BRAUDE
College of Engineering, karmiel

Capstone Project Phase A

House Gan ++

Project number -23-2-R-14

Prof. Zeev Volkovich
Dr. Renata Avros

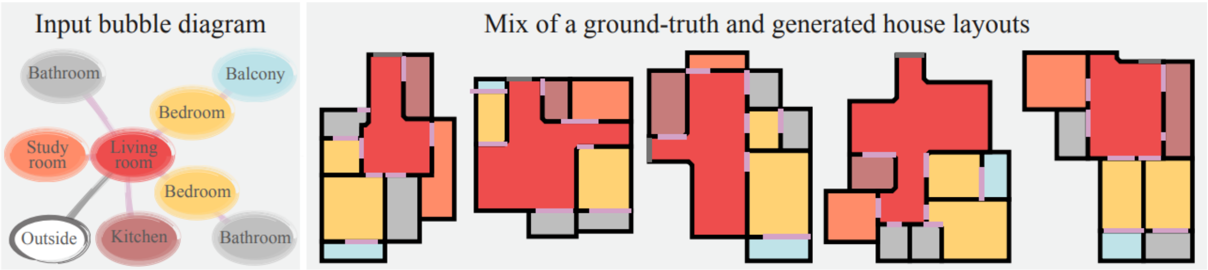
Mosa Hadish - moseshadish@gmail.com
Dorin Beery - dorinbeery@gmail.com

Table of Contents

Abstract

The main purpose of this project is to present a way for generating professional floor plans for houses. It extends and refines the House-GAN [1].
The discovery is a simple non-iterative training process, named component-wise GT-conditioning, that is effective in the learning curve of the generator.
Also, it provides further improvement by meta-optimization techniques.
Assessment is based on metrics used to evaluate, show significant improvements over the original House-GAN program, and even rises above ground-truth floorplans.

1. Introduction



[Figure 1] A floor plan is a diagram of a house viewed from above. They are a vital part of the house building process as they visualize the house without the need to visit it physically. They help to show the relationships between rooms and spaces and give a feeling of the flow of the house.[4]

The issue being addressed is the creation of house floor plans, a time-consuming process, mostly limited to professional architects which results in huge costs.

The goal is to generate professional floor plans automatically, based on the client's conditions, which will save a lot of time and money.

This project is based on the state of the art House-GAN, the novelty suggested here allows the model to cope with non-rectangular rooms, and achieve better results from previous models. It is possible due to improved architecture. The architecture includes training the GAN with component-wise ground truth-conditioning, where instead of providing the GAN with a professional ground-truth floor plan each iteration, a floor plan is provided with a probability of 0.5 at each iteration. And also this project suggests meta-optimization refinement-scheme which further improves the results compared to models.

The architecture is a mix of graph-constrained relational GAN and conditional GAN.

The term GAN [103], refers to a machine learning framework. Given a training set of ground truth floor plans designed by professionals, it learns to generate floor plans with the same properties.

2. Background and Related Work

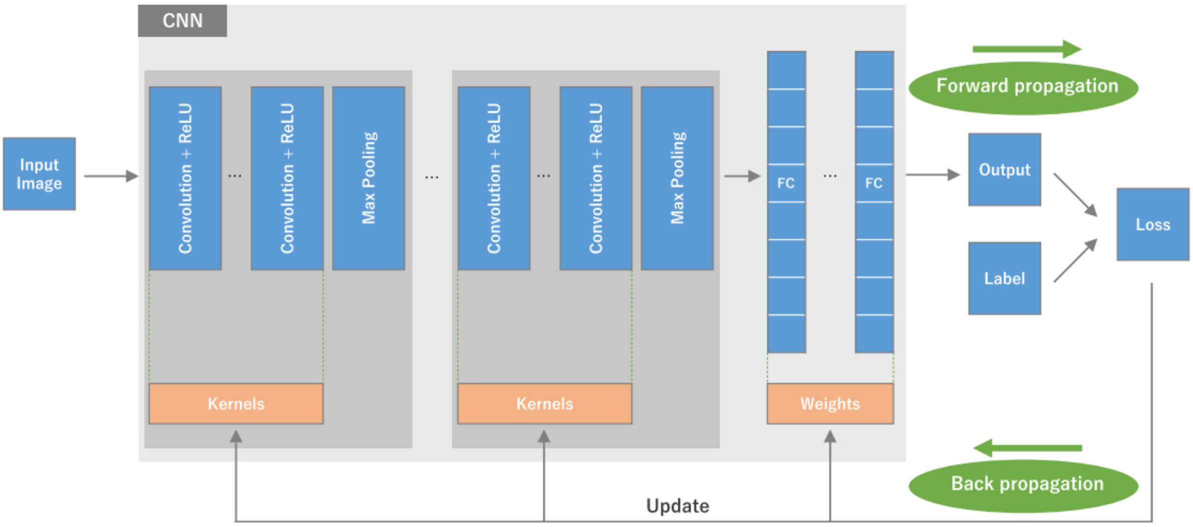
Many methods for generating structured data have been seen in the field. Our work revolves around deep learning, convolutional networks and image processing.

Neural Network

Neural networks are the backbone of all modern Artificial Intelligence (AI) systems. Their mechanism is similar to that of the human brain thus the name “neural”. The network is a massive amount of components named Neurons which are connected to each other and such that they form a network. The neurons receive input data, perform mathematical operation on it and forward it to the next layer of neurons through an activation function that determines which of the data is further forwarded and which not.

2.1 Convolutional Neural Network(CNN)

Is a type of deep learning model mainly for computer vision tasks. Their ability is to learn local stationary structures to generate hierarchical patterns without considering the spatial locations or size and it is their advantage against regular neural networks.



[Figure 2] Illustrates training process of CNN. Input image is a matrix called a tensor and it undergoes linear kernel operation which is a smaller array of numbers. An element wise product between the kernel and tensor and then summation produces an output tensor called feature map. The process repeats on the whole input image to create feature maps each representing a different property of the input image. These are passed to a non linear activation function to simplify the amount of calculations by the model, then, a pooling layer is obtained on these in order to downsample the model. The generated feature maps of the

convolutions are joined to a fully connected layer. The last fully connected layer undergoes a last activation function to provide the probability of classification to a class.
Lastly, the model performance is measured by the loss function through forward propagation, and the learned parameters are updated according to it through backpropagation [7][9].

2.2 Generative Adversarial Network (GAN)

A GAN consists of a generator and a discriminator, where each one of the mentioned is a neural network. This network is usually trained using adversarial training
The generator and the discriminator have 2 different roles:
The generator's role is to generate samples that are indistinguishable from real data, using random noise vectors as input. A generator is typically composed of several layers (such as convolutional layers, etc)
The discriminator's role is to output a probability score, indicating the likelihood of the input being real.
The generator is trained using real data, whereas the discriminator is trained using both real and fake data.

In order to learn a generator's distribution p_g over data x, a prior distribution is defined on input noise variables $p_z(z)$, which defines the characteristics of the input noise, which is then mapped to the data space as $G(z;\theta_g)$, where G is a differentiable function which is represented by a multilayer perceptron with parameters θ_g.
A second perceptron is defined, $D(x;\theta_d)$, which outputs a scalar. D(x) represents the probability that x is real data, not fake. The training goal is for D to maximize its probability of assigning the correct label to both sample types from G (real and fake), while G is simultaneously trained to minimize log(1-D(G(Z)).
This is usually referred to as a two-player minmax game with a value function V(G,D) (often referred to as objective function).

V(D,G) refers to the value function
G(x) refers to the data space generated from x
D(x) refers to the output representing that x came from training data
$E_{x\sim Pdata(x)}$ refers to Expectation function

The generator tries to minimize the value function, while the discriminator tries to maximize it.

In practice, the game is implemented using an iterative, numerical approach. The optimization of D to completion in the inner loop of training is prohibitive, and on finite datasets it would result in overfitting. The inner loop refers to the part of the algorithm that D is trained in to discriminate samples from data. In order to avoid that, the optimization of D and is alternated between k steps, and one step of optimizing G. As long as G changes slowly enough, D is maintained near its optimal solution.

After several steps of training, G and D will both reach a point at which they both cannot improve, meaning the generator outputs fake samples that are almost indistinguishable from real data, causing the discriminator to be unable to differentiate between fake and real data, i.e $D(x) = ½$

The generator tries to deceive the discriminator, meaning that the generator passes on data, some of which is generated by the generator, while the rest is real data, to the discriminator, and the discriminator tries to differentiate between real and fake data. The network updates its parameters by using a process called Backpropagation.

Backpropagation is a process that enables the network to update its parameters for further optimization. There are two key steps in the backpropagation process:

Forward Pass - the process of feeding an input to the generator, as it travels each layer in the network, generating an output sample.

Backward Pass - the gradients of the loss function of the network's output are computed, and propagated backwards through each layer, which are then used to update the parameters of the network using an optimization algorithm.

2.3 Conditional GAN (CGAN)

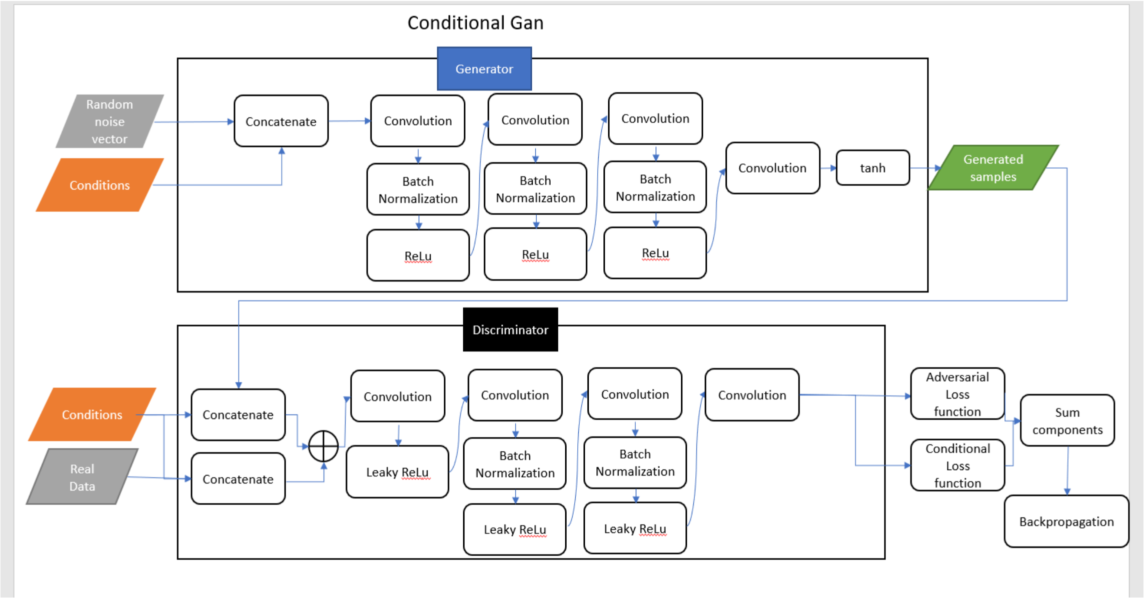
In this variant of a GAN, both the generator and discriminator receive additional input (denoted y) relevant to the task. This allows the user to influence the generated output by the conditioning of the GAN, passing as input the desired characteristics, and the generation is more controlled.
This results in more targeted and controlled sample generation
In HouseGAN++, the cGAN takes in as input the requirements of the layout, such as room types, number of rooms, etc..

However, cGANs don't train easily for context prediction tasks as the discriminator easily exploits the generated regions and the original to easily classify fake versus real samples.

In the generator (denoted as G), the random input noise vector (denoted as z) and y are combined in a joint hidden representation, and the framework allows for flexibility in how this representation is composed. In HouseGAN++, z and y are concatenated. In the discriminator (denoted as D), the data and y are presented as inputs.

The value function of a cGAN is as follows:

$$\min_G \max_D V(D, G) = Ex\text{~}p_{data(x)} [\log D(x|y)] + E_{z\text{~}Pz(z)} [\log(1 - D(G(z|y)))]$$



[Figure 3]

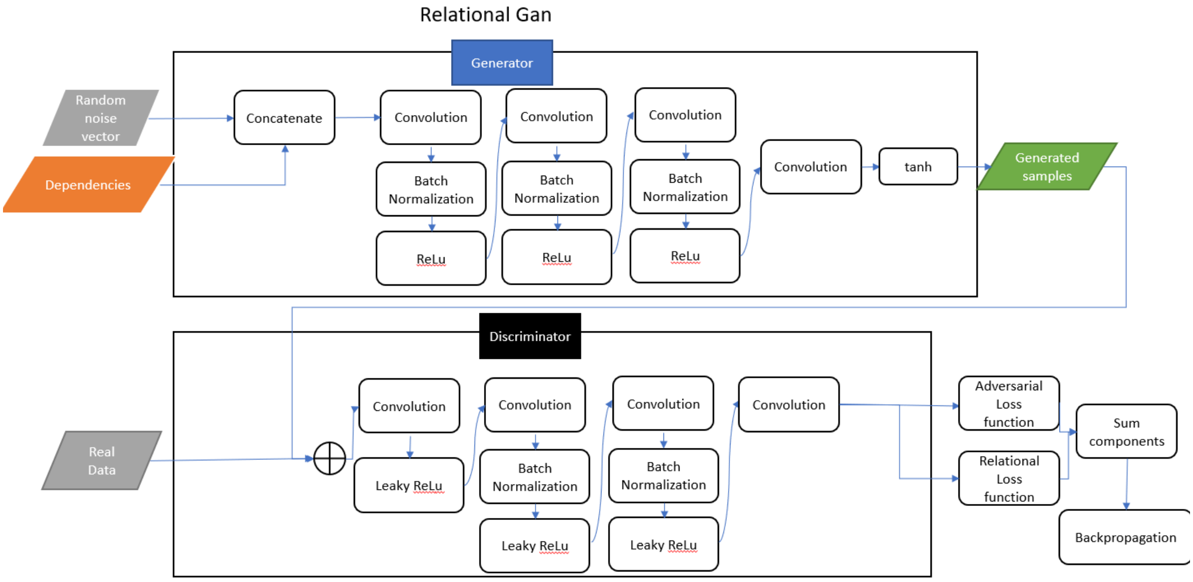
Batch normalization is a training technique that allows deep neural networks to standardize the inputs to a layer for each mini-batch. This stabilizes the learning process and drastically reduces the number of training epochs required to train the networks, which speeds up the training process.
This is done by adding new extra layers. Those new layers perform the standardizing and normalizing operations on the input coming from a previous layer.

2.4 Relational GAN (RelGAN)

The idea behind a Relational Gan is to enhance the generation process by considering the dependencies and interactions between elements of the generated output. The generator has an additional input, which refers to the relational information or spatial dependencies, to guide the generation process. This can be useful in various tasks such as generating images in specific spatial arrangements, or generating data with complex dependencies and more.

This enables a Relational GAN to output more complex and realistic samples compared to standard GANs.

In HouseGAN++, the RelGAN's generator takes as input the relationships between different rooms in the layout, doors, etc..



[Figure 4]

3. Expected Achievements

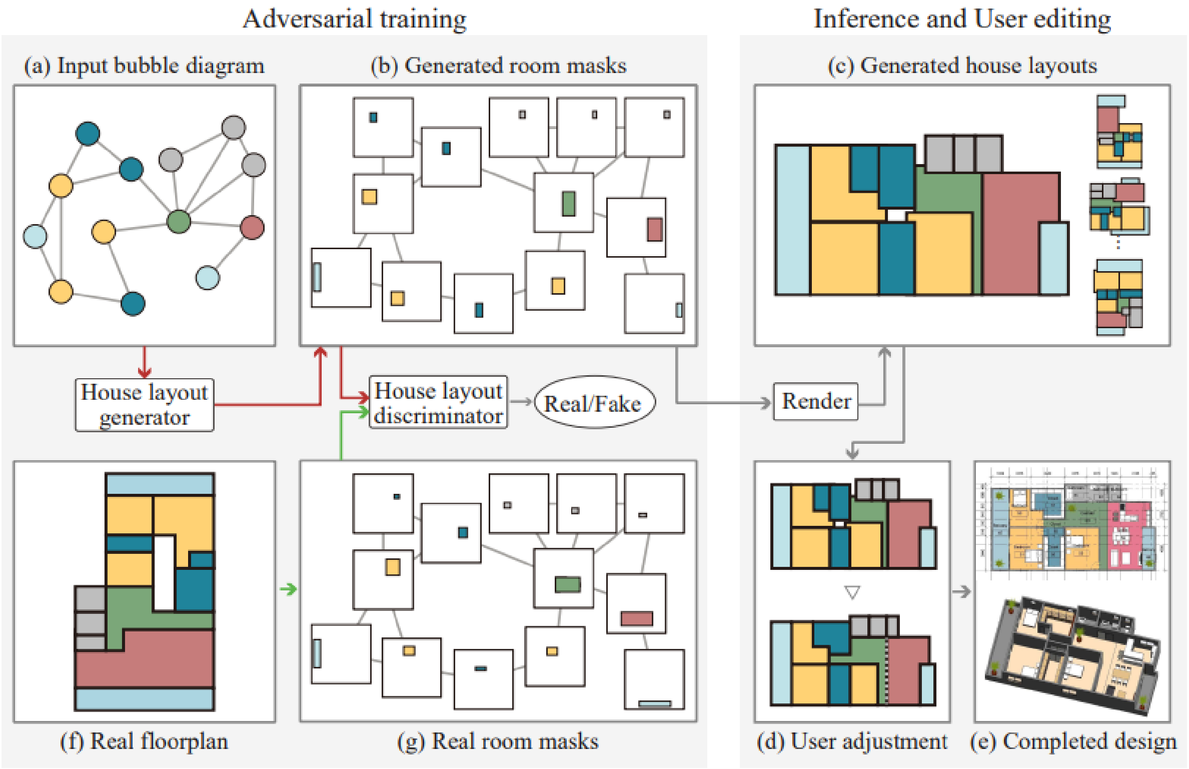
The purpose of the project is to create floor plans in an automatic manner without intervention of architects. It requires research in the fields of deep learning, image processing, and neural networks.

The expectation is to achieve a floor plan from the system that is realistic, and matches the relationships of the bubble diagram that are given.

4. Research Process

4.1 Process

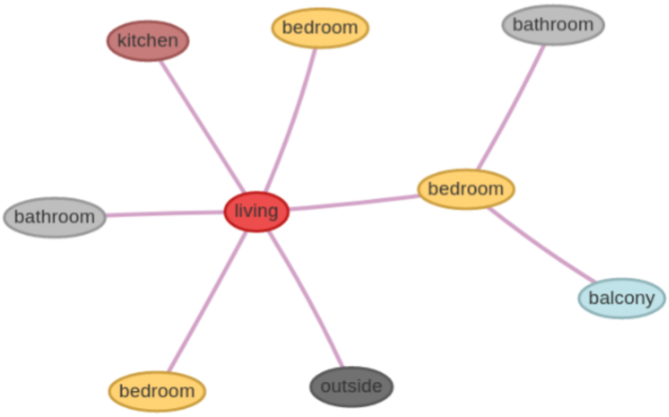
The project purpose is to generate automatic floor plans from a set of conditions given by the user.



[Figure 5] [1] An overview for the process of adversarial training.

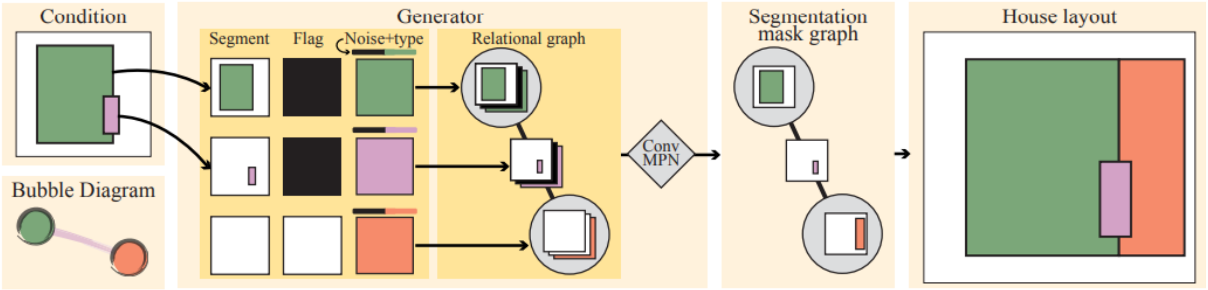
4.1.1 Training Input

The input for the training process is a parsed RPLAN dataset consisting of 60,000 vector-graphics floor plans that were designed by professionals, each was parsed into corresponding bubble diagrams. The conditions are the bubble diagram image The bubble diagram is converted to a graph where each room is parametrized to a node in a graph, and a door is an edge between two nodes. The generator and discriminator start the initial training on the preprocessed data.



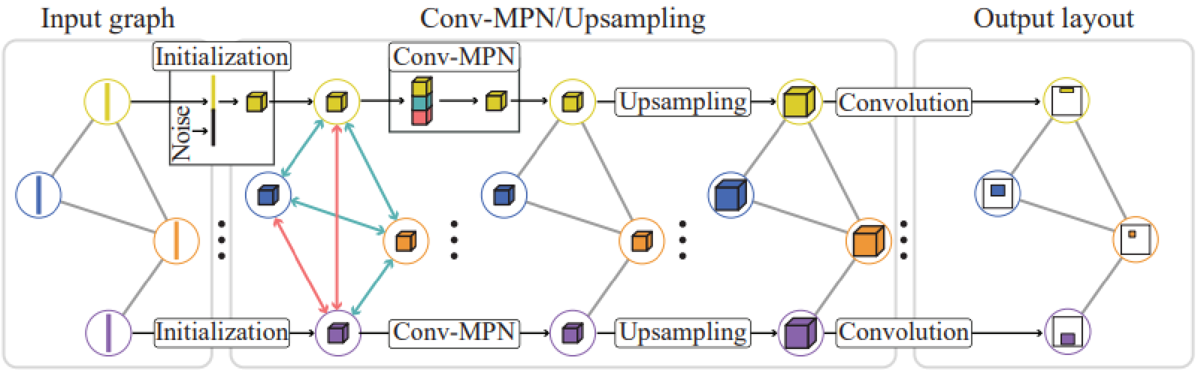
[Figure 6] [104] Introducing the bubble diagram, naturally divided into nodes and edges.

4.1.2. Generator



[Figure 7] [4] Illustrates the process of the generator, including the concatenation of the noise vector and transformation of the segmentation mask and feature maps. Also, the relational graph is included, for the tracking of the relationships between components in the floor.

The generator's role is to generate floor plans such that the discriminator will output that they are human made. The generator has 3 steps using the Conv-MPN module.



[Figure 8] [1] demonstrating the process of the Conv-MPN. Input graph (nodes+edges) is delivered to the Conv-MPN module, maintaining the relational structure of the rooms and doors. Each room is given with additional information, and then the extended data graph undergoes 3 operations (concatenation and convolution) the result gets the last convolution and output layer of a graph of segmentation masks that will be provided to the discriminator.

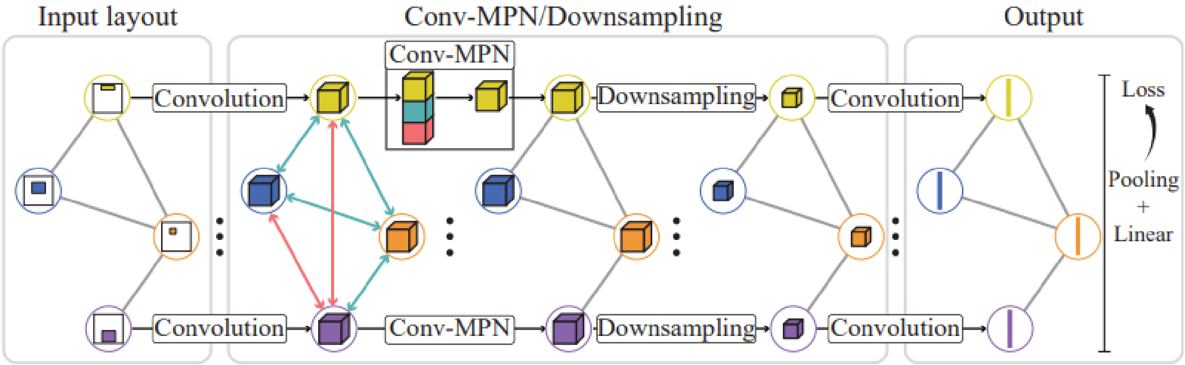
1. It will handle the creation of feature volumes. It preprocesses the graph such that each node (with its' segmentation mask of wall boundaries) is concatenated with additional data of a 12-d one-hot vector that determines the type of room/door associated with the node and concatenates to it a noise vector sampled from normal distribution to enable self learning. An L1 linear reshape is applied to transform to a feature volume, also, the condition image is given another layer (64x64x2), where the first channel provides the segmentation mask, and the second channel is 1 for every pixel if the segmentation mask is specified, otherwise 0. When the pixels are 1, The segmentation mask will be given to the conditional part of the GAN. It is provided with probability of 0.5, this method is dubbed component-wise gt-conditioning.

2. It applies CNN on 3 concatenated components: feature volume, sum pooled features from connected nodes, and sum pooled features of non connected nodes.

$$\text{feature-volume}_r \longleftarrow \text{CNN}[\ \text{feature-volume}_r\ ;\ \text{Pool(connected)}_{s \in u\ ;}\ \text{Pool(Unconnected)}_{s \in \bar{u}}\ ]$$

3. The final segmentation mask is computed as such: Apply a transposed convolution (kernel=4, stride=2, padding=1), this upsamples features by a factor of 2, while maintaining the number of channels. The generator has three rounds of this upsampling, making the final feature volume g_r of size (64x64x32).
Lastly, a shared 3-layer CNN converts that same final feature volume into a room segmentation mask of size (64x64x1), which will then be passed on to the discriminator during training.

4.1.3 Discriminator



[Figure 9] [1] Demonstrating the process of the discriminator, similar to the generator.

The discriminator is similar to the generator. The input is a graph of room segmentation masks. A segmentation mask is of size (64x64x1). A 12-d one-hot vector is taken, and a linear layer is applied to expand it, then reshaped to a (64x64x8) tensor, which is concatenated to the segmentation mask resulting in a (64x64x9) tensor.
A shared three-layer CNN is used to convert the feature into a size (64x64x16), and then three rounds of Conv-MPN and downsampling. The downsampling is done by a factor of 2 using a convolution layer, resulting in a tensor of size (8x8x16), this is then converted into a 128-d room vector. Lastly, all the room vectors are sum-pooled, and a single linear layer is added in order to output a scalar d.

Avoiding overfitting:
In order to avoid a network from memorizing and copying layouts, the same method is used as in House-GAN, which is k-fold cross validation, while dividing the samples into four groups, according to the number of rooms (5,6,7,8). A model is trained while excluding samples in the same group, so that it cannot simply memorize a layout. For example, when training for the generation of layouts with 8 rooms, the samples of the other groups are used in training.[4]

4.2. Architecture Overview

4.2.1 Convolutional Message Passing Network (Conv-MPN)

This network is used in order to find the boundaries of the rooms.
MPN is a type of neural network designed to propagate information through nodes and edges of a graph. This allows the exchange information between nodes to capture relational dependencies and interactions between them. Each node receives information from its neighbors, then updates its own state by incorporating the features of the neighbors.
conv-mpn differs from traditional mpn by that the features of a node are no more 1-d vector but a feature volume of 3D as in CNN. It learns to infer relationships of nodes by exchanging messages.
Its' main work is by recognizing candidate room edges and room corners, then a feature volume is initialized for each room candidate, convolution filters are applied three times and a convolution decoder returns the confidence of the node being a room corner, or an edge that connects two rooms [8].

4.2.2 Refinement-Scheme

House-Gan++ is based on state of the art House-GAN paper. It differs in the architecture of the GAN and in its meta-optimizing refinement scheme.
The refinement scheme discussed here has three strategies: Fixed Heuristics, Static Scheme and Dynamic Scheme.
The last two schemes can also be meta-optimized as will be described in the next chapter.[4]

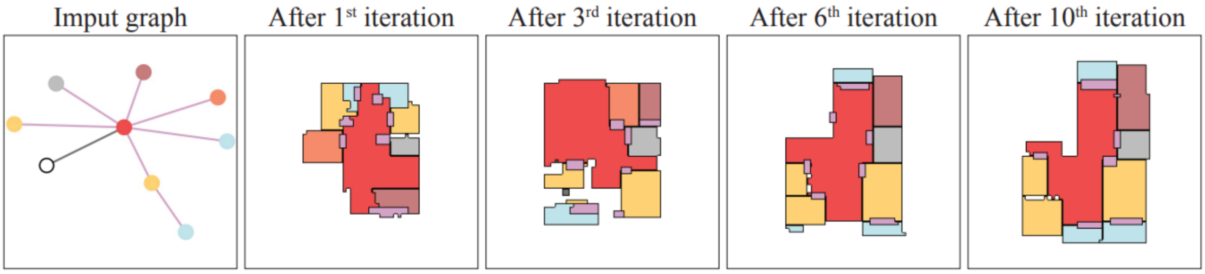
Fixed Heuristics

A fixed heuristic is a predefined rule that does not get changed during the running of the code, this is why this method is not going to undergo a process of meta-optimizing. The rule is providing the segmentation mask to the generator every iteration at a 50% chance.

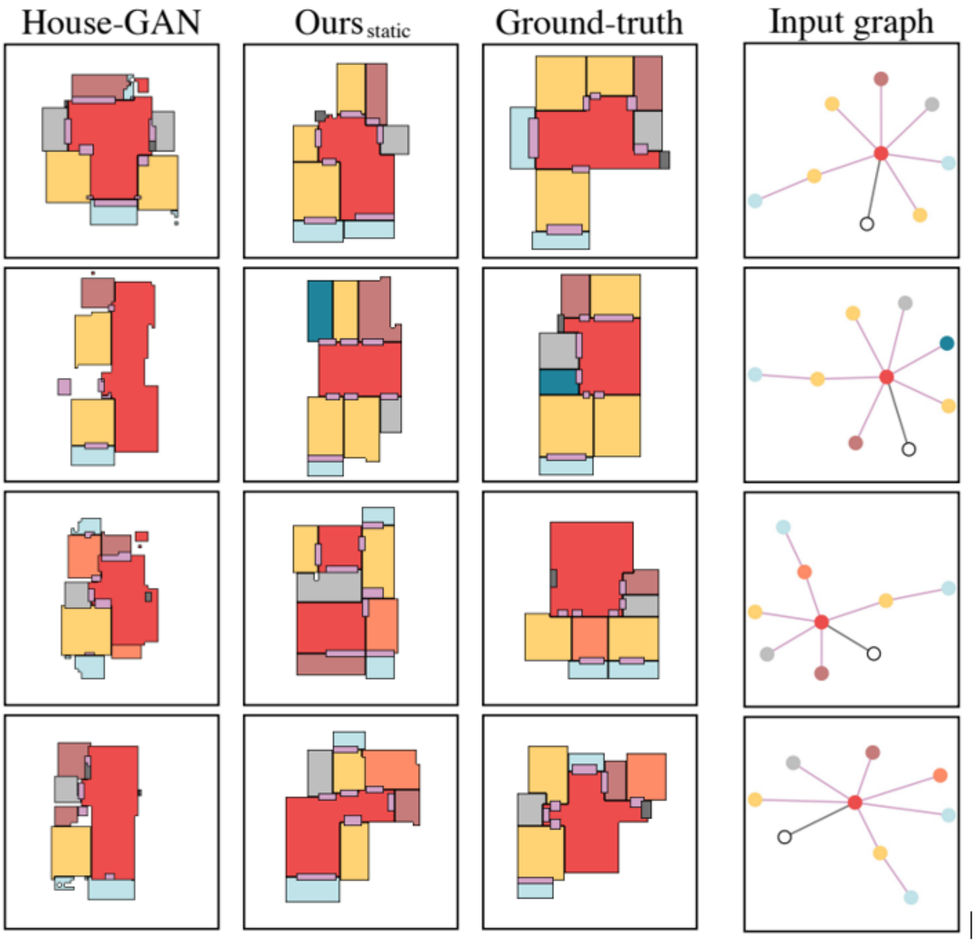
Static Scheme

This scheme is based on the reduction that architects start from designing specific rooms before other rooms. Keep in mind that the model is adding another room at each iteration. Intuitively, the master-bedroom is being designed before a bathroom.
The scheme is parametrized by a 12-d vector of 10 rooms and 2 doors. For example, if V[3] is 4, then it means that the previous segmentation mask that the generator manufactured will be transferred as a condition for the 3rd room type, only after the fourth iteration.
This method compels the generator to receive previously generated segmentation masks only after certain rooms were already placed.

For example, the living room was the first room to be placed, then the kitchen, and lastly, the bathroom. In the iteration of the bathrooms, the segmentation mask was provided to the generator, this obliges the generator to stick to the wall boundaries that were already set for the previously settled rooms.



[Figure 10] [4] Shows the iterative process of the static refinement scheme. The living room (red) is seemingly static in its position while the balcony is moving from iteration to iteration. This probably indicates that the balcony's segmentation mask was handled in late iterations.



[Figure 11] [4] This figure shows how the process of providing the segmentation mask only 50% of the time affected the output floor plan. We can see that the floorplan generated with the Ours static is much more realistic, with less gaps between rooms and less isolated rooms. In terms of realism, Ours static method is competitive against ground truth layouts.

Dynamic Scheme

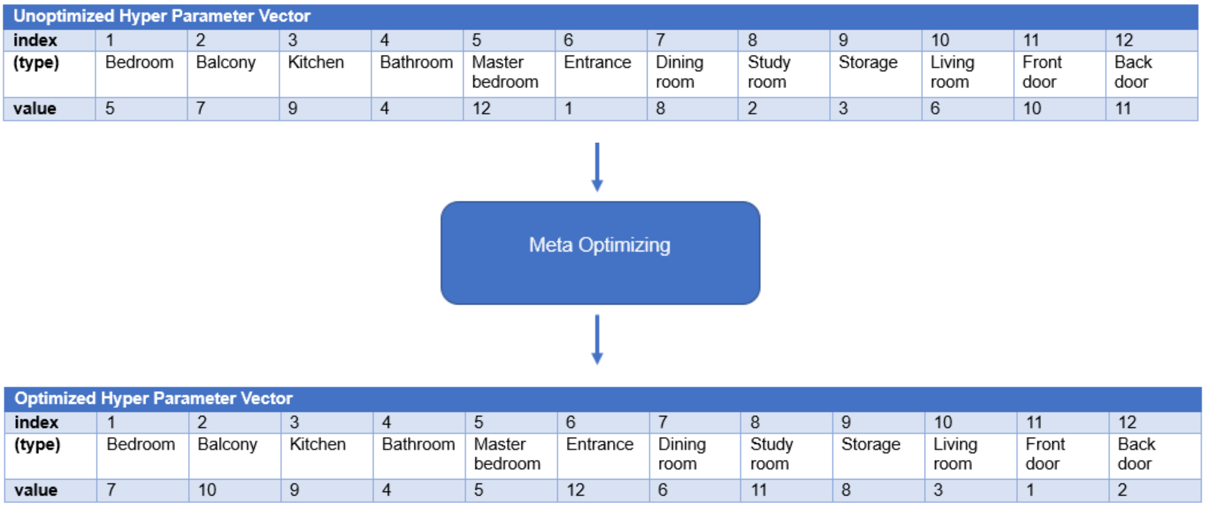
This scheme is based on the compatibility of the current model with the input bubble-diagram. It'll be parameterized by two 12-d vectors dubbed T and U where T is for rooms/doors that are compatible with the input-diagram and T is the opposite. The rest is the same as the static scheme - the previously generated segmentation mask is provided to the generation in the making of the i'th room after the T[i] iteration.

4.2.3 Meta-Optimizing

4.2.3.1 Introduction

Meta-Optimizing is the process of optimizing another optimization algorithm, here is optimizing the refinement-scheme technique. Meta-Optimizing claims to handle two challenges. Finding the group of parameters that provide the best results.[4] Distinguish whether an algorithm performance is genuinely great or just very good fine-tuned for certain data sets.

As discussed, the schemes that are being meta-optimized are the static and dynamic schemes, both of them use a vector. Each cell of it signifies at which iteration the generator is given with the previously generated segmentation mask.



Unoptimized Hyper Parameter Vector

index	1	2	3	4	5	6	7	8	9	10	11	12
(type)	Bedroom	Balcony	Kitchen	Bathroom	Master bedroom	Entrance	Dining room	Study room	Storage	Living room	Front door	Back door
value	5	7	9	4	12	1	8	2	3	6	10	11

Optimized Hyper Parameter Vector

index	1	2	3	4	5	6	7	8	9	10	11	12
(type)	Bedroom	Balcony	Kitchen	Bathroom	Master bedroom	Entrance	Dining room	Study room	Storage	Living room	Front door	Back door
value	7	10	9	4	5	12	6	11	8	3	1	2

[Figure 12] In this vector we made, it can be seen that in the first place, the chosen order for providing segmentation rooms has no logic in it, the living room is one of the most important rooms and should be considered in the first iterations. After meta-optimizing, the doors are created first, the living room afterward. It was found that rooms had little effect on the loss function. [4]

4.2.3.2 Bayesian Optimization

The main character in meta-optimizing is Bayesian optimization. Its tuning uses a continually updated probability model to pick promising hyperparameters by concluding from results that were accumulated from the past.[2] this approach has four conceptual parameters:

4.2.3.3 Null Distribution specification language

Starting configurations that will be used for an unsearched space. The researcher has its own intuitive approaches to the problem so he may feed the system with his already-searched hyper parameters. These parameters specify a joint distribution space, we will name it G.

4.2.3.4. Loss Function

The loss function also named Objective function is the target to minimize. In House-GAN++ it is necessarily to have the lowest amounts of positive diversity and compatibility from generated floor plan to a ground-truth floor plan so the role of the loss function is to eliminate G specifications that exceed a threshold for diversity and compatibility. The loss function is being used during the HOA phase as described next.

4.2.3.5 Hyper Parameter Optimization Algorithm (HOA)

HOA takes initial distribution G alongside with H - the history of pairs of (G, H) of the loss function and returns what is the next ideal g to examine by finding the minimal ratio.
In House-GAN++ TPE is used as HOA. It uses values from the Null Description language, it uses Gaussian Mixture Models (GMM). It is a probabilistic model which assumes all data points are generated from a mixture of gaussian distributions (normal distribution, bell) with unknown mean and variance (parameters).

TPE will sample hyperparameters from the null description space by running the objective function with the hyperparameters he chose, then it creates two models:
l(x) - for hyperparameters that performed well - smallest loss function
g(x) - for hyperparameters that performed bad - highest loss function.
HOA chooses the hyperparameter x that maximizes the ratio between the first model and the second model - $\frac{l(x)}{g(x)}$.

4.2.4. Metrics

Explaining about the different schemes for HOA requires a brief explanation about the evaluation of the system which is based on three metrics: realism, diversity and compatibility.[4]

4.2.4.1. Realism

Realism is the average rating of amateurs and professionals about how practical the layouts are. The users are asked about how realistic the output is for both ground truth and output layouts.

4.2.4.2. Diversity

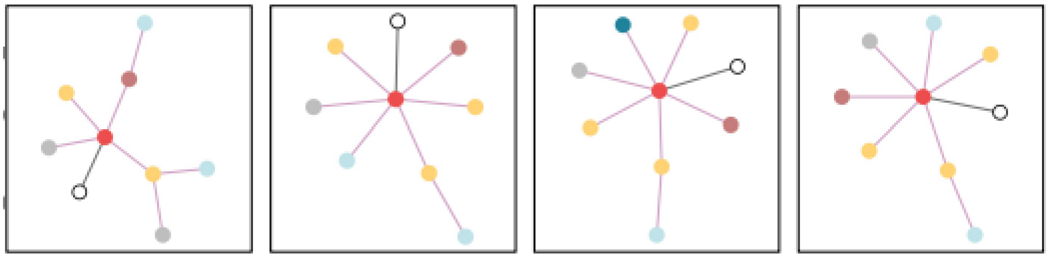
Diversity is measured by Frecht Inception Distance (FID). FID compares the distribution of generated images with the distribution of a set of real images ("ground truth"). the FID is calculated as follows: for

any two probability distributions μ, ν over R^n having finite mean and variances, their Fréchet distance is: $d_F(\mu, \nu) = inf_{\gamma \in \Gamma(\mu, \nu)} \int_{R^n \times R^n} ||x - y||^2 d_\gamma(x, y)^{1/2}$, it is the difference between the mean and variance. [102]

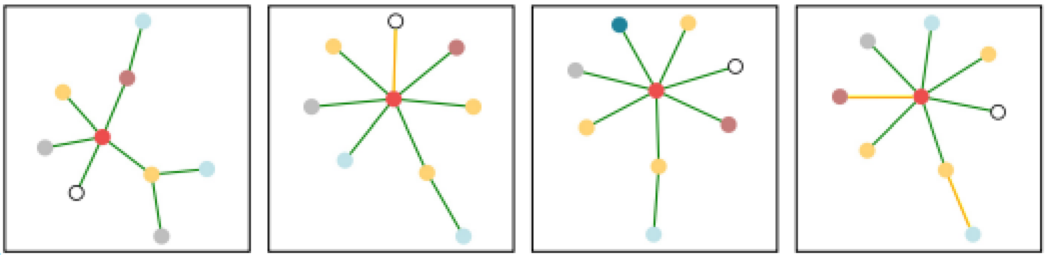
4.2.4.3 Compatibility

Compatibility is the graph edit distance (GED) between the input bubble-diagram and the one constructed from the output layout. It is a measure of the similarity between two graphs computed by the minimum cost of a set of graph edit operations that transform one graph to another. Graph edit operations examples are adding and removing nodes. [3]

Input graphs:



Output graphs:



	Compatibility (lower the better)			
	5 rooms	6 rooms	7 rooms	8 rooms
House-GAN	2.5±0.1	2.4±0.1	3.2±0.0	5.3±0.0

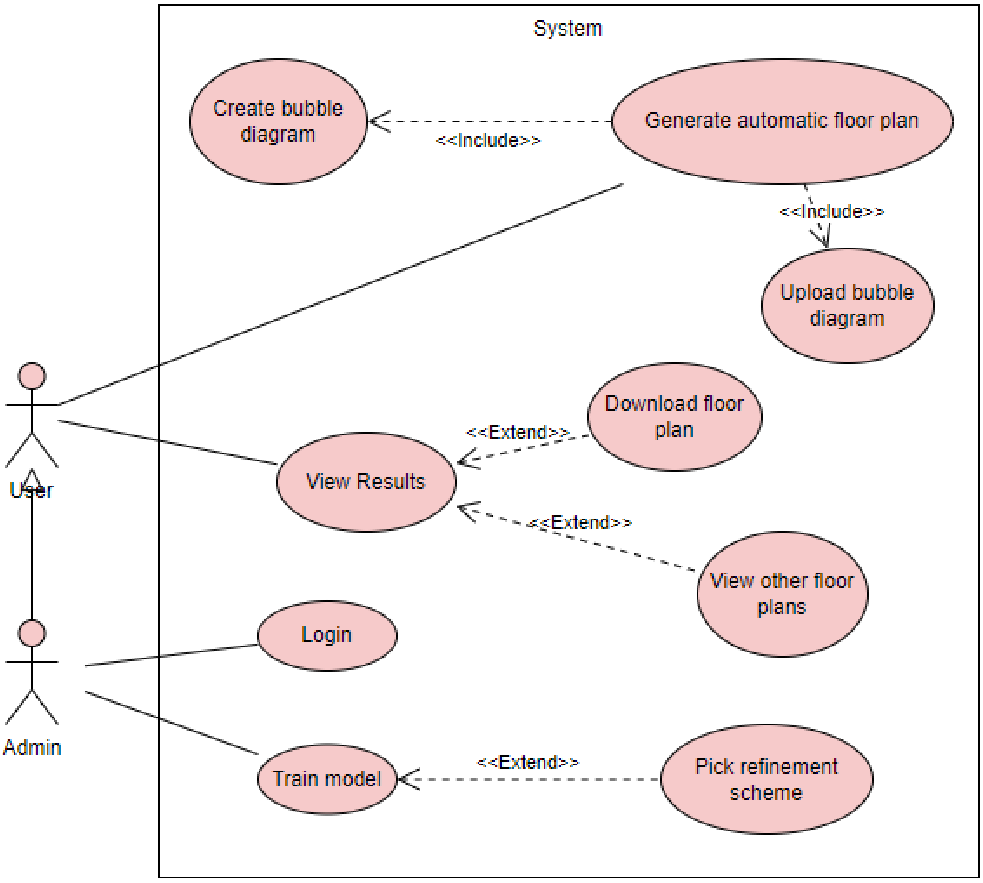
$Ours_{50\%Heur}$	1.9±0.3	2.2±0.3	2.4±0.3	3.9±0.5

[Figure 13] Graph input and output of the system, showing the input graphs to the House GAN ++ system, and the graph that it outputs, are very similar. Also, from the table it is seen the Ours is better than of the original House-GAN.[4]

4.3 Product

The product is a system that implements the model that generates automatic floor plans for any given bubble diagram of constraints. The system will be able to accept hyper parameters. The system will be able to train the model.

4.3.1. Use Case Diagram

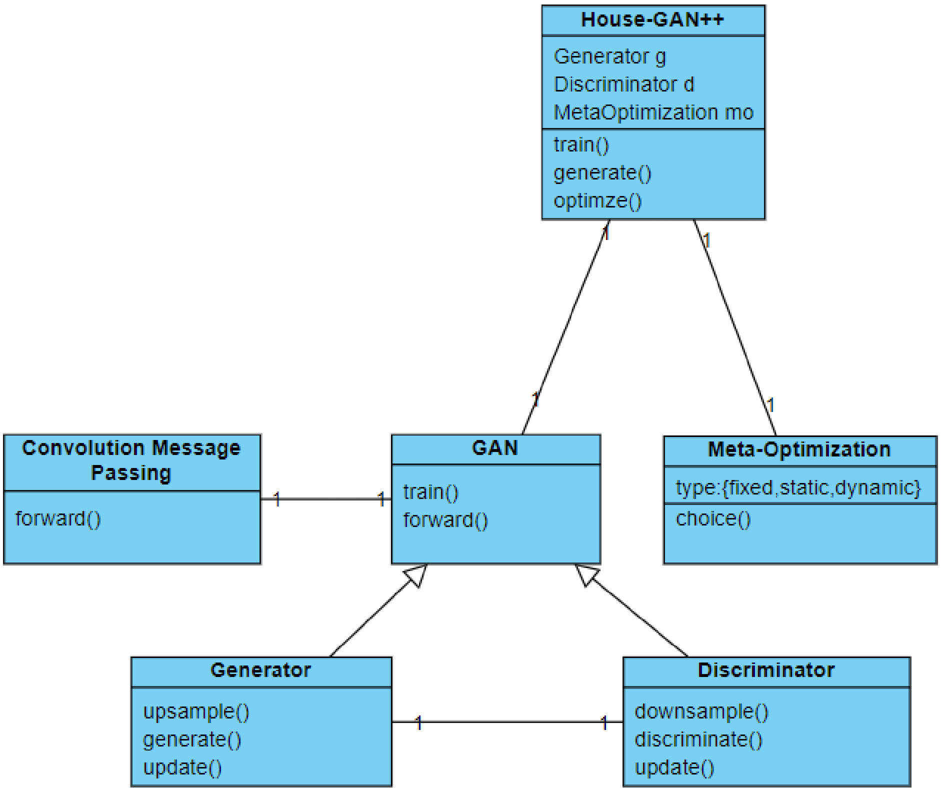


[Figure 14] use-case diagram

Use case	Generate automatic floor plans
Actors and goals	User: create automatic floor plans.
Description	The user can use the model to generate floor plans.

Precondition	User has an idea of the relationships between rooms.
Postcondition	User can view and download the floor plan as jpg.
Trigger	User clicks on the Upload bubble diagram button
Main Success Scenario	1. User opens the application and selects the “Upload bubble diagram” button. 2. System is showing a file selector window. 3. User selects desired bubble diagram from pop-up window. 4. System retrieves the image 5. User can edit the bubble diagram, or select “Generate”. 6. Application uses trained model and feeds it with bubble diagram of the user. Then, outputs a floor plan. 7. User can view and download the generated floor plan.
Branching	1. The selected file is corrupted. 2. The model has not been trained. 3. User exited the program.

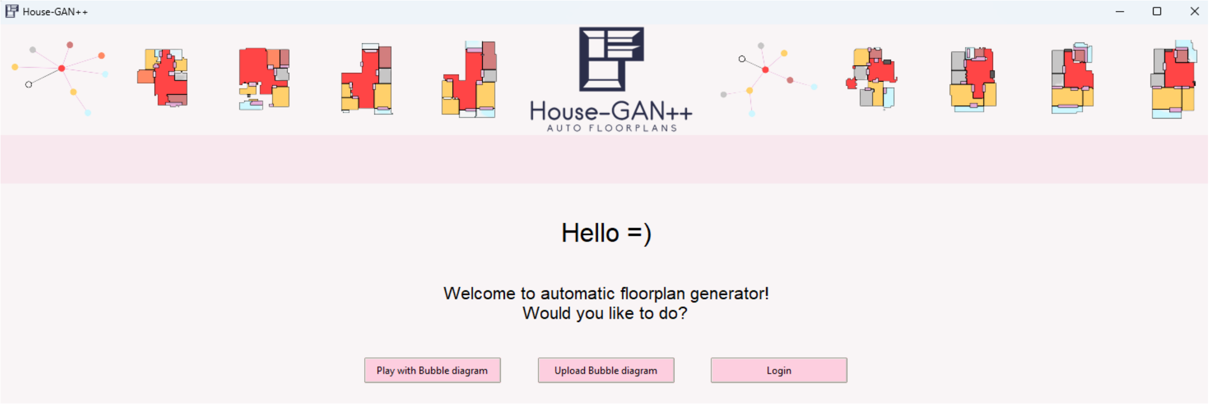
4.3.2. Class Diagram



6. GUI

GUI is developed with python using the tkinter library and a repo for creating graphs[105].

Home Screen: The screen that opens when the application starts.
The user has 3 options: Create a bubble diagram of his own desire, Upload a bubble diagram, or Login as administrator (admin, admin) to achieve admin privileges.

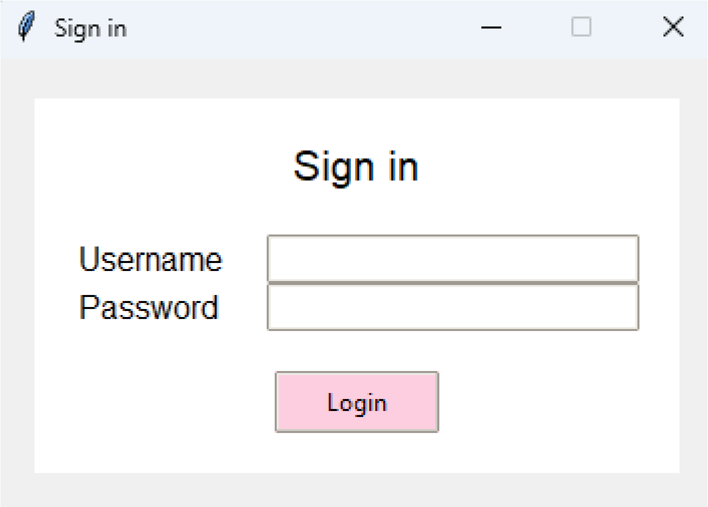


Hello =)

Welcome to automatic floorplan generator!
Would you like to do?

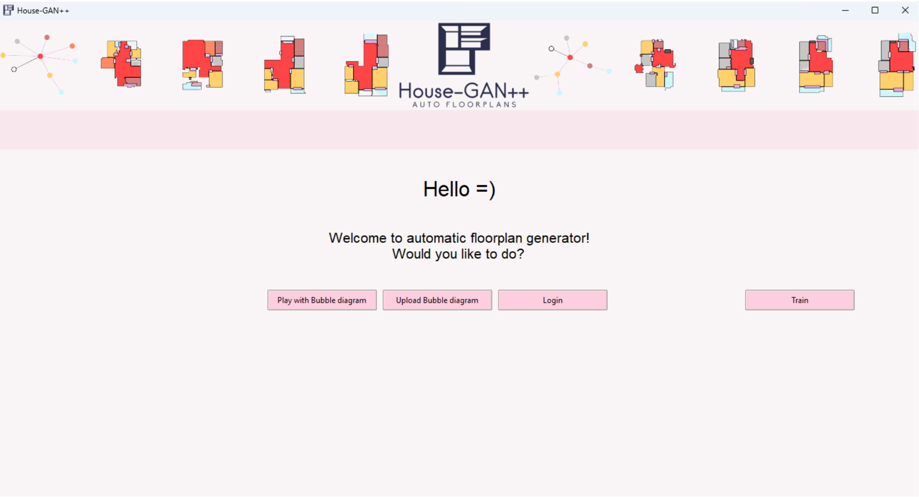
[Figure 15] home screen

Login: When clicking on Login button, a sign in window opens. Use the admin, admin username and password and click on Login.



[Figure 16] login page

Admin Home Screen: Entering the admin username and password will enable admin permissions. Now the admin can choose to train the model.

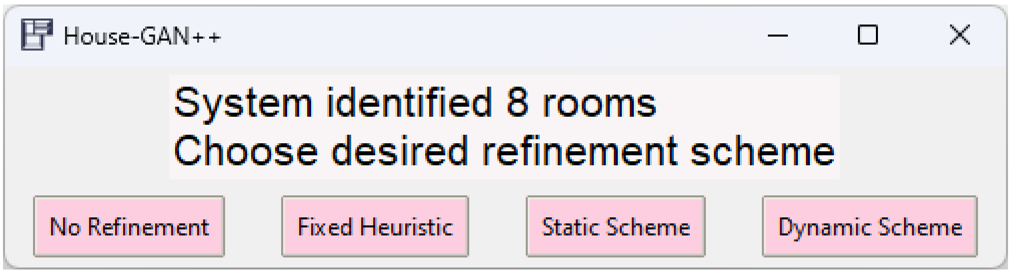


Hello =)

Welcome to automatic floorplan generator!
Would you like to do?

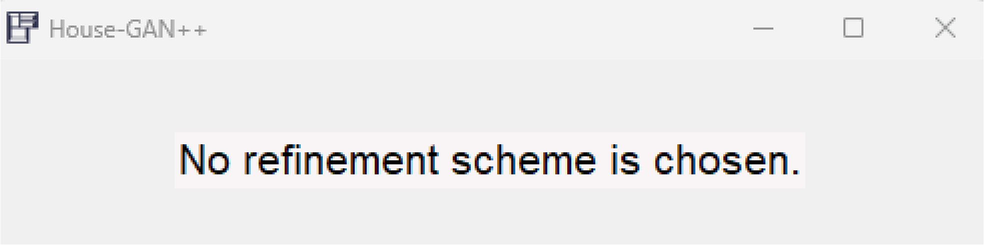
[Figure 17] Admin home screen

Initialize refinement scheme Screen: When admin decides to train the model, he may choose the desired meta-optimization refinement scheme.



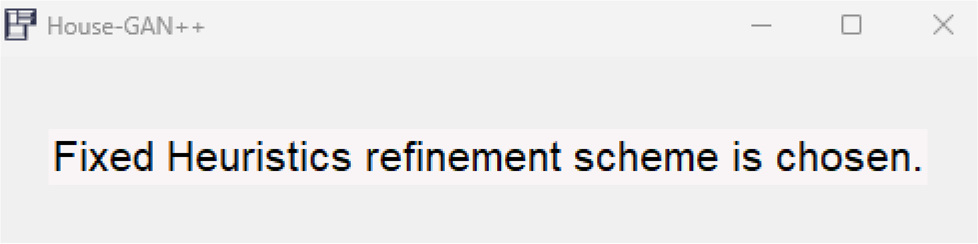
System identified 8 rooms
Choose desired refinement scheme

[Figure 19] Select refinement scheme



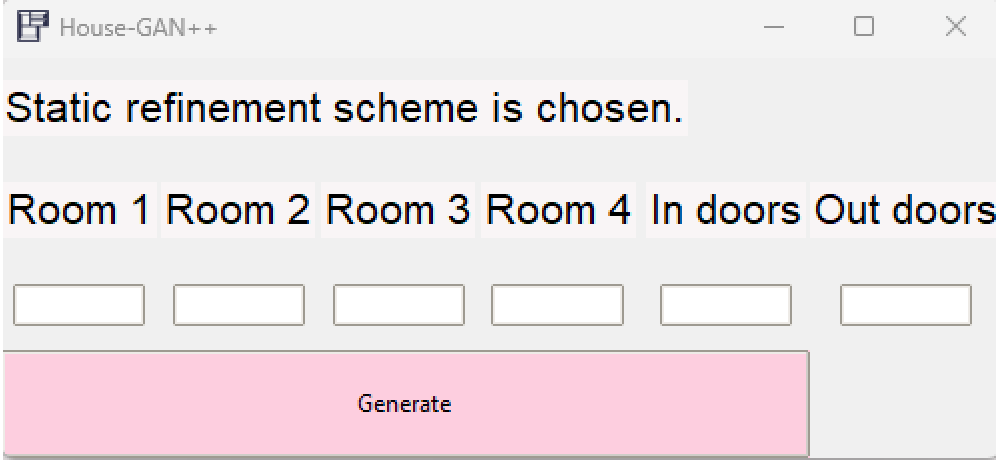
No refinement scheme is chosen.

[Figure 20] Selecting no refinement scheme



Fixed Heuristics refinement scheme is chosen.

[Figure 21] Selecting fixed heuristic refinement scheme



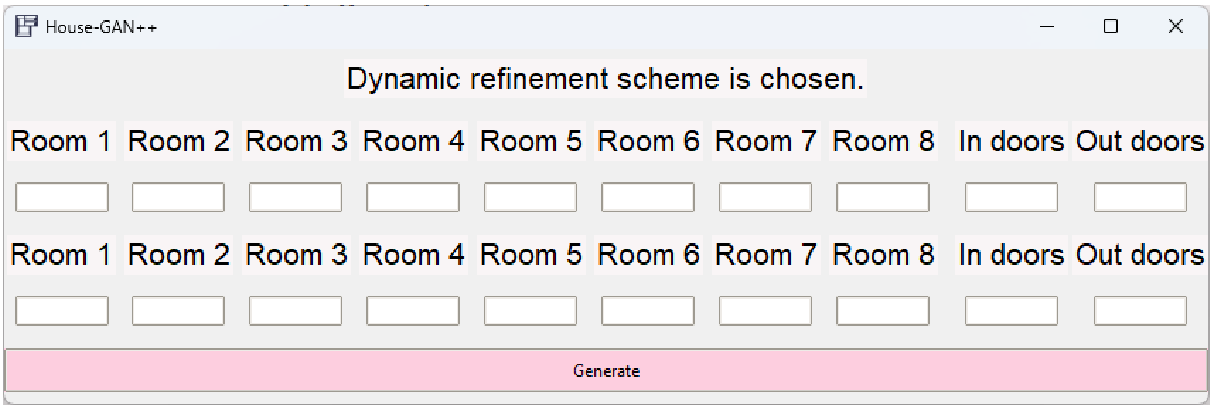
House-GAN++

Static refinement scheme is chosen.

Room 1 Room 2 Room 3 Room 4 In doors Out doors

Generate

[Figure 22] Select static refinement scheme



House-GAN++

Dynamic refinement scheme is chosen.

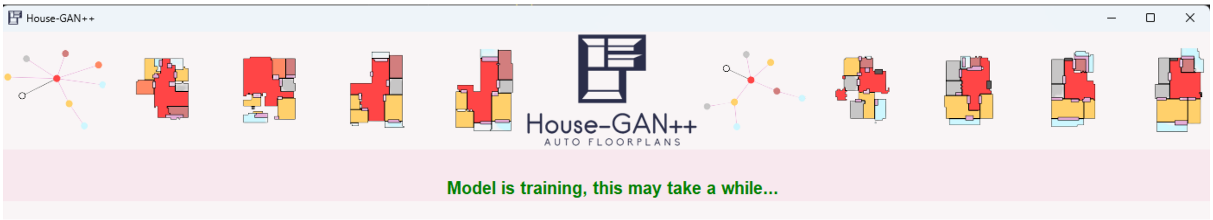
Room 1 Room 2 Room 3 Room 4 Room 5 Room 6 Room 7 Room 8 In doors Out doors

Room 1 Room 2 Room 3 Room 4 Room 5 Room 6 Room 7 Room 8 In doors Out doors

Generate

[Figure 23] Selecting dynamic refinement scheme

When clicking the train model, an appropriate message appears:

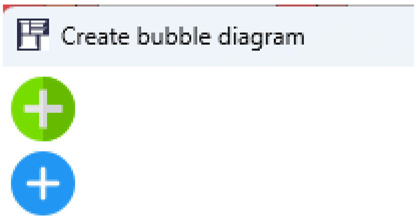


House-GAN++

Model is training, this may take a while...

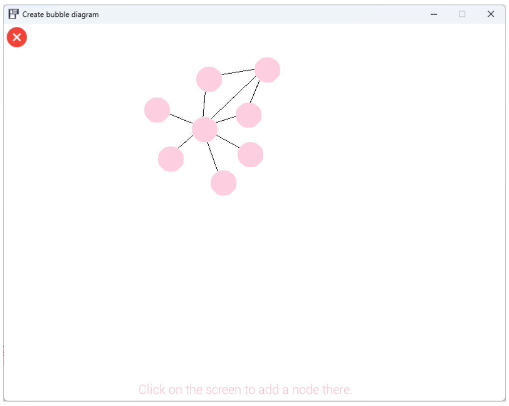
[Figure 24]

Play with bubble diagram: When user or admin click the Play with bubble diagram button, a screen for adding nodes and edges opens



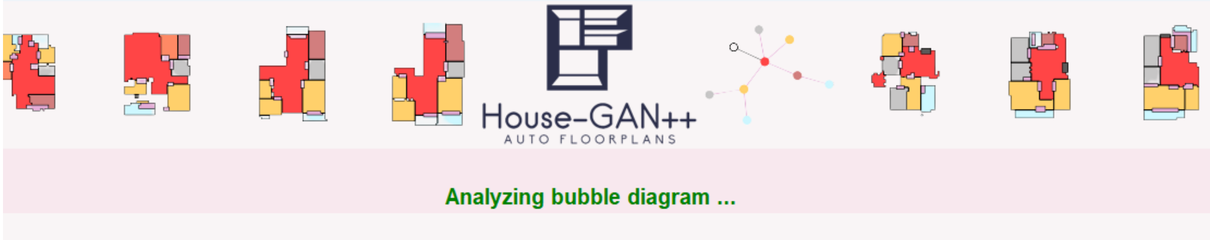
Create bubble diagram

[Figure 24]



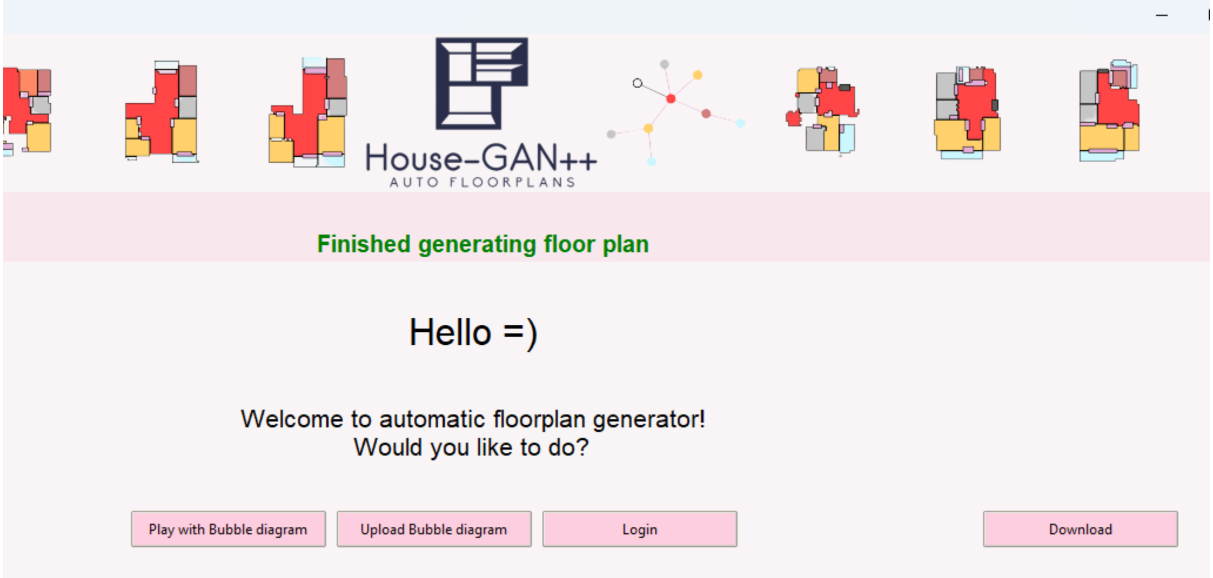
Click on the screen to add a node there.

[Figure 25]



Analyzing bubble diagram ...

[Figure 25] indicates that the application is analyzing the given bubble diagram

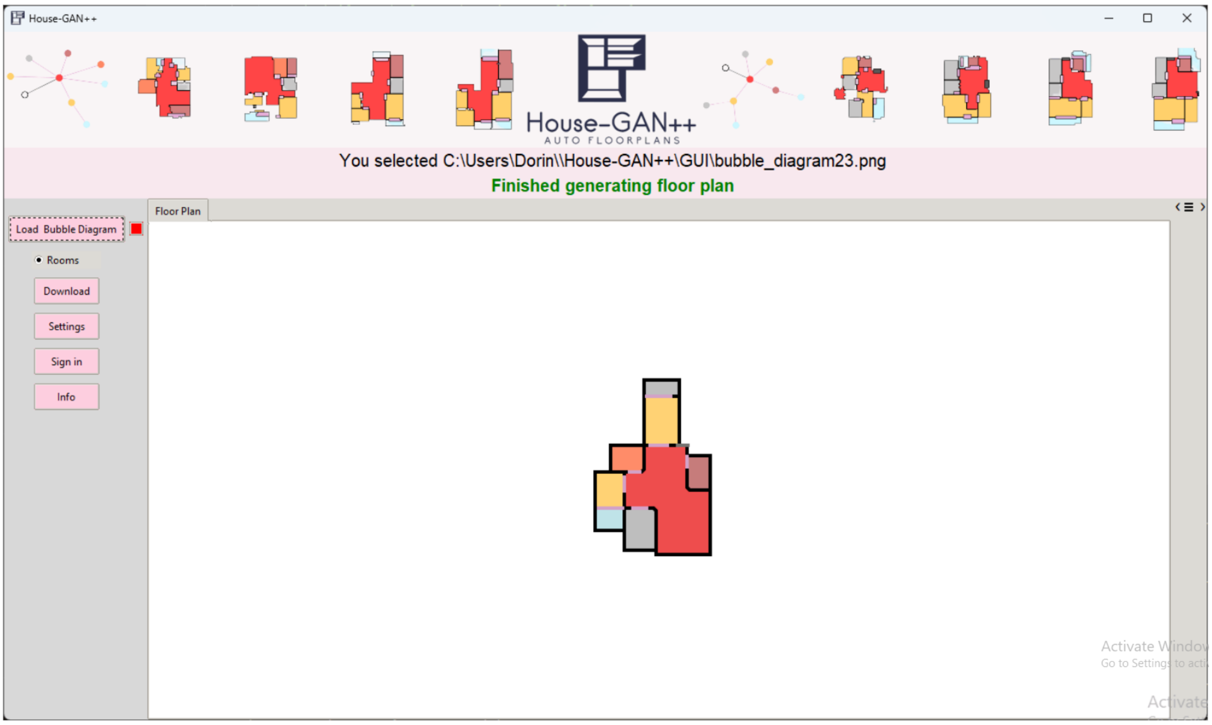


Finished generating floor plan

Hello =)

Welcome to automatic floorplan generator!
Would you like to do?

[Figure 26] User can download the generated floor plan.



You selected C:\Users\Dorin\\House-GAN++\GUI\bubble_diagram23.png

Finished generating floor plan

[Figure 27] The user can watch the floor plan and generate more floor plans that will be added to new tabs

*Clicking on the Download button will save the picture in the users' application location.

7. Evaluation Plan

Test Subject	Expected Result
Generate floor plans from bubble diagram images	A success message is shown, and the generated floor plan is shown on screen.
Download the created floor plan	An image of the floorplan is saved on the selected path of the users' computer.
Loading bubble diagram but not selected a file	A pop up alert is shown to the user.
Loading bubble diagram and selecting file	System analyzes the bubble diagram
Selecting Play with bubble diagram	A window of editing graph opens
Select generate after creating bubble diagram	System analyzes the bubble diagram
Selecting bubble diagram with an untrained model	Red message is shown in the messages frame, indicating that the model is not

	trained yet.
Clicking the stop button	The process currently running is stopped and the selected files are cleared from the program.
Use the Login button	Privilege is added.
Selecting to train the model	The model will be trained.
Selecting static refinement scheme	A window to insert room orders opens.
Selecting dynamic refinement scheme	A window to insert room orders for compatible and non-compatible iterations.

8. References

papers:

[1] Nelson Nauata 1 , Kai-Hung Chang 2 , Chin-Yi Cheng 2 , Greg Mori 1 , and Yasutaka Furukawa 1 House-GAN: Relational Generative Adversarial Networks for Graph-constrained House Layout Generation.

[2] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In International conference on machine learning, pages 115–123, 2013. 5

[3] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. 2015. 3

[4] Nelson Nauata, Sepidehsadat Hosseini, Kai-Hung Chang , Hang Chu, Chin-Yi Cheng and Yasutaka Furukawa House-GAN++ Generative Adversarial Layout Refinement Network towards Intelligent Computational Agent for Professional Architects

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio Generative Adversarial Nets

[6] Mehdi Mirza Conditional Generative Adversarial Nets

[7] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do Kaori Togashi Convolutional neural networks: an overview and application in radiology

[8] Fuyang Zhang∗ , Nelson Nauata∗ and Yasutaka Furukawa Simon Fraser, Conv-MPN: Convolutional Message Passing Neural Network for Structured Outdoor Architecture Reconstruction University, BC, Canada

[9] Michaël Defferrard Xavier Bresson Pierre Vandergheynst, Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

other references:

[100] KNN implementation

[101] How Much Does It Cost To Hire A Floor Plan Designer? Mark Wolfe,Samantha Allen, forbes

[102] Fréchet inception distance

[103] Generative adversarial network, Wikipedia

[104] House-GAN++ Github

[105] Graph-Editor-Tool Github