. Then, the walls are decomposed into vertical and horizontal wall blocks, which are represented as their bounding boxes. Finally, these wall blocks are transformed with a fixed wall width, as shown in Fig. 8 (b).
- Connect and align the wall blocks to recover the complete walls (Fig. 8 (c)). Separated blocks are connected by computing the intersection of horizontal and vertical wall blocks. For further optimization, we adjust the wall blocks locally: (1) close wall blocks within a certain threshold are merged together; and (2) wall blocks are moved to align with other wall blocks or the exterior walls.
- Obtain the label for each pixel based on the predicted rooms and recovered walls. To derive room geometry from the normalized wall map, the predicted room locations are used to generate pixel-wise semantics according to the connection between pixels. We also compute the semantics of exterior walls, entrance, and interior walls. In Fig. 8 (d), we show the semantics.
- Set doors and windows. The connectivity is determined based on the key observation that most doors are connected to the living room. We use this prior information to add passageways between the living room and other rooms. In addition, a passageway between any other connected rooms is added. Two empirical rules are proposed to place the passageways:
 - Open-walls are placed in public rooms (i.e., kitchen and balcony); otherwise, we place ordinary doors.
 - We place the door in the wall that minimizes the distance from the door to the front door.

The windows are placed based on two empirical rules:

- We set the French windows for the living room and small windows for the bathroom in the consideration of privacy; otherwise, we set the ordinary windows.

- Except for the bathroom, windows are laid out along the longest exterior wall segments within each room. We set at most one window at the centre of the wall segments. These heuristics are simple but available. Fig. 8 (d) shows the final result of the floor plan. A learning-based method is expected to improve this process.

These heuristics are simple but available. Fig. 8 (d) shows the final result of the floor plan. A learning-based method is expected to improve this process.

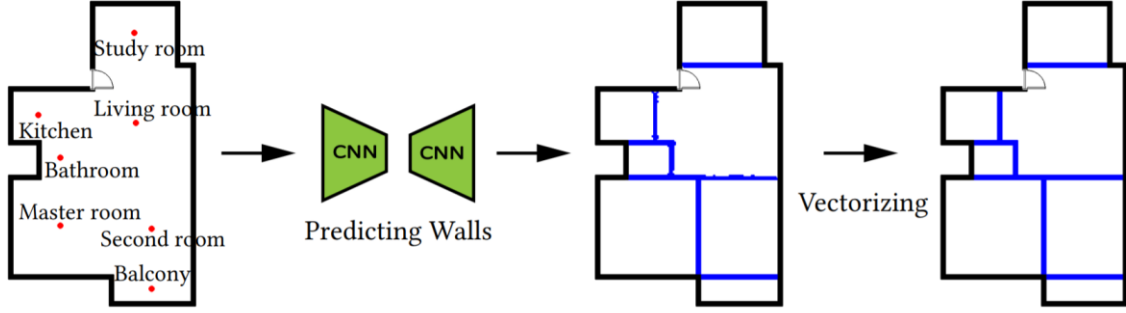


Figure 7

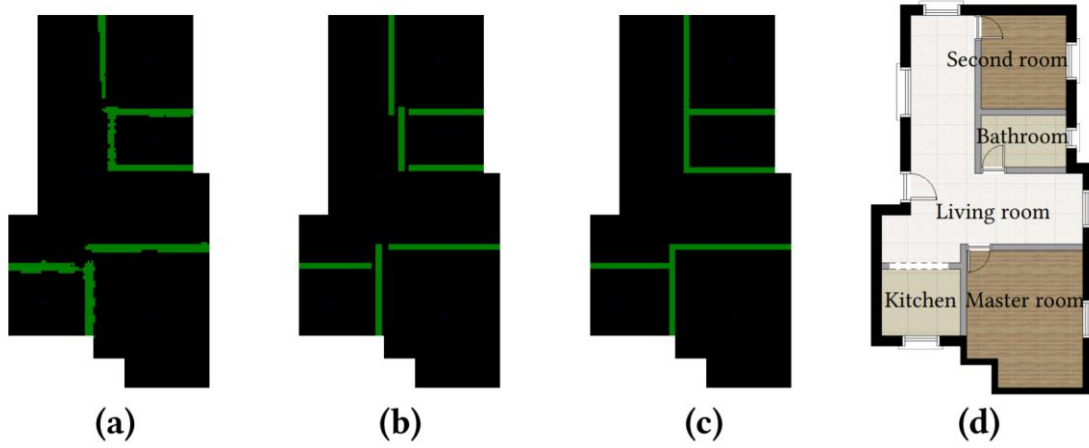


Figure 8

4 ANALYSIS

4.1 COMPETITORS

We select state-of-the-art methods as the competitors. The networks for indoor scene synthesis [14] can be used to locate rooms. We denote the network of [14] as ISSNet. Given the room locations, the MIQP-based method [19] can be used to determine the wall positions. Then, the method first uses ISSNet to locate rooms and then uses MIQP to locate walls is the first competitor (denoted as ISSNet+MIQP). Replacing the ISSNet of ISSNet+MIQP with our first stage approach for locating rooms is the second competitor (denoted as Stage1+MIQP). We substitute the MIQP of ISSNet+MIQP with our second stage approach for locating walls to define the third competitor (denoted as ISSNet+Stage2). The human-created is the fourth competitor (denoted as Human). We provide more details for ISSNet and MIQP in the supplementary material.

4.2 USER STUDIES

Given a pair of floor plans that share the same boundary, the forced-choice comparison task is designed, similar to [14]. In each task, each participant should choose the floor plan that they think is more plausible. To be fair, we randomly choose examples used for user studies from our generated results.

For each pair, the order of floor plans is randomized. We provide a questionnaire used for comparison to human-created floor plans in the supplementary material. We use the same user study design for four competitors. Each user study includes 30 forced-choice comparison tasks. In one out of each of the 15 tasks, we perform a “vigilance test”, in which an obviously wrong answer (specially, one floor plan with a randomized, jumbled arrangement of random rooms) is displayed. For each user study, the number of participants enrolled is 86, 81, 85 and 99, respectively. The participants are classified into general users and designers who practice interior design as a profession. If one participant does not achieve 100% accuracy on the vigilance tests, we discard this response. The statistics of the participants in each comparison study are shown in Table 1. For each participant, we record the number (denoted as N) of floor plans that are generated by our method and preferred by that participant.

4.3 RESULTS

The average and standard deviation for all the general users or designers in one user study is denoted as N_{avg} and N_{std} , respectively. The histograms in Fig. 10 show the distributions of N . Note that in Table 1 and Fig. 10, we only record the data from participants who pass the vigilance tests in each user study. A score of around 14 (28 non-vigilance choices in total) indicates that the two methods are comparable. From the distributions in Fig. 10, our method is comparable to the competitor Human and outperforms the other three competitors. Some participants failed the vigilance tests. We found about 7% of participants spent less than 1 minute and 21% of participants spent less than 2 minutes finishing comparison tasks. So quite a few participants are perfunctory and dropped by the vigilance test. We also observe that the average score of general users is less than designers. For example, in the results of comparisons of Human, N_{avg} is 13.39 for general users and 11.70 for designers. Designers usually focus on the details (e.g., orientations and relative sizes between rooms) of the floor plans, while the general users often judge the plausibility with personal preference. Our method may not learn those details very well. So, designers favoured the human designed layouts more than general users.

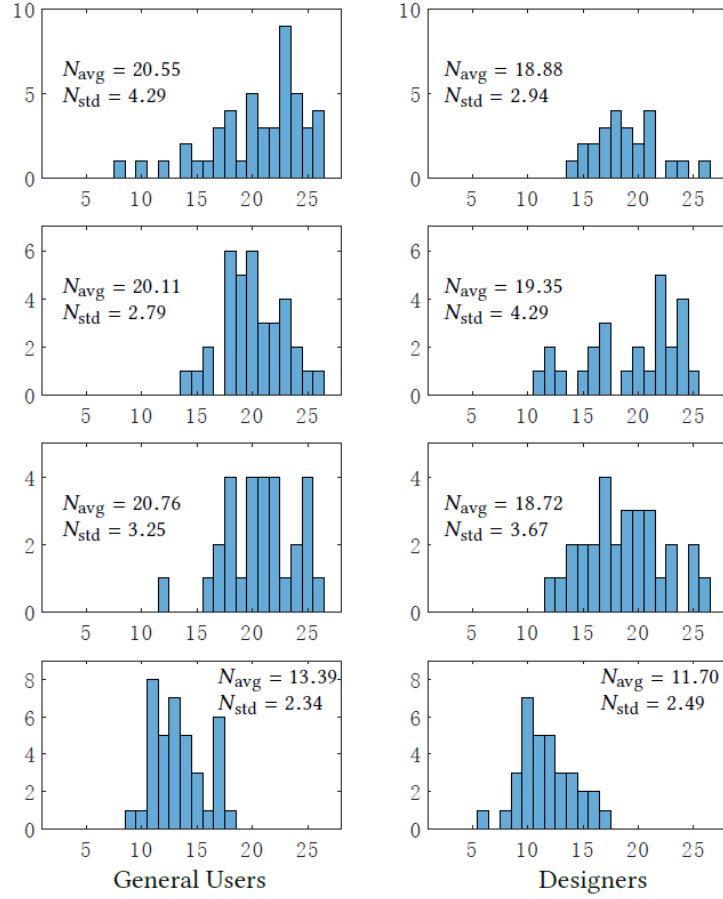


Figure 9

4.4 COMPARISON WITH OTHER MODELS

4.4.1 Comparison to ISS+MIQP



Figure 10

ISSNet computes category probabilities on a single pixel, which operates like an image-to-pixel function, and has problems with global consistency and noise suppression when sampling. MIQP is a

hierarchical optimization with geometric constraints and topology constraints. To this end, room locations predicted by ISSNet are formulated into position constraints for MIQP. We do not use any connection constraints between two rooms since the connections are unclear. Moreover, too many constraints for MIQP may result in contradictions and lead to no solutions. In the user study, average 20.55 floor plans of our method are preferred by general users and average 18.88 by designers, which indicates that participants preferred floor plans generated by our method to those generated by ISSNet+MIQP. Fig. 10 shows some representative results generated by ISSNet+MIQP and our method. ISSNet+MIQP generates floor plans with some necessary rooms missing which leads to sparse space allocation (columns (a) and (b)), and some rooms with unreasonable geometric sizes and shapes (columns (c) and (d)). In contrast, our method performs better in terms of global consistency and achieves better plausibility.

4.4.2 Comparison to Stage1+MIQP

Stage1+MIQP uses our first stage approach to locate rooms and then uses MIQP to generate walls. Similar to ISSNet+MIQP, the room locations predicted by our network serve as the location constraints for MIQP. We also do not add any connection constraints between two rooms to avoid the issue of constraint contradiction. In the user study, average 20.11 floor plans of our method are preferred by general users and average 19.35 by designers, which indicates that participants preferred results generated by our method over those generated by Stage1+MIQP. The room types and locations predicted by our network become the initializations for MIQP.



Figure 11

Although our network performs well at predicting room types and locations, MIQP fails in many examples with a few issues. The first issue is accessibility. In Fig. 11, columns (a) and (b) show that the master rooms generated by Stage1+MIQP are blocked by other rooms and cannot be entered. In column (c), the entrance is incorrectly connected to the bathroom. Geometric dimensions are another problem. In columns (d) and (e), MIQP generates some rooms with abnormal sizes, which are not suitable for residential buildings.

4.4.3 Comparison to ISSNet+Stage2

While ISSNet serves as an image-to-pixel function, our room locating network can be treated as an image-to-image function, which predicts the possible locations of all types of rooms in an input image. Our network has greater integrity and consistency in global layouts compared to ISSNet. We then compare our method to ISSNet+Stage2 using ISSNet for locating rooms and our wall locating network to generate walls. In the user study, average 20.76 floor plans of our method are preferred by

general users and average 18.72 by designers, which indicates that participants preferred results generated by our method. ISSNet has the global consistency problem and may introduce noise due to its image-to-pixel learning process, which causes that necessary rooms are omitted in many cases, as shown in Fig. 12 (a) and (b). Column (c) shows ISSNet+Stage2 generates a floor plan with only four rooms while our method generates a six-room floor plan.



Figure 12

Although a four-room floor plan is acceptable, participants preferred the more intensively populated floor plan generated by our method. In column (e), ISSNet+Stage2 synthesizes an abnormal floor plan where the balcony is connected to the bathroom, which violates the privacy of the bathroom. Benefiting from the image-to-image learning process, our method performs better in terms of room distribution and thus achieves better global integrity and consistency compared to ISSNet+Stage2.

4.4.4 Comparison to Human

We finally compare the floor plans generated by our method with original floor plans from our dataset. These floor plans are designed manually using a combination of intuition, prior experience, and professional knowledge. In the user study, average 13.39 floor plans of our method are preferred by general users and average 11.70 by designers. Participants show a slight preference for the human-designed floor plans to those generated by our method.



Figure 13

In Fig. 13, columns (d) and (e) illustrate that our method generates floor plans similar to those created by humans with only few differences, which proves the validity of our method. For columns (a), (b), and (c), our method provides different design options compared to the original human-created floor plans.

Note that the human-created results of Fig. 10, Fig. 11, and Fig. 12 are shown in the supplementary material.

4.5 PERFORMANCE LOGS

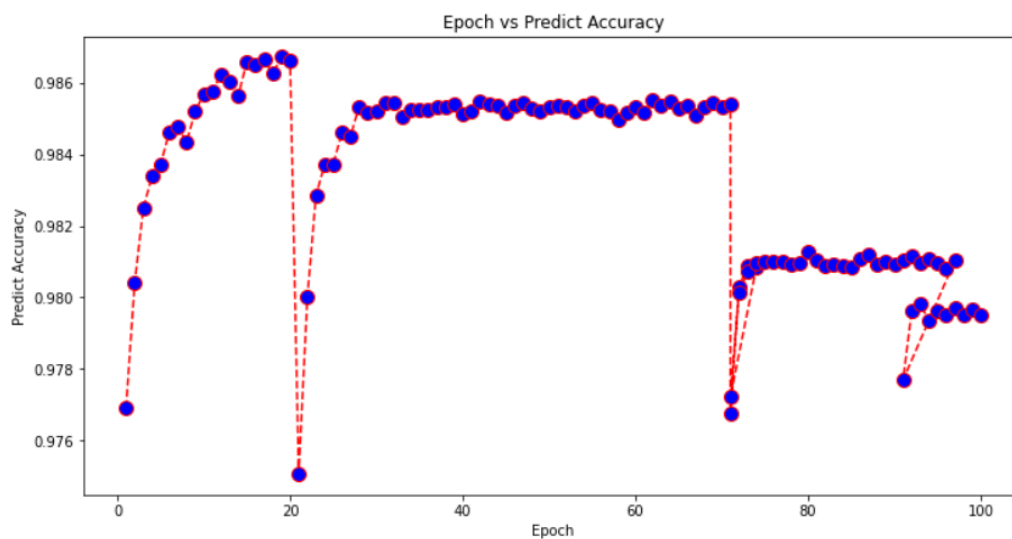
4.5.1 Continuation/Saturation Module

Epoch	Average Loss	Predict Accuracy	Number of Predict Category Right	Number of Target Category	Number of Predict Category	Category Accuracy	Category Proportion
1	0.1906	0.97691	1537	2471	1908	0.62202	0.80556
2	0.0758	0.98041	1721	2471	2211	0.69648	0.77838
3	0.06497	0.98249	1669	2471	1969	0.67544	0.84764
4	0.06002	0.98339	1739	2471	2115	0.70376	0.82222
5	0.05612	0.98372	1720	2471	2069	0.69607	0.83132
6	0.05392	0.98461	1736	2471	2049	0.70255	0.84724
7	0.05096	0.98479	1870	2471	2336	0.75678	0.80051
8	0.04913	0.98434	1768	2471	2141	0.7155	0.82578
9	0.04643	0.98519	1732	2471	2005	0.70093	0.86384
10	0.04423	0.98569	1881	2471	2290	0.76123	0.8214
11	0.04203	0.98576	1813	2471	2162	0.73371	0.83858
12	0.03898	0.98624	1836	2471	2179	0.74302	0.84259
13	0.03597	0.98604	1908	2471	2354	0.77216	0.81054

14	0.03211	0.98564	1923	2471	2402	0.77823	0.80058
15	0.02908	0.9866	1896	2471	2285	0.7673	0.82976
16	0.02623	0.98649	1913	2471	2333	0.77418	0.81997
17	0.02345	0.98665	1889	2471	2270	0.76447	0.83216
18	0.0207	0.98626	1937	2471	2402	0.78389	0.80641
19	0.01883	0.98672	1904	2471	2294	0.77054	0.82999
20	0.01662	0.98663	1900	2471	2296	0.76892	0.82753
21	0.18999	0.97507	1540	2471	1919	0.62323	0.8025
22	0.07562	0.98	1674	2471	2138	0.67746	0.78297
23	0.06458	0.98286	1735	2471	2118	0.70214	0.81917
24	0.05818	0.98371	1675	2471	1941	0.67786	0.86296
25	0.05457	0.98373	1750	2471	2129	0.70822	0.82198
26	0.05132	0.9846	1782	2471	2131	0.72117	0.83623
27	0.04842	0.98451	1857	2471	2305	0.75152	0.80564
28	0.04657	0.98534	1818	2471	2160	0.73573	0.84167
29	0.04511	0.98518	1831	2471	2206	0.741	0.83001
30	0.04426	0.98522	1831	2471	2200	0.741	0.83227
31	0.04335	0.98543	1840	2471	2204	0.74464	0.83485
32	0.04239	0.98546	1847	2471	2217	0.74747	0.83311
33	0.04305	0.98506	1830	2471	2213	0.74059	0.82693
34	0.04251	0.98525	1842	2471	2223	0.74545	0.82861
35	0.04244	0.98524	1840	2471	2221	0.74464	0.82846
36	0.04276	0.98523	1842	2471	2226	0.74545	0.82749
37	0.04266	0.98532	1844	2471	2222	0.74626	0.82988
38	0.04191	0.98532	1851	2471	2239	0.74909	0.82671
39	0.04224	0.98541	1848	2471	2224	0.74788	0.83094
40	0.04271	0.98514	1832	2471	2211	0.7414	0.82858
41	0.04247	0.98521	1838	2471	2216	0.74383	0.82942
42	0.04236	0.98549	1850	2471	2223	0.74868	0.83221
43	0.04287	0.9854	1844	2471	2218	0.74626	0.83138
44	0.04237	0.98535	1850	2471	2233	0.74868	0.82848
45	0.04214	0.98515	1842	2471	2230	0.74545	0.82601
46	0.04187	0.98535	1849	2471	2229	0.74828	0.82952
47	0.04307	0.98543	1849	2471	2224	0.74828	0.83138
48	0.04222	0.98529	1846	2471	2228	0.74707	0.82855
49	0.04282	0.98521	1842	2471	2226	0.74545	0.82749
50	0.04271	0.98532	1847	2471	2228	0.74747	0.82899
51	0.0421	0.98536	1848	2471	2228	0.74788	0.82944
52	0.04272	0.98533	1844	2471	2221	0.74626	0.83026
53	0.04231	0.98519	1839	2471	2223	0.74423	0.82726
54	0.04214	0.98536	1846	2471	2223	0.74707	0.83041
55	0.04199	0.98543	1851	2471	2225	0.74909	0.83191

56	0.0422	0.98526	1838	2471	2215	0.74383	0.8298
57	0.04272	0.98522	1838	2471	2218	0.74383	0.82867
58	0.04287	0.98498	1825	2471	2210	0.73857	0.82579
59	0.04284	0.98517	1843	2471	2231	0.74585	0.82609
60	0.04271	0.98532	1848	2471	2230	0.74788	0.8287
61	0.04276	0.98515	1839	2471	2227	0.74423	0.82577
62	0.04177	0.98551	1856	2471	2233	0.75111	0.83117
63	0.04223	0.98535	1847	2471	2226	0.74747	0.82974
64	0.04261	0.98549	1849	2471	2218	0.74828	0.83363
65	0.0424	0.98527	1845	2471	2228	0.74666	0.8281
66	0.04187	0.98538	1850	2471	2231	0.74868	0.82922
67	0.04232	0.98508	1838	2471	2230	0.74383	0.82422
68	0.04241	0.98532	1841	2471	2217	0.74504	0.8304
69	0.04277	0.98546	1851	2471	2228	0.74909	0.83079
70	0.04261	0.98532	1840	2471	2212	0.74464	0.83183
71	0.04294	0.98539	1846	2471	2221	0.74707	0.83116
71	0.18787	0.97677	1555	2471	1954	0.6293	0.7958
72	0.07627	0.98031	1578	2471	1888	0.63861	0.83581
73	0.07281	0.98088	1624	2471	1963	0.65722	0.82731
74	0.07039	0.98083	1624	2471	1968	0.65722	0.8252
71	0.18904	0.97722	1578	2471	2004	0.63861	0.78743
72	0.07618	0.98013	1574	2471	1902	0.63699	0.82755
73	0.07173	0.98072	1596	2471	1925	0.64589	0.82909
74	0.0715	0.98096	1624	2471	1965	0.65722	0.82646
75	0.07037	0.98099	1624	2471	1968	0.65722	0.8252
76	0.07023	0.98099	1619	2471	1958	0.6552	0.82686
77	0.07097	0.981	1628	2471	1968	0.65884	0.82724
78	0.07091	0.98094	1617	2471	1954	0.65439	0.82753
79	0.07074	0.98097	1619	2471	1958	0.6552	0.82686
80	0.07105	0.98127	1639	2471	1973	0.66329	0.83071
81	0.07134	0.98103	1621	2471	1958	0.65601	0.82789
82	0.0707	0.98087	1617	2471	1956	0.65439	0.82669
83	0.06992	0.98091	1619	2471	1963	0.6552	0.82476
84	0.07105	0.98089	1622	2471	1968	0.65641	0.82419
85	0.07117	0.98084	1618	2471	1964	0.6548	0.82383
86	0.07148	0.98109	1626	2471	1964	0.65803	0.8279
87	0.0711	0.98118	1631	2471	1968	0.66006	0.82876
88	0.07099	0.98094	1619	2471	1959	0.6552	0.82644
89	0.07109	0.98101	1631	2471	1976	0.66006	0.8254
90	0.07106	0.98094	1617	2471	1955	0.65439	0.82711
91	0.07178	0.98103	1627	2471	1969	0.65844	0.82631
92	0.07044	0.98115	1630	2471	1963	0.65965	0.83036

93	0.07088	0.98095	1621	2471	1965	0.65601	0.82494
94	0.07044	0.98107	1628	2471	1964	0.65884	0.82892
95	0.07052	0.98096	1614	2471	1950	0.65318	0.82769
96	0.07081	0.98082	1615	2471	1961	0.65358	0.82356
97	0.07024	0.98103	1623	2471	1962	0.65682	0.82722
91	0.19215	0.97769	1431	2471	1665	0.57912	0.85946
92	0.07852	0.97961	1562	2471	1890	0.63213	0.82646
93	0.07673	0.97981	1577	2471	1899	0.6382	0.83044
94	0.07797	0.97935	1553	2471	1882	0.62849	0.82519
95	0.0771	0.97964	1572	2471	1899	0.63618	0.8278
96	0.07675	0.97949	1551	2471	1872	0.62768	0.82853
97	0.07694	0.97969	1561	2471	1879	0.63173	0.83076
98	0.07759	0.97952	1551	2471	1869	0.62768	0.82986
99	0.07724	0.97968	1555	2471	1870	0.6293	0.83155
100	0.07783	0.97949	1560	2471	1886	0.63132	0.82715



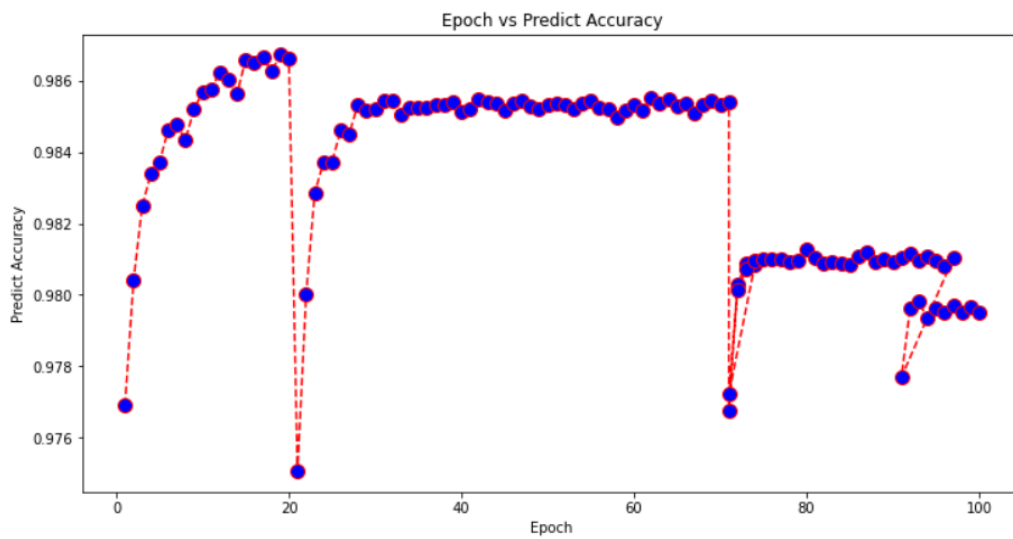
4.5.2 Locating Other rooms Module

Epoch	Average Loss	Predict Accuracy	Number of Predict Category Right	Number of Target Category	Number of Predict Category	Category Accuracy	Category Proportion
1	0.1906	0.97691	1537	2471	1908	0.62202	0.80556
2	0.0758	0.98041	1721	2471	2211	0.69648	0.77838
3	0.06497	0.98249	1669	2471	1969	0.67544	0.84764
4	0.06002	0.98339	1739	2471	2115	0.70376	0.82222
5	0.05612	0.98372	1720	2471	2069	0.69607	0.83132
6	0.05392	0.98461	1736	2471	2049	0.70255	0.84724
7	0.05096	0.98479	1870	2471	2336	0.75678	0.80051
8	0.04913	0.98434	1768	2471	2141	0.7155	0.82578

9	0.04643	0.98519	1732	2471	2005	0.70093	0.86384
10	0.04423	0.98569	1881	2471	2290	0.76123	0.8214
11	0.04203	0.98576	1813	2471	2162	0.73371	0.83858
12	0.03898	0.98624	1836	2471	2179	0.74302	0.84259
13	0.03597	0.98604	1908	2471	2354	0.77216	0.81054
14	0.03211	0.98564	1923	2471	2402	0.77823	0.80058
15	0.02908	0.9866	1896	2471	2285	0.7673	0.82976
16	0.02623	0.98649	1913	2471	2333	0.77418	0.81997
17	0.02345	0.98665	1889	2471	2270	0.76447	0.83216
18	0.0207	0.98626	1937	2471	2402	0.78389	0.80641
19	0.01883	0.98672	1904	2471	2294	0.77054	0.82999
20	0.01662	0.98663	1900	2471	2296	0.76892	0.82753
21	0.18999	0.97507	1540	2471	1919	0.62323	0.8025
22	0.07562	0.98	1674	2471	2138	0.67746	0.78297
23	0.06458	0.98286	1735	2471	2118	0.70214	0.81917
24	0.05818	0.98371	1675	2471	1941	0.67786	0.86296
25	0.05457	0.98373	1750	2471	2129	0.70822	0.82198
26	0.05132	0.9846	1782	2471	2131	0.72117	0.83623
27	0.04842	0.98451	1857	2471	2305	0.75152	0.80564
28	0.04657	0.98534	1818	2471	2160	0.73573	0.84167
29	0.04511	0.98518	1831	2471	2206	0.741	0.83001
30	0.04426	0.98522	1831	2471	2200	0.741	0.83227
31	0.04335	0.98543	1840	2471	2204	0.74464	0.83485
32	0.04239	0.98546	1847	2471	2217	0.74747	0.83311
33	0.04305	0.98506	1830	2471	2213	0.74059	0.82693
34	0.04251	0.98525	1842	2471	2223	0.74545	0.82861
35	0.04244	0.98524	1840	2471	2221	0.74464	0.82846
36	0.04276	0.98523	1842	2471	2226	0.74545	0.82749
37	0.04266	0.98532	1844	2471	2222	0.74626	0.82988
38	0.04191	0.98532	1851	2471	2239	0.74909	0.82671
39	0.04224	0.98541	1848	2471	2224	0.74788	0.83094
40	0.04271	0.98514	1832	2471	2211	0.7414	0.82858
41	0.04247	0.98521	1838	2471	2216	0.74383	0.82942
42	0.04236	0.98549	1850	2471	2223	0.74868	0.83221
43	0.04287	0.9854	1844	2471	2218	0.74626	0.83138
44	0.04237	0.98535	1850	2471	2233	0.74868	0.82848
45	0.04214	0.98515	1842	2471	2230	0.74545	0.82601
46	0.04187	0.98535	1849	2471	2229	0.74828	0.82952
47	0.04307	0.98543	1849	2471	2224	0.74828	0.83138
48	0.04222	0.98529	1846	2471	2228	0.74707	0.82855
49	0.04282	0.98521	1842	2471	2226	0.74545	0.82749
50	0.04271	0.98532	1847	2471	2228	0.74747	0.82899

51	0.0421	0.98536	1848	2471	2228	0.74788	0.82944
52	0.04272	0.98533	1844	2471	2221	0.74626	0.83026
53	0.04231	0.98519	1839	2471	2223	0.74423	0.82726
54	0.04214	0.98536	1846	2471	2223	0.74707	0.83041
55	0.04199	0.98543	1851	2471	2225	0.74909	0.83191
56	0.0422	0.98526	1838	2471	2215	0.74383	0.8298
57	0.04272	0.98522	1838	2471	2218	0.74383	0.82867
58	0.04287	0.98498	1825	2471	2210	0.73857	0.82579
59	0.04284	0.98517	1843	2471	2231	0.74585	0.82609
60	0.04271	0.98532	1848	2471	2230	0.74788	0.8287
61	0.04276	0.98515	1839	2471	2227	0.74423	0.82577
62	0.04177	0.98551	1856	2471	2233	0.75111	0.83117
63	0.04223	0.98535	1847	2471	2226	0.74747	0.82974
64	0.04261	0.98549	1849	2471	2218	0.74828	0.83363
65	0.0424	0.98527	1845	2471	2228	0.74666	0.8281
66	0.04187	0.98538	1850	2471	2231	0.74868	0.82922
67	0.04232	0.98508	1838	2471	2230	0.74383	0.82422
68	0.04241	0.98532	1841	2471	2217	0.74504	0.8304
69	0.04277	0.98546	1851	2471	2228	0.74909	0.83079
70	0.04261	0.98532	1840	2471	2212	0.74464	0.83183
71	0.04294	0.98539	1846	2471	2221	0.74707	0.83116
71	0.18787	0.97677	1555	2471	1954	0.6293	0.7958
72	0.07627	0.98031	1578	2471	1888	0.63861	0.83581
73	0.07281	0.98088	1624	2471	1963	0.65722	0.82731
74	0.07039	0.98083	1624	2471	1968	0.65722	0.8252
71	0.18904	0.97722	1578	2471	2004	0.63861	0.78743
72	0.07618	0.98013	1574	2471	1902	0.63699	0.82755
73	0.07173	0.98072	1596	2471	1925	0.64589	0.82909
74	0.0715	0.98096	1624	2471	1965	0.65722	0.82646
75	0.07037	0.98099	1624	2471	1968	0.65722	0.8252
76	0.07023	0.98099	1619	2471	1958	0.6552	0.82686
77	0.07097	0.981	1628	2471	1968	0.65884	0.82724
78	0.07091	0.98094	1617	2471	1954	0.65439	0.82753
79	0.07074	0.98097	1619	2471	1958	0.6552	0.82686
80	0.07105	0.98127	1639	2471	1973	0.66329	0.83071
81	0.07134	0.98103	1621	2471	1958	0.65601	0.82789
82	0.0707	0.98087	1617	2471	1956	0.65439	0.82669
83	0.06992	0.98091	1619	2471	1963	0.6552	0.82476
84	0.07105	0.98089	1622	2471	1968	0.65641	0.82419
85	0.07117	0.98084	1618	2471	1964	0.6548	0.82383
86	0.07148	0.98109	1626	2471	1964	0.65803	0.8279
87	0.0711	0.98118	1631	2471	1968	0.66006	0.82876

88	0.07099	0.98094	1619	2471	1959	0.6552	0.82644
89	0.07109	0.98101	1631	2471	1976	0.66006	0.8254
90	0.07106	0.98094	1617	2471	1955	0.65439	0.82711
91	0.07178	0.98103	1627	2471	1969	0.65844	0.82631
92	0.07044	0.98115	1630	2471	1963	0.65965	0.83036
93	0.07088	0.98095	1621	2471	1965	0.65601	0.82494
94	0.07044	0.98107	1628	2471	1964	0.65884	0.82892
95	0.07052	0.98096	1614	2471	1950	0.65318	0.82769
96	0.07081	0.98082	1615	2471	1961	0.65358	0.82356
97	0.07024	0.98103	1623	2471	1962	0.65682	0.82722
91	0.19215	0.97769	1431	2471	1665	0.57912	0.85946
92	0.07852	0.97961	1562	2471	1890	0.63213	0.82646
93	0.07673	0.97981	1577	2471	1899	0.6382	0.83044
94	0.07797	0.97935	1553	2471	1882	0.62849	0.82519
95	0.0771	0.97964	1572	2471	1899	0.63618	0.8278
96	0.07675	0.97949	1551	2471	1872	0.62768	0.82853
97	0.07694	0.97969	1561	2471	1879	0.63173	0.83076
98	0.07759	0.97952	1551	2471	1869	0.62768	0.82986
99	0.07724	0.97968	1555	2471	1870	0.6293	0.83155
100	0.07783	0.97949	1560	2471	1886	0.63132	0.82715

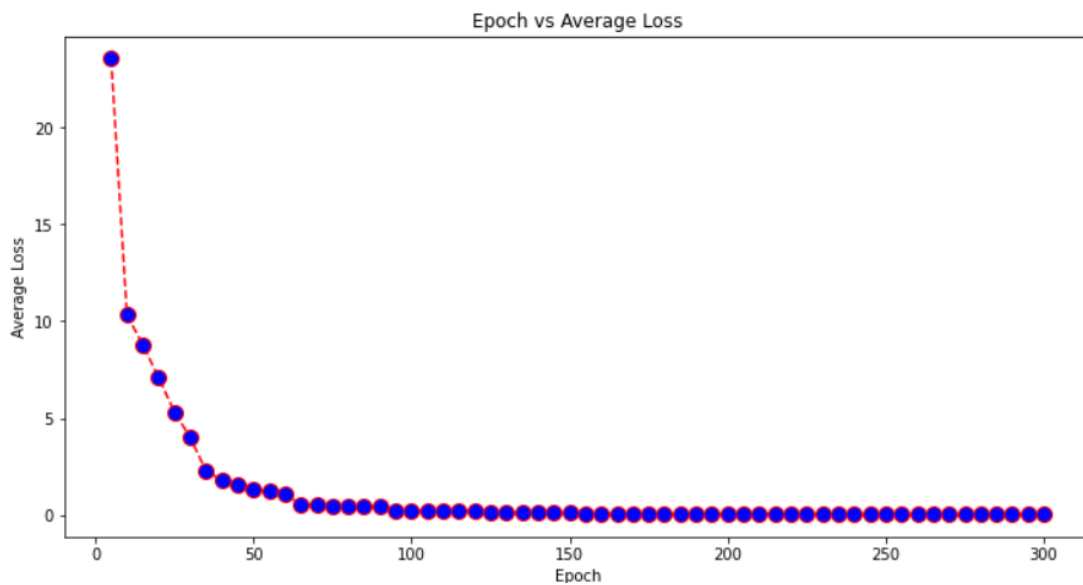


4.5.3 Living Room Location Module

Epoch	Average Loss	Val Loss
5	23.55354	2.10555
10	10.32724	1.23546
15	8.79	1.00845
20	7.10077	1.04762
25	5.31423	0.99463

30	4.00245	1.06215
35	2.27336	1.00154
40	1.79767	0.97625
45	1.52703	0.9734
50	1.32854	0.96662
55	1.22033	0.97084
60	1.05748	0.96063
65	0.54087	0.94842
70	0.48536	0.94547
75	0.48019	0.94238
80	0.45848	0.94493
85	0.43043	0.94139
90	0.41032	0.94554
95	0.2196	0.94007
100	0.20971	0.94237
105	0.19167	0.93961
110	0.19625	0.93714
115	0.18487	0.94077
120	0.19092	0.939
125	0.10712	0.93862
130	0.12038	0.93933
135	0.10662	0.93785
140	0.11648	0.93759
145	0.10911	0.9394
150	0.10351	0.93686
155	0.07786	0.93627
160	0.08011	0.93771
165	0.07552	0.93783
170	0.07398	0.93752
175	0.07405	0.9379
180	0.06883	0.93688
185	0.06175	0.93717
190	0.06222	0.93701
195	0.06282	0.93663
200	0.06255	0.93687
205	0.05932	0.93714
210	0.06104	0.9376
215	0.05475	0.93659
220	0.05769	0.93677
225	0.05724	0.93771
230	0.05634	0.9368
235	0.0563	0.93791
240	0.05452	0.93675

245	0.05333	0.93706
250	0.05203	0.93786
255	0.05724	0.93756
260	0.05244	0.93682
265	0.04983	0.93662
270	0.05301	0.93728
275	0.05083	0.93719
280	0.05203	0.93728
285	0.05005	0.93703
290	0.05255	0.93837
295	0.05064	0.93722
300	0.05018	0.9371



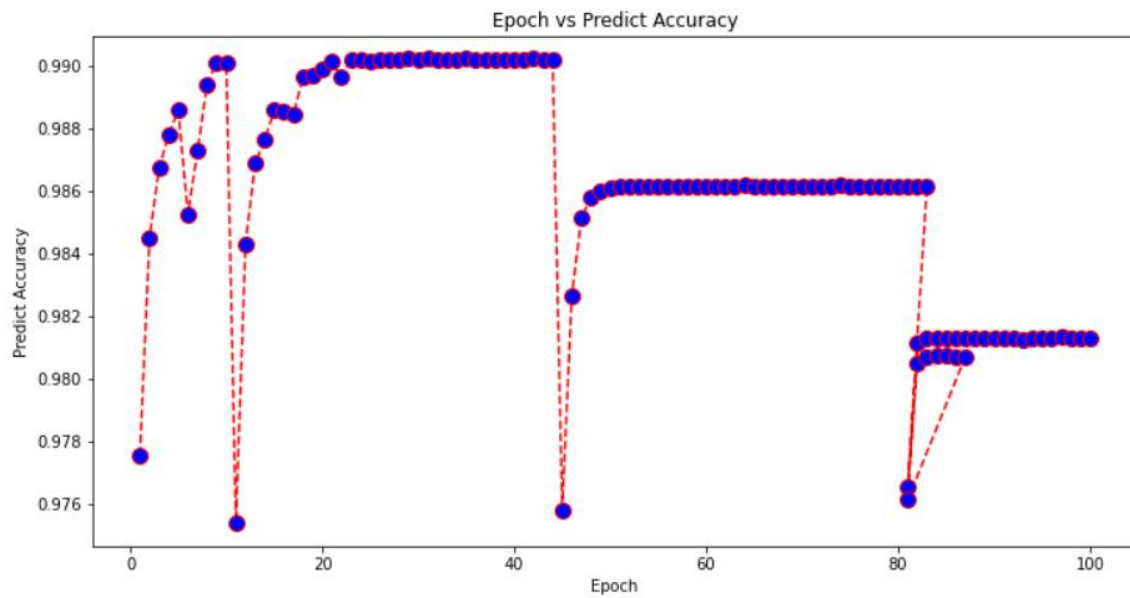
4.5.4 Wall Generation Module

Epoch	Average Loss	Predict Accuracy	Number of Predict Wall Right	Number of Target Wall	Number of Predict Wall	Wall Accuracy	Wall Proportion
1	0.10817	0.97754	778	1711	1133	0.4547	0.68667
2	0.04474	0.98451	1269	1711	1752	0.74167	0.72432
3	0.03451	0.98672	1277	1711	1645	0.74635	0.77629
4	0.02951	0.9878	1350	1711	1735	0.78901	0.7781
5	0.02615	0.98858	1331	1711	1657	0.77791	0.80326
6	0.02306	0.98525	1553	1711	2323	0.90766	0.66853
7	0.0205	0.98726	1545	1711	2181	0.90298	0.70839
8	0.01804	0.9894	1277	1711	1508	0.74635	0.84682
9	0.01616	0.9901	1446	1711	1806	0.84512	0.80066
10	0.01396	0.99006	1317	1711	1553	0.76973	0.84804

11	0.10843	0.97539	959	1711	1463	0.56049	0.6555
12	0.04477	0.98428	1046	1711	1323	0.61134	0.79063
13	0.03422	0.98686	1292	1711	1666	0.75511	0.77551
14	0.0281	0.98764	1263	1711	1558	0.73816	0.81065
15	0.02378	0.98856	1389	1711	1780	0.81181	0.78034
16	0.01955	0.98851	1440	1711	1887	0.84161	0.76312
17	0.01587	0.98844	1503	1711	2021	0.87843	0.74369
18	0.01288	0.98965	1428	1711	1795	0.8346	0.79554
19	0.01039	0.98968	1278	1711	1493	0.74693	0.85599
20	0.00854	0.98986	1297	1711	1519	0.75804	0.85385
21	0.00711	0.99013	1362	1711	1632	0.79603	0.83456
22	0.0061	0.98962	1241	1711	1426	0.72531	0.87027
23	0.00549	0.99016	1350	1711	1609	0.78901	0.83903
24	0.00514	0.99018	1376	1711	1659	0.80421	0.82942
25	0.00485	0.99014	1339	1711	1588	0.78258	0.8432
26	0.00466	0.99019	1362	1711	1631	0.79603	0.83507
27	0.00455	0.9902	1392	1711	1690	0.81356	0.82367
28	0.00446	0.99019	1361	1711	1629	0.79544	0.83548
29	0.00443	0.99021	1368	1711	1642	0.79953	0.83313
30	0.00438	0.9902	1366	1711	1638	0.79836	0.83394
31	0.00438	0.99021	1374	1711	1654	0.80304	0.83071
32	0.00437	0.99019	1366	1711	1640	0.79836	0.83293
33	0.00437	0.9902	1366	1711	1638	0.79836	0.83394
34	0.00435	0.9902	1372	1711	1651	0.80187	0.83101
35	0.00435	0.99021	1374	1711	1655	0.80304	0.83021
36	0.00437	0.9902	1365	1711	1636	0.79778	0.83435
37	0.00437	0.9902	1367	1711	1640	0.79895	0.83354
38	0.00436	0.9902	1369	1711	1644	0.80012	0.83273
39	0.00437	0.9902	1380	1711	1666	0.80655	0.82833
40	0.00435	0.9902	1376	1711	1658	0.80421	0.82992
41	0.00437	0.99019	1362	1711	1632	0.79603	0.83456
42	0.00435	0.99021	1378	1711	1663	0.80538	0.82862
43	0.00434	0.9902	1372	1711	1650	0.80187	0.83152
44	0.00436	0.9902	1373	1711	1652	0.80245	0.83111
45	0.10974	0.97581	1121	1711	1869	0.65517	0.59979
46	0.04661	0.98266	1326	1711	1973	0.77499	0.67207
47	0.03706	0.98512	1288	1711	1757	0.75278	0.73307
48	0.03302	0.98579	1272	1711	1688	0.74342	0.75355
49	0.03125	0.98599	1298	1711	1728	0.75862	0.75116
50	0.0305	0.98607	1300	1711	1728	0.75979	0.75231
51	0.03021	0.98612	1291	1711	1707	0.75453	0.7563
52	0.03012	0.98614	1287	1711	1697	0.75219	0.7584

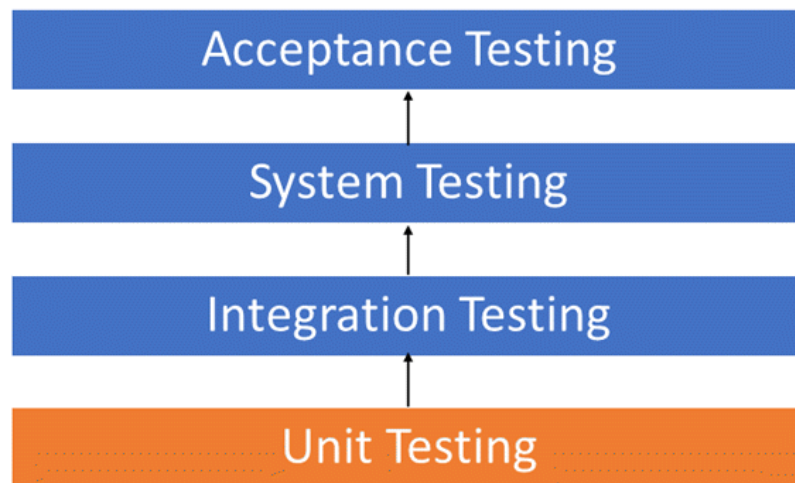
53	0.03006	0.98613	1288	1711	1701	0.75278	0.7572
54	0.03008	0.98614	1287	1711	1697	0.75219	0.7584
55	0.03005	0.98612	1294	1711	1713	0.75628	0.7554
56	0.03006	0.98615	1285	1711	1693	0.75102	0.75901
57	0.03008	0.98614	1290	1711	1704	0.75395	0.75704
58	0.03007	0.98612	1294	1711	1713	0.75628	0.7554
59	0.03006	0.98613	1291	1711	1706	0.75453	0.75674
60	0.03006	0.98613	1291	1711	1707	0.75453	0.7563
61	0.03004	0.98612	1295	1711	1716	0.75687	0.75466
62	0.03008	0.98612	1296	1711	1717	0.75745	0.7548
63	0.03005	0.98613	1294	1711	1712	0.75628	0.75584
64	0.03007	0.98616	1278	1711	1678	0.74693	0.76162
65	0.03006	0.98612	1295	1711	1714	0.75687	0.75554
66	0.03006	0.98614	1287	1711	1697	0.75219	0.7584
67	0.03007	0.98612	1293	1711	1710	0.7557	0.75614
68	0.03009	0.98611	1301	1711	1727	0.76037	0.75333
69	0.03008	0.98614	1288	1711	1700	0.75278	0.75765
70	0.03006	0.98614	1286	1711	1694	0.75161	0.75915
71	0.03008	0.98612	1294	1711	1713	0.75628	0.7554
72	0.03005	0.98611	1298	1711	1721	0.75862	0.75421
73	0.03008	0.98612	1296	1711	1716	0.75745	0.75524
74	0.0301	0.98616	1272	1711	1664	0.74342	0.76442
75	0.03007	0.98614	1285	1711	1693	0.75102	0.75901
76	0.03007	0.98611	1298	1711	1722	0.75862	0.75377
77	0.03007	0.98612	1294	1711	1713	0.75628	0.7554
78	0.03006	0.98613	1291	1711	1705	0.75453	0.75718
79	0.03007	0.98611	1294	1711	1713	0.75628	0.7554
80	0.03006	0.98612	1297	1711	1720	0.75804	0.75407
81	0.03007	0.98615	1281	1711	1684	0.74868	0.76069
82	0.03006	0.98613	1293	1711	1710	0.7557	0.75614
83	0.03006	0.98614	1282	1711	1687	0.74927	0.75993
81	0.10926	0.97652	863	1711	1295	0.50438	0.66641
82	0.05005	0.98049	977	1711	1388	0.57101	0.70389
83	0.04705	0.98068	1081	1711	1588	0.63179	0.68073
84	0.04676	0.98072	1066	1711	1556	0.62303	0.68509
85	0.04674	0.98072	1052	1711	1526	0.61485	0.68938
86	0.04675	0.98069	1080	1711	1585	0.63121	0.68139
87	0.04674	0.98069	1076	1711	1578	0.62887	0.68188
81	0.10797	0.97612	930	1711	1446	0.54354	0.64315
82	0.04876	0.98112	1070	1711	1539	0.62537	0.69526
83	0.04579	0.98128	1096	1711	1580	0.64056	0.69367
84	0.04551	0.98128	1097	1711	1583	0.64115	0.69299

85	0.04549	0.9813	1089	1711	1565	0.63647	0.69585
86	0.04548	0.98128	1098	1711	1585	0.64173	0.69274
87	0.04548	0.9813	1077	1711	1541	0.62946	0.6989
88	0.04549	0.98128	1102	1711	1592	0.64407	0.69221
89	0.04547	0.98129	1088	1711	1564	0.63589	0.69565
90	0.04548	0.98129	1092	1711	1572	0.63822	0.69466
91	0.04547	0.9813	1079	1711	1546	0.63063	0.69793
92	0.04548	0.98131	1076	1711	1539	0.62887	0.69916
93	0.04548	0.98126	1104	1711	1596	0.64524	0.69173
94	0.04548	0.9813	1091	1711	1570	0.63764	0.6949
95	0.0455	0.98129	1093	1711	1574	0.63881	0.69441
96	0.0455	0.98128	1098	1711	1585	0.64173	0.69274
97	0.0455	0.98132	1075	1711	1536	0.62829	0.69987
98	0.04548	0.98129	1093	1711	1574	0.63881	0.69441
99	0.04548	0.98129	1097	1711	1582	0.64115	0.69343
100	0.04547	0.98129	1093	1711	1573	0.63881	0.69485



5 TESTING

5.1 UNIT TESTING



Unit testing is the first level of testing and is often performed by the developers themselves. It is the process of ensuring individual components of a piece of software at the code level are functional and work as they were designed to. Developers in a test-driven environment will typically write and run the tests prior to the software or feature being passed over to the test team. Unit testing can be conducted manually, but automating the process will speed up delivery cycles and expand test coverage. Unit testing will also make debugging easier because finding issues earlier means they take less time to fix than if they were discovered later in the testing process.

Our model has 4 modules and they have inbuilt tests and loss functions to see if the training is of the best standard or not. The modules are namely 'living_train', 'wall_train', 'location_train' and 'continue_train'. All these modules go rigorous testing for every iteration and every epoch they go through. This happens due to inbuilt testing of the training module.

The modules that follow training modules are Vectorization and development module. These modules are based on GAN and visualization that also have their loss functions but don't have evaluation metrics as there are no metrics that are unanimously observed as the evaluation metrics for such models.

5.1.1 Why to do Unit Testing

- Unit tests help to fix bugs early in the development cycle and save costs.
- It helps the developers to understand the testing code base and enables them to make changes quickly
- Good unit tests serve as project documentation
- Unit tests help with code re-use. Migrate both your code and your tests to your new project. Tweak the code until the tests run again.
- Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit API.
- Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly (i.e., Regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed.
- Due to the modular nature of the unit testing, we can test parts of the project without waiting for others to be completed.

5.1.2 Common Practices Applied in Unit Testing of our Model

- Unit Test cases should be independent. In case of any enhancements or change in requirements, unit test cases should not be affected.
- Test only one code at a time.
- Follow clear and consistent naming conventions for your unit tests
- In case of a change in code in any module, ensure there is a corresponding unit Test Case for the module, and the module passes the tests before changing the implementation
- Bugs identified during unit testing must be fixed before proceeding to the next phase in SDLC
- Adopt a "test as your code" approach. The more code you write without testing, the more paths you have to check for errors.

5.1.3 Code Coverage

The Unit Testing Techniques are mainly categorized into three parts which are Black box testing that involves testing of user interface along with input and output, White box testing that involves testing the functional behaviour of the software application and Gray box testing that is used to execute test suites, test methods, test cases and performing risk analysis.

5.1.3.1 Statement Coverage

Statement Coverage is a white box testing technique in which all the executable statements in the source code are executed at least once. It is used for calculation of the number of statements in source code which have been executed. The main purpose of Statement Coverage is to cover all the possible paths, lines and statements in source code. Statement coverage is used to derive scenario based upon the structure of the code under test.

$$\text{Statement Coverage} = \frac{\text{Number of Executed Statements}}{\text{Total number of Statements}} \times 100$$

In White Box Testing, the tester is concentrating on how the software works. In other words, the tester will be concentrating on the internal working of source code concerning control flow graphs or flow charts.

Generally, in any software, if we look at the source code, there will be a wide variety of elements like operators, functions, looping, exceptional handlers, etc. Based on the input to the program, some of the code statements may not be executed. The goal of Statement coverage is to cover all the possible path's, line, and statement in the code.

5.1.3.2 Decision Coverage

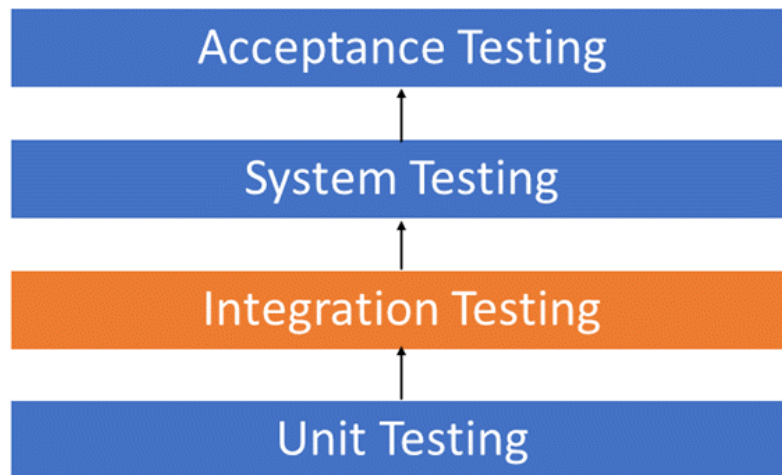
Decision Coverage is a white box testing technique which reports the true or false outcomes of each Boolean expression of the source code. The goal of decision coverage testing is to cover and validate all the accessible source code by checking and ensuring that each branch of every possible decision point is executed at least once.

In this coverage, expressions can sometimes get complicated. Therefore, it is very hard to achieve 100% coverage. That's why there are many different methods of reporting this metric. All these methods focus on covering the most important combinations. It is very much similar to decision coverage, but it offers better sensitivity to control flow.

$$\text{Decision Coverage} = \frac{\text{Number of Decision Outcomes Exercised}}{\text{Total Number of Decision Outcomes}}$$

There are more models in Code Coverage but our model and its modules are too complicated for unit testing with line to line, condition to condition testing of our code.

5.2 INTEGRATION TESTING



After each unit is thoroughly tested, it is integrated with other units to create modules or components that are designed to perform specific tasks or activities. These are then tested as group through integration testing to ensure whole segments of an application behave as expected (i.e., the interactions between units are seamless). These tests are often framed by user scenarios, such as logging into an application or opening files. Integrated tests can be conducted by either developers or independent testers and are usually comprised of a combination of automated functional and manual tests. Although each software module is unit tested, defects still exist for various reasons like:

- A Module, in general, is designed by an individual software developer whose understanding and programming logic may differ from other programmers. Integration Testing becomes necessary to verify the software modules work in unity
- At the time of module development, there are wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence system integration Testing becomes necessary.
- Interfaces of the software modules with the database could be erroneous
- External Hardware interfaces, if any, could be erroneous
- Inadequate exception handling could cause issues.

5.2.1 Big Bang Testing

Big Bang Testing is an Integration testing approach in which all the components or modules are integrated together at once and then tested as a unit. This combined set of components is considered as an entity while testing. If all of the components in the unit are not completed, the integration process will not execute.

5.2.1.1 Advantages

- Convenient for small systems.

5.2.1.2 Disadvantages

- Fault Localization is difficult.
- Given the sheer number of interfaces that need to be tested in this approach, some interfaces link to be tested could be missed easily.
- Since the Integration testing can commence only after "all" the modules are designed, the testing team will have less time for execution in the testing phase.
- Since all modules are tested at once, high-risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.

5.2.2 Incremental Testing

In the Incremental Testing approach, testing is done by integrating two or more modules that are logically related to each other and then tested for proper functioning of the application. Then the other related modules are integrated incrementally and the process continues until all the logically related modules are integrated and tested successfully. Incremental Approach, in turn, is carried out by two different Methods but in our case only the latter holds relevance:

- Bottom Up
- Top Down

5.2.2.1 Stubs and Drivers

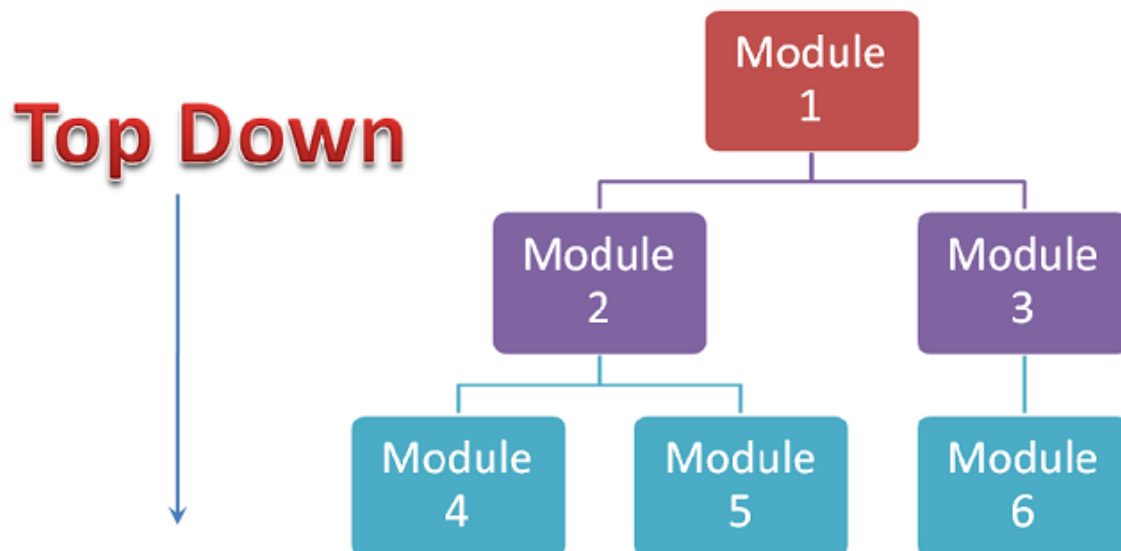
Stubs and Drivers are the dummy programs in Integration testing used to facilitate the software testing activity. These programs act as substitutes for the missing models in the testing. They do not implement the entire programming logic of the software module but they simulate data communication with the calling module while testing.

- Stub: Is called by the Module under Test
- Driver: Calls the Module to be tested

5.2.3 Top-down Integration Testing

Top-Down Integration Testing is a method in which integration testing takes place from top to bottom following the control flow of software system. The higher-level modules are tested first and then lower-level modules are tested and integrated in order to check the software functionality. Stubs are used for testing if some modules are not ready.

This is the model that holds the most relevance for us. When training GAN models you can't rely on stubs and drivers to make your case for the trained models. But in our case, that is not how it goes. We have also included a pre-trained Resnet model to integrate the training modules so that we can go top down all the times we want to. This even though might completely mess up the accuracy and runtime but gives us an understanding of the integration testing method.



5.2.3.1 Advantages

- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

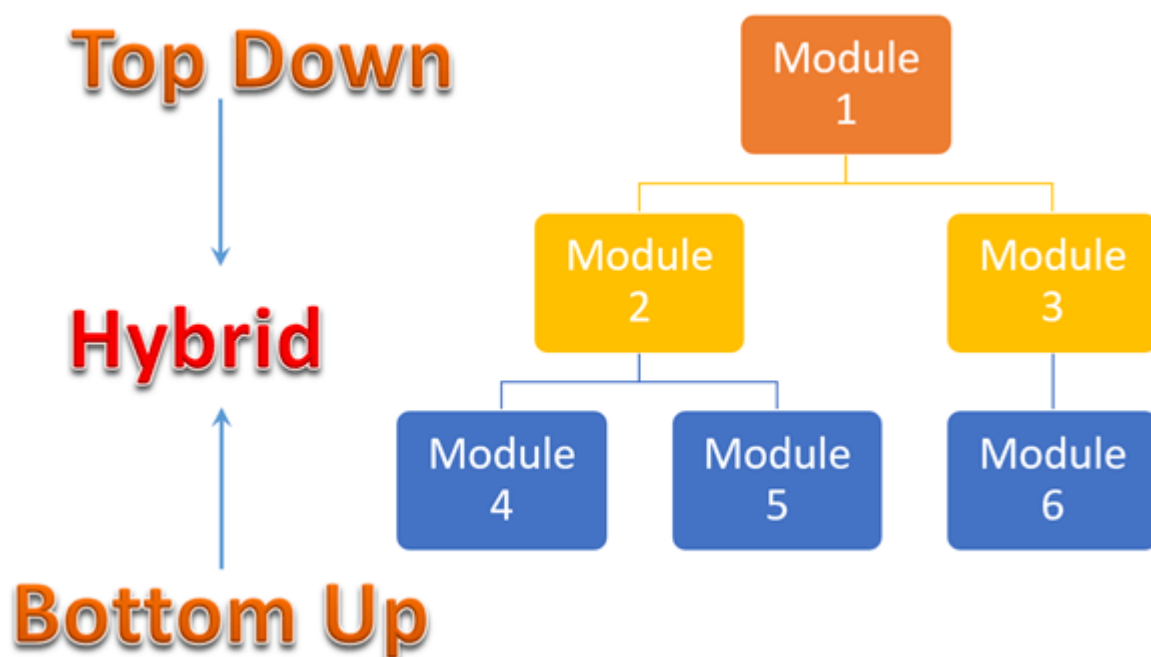
5.2.3.2 Disadvantages

- Needs many Stubs.
- Modules at a lower level are tested inadequately.

5.2.4 Sandwich Testing

Sandwich Testing is a strategy in which top level modules are tested with lower-level modules at the same time lower modules are integrated with top modules and tested as a system. It is a combination of Top-down and Bottom-up approaches therefore it is called Hybrid Integration Testing. It makes use of both stubs as well as drivers.

We have used this testing widely as we have a vectorization module that cannot be trained without all the modules below it being trained. So, we compile all the models and do the testing simultaneously on remote machines with same hardware and software configuration to achieve good results on its compilation.



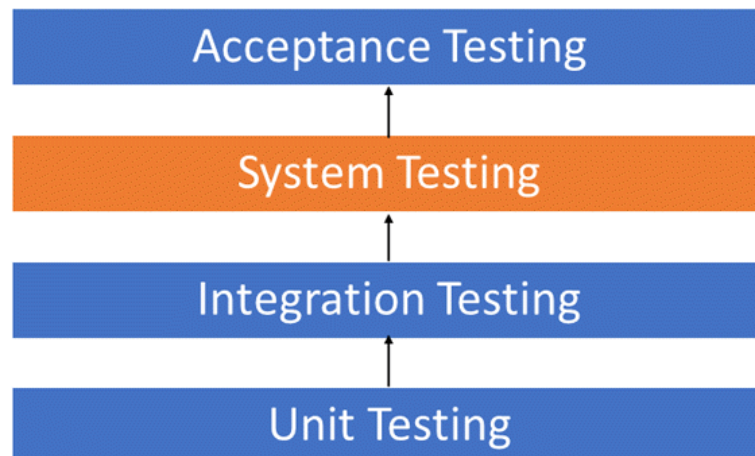
5.2.4.1 Entry Criteria

- Unit Tested Components/Modules
- All High prioritized bugs fixed and closed
- All Modules to be code completed and integrated successfully
- Integration tests Plan, test case, scenarios to be signed off and documented
- Required Test Environment to be set up for Integration testing

5.2.4.2 Exit Criteria

- Successful Testing of Integrated Application.
- Executed Test Cases are documented
- All High prioritized bugs fixed and closed
- Technical documents to be submitted followed by release notes

5.3 SYSTEM TESTING



System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

5.3.1 What do we test in black box system testing?

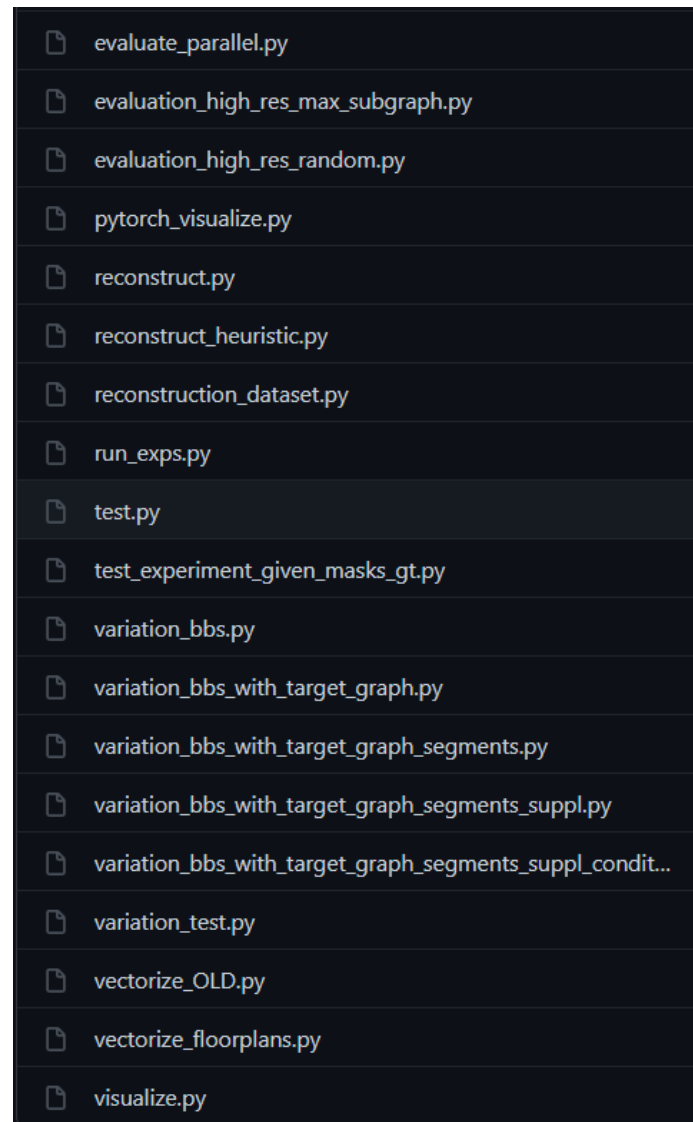
- Testing the fully integrated applications including external peripherals in order to check how components interact with one another and with the system as a whole. This is also called End to End testing scenario.
- Verify thorough testing of every input in the application to check for desired outputs.
- Testing of the user's experience with the application.




















5.3.2 All the System Testing Methodologies used in the process

- **All-pairs Testing:** Combinatorial testing method that tests all possible discrete combinations of input parameters. It is performed by the testing teams.
- **Big Bang Integration Testing:** Testing technique which integrates individual program modules only when everything is ready. It is performed by the testing teams.
- **Branch Testing:** Testing technique in which all branches in the program source code are tested at least once. This is done by the developer.
- **Compatibility Testing:** Testing technique that validates how well a software performs in a particular hardware/software/operating system/network environment. It is performed by the testing teams.
- **Performance Testing:** Functional testing conducted to evaluate the compliance of a system or component with specified performance requirements. It is usually conducted by the performance engineer.
- **Volume Testing:** Testing which confirms that any values that may become large over time (such as accumulated counts, logs, and data files), can be accommodated by the program and will not cause the program to stop working or degrade its operation in any manner. It is usually conducted by the performance engineer.
- **Negative Testing:** Also known as "test to fail" - testing method where the tests' aim is showing that a component or system does not work. It is performed by manual or automation testers.

5.4 LOCAL SOFTWARE TESTING SCRIPTS

The names of the files are self-explanatory and are the scripts that we ran for the testing that was extensive and deep. For e.g., test.py runs the integration testing big-bang approach when file location is the trained model and performs the top-down approach when file location is pre-trained model.



 evaluate_parallel.py
 evaluation_high_res_max_subgraph.py
 evaluation_high_res_random.py
 pytorch_visualize.py
 reconstruct.py
 reconstruct_heuristic.py
 reconstruction_dataset.py
 run_exps.py
 test.py
 test_experiment_given_masks_gt.py
 variation_bbs.py
 variation_bbs_with_target_graph.py
 variation_bbs_with_target_graph_segments.py
 variation_bbs_with_target_graph_segments_suppl.py
 variation_bbs_with_target_graph_segments_suppl_condit...
 variation_test.py
 vectorize_OLD.py
 vectorize_floorplans.py
 visualize.py

6 CONCLUSION

Generating floor models has been a largely manual process. Hence, our aim was to automate it. Our model and program are able to successfully determine floor plans, only based on the boundary of the floor. Other models in existence need a lot more attributes like number of rooms, their size and shape, etc. Deciding such things, is not something a non-professional can do.

Generative Adversarial Networks was the fundamental framework used. The particular GAN used was ‘Layout GAN’. [5] It was found the most suitable because it imitates human learning. The generator of the network is, after training, able to produce plausible floor plans just like a professional would. It did so in only about 4 seconds, which is way less than what an interior designer would take.

7 REFERENCES

1. Wu, W., Fu, X. M., Tang, R., Wang, Y., Qi, Y. H., & Liu, L. (2019). Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics (TOG)*, 38(6), 1-12.
2. Hua, H. (2016). Irregular architectural layout synthesis with graphical inputs. *Automation in Construction*, 72, 388-396.
3. Liu, H., Yang, Y. L., AlHalawani, S., & Mitra, N. J. (2013). Constraint-aware interior layout exploration for pre-cast concrete-based buildings. *The Visual Computer*, 29(6), 663-673.
4. Merrell, P., Schkufza, E., & Koltun, V. (2010). Computer-generated residential building layouts. In *ACM SIGGRAPH Asia 2010 papers* (pp. 1-12).
5. Li, J., Yang, J., Hertzmann, A., Zhang, J., & Xu, T. (2019). Layoutgan: Generating graphic layouts with wireframe discriminators. *arXiv preprint arXiv:1901.06767*.
6. Chang, K. H., Cheng, C. Y., Luo, J., Murata, S., Nourbakhsh, M., & Tsuji, Y. (2021). Building-GAN: Graph-Conditioned Architectural Volumetric Design Generation. *arXiv preprint arXiv:2104.13316*.
7. Zheng, H., An, K., Wei, J., & Ren, Y. (2020). Apartment Floor Plans Generation via Generative Adversarial Networks.
8. Navarro-Mateu, D., Carrasco, O., & Cortes Nieves, P. (2021). Color-Patterns to Architecture Conversion through Conditional Generative Adversarial Networks. *Biomimetics*, 6(1), 16.
9. Nauata, N., Chang, K. H., Cheng, C. Y., Mori, G., & Furukawa, Y. (2020, August). House-gan: Relational generative adversarial networks for graph-constrained house layout generation. In *European Conference on Computer Vision* (pp. 162-177). Springer, Cham.
10. Di, X., Yu, P., Yang, D., Zhu, H., Sun, C., & Liu, Y. (2020). End-to-end Generative Floor-plan and Layout with Attributes and Relation Graph. *arXiv preprint arXiv:2012.08514*.
11. Qian, Y., Zhang, H., & Furukawa, Y. (2020). Roof-GAN: Learning to Generate Roof Geometry and Relations for Residential Houses. *arXiv preprint arXiv:2012.09340*.
12. Rengel, R. J. (2011). *The interior plan: Concepts and exercises*. A&C Black.
13. Ritchie, D., Wang, K., & Lin, Y. A. (2019). Fast and flexible indoor scene synthesis via deep convolutional generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 6182-6190).
14. Wang, K., Savva, M., Chang, A. X., & Ritchie, D. (2018). Deep convolutional priors for indoor scene synthesis. *ACM Transactions on Graphics (TOG)*, 37(4), 1-14.
15. Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial networks. *arXiv preprint arXiv:1406.2661*.
16. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
17. Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision* (pp. 1440-1448).
18. Chen, L. C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2017). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4), 834-848.
19. Wu, W., Fan, L., Liu, L., & Wonka, P. (2018, May). Miqp-based layout design for building interiors. In *Computer Graphics Forum* (Vol. 37, No. 2, pp. 511-521).

8 DISCUSSIONS

8.1 REAL LIFE CONSTRAINTS

This model is very generic, intentionally. Real life constraints and the knowledge of the person using the model vary. Still, in actual designing, there need to be at least some constraints to keep everything neat and get desired results faster. After all, there is no point in generating a house with 2 bedrooms when 3 are needed. A simple solution is to introduce more generative models for these additional constraints. Another solution is to transform these constraints that are unknown to our model into something that is actually known.

8.2 STAIRS

Currently, our method only designs floor plans for one-story residential buildings. For multi-story homes, stairs are necessary to connect two consecutive floors. Our method can be applied by conceptualizing the stairs as a type of room. We generate the first-floor plan containing the stairs, and then the second-floor plan is generated based on the first. The stairs generated in the first floor should serve as a constraint for the generation of the second floor. However, in the real world, stairs have special shapes and location considerations, and it may be hard for our deep networks to control the generation of stairs.

8.3 OTHER TYPES OF BUILDINGS

This two-phase approach, without the living-room-first-strategy, can be extended to other types of buildings, like offices, supermarkets and shopping malls. However, we will need to have a large dataset of their blueprints and people and businesses willing to give it to us.